Efficient Training and Memory Optimization Strategies for Margin Propagation Based Networks

by Viraj Kotak
Masters in Technology

Abstract

Deep learning models have revolutionized artificial intelligence but rely on multiplication operations that consume substantial energy, limiting their efficiency on resource-constrained devices. This thesis addresses challenges in scaling Margin Propagation (MP) networks, a multiplication-free neural network framework that uses only addition, subtraction, and thresholding operations. While MP networks significantly reduce energy consumption and maintain competitive accuracy (93% on MNIST), they suffer from increased memory requirements during training due to their dual representation of values as margin pairs and sorting operation. We present two key contributions to overcome these memory constraints: (1) theoretical error bounds between MP formulation and standard vector multiplication, enabling efficient weight porting from pre-trained MLPs to untrained MP networks, and (2) a novel connection between simplex projection and MP equation that replaces memory-intensive operations with optimized simplex algorithms. Our approaches substantially reduce the memory footprint required for training MP networks while preserving their energy efficiency benefits, making these multiplication-free networks practical for deployment at scale.

Guided by
Prof. Chiranjib Bhattacharyya
Department of Computer Science and Automation
Indian Institute of Science (IISc),
Bangalore, India

[illegible]

Signature

Acknowledgements

I would like to express my heartfelt gratitude to **Prof. Chiranjib Bhattacharya** and **Prof. Chetan Singh Thakur** for their invaluable guidance and constant support throughout this work. Their mentorship, insightful feedback, and unwavering encouragement played a pivotal role at every stage of this research. It has been a privilege to learn under their supervision.

I am also deeply thankful to my labmate, **Madhuvanthi Srivatsav**, for her patience in answering all my questions, no matter how small and for always being a source of encouragement and inspiration.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Motivation

Deep learning models have revolutionized artificial intelligence across numerous domains, achieving remarkable performance in tasks ranging from computer vision to natural language processing. However, these advances come with a significant computational cost. Traditional Multilayer Perceptron (MLP) networks rely heavily on multiplication operations that consume substantial energy, making them inefficient for deployment on resource-constrained devices.

The multiply-accumulate (MAC) operation, central to neural network computation, is particularly energy-intensive. Research shows that a 32-bit floating-point multiplication consumes approximately 3.7 pJ of energy, while an addition operation requires only 0.9 pJ—roughly 4 times less energy (Horowitz, 2014). When scaled to modern neural networks with billions of parameters, these multiplication operations account for a majority of the energy consumption. For instance, BERT-large with 340 million parameters requires approximately 6.8×10^{10} MAC operations for a single forward pass (Strubell, Ganesh, and McCallum, 2019), translating to significant energy demands.

To address this energy efficiency challenge,(Nair et al., 2022) introduced Margin Propagation (MP)—a novel computational framework that eliminates multiplication operations in neural networks. Their approach replaces standard vector multiplication with operations limited to addition, subtraction, and thresholding, significantly reducing the computational burden. Their work demonstrates that MP networks can achieve competitive accuracy across various tasks while substantially decreasing energy requirements.

The MP formulation reimagines neural network computation by encoding numerical values as the difference between two non-negative values (margins). This approach is illustrated and derived in detail in Section 2. MP formulation allows the network to perform vector multiplications through purely additive operations. The authors implemented this framework for 3 layer MLP Network. Their experimental results showed performance across several benchmarks, including 93% accuracy on MNIST, approaching the comparable performance of traditional multiplication-based networks while eliminating multiplication operations.

Despite these advantages in energy efficiency, the MP formulation presents significant challenges in training. The dual representation of values as margin pairs substantially increases memory requirements, making it difficult to scale these networks to larger architectures. As model size increases, the memory footprint grows prohibitively large, particularly on GPU hardware where memory is often the limiting factor for training complex models. This memory constraint creates a practical barrier to applying MP networks to large-scale problems that could benefit most from their energy efficiency.

This thesis addresses the memory efficiency challenges of MP networks through two primary contributions. In Section 3, we derive theoretical error bounds between the MP formulation and standard vector multiplication. Using these bounds, we develop a novel weight-porting technique that efficiently transfers learned parameters from pre-trained MLP networks to untrained MP networks, dramatically reducing the training resources required. In Section 4, we explore the mathematical connection between simplex projection and the MP equation, allowing us to replace memory-intensive operations in the training process with optimized simplex algorithms. Together, these approaches enable more memory-efficient training of MP networks, paving the way for their application to larger-scale problems. In Section 5, we explored the parallels between the classical perceptron neuron and the non-leaky integrate-and-fire (n-LIF) spiking neuron model. The goal was to understand how the decision-making behavior of the perceptron could be replicated using spiking dynamics.

Chapter 2

Margin Propagation (MP) Formulation

The Margin Propagation method provides a multiplierless approach to approximate the inner product of two vectors (Nair et al., 2022). This method is particularly advantageous for hardware implementations due to its reliance on simple arithmetic operations like addition, subtraction, and thresholding. Below, we outline the MP method, provide its derivation, and discuss the key challenges associated with training MP-based networks.

2.1 MP Method

Let $w, x \in \mathbb{R}^D$ be two D-dimensional vectors. To approximate their inner product $w^T x$ using MP method, we begin by defining two intermediate vectors $p, q \in \mathbb{R}^{2D}$ as:

$$p = \begin{bmatrix} w + x \\ -w - x \end{bmatrix}, \qquad q = \begin{bmatrix} w - x \\ -w + x \end{bmatrix}$$

We introduce a positive scalar hyperparameter $\gamma > 0$ to control the scaling of the approximation. Then, we define two auxiliary scalars α and β as the unique solutions to the following equations:

$$\sum_{i=1}^{2D} [p_i - \alpha]_+ = \gamma, \qquad \sum_{i=1}^{2D} [q_i - \beta]_+ = \gamma \tag{2.1}$$

where $[\cdot]_+$ denotes the rectified linear unit (ReLU) activation function.
Using these solutions, we define the MP values as:

$$\mathrm{MP}(p, \gamma) := \gamma D \alpha, \qquad \mathrm{MP}(q, \gamma) := \gamma D \beta$$

Finally, the approximation of the inner product $w^T x$ is given by:

$$y_{MP} = \frac{1}{2} \left(\mathrm{MP}(p, \gamma) - \mathrm{MP}(q, \gamma) \right) \approx w^T x$$

2.2 Derivation of MP method

Margin Propagation (MP) approximation involves two implicit approximations: (1) approximating a quadratic function using the Log-Sum-Exp (LSE) function and (2) approximating the exponential function via its first-order Taylor expansion. To compute the exact dot product of two vectors, we can start with a quadratic function:

$$f(x) = \frac{1}{2}x^T x,$$

Now, consider evaluating the expression $\frac{1}{2}(f(w+x) - f(w-x))$ using the quadratic function defined above. This yields the exact value of the dot product $w^T x$. However, implementing this approach on digital hardware requires computing quadratic functions that involve multiplications, which is computationally intensive and inefficient, particularly for low-precision hardware. To overcome these limitations, we employ the Log-Sum-Exponential (LSE) function as an approximation of the quadratic function. Introducing a hyperparameter $\gamma > 0$, the function is defined as:

$$f(x,\gamma) = \gamma^2 D \log\left(\sum_{i=1}^{D} e^{\frac{x_i}{\gamma}} + e^{-\frac{x_i}{\gamma}}\right) \tag{2.2}$$

The modified expression to evaluate becomes:

$$y_{LSE} = \frac{1}{2}(f(w+x,\gamma) - f(w-x,\gamma)) \tag{2.3}$$

An important observation from the MP approximation presented in the original paper is that the function f was scaled by a factor of γ, rather than the more accurate $\gamma^2 D$ as derived in our analysis. The rationale behind applying the $\gamma^2 D$ scaling becomes evident upon expanding the approximation function using a Taylor series. This expansion reveals an inherent scaling factor of $1/(\gamma D)$ in the leading quadratic term.

To compensate for this and reduce the approximation error between the smoothed output y_{LSE} and $\mathbf{w}^\top \mathbf{x}$, we apply the $\gamma^2 D$ scaling explicitly. Empirical results, summarized in Table 2.1, demonstrate that this correction significantly improves the fidelity of the approximation.

D	$w^T x$	y_{LSE} (w/o scalling)	y_{LSE} (with scalling)	LSE Error	Modified LSE Error
100	3.1168	0.0302	3.0244	3.0866	0.0924
500	1.9325	0.0038	1.8780	1.9287	0.0545
1000	-0.6336	-0.0006	-0.6094	0.6330	0.0242

TABLE 2.1: Comparison of original and modified LSE values and errors for different values of D

We begin by analyzing the term $f(w+x,\gamma)$; a similar analysis applies to $f(w-x,\gamma)$.

$$f(w+x,\gamma) = \gamma^2 D \log\left(\sum_{i=1}^{D} e^{\frac{w_i+x_i}{\gamma}} + e^{-\frac{w_i+x_i}{\gamma}}\right) \tag{2.4}$$

Rewriting the expression, we obtain:

$$\sum_{i=1}^{D} e^{\frac{w_i+x_i}{\gamma} - \frac{f(w+x,\gamma)}{\gamma^2 D}} + \sum_{i=1}^{D} e^{-\left(\frac{w_i+x_i}{\gamma} + \frac{f(w+x,\gamma)}{\gamma^2 D}\right)} = 1 \tag{2.5}$$

We now apply a first-order Taylor approximation for the exponential functions: $e^x \approx [1+x]_+$ and $e^{-x} \approx [1-x]_+$. Substituting, we get:

$$\sum_{i=1}^{D} \left[1 + \frac{w_i+x_i}{\gamma} - \frac{f(w+x,\gamma)}{\gamma^2 D}\right]_+ + \sum_{i=1}^{D} \left[1 - \left(\frac{w_i+x_i}{\gamma} + \frac{f(w+x,\gamma)}{\gamma^2 D}\right)\right]_+ = 1$$

$$\sum_{i=1}^{D} \left[\gamma + w_i + x_i - \frac{f(w+x,\gamma)}{\gamma D}\right]_+ + \sum_{i=1}^{D} \left[\gamma - w_i - x_i - \frac{f(w+x,\gamma)}{\gamma D}\right]_+ = \gamma$$

Define the vector $p \in \mathbb{R}^{2D}$ as $p = \begin{bmatrix} w+x \\ -w-x \end{bmatrix}$ and let $\alpha = \frac{f(w+x,\gamma)}{\gamma D}$. Then the equation simplifies to:

$$\sum_{i=1}^{2D} [\gamma + p_i - \alpha]_+ = \gamma \tag{2.6}$$

Similarly, defining $q = \begin{bmatrix} w-x \\ -w+x \end{bmatrix}$ and $\beta = \frac{f(w-x,\gamma)}{\gamma D}$, we obtain:

$$\sum_{i=1}^{2D} [\gamma + q_i - \beta]_+ = \gamma \tag{2.7}$$

The solutions to the equations (8) and (9) are approximations of $\frac{f(w+x,\gamma)}{\gamma D}$ and $\frac{f(w-x,\gamma)}{\gamma D}$, respectively. We define the scaled version of these solutions as $\text{MP}(\gamma+p,\gamma)$ and $\text{MP}(\gamma+q,\gamma)$. Notably, the difference eliminates the common offset γ inside the ReLU operations, demonstrating that the operation is offset-invariant. Specifically,

$$\text{MP}(\gamma+p,\gamma) - \text{MP}(\gamma+q,\gamma) = \text{MP}(p,\gamma) - \text{MP}(q,\gamma)$$

Therefore, we can define MP operator as,

$$\text{MP}(p,\gamma) := \gamma D\alpha \approx f(w+x,\gamma), \qquad \text{MP}(q,\gamma) := \gamma D\beta \approx f(w-x,\gamma)$$

Putting these approximations in equation (2.3) we get MP approximation of dot product $w^T x$ as,

$$y_{MP} = \frac{1}{2}\left(\text{MP}(p,\gamma) - \text{MP}(q,\gamma)\right) \tag{2.8}$$

2.3 Memory Consumption in MP Networks

MP uses differential formulation of inputs and weights where they are divided into two positive counterparts, difference of which gives back original values. This requires twice the amount of memory which is primary bottleneck for scalling MP networks upto VGG16 like architecture. In a MP network pseudocode, we define the following values for input x and weight w:

$$\begin{aligned}
x^+ &= a + x, \\
x^- &= a - x, \\
w^+ &= \text{ReLU}(w), \\
w^- &= \text{ReLU}(-w), \\
z^+ &= \text{concatenate}(x^+ + w^+,\ x^- + w^-), \\
z^- &= \text{concatenate}(x^+ + w^-,\ x^- + w^+).
\end{aligned}$$

Here, a is a parameter that is used to convert input x into differential form. For an input of shape $[128, 28, 28]$, flattened to $[128, 784]$, and a three-layer network with dimensions $[784, 8000, 10]$, the computation of the hidden layer tensors z^+ and z^- results in significant memory usage. Each z tensor (both z^+ and z^-) involves a concatenation of two branches of size $(128 \times 784 \times 8000)$. In total, both tensors require:

$$\begin{aligned}
\text{Memory} &= \frac{128 \times (784 + 784) \times 8000 \times 4}{1024^2 \times 1000} \times 2 \\
&= \frac{128 \times 1568 \times 8000 \times 4}{1024^2 \times 1000} \\
&\approx 3\,\text{GB}.
\end{aligned}$$

The term $(784 + 784)$ accounts for concatenation of the x^+ and x^- branches, the factor 4 corresponds to the use of `float32` data type (i.e., 4 bytes per element) and the final multiplication by 2 accounts for both z^+ and z^- tensors being computed and stored.

This shows that even for relatively modest hidden layer sizes, the memory consumption in MP networks is very high. This becomes a significant bottleneck, especially when scaling to large-scale MP networks, where such intermediate representations dominate memory usage.

Another major contributor to GPU memory consumption and computational overhead is the sorting operation. To solve the equation

$$\sum_{i=1}^{2D} [p_i - \alpha]_+ = \gamma,$$

we must first determine which elements of the vector $p = (p_1, p_2, \ldots, p_{2D})$ will be active (i.e., positive) after the thresholding operation. Since the operator $[\,\cdot\,]_+$ retains only values greater than zero, the summation effectively includes only those p_i for which $p_i > \alpha$. To identify this subset efficiently, it is necessary to sort the elements p_i in descending order. Once sorted, we can sequentially examine prefixes of the sorted list to compute the corresponding values of α that satisfy the equality constraint. However doing this operation for every neuron in the network consumes significant amount of memory, making it a primary bottleneck for training large-scale MP Networks.

Chapter 3

Efficient MP Network Initialization

One promising direction to address the high memory consumption in MP networks is to first train a conventional multilayer perceptron (MLP) and subsequently port the learned weights into an MP network. Since the MP formulation inherently approximates the dot product, it offers a natural pathway for efficient weight transfer. To enable this, we begin by analyzing in detail the approximation error between the standard dot product and the MP formulation. This analysis helps in selecting appropriate hyperparameters for the MP method, thereby allowing the MP network to achieve performance comparable to that of the original MLP.

3.1 Error Analysis

As mentioned earlier, MP approximation involves two implicit approximations. In the first section, we will try to find an analytical bound for the error in first approximation that is, using LSE to approximate the quadratic function.

3.1.1 Error Analysis of LSE Approximation to Dot Product

As described in section 2.2 we can use LSE function to approximate quadratic functions. To find the bound on the error, we first expand the terms inside log function using taylor series,

$$\begin{aligned} f(x,\gamma) &= \gamma^2 D \log\left(\frac{1}{2D}\sum_{i=1}^{D} e^{\frac{x_i}{\gamma}} + e^{\frac{-x_i}{\gamma}}\right) \\ &= \gamma^2 D \log\left(\frac{1}{D}\sum_{i=1}^{D}\left(1 + \frac{x_i^2}{2!\gamma^2} + \frac{x_i^4}{4!\gamma^4} + \cdots\right)\right) \\ &= \gamma^2 D \log\left(1 + \frac{\|x\|_2^2}{2!\gamma^2 D} + \frac{\|x\|_4^4}{4!\gamma^4 D} + \cdots\right) \end{aligned}$$

We can approximate log function using taylor series as following,

$$\log(1+y) = y + O\left(y^2\right)$$

Let's say $y = y_1 + \epsilon_1$ where $y_1 = \frac{\|x\|_2^2}{2!\gamma^2 D}$ and $\epsilon_1 = \frac{\|x\|_4^4}{4!\gamma^4 D} + \frac{\|x\|_8^8}{8!\gamma^8 D} + \cdots$. We know that $\|x\|_2 \leq \|x\|_4 \leq \|x\|_8 \leq$ .. so if we assume that $\|x\|_2^2 \ll \gamma^2 D$ then we can drop ϵ_1 term and we can write the log term as,

$$\log\left(1 + \frac{\|x\|_2^2}{2!\gamma^2 D} + \frac{\|x\|_4^4}{4!\gamma^4 D} + \cdots\right) = \frac{\|x\|_2^2}{2!\gamma^2 D} + O\left(\frac{\|x\|_2^4}{\gamma^4 D^2}\right)$$

Therefore we can write,

$$f(x,\gamma) = \frac{\|x\|_2^2}{2} + O\left(\frac{\|x\|_2^4}{\gamma^2 D}\right) \tag{3.1}$$

Substituting this in equation (2.3) we get,

$$y_{LSE} = \frac{1}{2}\left[\frac{1}{2}\left(\|w+x\|^2 - \|w-x\|^2\right) + \left(O\left(\frac{\|w+x\|^4}{\gamma^2 D}\right) - O\left(\frac{\|w-x\|^4}{\gamma^2 D}\right)\right)\right].$$

$$y_{LSE} = w^T x + O\left(\frac{\|w+x\|^4}{\gamma^2 D}\right) - O\left(\frac{\|w-x\|^4}{\gamma^2 D}\right).$$

If $\|w\| \leq K$ and $\|x\| \leq K$, then $\|w \pm x\| = O(K)$ and hence

$$O\left(\frac{\|w \pm x\|^4}{\gamma^2 D}\right) = O\left(\frac{K^4}{\gamma^2 D}\right).$$

Thus, the error in approximating $w^T x$ is bounded by

$$\left|y_{LSE} - w^T x\right| \leq O\left(\frac{K^4}{\gamma^2 D}\right) \tag{3.2}$$

The empirical results for this bound when w and x are sampled from uniform distribution is plotted.

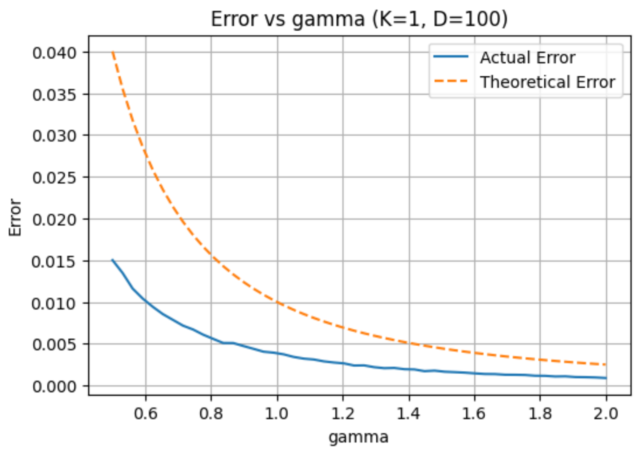

FIGURE 3.1: Absolute error between y_{LSE} and $w^\top x$ with fixed K and D, varying γ.

3.1.2 Error Analysis of MP Approximation to LSE function

For convenience, let's define LSE function in equation (6) using concatenated vector p as following.

$$f(\mathbf{p},\gamma) = \gamma^2 D \log\left(\sum_{i=1}^{2D} e^{p_i/\gamma}\right).$$

From this, we obtain

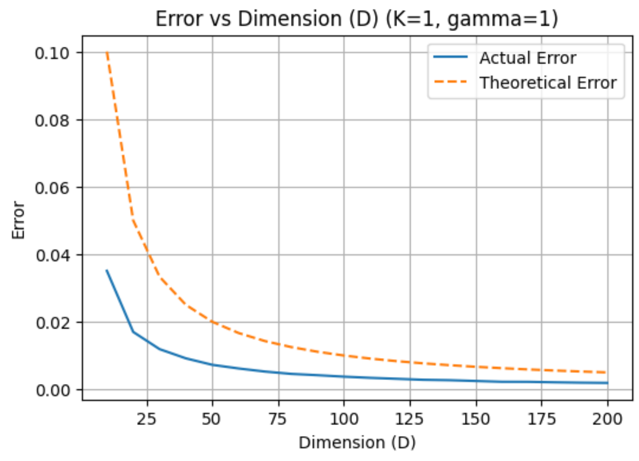

FIGURE 3.2: Absolute error between y_{LSE} and $w^\top x$ with fixed K and γ, varying D.

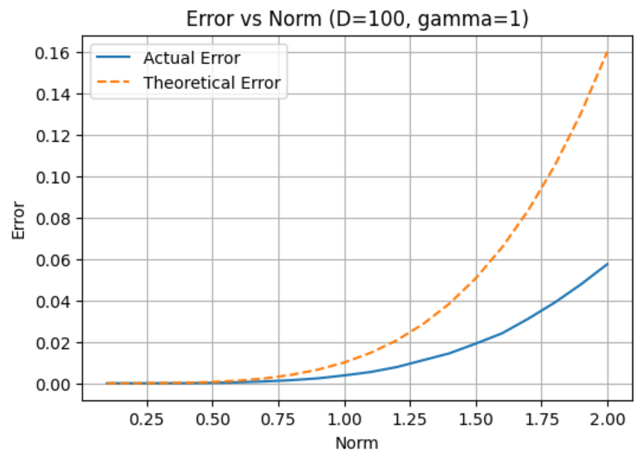

FIGURE 3.3: Absolute error between y_{LSE} and $w^\top x$ with fixed D and γ, varying K.

$$\sum_{i=1}^{2D} e^{\frac{p_i}{\gamma} - \frac{f(\mathbf{p},\gamma)}{\gamma^2 D}} = 1. \tag{3.3}$$

Define the term, $\alpha_{\mathrm{exact}} := \frac{f(\mathbf{p},\gamma)}{\gamma D}$. then we get,

$$\sum_{i=1}^{2D} e^{\frac{p_i - \alpha_{\mathrm{exact}}}{\gamma}} = 1. \tag{3.4}$$

As described in section 2.2, we have employed the first-order approximation $e^x \approx [1+x]_+$, which holds under the assumption that $|x| \ll 1$. Substituting the approximation into equation (3.4), we get

$$\sum_{i=1}^{2D} \left[1 + \frac{p_i - \alpha_{\mathrm{approx}}}{\gamma}\right]_+ = 1, \tag{3.5}$$

where $\alpha_{\mathrm{approx}} := \frac{\mathrm{MP}(\mathbf{p},\gamma)}{\gamma D}$ denotes the MP estimate of α_{exact}.

However, this approximation incurs significant error when this condition is violated, particularly for values of x far from zero. The pointwise approximation error for some $\xi \in (0, x)$ is given by,

$$e^x - [1+x]_+ = \begin{cases} \frac{x^2 e^{\xi}}{2}, & \text{if } x > -1, \\ e^x, & \text{if } x \leq -1. \end{cases} \tag{3.6}$$

Let's analyze this error in MP framework. In our context, the variable of interest is the term,

$$\frac{p_i - \alpha_{\text{exact}}}{\gamma}$$

By substituting α_{exact} we get,

$$\begin{aligned} \frac{p_i - \alpha_{\text{exact}}}{\gamma} &= \frac{p_i}{\gamma} - \log\left(\sum_{i=1}^{D} e^{\frac{p_i}{\gamma}} + e^{-\frac{p_i}{\gamma}}\right) \\ &= \log e^{\frac{p_i}{\gamma}} - \log\left(\sum_{i=1}^{D} e^{\frac{p_i}{\gamma}} + e^{-\frac{p_i}{\gamma}}\right) \\ &= \log\left(\frac{e^{\frac{p_i}{\gamma}}}{\sum_{i=1}^{D} e^{\frac{p_i}{\gamma}} + e^{-\frac{p_i}{\gamma}}}\right) \\ &= \log\left(\text{softmax}\left(\frac{p_i}{\gamma}\right)\right). \end{aligned}$$

Now let's analyze the range of this term so we can use point-wise error accordingly. Firstly since softmax is always less than or equal to 1 the log $softmax$ will always be negative. Now if,

$$\log\left(\text{softmax}\left(\frac{p_i}{\gamma}\right)\right) > -1$$

$$\text{softmax}\left(\frac{p_i}{\gamma}\right) > \frac{1}{e}$$

But we know that

$$\sum_{i=1}^{2D} \text{softmax}\left(\frac{p_i}{\gamma}\right) = 1$$

$$\sum_{i=1}^{2D} \frac{1}{e} \leq 1$$

$$D \leq \frac{e}{2}$$

Since D is an integer the only value of D that satisfies this condition is when $D = 1$ which is a trivial case. Hence, we will be using the pointwise error of 2^{nd} case in equation (3.6) i.e., $x \leq -1$.

$$e^x - [1+x]_+ \leq e^{-1}$$

$$e^{\frac{p_i - \alpha_{\text{approx}}}{\gamma}} - \left[1 + \frac{p_i - \alpha_{\text{approx}}}{\gamma}\right]_+ \leq \frac{1}{e} \tag{3.7}$$

Define, $\delta := \frac{\alpha_{\text{exact}} - \alpha_{\text{approx}}}{\gamma}$ and using this in equation (3.4) we get,

$$\sum_{i=1}^{2D} e^{\frac{p_i - \alpha_{\text{approx}}}{\gamma} - \delta} = 1,$$

$$\sum_{i=1}^{2D} e^{\frac{p_i - \alpha_{\text{approx}}}{\gamma}} = e^{\delta}.$$

Using the pointwise bound from equation (3.7) we estimate,

$$\sum_{i=1}^{2D} \left[1 + \frac{p_i - \alpha_{\text{approx}}}{\gamma}\right]_+ + \sum_{i=1}^{2D} \frac{1}{e} \geq e^{\delta},$$

$$1 + \frac{2D}{e} \geq e^{\delta}$$

$$\delta \leq \ln\left(1 + \frac{2D}{e}\right),$$

$$\alpha_{\text{exact}} - \alpha_{\text{approx}} \leq \gamma \ln\left(1 + \frac{2D}{e}\right), \tag{3.8}$$

which implies the final bound on the difference between LSE function and its MP approximation is,

$$|f(\mathbf{p}, \gamma) - \text{MP}(\mathbf{p}, \gamma)| \leq \gamma^2 D \ln\left(1 + \frac{2D}{e}\right) \tag{3.9}$$

This bound characterizes the worst-case error introduced by the first-order exponential approximation. Empirical validation of this bound, using w and x sampled from a uniform distribution, is presented in the subsequent figure.

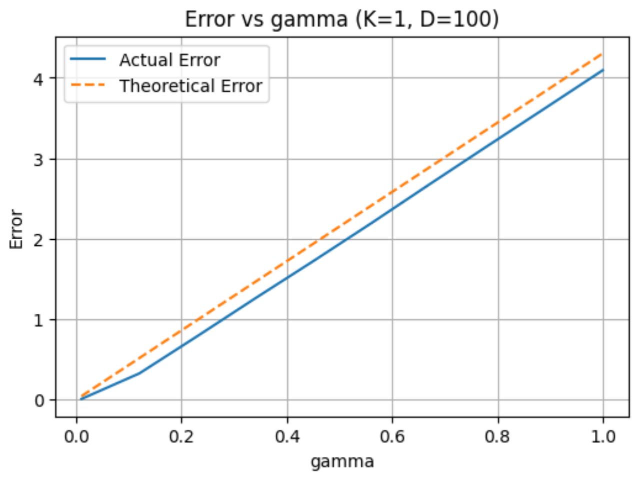

FIGURE 3.4: Absolute error between $f(\mathbf{p}, \gamma)$ and $\text{MP}(\mathbf{p}, \gamma)$ with fixed D, varying γ.

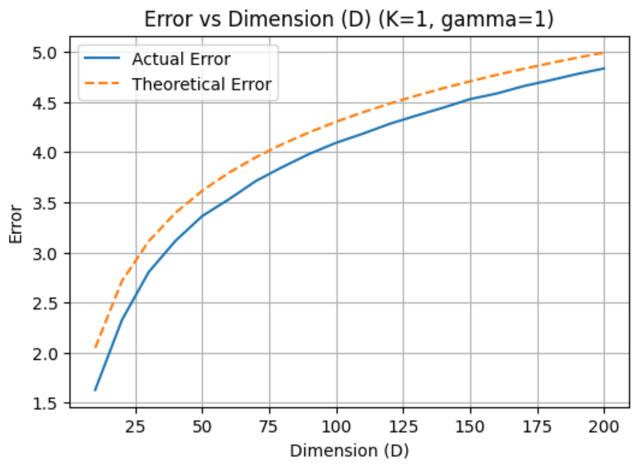

FIGURE 3.5: Absolute error between $f(\mathbf{p},\gamma)$ and $\mathrm{MP}(\mathbf{p},\gamma)$ with fixed γ, varying D.

Now we can use this to calculate error between y_{LSE} and y_{MP}.

$$\begin{aligned}|y_{LSE}-y_{MP}| &= \frac{1}{2}|\left(f(w+x,\gamma)-f(w-x,\gamma)\right)-\left(\mathrm{MP}(p,\gamma)-\mathrm{MP}(q,\gamma)\right)| \\ &= \frac{1}{2}|\left(f(w+x,\gamma)-\mathrm{MP}(p,\gamma)\right)-\left(f(w-x,\gamma)-\mathrm{MP}(q,\gamma)\right)| \\ &\leq \frac{1}{2}|f(w+x,\gamma)-\mathrm{MP}(p,\gamma)|+\frac{1}{2}|f(w-x,\gamma)-\mathrm{MP}(q,\gamma)|\end{aligned}$$

Using equation (3.9) we get a final bound as,

$$|y_{LSE}-y_{MP}| \leq \gamma^2 D \ln\left(1+\frac{2D}{e}\right) \tag{3.10}$$

Finally using equation (3.2) and (3.10) we can write,

$$\boxed{\left|y_{\mathrm{MP}}-w^T x\right| \leq O\left(\frac{K^4}{\gamma^2 D}\right)+\gamma^2 D \ln\left(1+\frac{2D}{e}\right)} \tag{3.11}$$

3.2 Direct weight transfer from trained MLP Network

Using the theoretical error bound derived in the previous section, we select an optimal value of the scaling parameter γ that minimizes the approximation error between the MP function and the original neural activation.

To empirically validate this approach, we train a fully connected multi-layer perceptron (MLP) consisting of approximately 1.5 million parameters on the MNIST dataset. The MLP takes flattened 28×28 grayscale images as input (784 dimensions), and comprises four hidden layers with dimensions $[1024, 512, 256, 128]$, followed by a 10-way classification output layer. The learned weights from this network are then directly ported to the MP network without any additional training. We evaluate the classification accuracy of the resulting MP network across a range of γ values and report the results below.

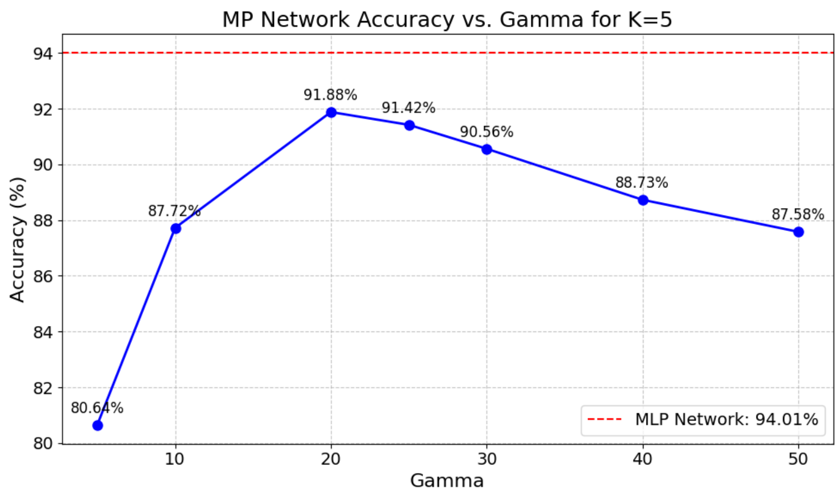

FIGURE 3.6: MP Network accuracy with MLP-trained weights for varying γ.

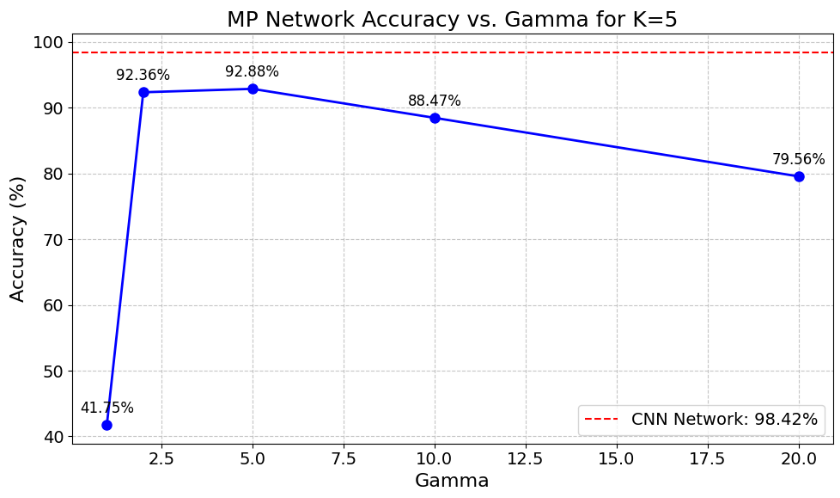

FIGURE 3.7: MP Network accuracy with Conv-trained weights for varying γ.

To assess generalization across architectures, we repeat this procedure using a fully convolutional neural network (CNN). The CNN consists of four convolutional layers with kernel sizes 5, 5, 5, and 3, and output channels 6, 16, 120, and 84 respectively, interleaved with max pooling after the first two convolutions. The final feature map is flattened and passed through a fully connected layer with 84 input units and 10 output classes. As with the MLP, the trained CNN weights are transferred to the MP network, and accuracy is measured as a function of γ.

In addition to accuracy measurements, we further analyze the MP network by computing the correlation between the hidden layer activations of the original MLP and its MP counterpart. Specifically, we calculate the Pearson correlation coefficient between activation vectors and also visualize neuron-wise correlation to assess fidelity in internal representations across the networks.

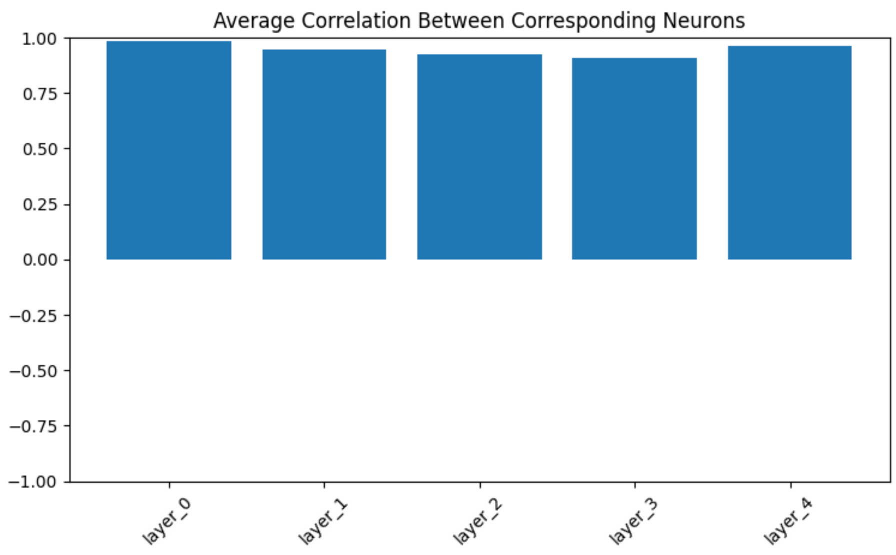

FIGURE 3.8: Correlation of average hidden layer activations between MP and MLP networks.

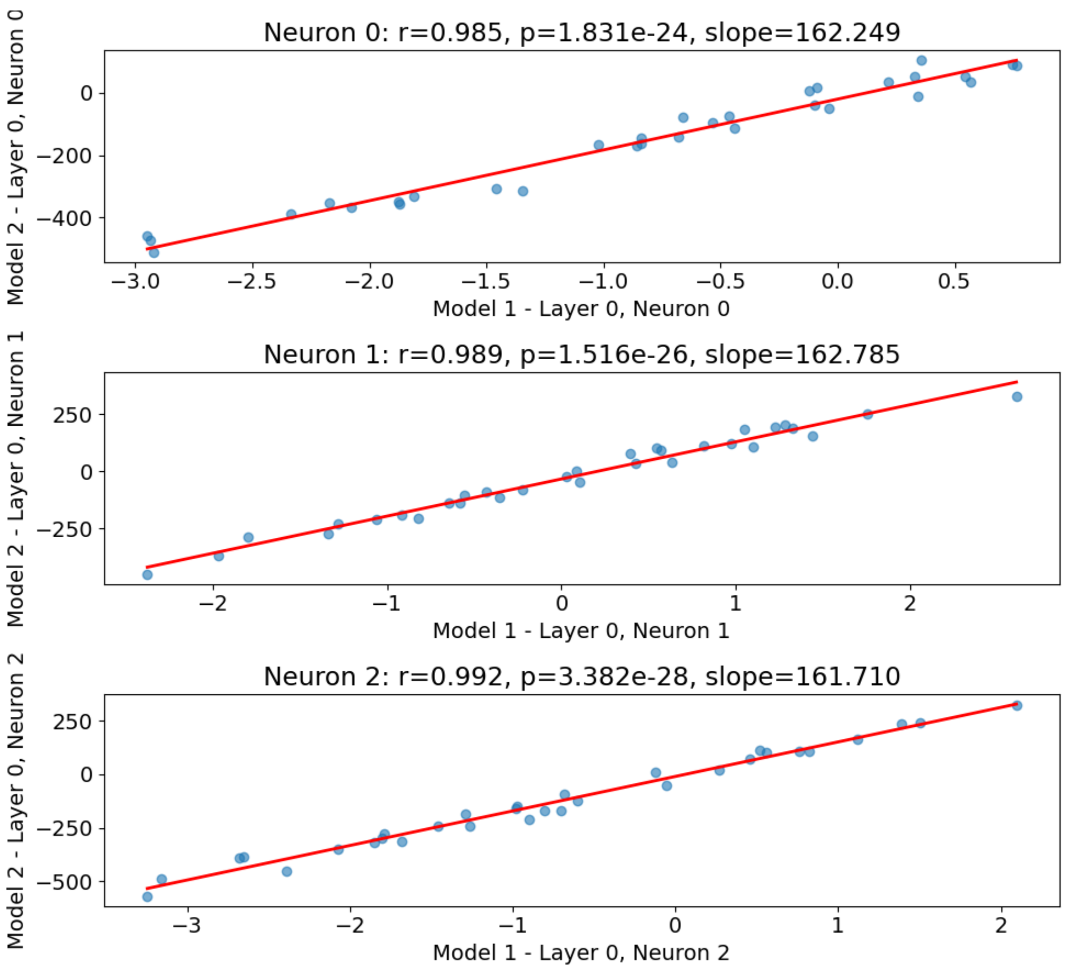

FIGURE 3.9: Correlation of individual neuron activations between MP and MLP networks.

Chapter 4

Connecting Simplex Projection with MP Equation

4.1 Accelerating the MP Update via Simplex Projection

In this section, we exploit the equivalence between the MP update equation

$$\sum_{i=1}^{2D} \left[p_i - \alpha\right]_+ = \gamma \tag{4.1}$$

and the Euclidean projection onto the probability simplex. We show how this connection allows us to avoid expensive sorting of spike times (a major bottleneck in TEMP and other SNN simulations) by instead invoking a linear-time simplex-projection algorithm (Condat, 2016). We then present empirical results comparing memory usage and latency for three-layer MP networks of size $784 \times h \times 10$, as the hidden dimension h varies.

Connection to Simplex Projection

Recall from Section 2.1 that the MP approximation can be written in closed form as a "ReLU–sum" equation (4.1), which exactly matches the KKT conditions for projecting the vector $\frac{1}{\gamma}\, p$ onto the probability simplex of dimension $2D$. In particular, if we define

$$q_i = \frac{p_i}{\gamma}, \qquad \alpha = \tau_{\text{proj}}(q),$$

then the unique solution to

$$\arg\min_{u \in \mathbb{R}^{2D}} \left\|u - q\right\|^2 \quad \text{s.t.} \quad \sum_i u_i = 1,\ u_i \geq 0$$

satisfies

$$u_i = \left[q_i - \alpha\right]_+,$$

and summing over i recovers (4.1). Thus, computing the MP update reduces to projecting onto the simplex.

Replacing Sorting with Condat's Simplex-Projection Algorithm

In standard MP and SNN simulators, one must sort each neuron's incoming spike times before computing the update in (4.1). This $O(n \log n)$ sorting step is a well-known GPU-memory and compute bottleneck. By exploiting the equivalence to simplex projection, we replace explicit sorting with Condat's linear-time algorithm Condat, 2016. We implemented this algorithm in C, interfaced via Python, and substituted it for the usual GPU-based sort + prefix-sum implementation in our MP network code.

Memory–Latency Trade-off: Experimental Results

We evaluated three-layer MP networks of architecture $784 \times h \times 10$ on the MNIST test set, varying $h \in \{500, 1000, 2000, 4000\}$. For each h, we measured (i) peak GPU memory usage and (ii) per-batch inference latency, comparing the *sorting-based* MP against the *simplex-projection* MP. Results are plotted in Figure 4.1.

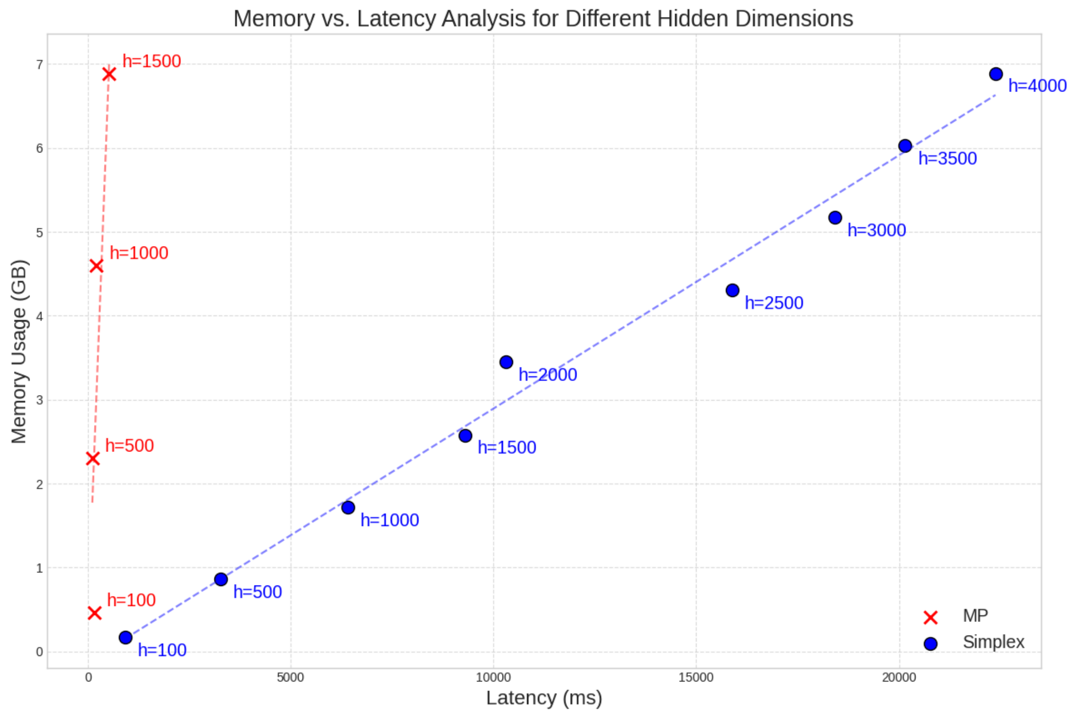

FIGURE 4.1: Memory consumption vs. latency for MP networks using sorting (red) and Condat simplex-projection (blue), as hidden dimension h varies.

As expected, the simplex-projection approach substantially reduces memory footprint (up to 30% for large h), since it avoids storing full sorted spike sequences. However, this saving comes at the expense of significantly increased latency, often by a factor of 2–3 due to the CPU-host call and per-neuron projection overhead.

While this algorithm eliminates the sorting-induced memory bottleneck, the resulting latency increase renders this approach impractical for training large-scale TEMP networks in real time.

Chapter 5

Perceptron-Inspired Interpretation of n-LIF Spiking Neurons

5.1 Perceptron-like Algorithm for n-LIF Neuron

Under certain assumptions, a spiking neuron can be interpreted as an extension or variant of a perceptron neuron. By drawing parallels between the two models, we can derive a perceptron-like formulation tailored to spiking dynamics. In the following section, we explore this correspondence in detail.

The perceptron algorithm (Gallant, 1990) is a simple supervised learning method for binary classification. It classifies data based on a linear decision boundary and updates weights iteratively to minimize classification errors. For input $\mathbf{x}$ and label $y \in \{-1, +1\}$, the weight update rule is:

$$\mathbf{w} \leftarrow \mathbf{w} + y\mathbf{x},$$

If the classifier misclassifies the input. The perceptron converges in a finite number of iterations if the data is linearly separable. It has been widely used in tasks like text classification and image recognition.

For an n-LIF neuron, the membrane potential at any time t is given by:

$$V(t) = K \sum_{j=1}^{d} w_j \left(1 - e^{-(t-t_j)}\right) H(t - t_j),$$

where t_j are spike times, w_j are synaptic weights, and $H(t - t_j)$ is the Heaviside function. Define $T = \max_j t_j$. The classifier is defined as follows:

- If $V(T) \geq v_{\text{th}}$, then $y_{\text{pred}} = +1$.
- Otherwise, $y_{\text{pred}} = -1$.

Here v_{th} is a hyperparameter. At $t = T$, $H(T - t_j) = 1$ for all j, reducing $V(T)$ to:

$$V(T) = K \sum_{j=1}^{d} w_j \left(1 - e^{-(T-t_j)}\right).$$

Defining $\phi_j = K(1 - e^{-(T-t_j)})$, we can write $V(T)$ as:

$$V(T) = \mathbf{w}^\top \boldsymbol{\phi},$$

where $\boldsymbol{\phi} = [\phi_1, \phi_2, \ldots, \phi_d]$. The classifier now becomes:

$$\mathbf{w}^\top \boldsymbol{\phi} - v_{\text{th}} \geq 0 \quad \Rightarrow \quad y_{\text{pred}} = +1,$$

$$\mathbf{w}^\top \boldsymbol{\phi} - v_{\text{th}} < 0 \quad \Rightarrow \quad y_{\text{pred}} = -1.$$

This resembles the perceptron classifier, where $\boldsymbol{\phi}$ acts as the input feature vector.

Aspect	**Perceptron**	**n-LIF Neuron Model**
Inputs	$x_j^{(i)}$	Spike times $t_j^{(i)} = x_j^{(i)}$
Weights	w_j	w_j
Hyperparameters	η	η, T, V_{th}
Hardware-Specific Parameters	Not applicable	τ, K
Feature Transformation	$x_j^{(i)}$	$\phi(t_j^{(i)}) = K\left(1 - e^{-\frac{T-t_j}{\tau}}\right) H(T - t_j)$
Decision Boundary	0	V_{th}
Decision Rule	$\hat{y} = \text{sign}(w \cdot x)$	$\hat{y} = \text{sign}(w \cdot \phi - V_{th})$
Linear Separability	$y^{(i)}(w \cdot x^{(i)}) \geq \gamma$	$y^{(i)}(w \cdot \phi^{(i)} - V_{th}) \geq \gamma$

TABLE 5.1: Comparison of Perceptron and n-LIF Neuron Models

Algorithm 1 n-LIF-Perceptron Training Algorithm

```
Require: Training data $\{(\phi^{(i)}, y^{(i)})\}_{i=1}^{N}$, learning rate $\eta$, threshold $V_{\text{th}}$
Ensure: Trained weights $\tilde{w}$
 1: Initialize weights $\tilde{w}$
 2: repeat
 3:     errors $\leftarrow 0$
 4:     for each training example $(\phi^{(i)}, y^{(i)})$ do
 5:         Compute prediction $\hat{y}^{(i)} \leftarrow \text{sign}(\tilde{w} \cdot \phi^{(i)} - V_{\text{th}})$
 6:         if $\hat{y}^{(i)} \neq y^{(i)}$ then
 7:             Update weights: $\tilde{w} \leftarrow \tilde{w} + \eta y^{(i)} \phi^{(i)}$
 8:             errors $\leftarrow$ errors $+1$
 9:         end if
10:     end for
11: until errors $= 0$ or maximum epochs reached
12: return $\tilde{w}$
```

The algorithm converges similar to perceptron algorithm if data is linearly separable in ϕ space and there exists a constant R such that $||\phi|| \leq R$.. The proof of convergence of perceptron algorithm is derived in next section.

5.2 Experiments and Analysis

Let's say data is coming from $X \sim U[a, b]$, the PDF of X is:

$$f_X(x) = \begin{cases} \frac{1}{b-a}, & a \leq x \leq b, \\ 0, & \text{otherwise.} \end{cases}$$

Define a transformation as following,

$$Y = K(1 - e^{-(T-X)/\tau})$$

where K,T and τ are constants. The inverse transformation is:

$$x = T - \tau \ln\left(1 - \frac{y}{K}\right),$$

with

$$\frac{dx}{dy} = \frac{\tau}{K-y}.$$

Then the PDF of Y is given by:

$$f_Y(y) = f_X\left(T - \tau \ln\left(1 - \frac{y}{K}\right)\right) \cdot \frac{\tau}{K-y}.$$

Substituting $f_X(x)$:

$$f_Y(y) = \begin{cases} \frac{\tau}{(b-a)(K-y)}, & y \in [K(1 - e^{-(T-b)/\tau}), K(1 - e^{-(T-a)/\tau})], \\ 0, & \text{otherwise.} \end{cases}$$

Now if we approximate $e^{-x} \approx 1 - x$ in Y, the transformation becomes:

$$Y \approx K \cdot \frac{T-X}{\tau}.$$

The inverse is:

$$x = T - \frac{\tau}{K}y, \quad \frac{dx}{dy} = -\frac{\tau}{K}.$$

And the PDF of Y:

$$f_Y(y) = f_X\left(T - \frac{\tau}{K}y\right) \cdot \left|-\frac{\tau}{K}\right|.$$

Substituting $f_X(x)$ we get:

$$f_Y(y) = \begin{cases} \frac{\tau}{(b-a)K}, & y \in \left[\frac{K}{\tau}(T-b), \frac{K}{\tau}(T-a)\right], \\ 0, & \text{otherwise.} \end{cases}$$

which is again a Uniform distribution For two classes with $X_1 \sim U[a_1, b_1]$ and $X_2 \sim U[a_2, b_2]$, Assume $a_1 < b_1 < a_2 < b_2$. For the approximation $e^{-x} \approx 1 - x$ we require $T - X \ll \tau$, for all X. Now if $T - a_1 \ll \tau$ then it previous inequality is satisfied for all X Let $T = b_2$ (as defined in the algorithm) and

$$\tau = c(b_2 - a_1)$$

where $c \gg 1$

To minimize misclassifications, set V_{th} near the midpoint of the separation region: $V_{th} \approx \frac{\text{mean}(Y_1)+\text{mean}(Y_2)}{2}$. which leads to

$$V_{th} = \frac{K}{\tau}\left(T - \frac{a_1 + b_1 + a_2 + b_2}{4}\right).$$

A binary classification task was conducted using a synthetic dataset, where samples from the first class were drawn uniformly from the interval $[0,1]$, and those from the second class from $[1,2]$. A Support Vector Machine (SVM) classifier from the `scikit-learn` library was employed to compute the optimal linear decision boundary. The margin was defined as the minimum distance between this boundary and the closest datapoint from either class. In parallel, the classical perceptron algorithm was applied, with the margin computed as:

$$\text{Margin} = \min_i \left(y^{(i)}(\mathbf{w} \cdot \mathbf{x}^{(i)})\right),$$

where $y^{(i)} \in \{-1, +1\}$ denotes the class label and $\mathbf{x}^{(i)}$ the input feature vector.

The dataset was then transformed into the $\boldsymbol{\phi}$-space representation, defined by $\phi_j = 1 - e^{-(T-t_j)}$, which models the response of spiking neurons. Both the SVM and a perceptron-like algorithm for n-LIF neurons (referred to as the LIF-perceptron) were applied in this transformed space. The LIF-perceptron incorporates membrane dynamics through parameters τ, T, and a threshold V_{th}, influencing the classification boundary.

Method	Original Data Margin	Transformed Data Margin
SVM	0.412	0.047
Perceptron	0.281	0.006

TABLE 5.2: Comparison of margins for different methods.

Empirical results showed that the margin in the ϕ-space was reduced relative to the original space. Notably, the LIF-perceptron exhibited a smaller margin than the standard perceptron, reflecting the restricted representational capacity imposed by the temporal dynamics of the n-LIF model. As expected, both perceptron-based methods yielded smaller margins than the SVM, which explicitly maximizes margin under linear separability assumptions.

5.3 Convergence Proof of Perceptron Algorithm

Assumption (Linear Separability): Suppose there exists $w^* \in \mathbb{R}^d$ and a margin $\gamma > 0$ such that for all i:

$$y^{(i)}(w^* \cdot x^{(i)}) \geq \gamma.$$

Assume without loss of generality that $\|w^*\| = 1$. To show that the algorithm converges, We show that the number of misclassifications is bounded. Since each misclassification leads to a weight update, and the number of updates cannot exceed a finite bound, the algorithm must stop making mistakes.

Let M be the total number of mistakes (updates). We track the evolution of w through these mistakes.
Initially, $w^{(1)} = 0$. At the k-th mistake, we have:

$$w^{(k+1)} = w^{(k)} + y^{(i_k)} x^{(i_k)}$$

for some misclassified example $(x^{(i_k)}, y^{(i_k)})$. Take the dot product with w^*:

$$w^{(k+1)} \cdot w^* = w^{(k)} \cdot w^* + y^{(i_k)} (w^* \cdot x^{(i_k)}).$$

By linear separability:

$$y^{(i_k)} (w^* \cdot x^{(i_k)}) \geq \gamma > 0.$$

Thus, each update increases $w \cdot w^*$ by at least γ. After M mistakes:

$$w^{(M+1)} \cdot w^* \geq M\gamma. \tag{5.1}$$

Also,

$$\begin{aligned} \|w^{(k+1)}\|^2 &= \|w^{(k)} + y^{(i_k)} x^{(i_k)}\|^2 \\ &= \|w^{(k)}\|^2 + \|x^{(i_k)}\|^2 + 2y^{(i_k)} w^{(k)} \cdot x^{(i_k)}. \end{aligned}$$

Since the example was misclassified, $y^{(i_k)}(w^{(k)} \cdot x^{(i_k)}) \leq 0$, so:

$$\|w^{(k+1)}\|^2 \leq \|w^{(k)}\|^2 + \|x^{(i_k)}\|^2.$$

Summing over M mistakes:

$$\|w^{(M+1)}\|^2 \leq \sum_{k=1}^{M} \|x^{(i_k)}\|^2 \leq MR^2,$$

Where $R = \max_i \|x^{(i)}\|$.

$$\|w^{(M+1)}\| \leq R\sqrt{M}.$$

By Cauchy–Schwarz we get,

$$w^{(M+1)} \cdot w^* \leq \|w^{(M+1)}\| \|w^*\| = \|w^{(M+1)}\| \leq R\sqrt{M}. \tag{5.2}$$

From (5.1) and (5.2) we get,

$$M\gamma \leq R\sqrt{M} \implies M \leq \frac{R^2}{\gamma^2}.$$

Since M cannot exceed this finite bound, the algorithm must stop making mistakes. Therefore, the the algorithm converges.

Adapting the Proof for L_1-Norm

To adapt the perceptron convergence proof for L_1-norm and , we need to carefully modify the arguments involving the norms and inequalities like the Cauchy-Schwarz inequality. The general structure of the proof remains the same, but the bounding and norm-related terms change. The update rule for the perceptron algorithm remains:

$$w^{(k+1)} = w^{(k)} + y^{(i)} x^{(i)}.$$

Taking the dot product with w^*, as before:

$$w^{(k+1)} \cdot w^* = w^{(k)} \cdot w^* + y^{(i)} (w^* \cdot x^{(i)}).$$

By linear separability:

$$y^{(i)} (w^* \cdot x^{(i)} \geq \gamma > 0.$$

Thus, after M mistakes:

$$w^{(M+1)} \cdot w^* \geq M\gamma.$$

Using the L_1-norm, we bound $\|w^{(k+1)}\|_1$:

$$\|w^{(k+1)}\|_1 = \|w^{(k)} + y^{(i)} x^{(i)}\|_1.$$

Using the triangle inequality:

$$\|w^{(k+1)}\|_1 \leq \|w^{(k)}\|_1 + \|x^{(i)}\|_1.$$

Summing over M updates:

$$\|w^{(M+1)}\|_1^2 \leq \sum_{k=1}^{M} \|x^{(i)}\|_1^2 \leq M R_1^2,$$

where $R_1 = \max_i \|x^{(i)}\|_1$.

For the L_1-norm, we use the Hölder inequality (a generalization of Cauchy-Schwarz):

$$|w^{(M+1)} \cdot w^*| \leq \|w^{(M+1)}\|_1 \|w^*\|_\infty.$$

Since $\|w^*\|_\infty = 1$ (by normalization):

$$w^{(M+1)} \cdot w^* \leq \|w^{(M+1)}\|_1.$$

Thus:

$$M\gamma \leq \|w^{(M+1)}\|_1 \leq \sqrt{M} R_1.$$

Equating terms:

$$M \leq \frac{R_1^2}{\gamma^2}$$

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This thesis has addressed a fundamental challenge in the deployment of deep neural networks on resource-constrained devices: the energy efficiency of computational operations. We focused on Margin Propagation (MP) networks, which replace multiplication operations with addition, subtraction, and thresholding operations, resulting in substantial energy savings. While previous work demonstrated the theoretical advantages of MP networks, achieving up to 93% accuracy on MNIST, our research tackled the primary barrier to their practical deployment, excessive memory requirements during training.

Our first contribution established theoretical error bounds between MP formulation and standard vector multiplication. These bounds provided a mathematical foundation for our novel weight-porting technique, which enables the efficient transfer of learned parameters from pre-trained MLP networks to untrained MP networks. By leveraging existing trained models, we dramatically reduced the computational resources required for deploying MP networks, making them accessible for scenarios where training from scratch would be prohibitively expensive.

Our second contribution explored the mathematical connection between simplex projection and the MP equation. We demonstrated that memory-intensive operations in MP training could be replaced with optimized simplex algorithms, significantly reducing the memory footprint of the training process. This approach preserved the energy efficiency benefits of MP networks while addressing their primary limitation—memory consumption during training.

Furthermore, our investigation into the parallels between classical perceptron neurons and non-leaky integrate-and-fire (n-LIF) spiking neurons provided insights into how perceptron decision-making behavior can be replicated using spiking dynamics. This connection establishes a bridge between traditional neural networks and neuromorphic computing paradigms, potentially leading to even more energy-efficient implementations.

6.2 Future Work

A natural extension of our work would be to develop hardware-specific implementations of MP networks tailored to different resource-constrained platforms. Future research could explore custom accelerator designs that directly leverage the addition-only nature of MP computations, potentially yielding even greater energy efficiency gains. Additionally, investigating implementations on existing neuromorphic hardware platforms could further capitalize on the energy benefits.

Despite our memory optimization techniques, scaling MP networks to very large architectures (comparable to state-of-the-art models with billions of parameters) remains challenging. Future work could explore hierarchical or modular MP network structures that allow for efficient distributed training across multiple devices, further reducing per-device memory requirements.

Finally, more rigorous theoretical analysis of MP networks could provide stronger guarantees about their convergence properties, generalization capabilities, and representational power compared to traditional neural networks. Such theoretical foundations would strengthen the case for wider adoption of MP networks in critical applications.

Bibliography

Comsa, Ioana-Maria et al. (2020). “Temporal Coding in Spiking Neural Networks with Alpha Synaptic Function”. In: *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing*. Barcelona, Spain, pp. 8529–8533. DOI: 10.1109/ICASSP40776.2020.9053856.

Condat, Laurent (2016). “Fast projection onto the simplex and the ℓ_1 ball”. In: *Mathematical Programming* 158.1, pp. 575–585. DOI: 10.1007/s10107-015-0946-6.

Gallant, S.I. (1990). “Perceptron-based learning algorithms”. In: *IEEE Transactions on Neural Networks* 1.2, pp. 179–191. DOI: 10.1109/72.80230.

Horowitz, Mark (2014). “1.1 Computing’s energy problem (and what we can do about it)”. In: *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. IEEE, pp. 10–14.

Li, Yujie et al. (2021). “Differentiable spike: Rethinking gradient-descent for training spiking neural networks”. In: *Advances in Neural Information Processing Systems*.

Madhuvanthi, R., Shantanu Chakrabartty, and Chetan Thakur (2023). *Neuromorphic Computing with AER using Time-to-Event-Margin Propagation*. DOI: 10.48550/arXiv.2304.13918. arXiv: 2304.13918 [cs.NE].

Martins, André F. T. and Ramón Fernandez Astudillo (2016). *From Softmax to Sparsemax: A Sparse Model of Attention and Multi-Label Classification*. arXiv: 1602.02068 [cs.CL]. URL: https://arxiv.org/abs/1602.02068.

Moss, F., Stan Gielen, and W. Gerstner (Jan. 2001). “A framework for spiking neuron models - the spike response model”. In.

Mostafa, Hesham (2017). “Supervised learning based on temporal coding in spiking neural networks”. In: *IEEE Transactions on Neural Networks and Learning Systems* 29.7, pp. 3227–3235.

Nair, Abhishek Ramdas et al. (2022). “Multiplierless MP-Kernel Machine for Energy-Efficient Edge Devices”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 30.11, pp. 1601–1614. DOI: 10.1109/TVLSI.2022.3189780.

Sjöström, J. and W. Gerstner (2010). “Spike-timing dependent plasticity”. In: *Scholarpedia* 5.2. revision #184913, p. 1362. DOI: 10.4249/scholarpedia.1362.

Strubell, Emma, Ananya Ganesh, and Andrew McCallum (2019). “Energy and Policy Considerations for Deep Learning in NLP”. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, pp. 3645–3650.

Zhou, Shiqi et al. (2021). “Temporal-Coded Deep Spiking Neural Network with Easy Training and Robust Performance”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 12, pp. 11143–11151. DOI: 10.1609/aaai.v35i12.17329.