Training Neuromorphic Neural Networks for Speech Recognition

Rihards Klotiņš

Supervisor: Dorian Florescu

May 20, 2025

Abstract

FILLER ABSTRACT IGNORE - 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.

Contents

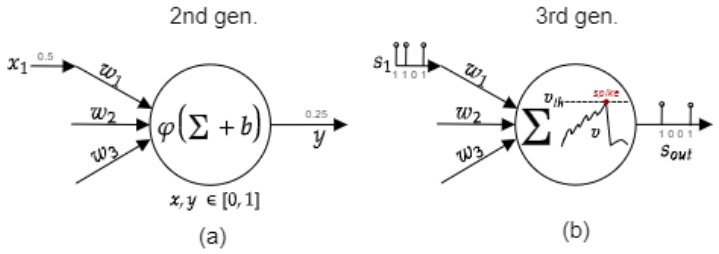


Figure 1: (a) Shows a 2nd generation neuron. (b) Shows a 3rd generation neuron.

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 1 (a)). For a third 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 1 (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.

2 Context

2.1 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.

2.1.1 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 2 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,

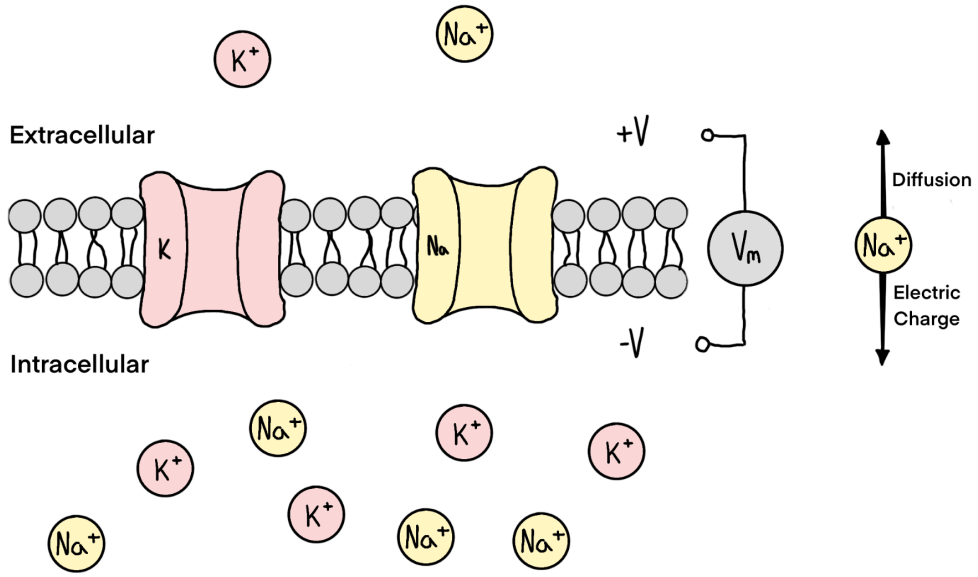


Figure 2: Diagram of neuron membrane.

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 2. 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} 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.

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

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

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

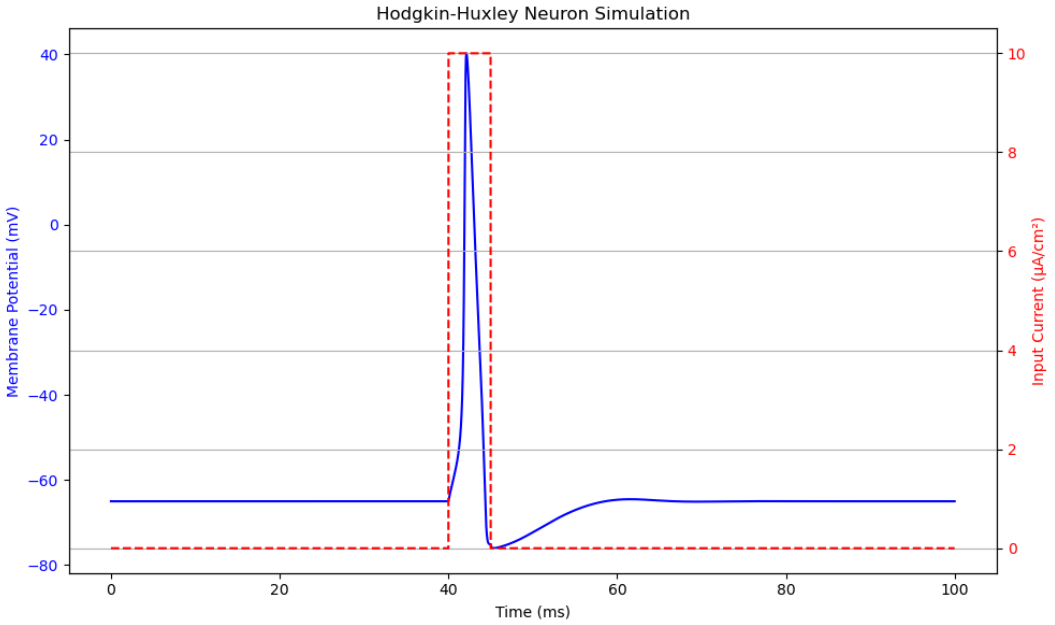


Figure 3: Response of the Hodgkin–Huxley model (blue) to a 5ms step function input current (red).

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 equation (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 $\bar{g}_{Na}m^3h$. The $\bar{g}_{Na}$ term is the conductance of the channel when it is fully open, i.e. the channels maximum conductance. The m^3h 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 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 3, 4, and 5 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.

2.1.2 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 1 (b)) - 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

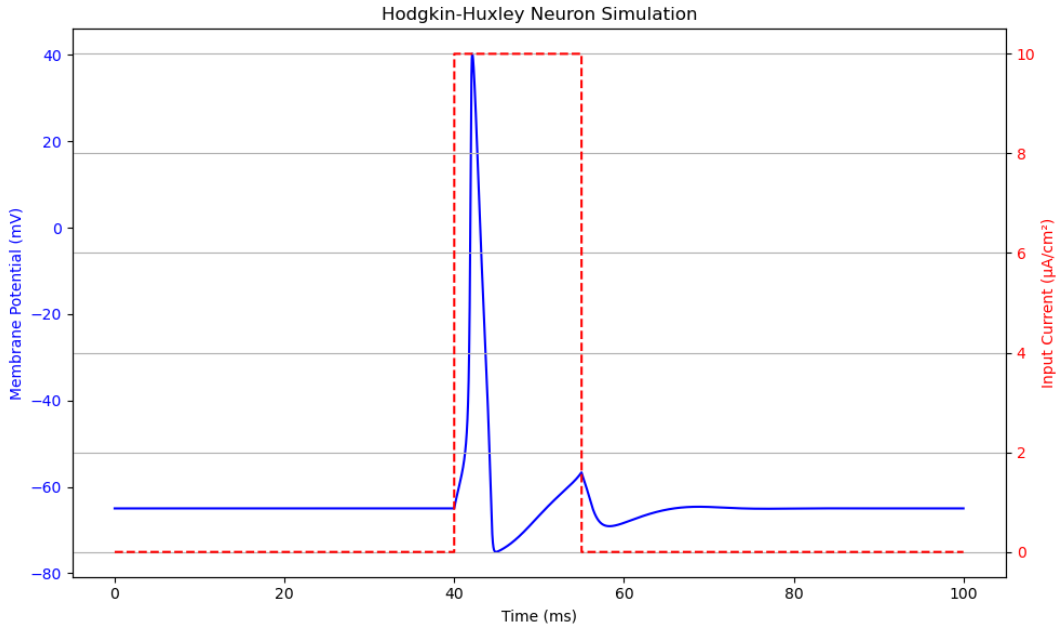


Figure 4: Response of the Hodgkin–Huxley model (blue) to a 10ms step function input current (red).

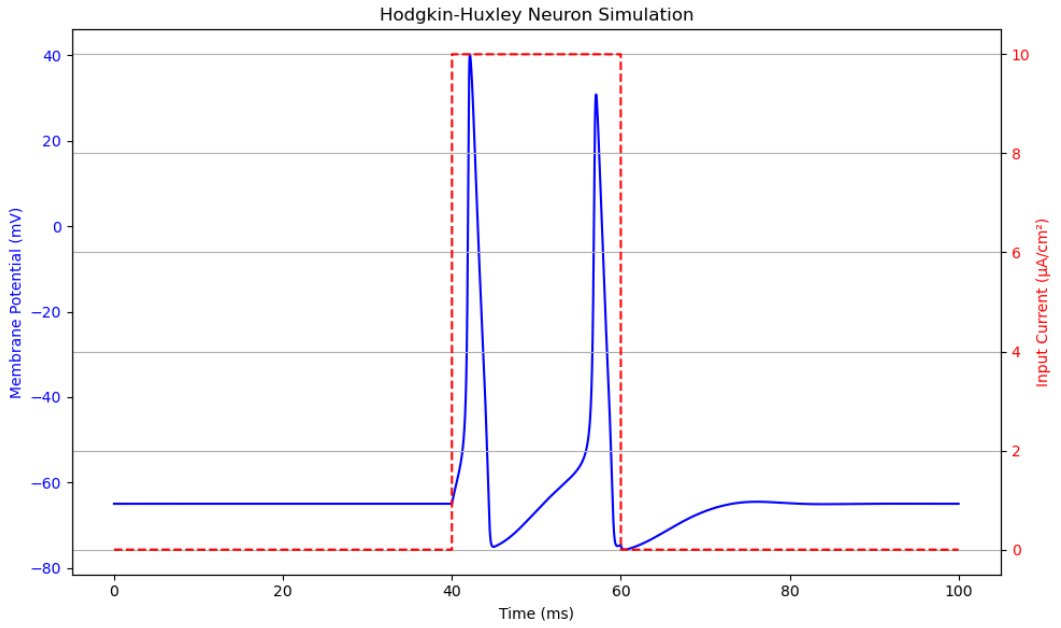


Figure 5: Response of the Hodgkin–Huxley model (blue) to a 15ms step function input current (red).

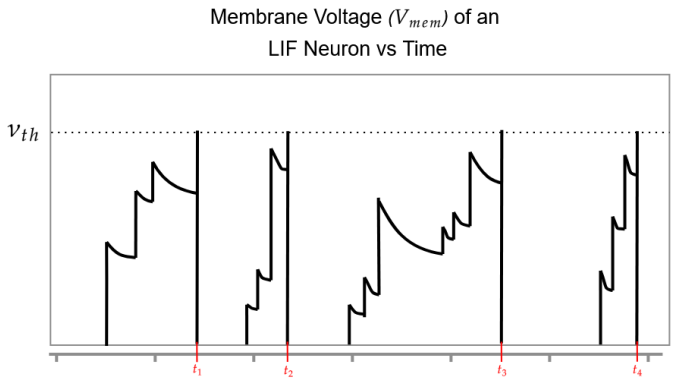


Figure 6: LIF neuron plot of V_{mem}.

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).

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

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

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

2.1.3 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 [6]. 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 [6]. 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 [7].

Mathematically, the model comprises these coupled ordinary differential equations:

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

$$\frac{du}{dt} = a(b * v - u), \tag{10}$$

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 [6]. 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 [7].

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.

2.1.4 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 7). 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 7. 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.

==RECURRENT NEURAL NETWORKS==

2.2 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 [8, 9]. 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.

2.2.1 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.

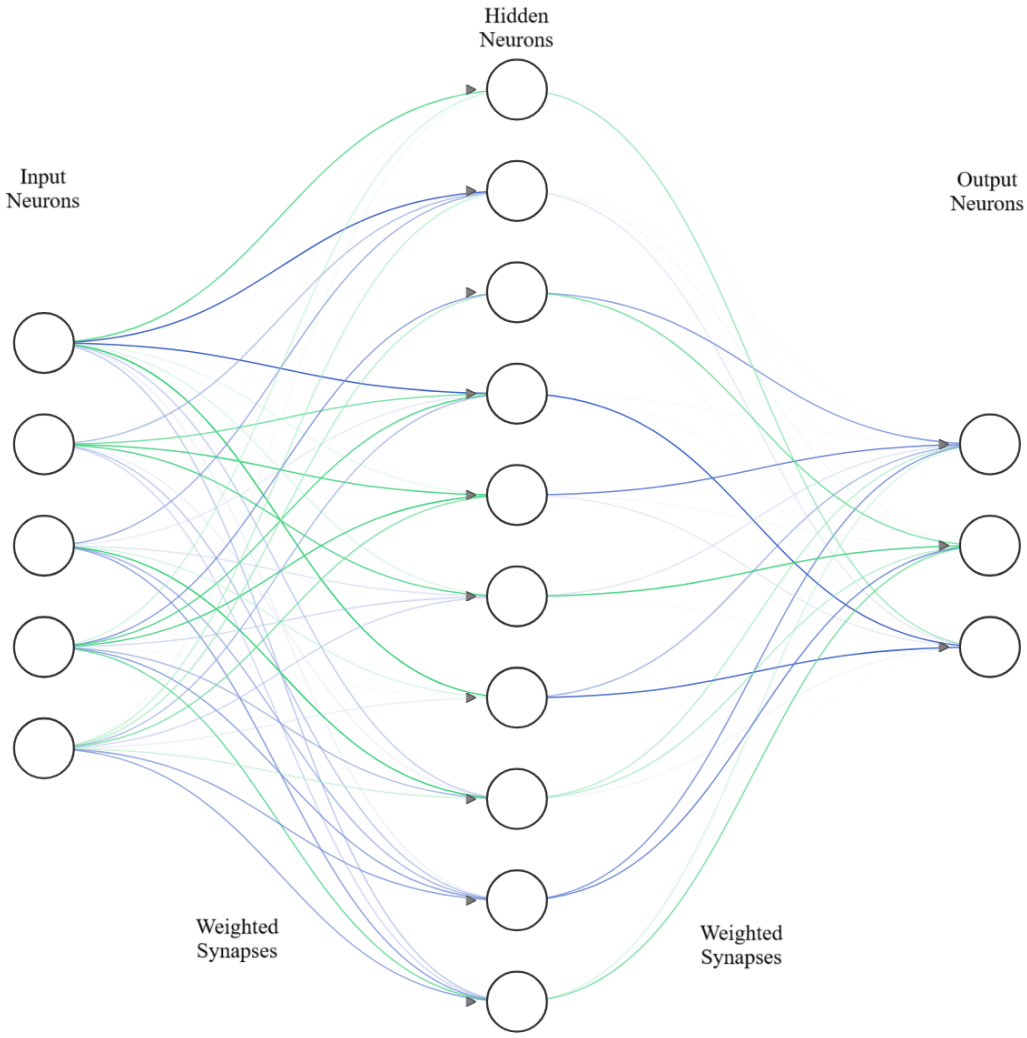


Figure 7: Diagram of a 3 layer neural network. The weights of the synapses are visualised by the colours of the connections.

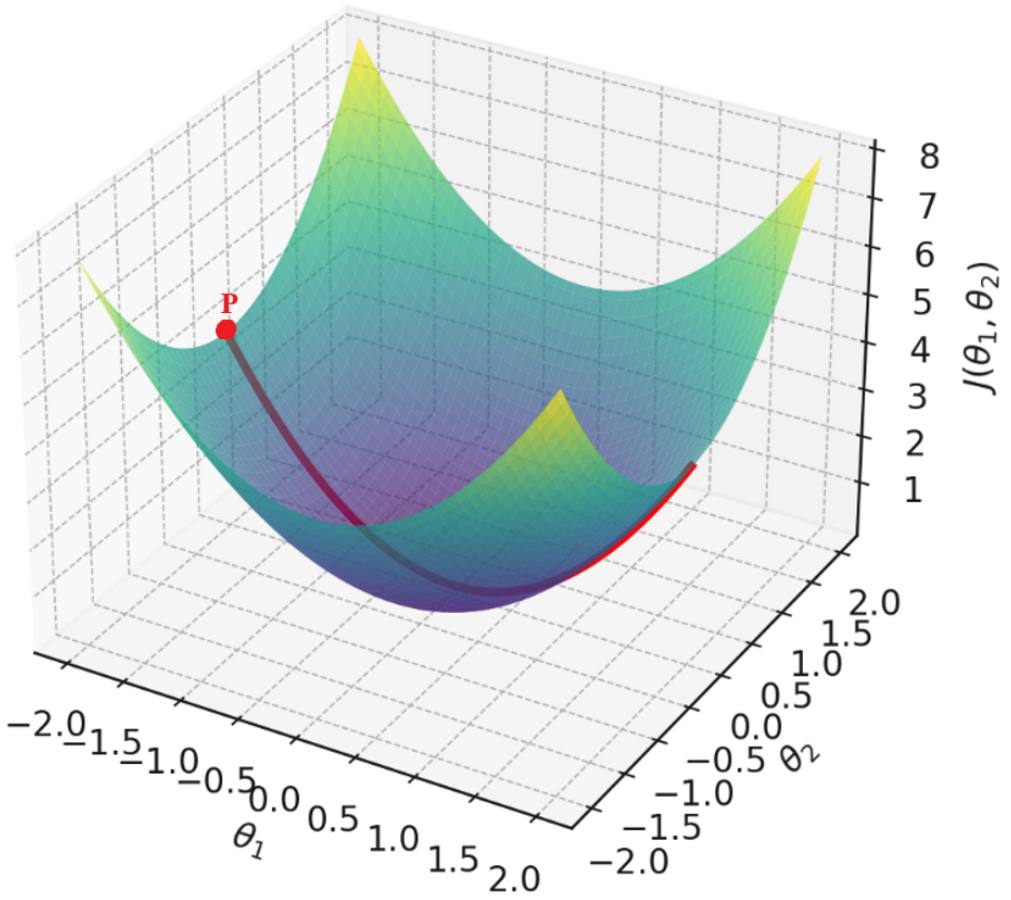


Figure 8: The loss as a function of the parameters θ_1 and θ_2 plotted.

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

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

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 8 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 a simplified, fully-connected network consisting of a layer of input neurons, hidden neurons, and output neurons (Figure 9). Fully-connected means each neuron in a layer is

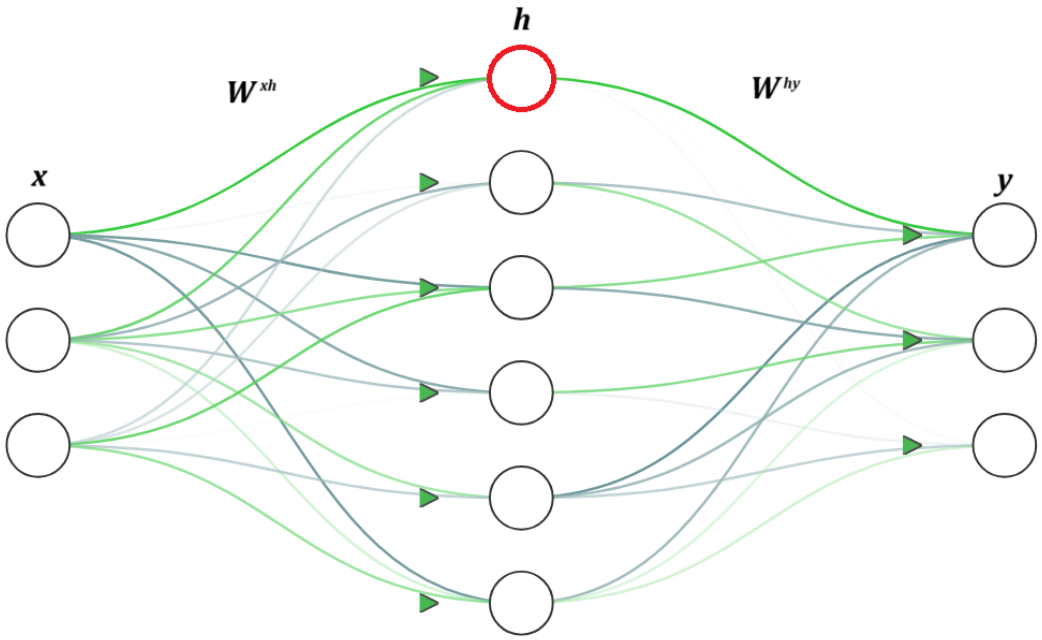


Figure 9: The input. $\mathbf{x}$, is a vector of length 3. $\mathbf{h}$ is a hidden layer of length 6. $\mathbf{W}^{xh}$ is the matrix containing the values of each of the weights connecting $\mathbf{x}$ and $\mathbf{h}$, thus it is of size 3 by 6. $\mathbf{y}$ is the output vector of length 3. $\mathbf{W}^{hy}$ is the matrix containing the values of each of the weights connecting $\mathbf{h}$ and $\mathbf{y}$. thus it is of size 6 by 3.

connected to all neurons in the following layer.

The highlighted neuron, $\mathbf{h}_1$, is connected to all input neurons, $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$. Therefore $\mathbf{h}_1$ is a weighted sum of all the input neurons (Equation 13).

$$\mathbf{h}_1 = \sum_{i=1}^{3} \mathbf{x}_i \times \mathbf{W}_i^1 \tag{13}$$

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

$$\mathbf{h} = \mathbf{x} \times \mathbf{W}^{xh}$$
$$\mathbf{y} = \mathbf{h} \times \mathbf{W}^{hy}$$

These two equations encompass all of the weighted summations that define the network. The relationship between the weighted summation equation (equation **??**) and the matrix multiplication representation of the network is highlighted in figure 10. You can see that each input neuron x_i is multiplied by the corresponding weight $w_{i,1}$ and summed to give h_1.

Matrix multiplications perform the *forward pass* of the network - i.e. the inference. This inferred output value, y, is compared with the true "label" value, $\mathbf{y}_{label}$. The comparison between the actual output and the desired output is dome by a loss function, $l()$. There are many ways of implementing the loss function, one way by finding the mean squared error (MSE):

$$L = l(\mathbf{y}, \mathbf{y}_{label}) \tag{14}$$

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:

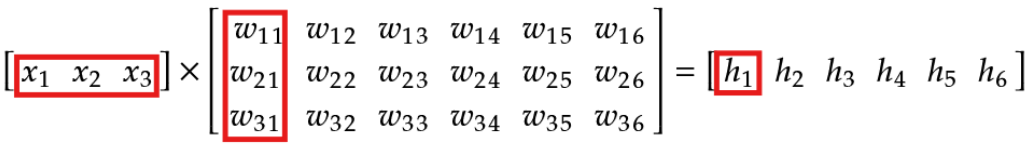


Figure 10: Matrix calculation of neural network values. The highlighted values of vector $\mathbf{x}$ are multiplied by highlighted values of the $\mathbf{W}$ matrix and summed to get $\mathbf{h}_1$.

$$\frac{\partial L}{\partial \mathbf{W}_{ij}} \tag{15}$$

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

$$\frac{\partial L}{\partial \mathbf{W}^{hy}} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial \mathbf{W}^{hy}} = \frac{\partial L}{\partial \mathbf{y}} \cdot \mathbf{h} \tag{16}$$

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

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

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:

$$\mathbf{w}_{ij} \leftarrow \mathbf{w}_{ij} - \eta \frac{\partial L}{\partial \mathbf{w}_{ij}} \tag{18}$$

where η is the learning rate. A higher η would mean faster learning but risks instability. A lower η would ensure stable convergence to the minima, but may make the training very slow.

Repeating this backpropagation 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 popular 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. If there are too many local minima, this approach may not be enough.

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 the following section, we introduce several of these methods, highlighting how they address the challenges of spike-based computation and enable effective learning in spiking systems.

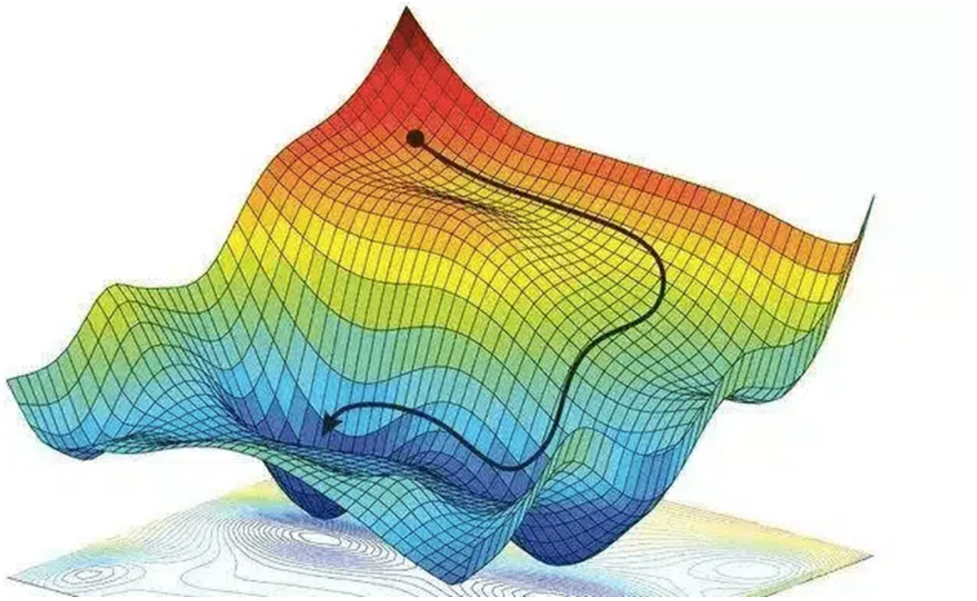


Figure 11: Loss function 3D surface being descended [**?**].

2.2.2 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 [10, 11]. 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 [12]. Thus, to fully harness the benefits of SNNs — from energy efficiency to novel architectures — effective direct training methods are essential.

2.2.3 Backpropagation-Through-Time

Similar to the ANN-to-SNN method, Backpropagation-Through-Time (BPTT) attempts to carry over the effectiveness of the backpropagation algorithm to SNNs. However, this time the SNNs get trained directly which allows them to learn dynamic patterns from temporal data such as speech. Familiar gradient-based methods are applied in tandem with "surrogate gradients" (SG) which are spike gradient approximations used to overcome the fact that spikes are non-differentiable. 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 [13]. 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 [12].

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 [11, 14]. Researchers have even swapped out recurrent layers in end-to-end speech models for surrogate gradient trained spiking modules, showing only small drops in word-error rate and offering a path toward low-power, real-time processing [11].

However, BPTT 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 [14]. Second, because the learning rule relies on a global error signal

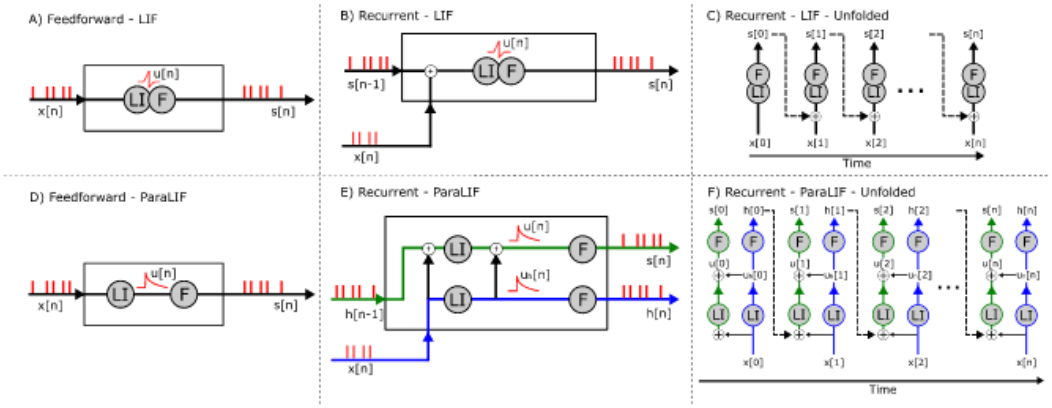


Figure 12: A) Feedforward LIF processing flow. B) Feedforward ParaLIF processing flow. [15]

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.

Parallelisable LIF The major bottleneck in training spiking neural networks (SNNs) using BPTT 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 orders of magnitudes of speed-ups in training, without altering the fundamental membrane-and-spike dynamics that give SNNs their event-driven efficiency [15, 16].

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 parallel computation [15].

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 [15, 16]. 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 [16].

While parallelising neurons has achieved impressive results on the SHD benchmark this method comes with notable disadvantages. By reorganizing the time dimension, ParaLIF departs from the continuous, step-by-step integration that real neurons exhibit, reducing its biological plausibility [17]. 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 modelled [18, 19]. Moreover, 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, a key potential application of SNNs.

2.2.4 Eligibility Propagation

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 while still taking inspiration from the clearly effective backpropagation algorithm. 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 [20] and can dramatically lower energy consumption on event-driven hardware [21, 22]. 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 [20]. 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 [23].

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 [24].

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 [22**?**]. Second, because it employs approximate surrogate gradients rather than true backpropagation, e-prop–trained models typically achieve lower accuracy than their BPTT-trained counterparts [20]. 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 [21].

2.2.5 Spike-Time-Dependent-Plasticity

Spike-timing-dependent plasticity (STDP) represents a training approach for SNNs that draws even greater inspiration from the learning rules observed in biological neurons. STDP is a biological learning rule that adjusts the strength of synaptic connections based on the precise timing of pre-synaptic and post-synaptic spikes. If a pre-synaptic neuron fires just before a post-synaptic neuron, the connection between them is strengthened. Conversely, if the order is reversed, the connection is weakened. This temporally sensitive form of Hebbian learning, often summarized by the maxim "cells that fire together, wire together," operates locally at each synapse and does not require global error signals. This makes it inherently biologically plausible, asynchronous, and capable of unsupervised learning.

Reward-modulated STDP (R-STDP) extends the basic STDP rule by incorporating a global reward or punishment signal that modulates the synaptic weight changes. This allows the network to learn task-specific features by reinforcing connections that contribute to correct outputs and weakening those associated with errors. R-STDP bridges the gap between unsupervised STDP and supervised learning, enabling SNNs to tackle more complex, goal-oriented tasks like speech recognition. The reward signal in R-STDP can potentially be implemented using neuromodulators, further enhancing the biological plausibility of SNN training.

STDP has been employed to train SNNs for speech recognition, often in conjunction with temporal coding schemes such as time-to-first-spike for efficient processing. It has been used for unsupervised feature extraction from speech signals, with subsequent classification using methods like Hidden Markov Models (HMMs) or tempotrons. STDP-

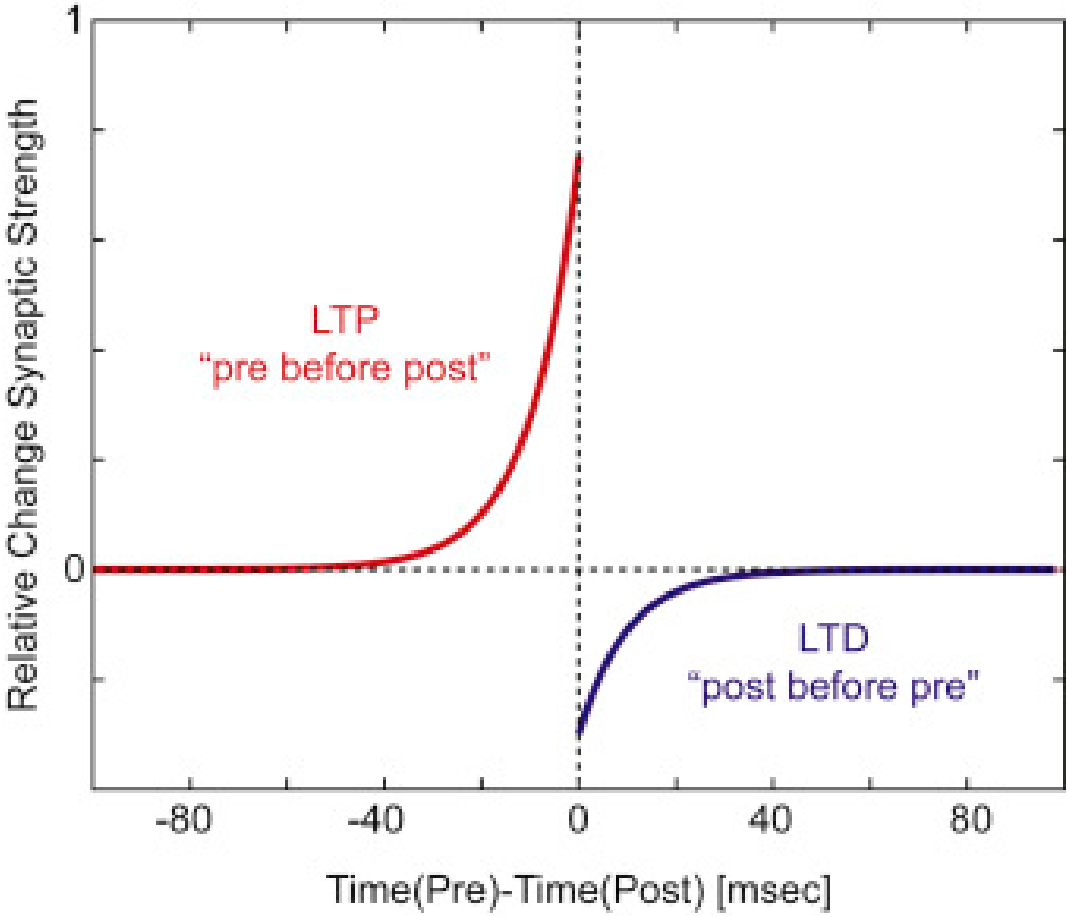


Figure 13: The change of synaptic weight plotted against the difference in timing of the pre- and post-synaptic neuron spikes, showing the long term potential and long term depression phenomena.

based convolutional SNNs have shown promise in achieving accurate and efficient speech recognition, with performance levels approaching those of traditional ANNs in certain scenarios.

Memristors are two-terminal devices whose conductance changes based on the history of voltage or current, closely mimicking how biological synapses adjust their efficacy [25]. 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 [26]. By collocating memory and computation, this approach avoids the von Neumann bottleneck and enables energy-efficient, on-device learning in neuromorphic hardware [27].

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 [28]. 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 [29].

While STDP offers advantages such as biological plausibility, local learning, and energy efficiency, it also has limitations. Training deep networks and achieving state-of-the-art accuracy on complex tasks can be challenging compared to backpropagation-based methods [30]. Directly applying STDP to supervised learning tasks often requires extensions such as R-STDP or classifiers which interpret the output of the model [31]. Furthermore, STDP can be sensitive to the choice of hyperparameters and network architecture [32], and it may tend to extract frequently occurring features that are not necessarily the most discriminative for a specific task [33]. Despite these limitations, ongoing research continues to explore variations and extensions of STDP to enhance its capabilities for diverse learning tasks.

2.2.6 Eventprop

A novel, accurate, and efficient training algorithm. Eventprop uses the adjoint method from optimisation theory to implement backpropagation in an efficient manner for spiking neural networks. Unlike other implementations of backpropagation for SNNs, Eventporp computes *exact* gradients to train the network, as opposed to approximate surrogate gradients typically used in BPTT. Moreover, it does so while using less compute and memory resources, reaching state-of-the-art (SOTA) performance for the SHD dataset while using 4x less memory and running 3x faster [34].

What allows Eventprop to be so efficient is its event-driven nature. Similar to the forward propagation of information in an SNN, the backward propagation of error signals in Eventprop occurs only at the precise times of recorded spikes [**?**], see how in figure 14 the backpropagated error signals occur at the same time in the backward propagation as the spike signals do in the forward propagation. This results in significant temporal and spatial sparsity in the computation, leading to reduced computational cost and improved efficiency, especially for parallel computing architectures such as the ones in GPUs. Furthermore, Eventprop offers favourable memory requirements compared to methods like BPTT. It only necessitates the storage of neuron states at the times of spike events, rather than at every time step, which can drastically reduce memory usage, particularly for long input sequences common in speech recognition tasks. This memory efficiency makes Eventprop well-suited for implementation on resource-constrained neuromorphic hardware, this was recently explored, with successful deployments on platforms like Intel's Loihi 2, showcasing its potential for low-power, event-based machine learning on specialized hardware [35].

Eventprop has demonstrated its effectiveness on challenging speech recognition benchmarks. It has achieved state-of-the-art performance on the Spiking Heidelberg Digits (SHD) dataset and shown good accuracy on the Spiking Speech Commands (SSC) dataset when training recurrent spiking neural networks. Notably, when compared to leading surrogate gradient-based SNN training methods, implementations of Eventprop have been shown to be significantly faster (up to 3x) and require considerably less memory (up to 4x) [34]. This efficiency allows for scaling SNN training to more complex tasks and larger models, which have yet to be explored and could potentially increase the accuracy of models further.

Eventprop's flexibility extends to its compatibility with a wider class of loss functions [34], including those commonly used with BPTT, allowing researchers to tailor the training process to specific task requirements. Furthermore, the Eventprop formalism can be extended to incorporate learnable delays within SNNs, which can be crucial for

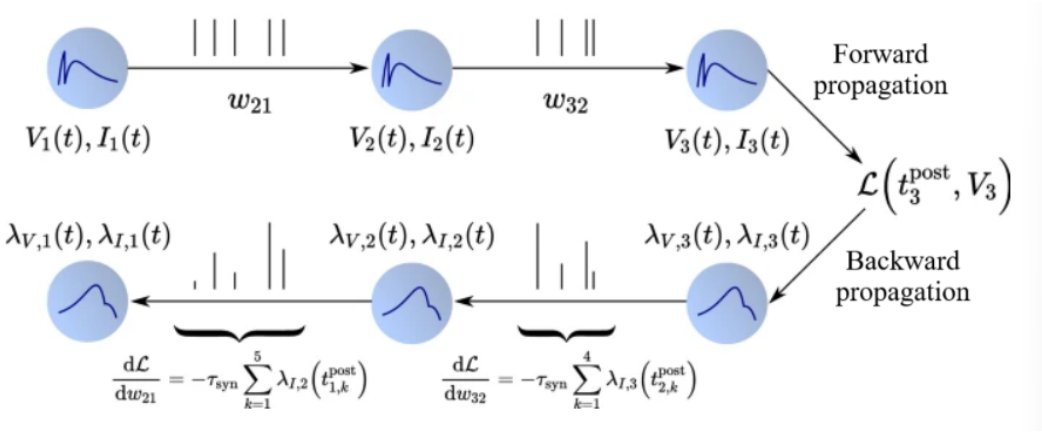


Figure 14: Spikes propagated forward according to LIF neuron dynamics. Error signals propagated backward via computing adjoint variables backwards in time.

capturing temporal dependencies in sequential data like speech. It has been shown that training neuron model parameters like the membrane time constant or the synapse time constants - which control how fast the neuron voltage and current decay - can achieve good results.

In summary, Eventprop is a direct training method - unlike ANN-to-SNN conversion - which enables it to train networks of architectures more suited to SNNs as, improving processing of dynamic data. Moreover, as a direct training method it doesn't require the costly conversion step. Eventprop performs more efficiently - both computationally and memory wise - than BPTT implementations like Spyx, scalling better for longer sample durations. In addition, unlike the Parallelisable-LIF implementation of BPTT - which does come with performance increases - Eventprop doesn't increase the complexity of neuronal model implementations which allows it do be implemented in neuromorphic hardware, unlocking the great potential of efficiency for SNNs. Not only does Eventprop scale better than eligibility propagation (e-prop) in terms of compute resources, it also doesn't use approximate gradients and thus achieves better performance. And finally, Eventprop is easier to train for complex tasks than spike-timing-dependent-plasticity (STDP) based approaches, requiring less computational resources and achieving better performance. For these reasons I have chosen to train spiking neural networks using Eventprop for speech recognition; it shows great potential for efficiently and accurately training larger models which can be used for efficient edge processing, for instance on smartphones for voice recognition.

Mathematics of Eventprop At the core of Eventprop is the "adjoint method" from optimisation theory. The adjoint method is a highly efficient mathematical technique for calculating the sensitivity of a function's output to the functions parameters. In the case of optimising neural networks, the adjoint method would calculate the sensitivity of the network's loss function to changes in the synaptic weights of the network; this sensitivity information is used to optimise the algorithm. The adjoint method is very computationally efficient, the cost of computing the gradient is nearly independent of the number of parameters; unlike in methods such as finite differences, where the computational cost scales linearly with the number of parameters. This linear scaling would become prohibitive in large networks.

An adjoint system to the leaky integrate-and-fire (LIF) network has been derived by [**?**] and is shown in figure 15. The adjoint variables for the input current and membrane voltage are λ_I and λ_V respectively. They represent how sensitive the loss function is to changes to the input current or membrane voltage for a given neuron at a given time. By solving the adjoint equations backwards in time, going from $t = T$ to $t = 0$, you are left with all the values of λ_I. Summing the values of λ_I at spike events which are received by neuron j and transmitted by neuron i, and multiplying this sum by $-\tau_{syn}$ you get the gradient of the loss function with respect to weight w_{ji}. This is described in equation (19).

$$\frac{d\mathcal{L}}{dw_{ji}} = -\tau_{syn} \sum_{spikes\ \ from\ \ i} (\lambda_I)_j \tag{19}$$

Free dynamics	Transition condition	Jumps at transition
Forward:		
(i) $\tau_{\text{mem}}\dot{V} = -V + I$	$(V)_n - \vartheta = 0,$	$(V^+)_n = 0$
(ii) $\tau_{\text{syn}}\dot{I} = -I$	$\left(\dot{V}\right)_n \neq 0$	$I^+ = I^- + We_n$
Backward:		
(iii) $\tau_{\text{mem}}\lambda_V' = -\lambda_V - \frac{\partial l_V}{\partial V}$	$t - t_k = 0$	(v) $(\lambda_V^-)_{n(k)} = (\lambda_V^+)_{n(k)} + \frac{1}{\tau_{\text{mem}}(V^-)_{n(k)}}\Big[\vartheta(\lambda_V^+)_{n(k)}$
(iv) $\tau_{\text{syn}}\lambda_I' = -\lambda_I + \lambda_V$		$+ \left(W^T(\lambda_V^+ - \lambda_I)\right)_{n(k)} + \frac{\partial l_p}{\partial t_k} + l_V^- - l_V^+\Big]$
Gradient of the loss:(vi) $\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}w_{ji}} = -\tau_{\text{syn}}\sum_{t\in t_{\text{spike}}(i)}\lambda_{I,j}(t)$		

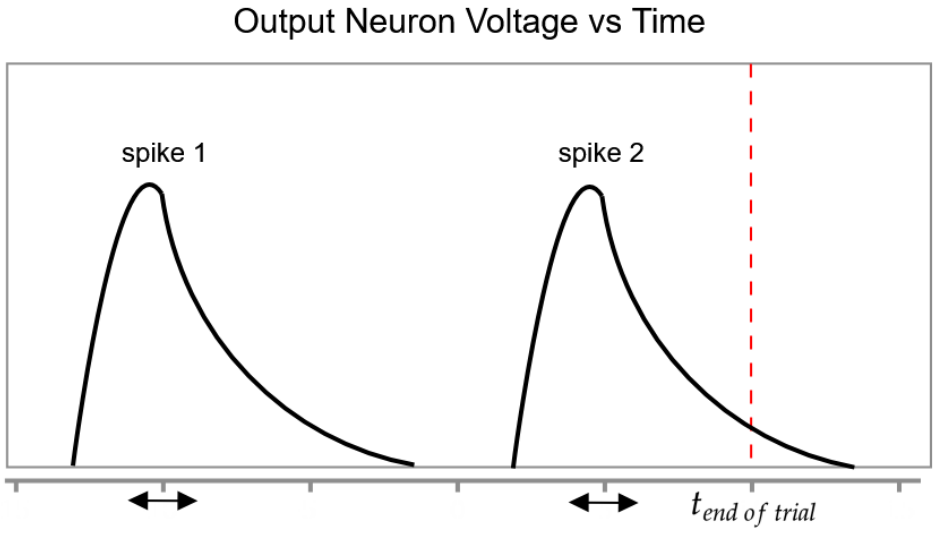
Figure 15: Spikes propagated forward according to LIF neuron dynamics. Error signals are propagated backword according to the adjoint system. ν

Figure 15 shows the maths for general loss functions; the spike time dependent loss, l_p, and the output neuron voltage dependent loss, l_V. This allows flexibility in how loss functions are defined, enabling the shaping of loss functions to suit the task. On the SHD dataset, Eventprop has achieved state-of-the-art accuracy by using a loss function based on the membrane voltage of the output neurons [34]. Initially, researchers tried using a loss function which takes the an integral of the output neuron's membrane voltage over the recording period, $\int_0^T V(t)dt$. Each neuron corresponds to a word that could have been said in the recording; according to this loss function, the output neuron with the greatest integral of membrane voltage should correspond to the word that was actually said in the recording. The loss function took this integral for each output neuron, and calculated the loss using log-softmax function, as in equation (20).

$$\mathcal{L}_{sum} = -\frac{1}{N_{batch}}\sum_{m=1}^{N_{batch}} log(\frac{exp(\int_0^T V_{l(m)}^m(t)dt)}{\sum_{k=1}^{N_{out}} exp(\int_0^T V_k^m(t)dt)}) \tag{20}$$

This loss function has shown better accuracy than alternatives. One alternative is based on the timing of the first spike - the first spike to arrive at an output neuron determines how the recording is classified. Another alternative classifies the recording based on the max membrane voltage reached in the output neurons.

While it had the best accuracy, it was found that the learning rate of this sum based loss function, $\mathcal{L}_{sum}$, was very slow. When spikes reach the output neurons, the membrane voltage of the output neuron increases, and then decays based on the membrane time constant, τ_{mem}. However, if spikes reach the output neurons near the end of the trial, the tail-end of the decaying voltage will be cut off, see figure **??**. Changing the weights between neurons affects the timing of the spikes they generate; a stronger synaptic connection will increase the voltage in the post-synaptic neuron faster, and thus it will reach the threshold voltage quicker. Changing the timing of the spikes occurring near the end of the recording will cause more or less of their area to be cut-off. Therefore, their timing will affect the loss - which is the result of the output neuron membrane voltage integrated over the duration of the trial - far more than earlier spikes. This disproportionate effect that later spikes have on the loss is an arbitrary artifact of the trial duration end, and causes learning to be slower and less efficient.



To improve gradient flow, a weighted sum based loss was proposed and tested, $\mathcal{L}_{sum\_exp}$ in equation (21). This weights earlier spikes exponentially more than the later spikes according to the $e^{-t/T}$ term. This loss function ended up achieving state-of-the-art accuracy.

$$\mathcal{L}_{sum\_exp} = -\frac{1}{N_{batch}} \sum_{m=1}^{N_{batch}} log(\frac{exp(\int_0^T e^{-t/T} V_{l(m)}^m(t)dt)}{\sum_{k=1}^{N_{out}} exp(\int_0^T e^{-t/T} V_k^m(t)dt)}) \tag{21}$$

However, I hypothesise that this loss function is still sub-optimal. While it deals with the edge case of the final spikes, it now over-emphasises early spikes. In particular, I am referring the spikes that happen instantaneously to the recording beginning, since at that time there is still no way to know what the word is, therefore they are just a result of noise. Moreover, they occur before the recurrent network's rich temporal dynamics have had a chance to play-out. I predict that this causes the rate of learning of the network to be slowed because random noise is emphasised and the actual signal is attenuated.

This is why I will explore deriving and implementing a novel loss function and Eventprop scheme. More on this in the methods section. # Methods

Marking Scheme: A+: Expertly applied i) mathematical and statistical methods, tools and notations, ii) quantitative and computational methods, and engineering principles in completing the project. Technical content of the highest quality - could easily provide the basis for a publication or could confidently and proudly be presented to a commercial client. Work of the level of a professional engineer.

2.3 Dataset used

The Spiking Heidelberg digits dataset [36] is derived from the Heidelberg Digits dataset, comprising approximately 10,000 high-quality recordings of spoken digits (0 to 9) in English and German. These recordings feature a balanced group of 12 speakers (6 males and 6 females) aged between 21 and 56 years, with a mean age of 29. The audio data is converted into spike trains using the Lauscher artificial cochlea model, resulting in 700 input channels that represent different frequency bands detected by the human ear. This transformation captures the temporal dynamics of speech, making the dataset suitable for training and evaluating SNNs that process time-dependent information.

The SSC dataset is a spiking version of the Google Speech Commands dataset, containing a large number of utterances across 35 word categories. These recordings are processed using the same Lauscher artificial cochlea model as in SHD, producing spike trains over 700 input channels [**?**]. The SSC dataset encompasses a broader vocabulary and a larger speaker base, providing a more extensive benchmark for SNNs in speech recognition tasks.

The TIMIT Acoustic-Phonetic Continuous Speech Corpus is a dataset containing a diverse group of English speakers reading phonetically-rich sentences with detailed transcription [**?**]. A downside of TIMIT is that it is not free, needing a license to be used. LibriSpeech ASR Corpus is a very large-scale dataset containing 1,000 hours of speech derived from audio books [**?**]. While it is free, the fact that it is read out loud means it may not generalise well to spontaneous conversational speech, such as might be used when engaging with a large language model through your smartphone or smart-glasses. Another prevelant speech dataset is the TIDIGITS dataset, containing recordings of digits spoken in sequence [**?**], making it more complex than the SHD dataset which contains recordings of single digits being spoken. However, a major issue with all of these alternatives to SHD and SSC is that they aren't inherently spike-based, this would cause less fair comparisons between competing training algorithms as researchers might have differing ways of converting the datasets to spiking datasets. This is why the SHD and SSC datasets are the most prevalent in the research of applying spiking neural networks to speech recognition, and it is why I will use them.

2.4 Software Libraries

There are multiple libraries I considered which implement Eventprop functionality. Their GitHub repositories were "timowunderlich/eventprop", "tnowotny/genn_eventprop", and "lolemacs/pytorch-eventprop".

"timowunderlich/eventprop"

What libraries there are for training a network using Eventprop.

Why I chose the GeNN based library.

Installing the library and tools needed for the project.

2.5 Accessing GPU Resources

Creating a software development pipeline using remote GPU.

Setting up GPU at my dads house in London.

Setting up the network to be able to access the GPU.

Dealing with firewall issues.

2.6 Splitting Training and Evaluation Data

Why data has been split the way it has.

2.7 Training the Network

First I implemented the model used in [34] to get state-of-the-art 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.

GRAPHS OF TRAINING GRAPHS OF WEIGHTS CHANGING OVER TIME?

Using the hyper-parameters from [34] allowed me to reproduce the state-of-the-art accuracy found in literature.

How are the results classified in literature?

Ran tests using previously implemented loss functions. Show what they look like and what they mean. Results reproduced.

2.8 Deriving and implementing a Novel Loss Function

As mentioned

The best loss function in the literature is exponential loss.

This is because the spikes occurring at the end of the recording affect loss significantly more, this is due to the fact that if you move them a little bit, forward of backward in time, the decaying membrane voltage tail they leave behind gets cut off. Therefore their timing is arbitrarily far more important than other spikes, arbitrarily because the recording end time should not affect the classification. Therefore a function was chosen to reduce the weight spiles happening later have on the loss.

This highly emphasises very early spikes. Spikes that occur whilst the word has barely begun to be said. Therefore, they cannot correlate with the correct answer - i.e. they are just noise. But their significant weighting sends big gradient adjustments based on this noise. A better loss function would lower the weight of the spikes at the end, and the spikes at the *very* beginning.

Show graphs of output spikes occurring at the very beginning of the recording.

I chose a new format for the loss function, more of a bell curve shape.

Derived the mathematics of the new eventprop scheme.

Then I implemented this function in the code (C++).

2.9 Neural Network Model Description

First I implemented the same model used in [34] 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.

2.10 Bayesian Optimisation of Hyperparameters

What are hyperparameters.

The library I use for training has weak functionality for finding optimum hyperparameters.

Implementing a useful feature would aid the research work to explore novel architectures, training methods and loss functions. It would support open source software.

Talk about some of the main ones and why I chose Bayesian optimisation.

-Grid search -Random search -Evolutionary Algorithms -Bayesian Optimization and why I chose it

How I implemented it.

3 Results

Show cool plots of spikes occurring in the network.

How the resources are utilised. Memory / compute

3.1 Reproducing State-Of-The-Art Accuracy

Before experimenting with novel loss functions I used the current best loss function found in the literature for Eventprop, $\mathcal{L}_{sum\_exp}$. To replicate the state-of-the-art results I ensure that the hyper-parameters of the network and training scheme parameters were set up correctly. To do this the parameters in the literature had to be matched to the variables in the software.

I used the development pipeline I had set up to run the training program, and got the correct evaluation value of 92% stated in literature. Putting me on the SHD benchmark leaderboard.

Show plot of the exp function learning

Show that it reaches SOTA accuracy.

3.2 Comparing Existing Loss Functions

I then implemented other loss functions found in literature and got the following results. This shows that the exp loss function is best.

Show plots of how quickly the existing loss functions learn.

Compare and explain.

3.3 New Loss Function Improves Training

Show that it achieves SOTA accuracy too!

Show that it learns faster than the previous loss functions due to the increased resilience to noise.

3.4 More Efficient Hyperparameter Optimisation Using Bayesian Optimisation

Show how the Bayesian algorithm found optimal values for hyperparameters.

Prove that this is faster than brute force, which was used previously.

4 Discussion

Eventprop Training for larger models.

Learnable delays for Eventprop.

lorem ipsum

5 Conclusion

lorem ipsum

6 Project Review

lorem ipsum

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] E. Izhikevich, "Simple model of spiking neurons," *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1569–1572, Nov. 2003.

[7] ——, "Which model to use for cortical spiking neurons?" *IEEE Transactions on Neural Networks*, vol. 15, no. 5, pp. 1063–1070, Sep. 2004.

[8] 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.

[9] 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.

[10] 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.

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

[12] 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.

[13] E. O. Neftci, M. Hesham, and Z. Friedemann, "Surrogate Gradient Learning in Spiking Neural Networks: Bringing the Power of Gradient-Based Optimization to Spiking Neural Networks," *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 51–63, Nov. 2019.

[14] 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.

[15] 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.

[16] S. Y. A. Yarga and S. U. N. Wood, "Accelerating SNN Training with Stochastic Parallelizable Spiking Neurons," in *2023 International Joint Conference on Neural Networks (IJCNN)*, Jun. 2023, pp. 1–8.

[17] W. Maas, "Networks of spiking neurons: The third generation of neural network models," *Trans. Soc. Comput. Simul. Int.*, vol. 14, no. 4, pp. 1659–1671, Dec. 1997.

[18] R. Koopman, A. Yousefzadeh, M. Shahsavari, G. Tang, and M. Sifalakis, "Overcoming the Limitations of Layer Synchronization in Spiking Neural Networks," Aug. 2024.

[19] Y. Zhong, R. Zhao, C. Wang, Q. Guo, J. Zhang, Z. Lu, and L. Leng, "SPikE-SSM: A Sparse, Precise, and Efficient Spiking State Space Model for Long Sequences Learning," Oct. 2024.

[20] 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.

[21] ——, "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.

[22] 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.

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

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] Y. Guo, X. Huang, and Z. Ma, "Direct learning-based deep spiking neural networks: A review," *Frontiers in Neuroscience*, vol. 17, p. 1209795, Jun. 2023.

[31] F. Liu, W. Zhao, Y. Chen, Z. Wang, T. Yang, and L. Jiang, "SSTDP: Supervised Spike Timing Dependent Plasticity for Efficient Spiking Neural Network Training," *Frontiers in Neuroscience*, vol. 15, p. 756876, Nov. 2021.

[32] C. Lee, P. Panda, G. Srinivasan, and K. Roy, "Training Deep Spiking Convolutional Neural Networks With STDP-Based Unsupervised Pre-training Followed by Supervised Fine-Tuning," *Frontiers in Neuroscience*, vol. 12, Aug. 2018.

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

[34] 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.

[35] T. Shoesmith, J. C. Knight, B. Mészáros, J. Timcheck, and T. Nowotny, "Eventprop training for efficient neuromorphic applications," Mar. 2025.

[36] "Spiking Heidelberg Digits and Spiking Speech Commands – Zenke Lab."