PPLNs: Parametric Piecewise Linear Networks for Event-Based Temporal Modeling and Beyond

Chen Song Zhenxiao Liang Bo Sun Qixing Huang
Department of Computer Science
The University of Texas at Austin
Austin, TX 78712
`{song, liangzx, bosun, huangqx}@cs.utexas.edu`

Abstract

We present Parametric Piecewise Linear Networks (PPLNs) for temporal vision inference. Motivated by the neuromorphic principles that regulate biological neural behaviors, PPLNs are ideal for processing data captured by event cameras, which are built to simulate neural activities in the human retina. We discuss how to represent the membrane potential of an artificial neuron by a parametric piecewise linear function with learnable coefficients. This design echoes the idea of building deep models from learnable parametric functions recently popularized by Kolmogorov–Arnold Networks (KANs). Experiments demonstrate the state-of-the-art performance of PPLNs in event-based and image-based vision applications, including steering prediction, human pose estimation, and motion deblurring. The source code of our implementation is available at https://github.com/chensong1995/PPLN.

1 Introduction

Event cameras are neuromorphic sensors that summarize the evolving world as a stream of *events*. Each event describes the pixel coordinates, time, and polarity of an intensity change. Thanks to the simplicity of this data representation, event cameras enjoy multiple advantages over conventional cameras, including but not limited to fast data rate and high dynamic range (Gallego et al., 2020). Over the past, researchers have developed event-based algorithms to solve various computer vision problems, such as motion deblurring (Pan et al., 2019, 2020; Wang et al., 2020; Song et al., 2022, 2024), human pose estimation (Calabrese et al., 2019), and autonomous driving (Binas et al., 2017; Hu et al., 2020). A recent survey (Zheng et al., 2023) indicates a rapidly increasing community interest in event-based research, with motion deblurring being the most popular task. Extensive experiments demonstrate that by utilizing events as an auxiliary input, algorithms perceive fine motion details that are absent from conventional image captures, leading to substantial performance gain over methods that make inferences from conventional images alone.

The success of event cameras demonstrates the power of imitating biological neuromorphic principles. The event camera comprises a rectangular array of *dynamic vision sensors*, each of which is analogous to a visual receptor neuron on the human retina dedicated to perceiving one specific location. The neuron experiences excitement and produces a spike when the environmental intensity varies significantly, corresponding to an event generated as a response to brightness changes.

This paper presents Parametric Piecewise Linear Networks (PPLNs) to understand the bio-inspired event data with a bio-inspired deep learning model. As opposed to a generic network design, we believe and verify in this paper that it is highly beneficial to build a processing network that caters to the principles of the data source (event cameras). As illustrated by Figure 1 (Middle), the key idea is to explicitly approximate the membrane potential of a neuron as a piecewise linear mapping from time to electric voltage. Figure 1 (Right) presents the sketch of a PPLN node, whose internal mechanism

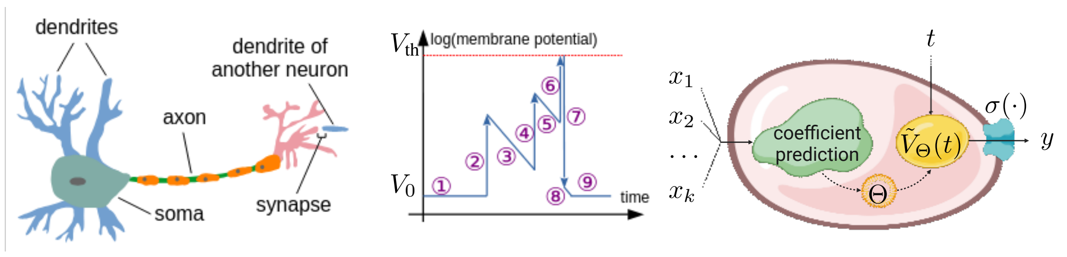

Figure 1: **(Left)**: A biological neuron has three main components: the dendrites (blue), the axon (orange, pink), and the soma (green). The dendrites are responsible for receiving external inputs. The axon transmits signals to the dendrites of other neurons through the synapses. The soma is the body of the cell and connects the dendrites to the axon. **(Middle)**: The membrane potential, defined as the voltage difference between the interior and the exterior of the cell, regulates the neuron's behavior and can be approximately modeled by a piecewise linear function. ① When the neuron is at rest, the potential stays at a constant level V_0. ② An external input is received by the dendrites, causing an instantaneous perturbation to the membrane potential. ③ The perturbation is not significant enough to excite the neuron, and the potential leaks over time exponentially (*i.e.*, linearly in the logarithmic space). ④ Another external input happens. ⑤ The input fails to excite the neuron. ⑥ A third input causes the membrane potential to exceed the threshold voltage V_{th}. The neuron becomes excited and generates a spike. ⑦ The excitement opens ion channels, and the ion flow causes a reset to the membrane potential. ⑧ After the excitement, the ion channels close again, and the potential continues to decay. ⑨ The neuron returns to the resting state, waiting for new inputs. **(Right)**: A PPLN node. Given inputs $\{x_i\}_{i=1}^{k}$, we predict the linear coefficients Θ for the membrane potential function, including the slope, intercept, and endpoints of each line segment. The resulting parametric function $\tilde{V}_\Theta$ is then used to evaluate the neuron output at the timestamp of interest $y(x_1, \dots, x_k, t) = \sigma(\tilde{V}_\Theta(t))$, where $\sigma(\cdot)$ is the integral normalization defined in Section 3.4.

is explained thoroughly in Section 3. While the event camera imitates how a single layer of visual receptor neurons react to external intensity changes, PPLNs are motivated by observations of how layers of biological neurons communicate. Inspired by the Leaky Integrate-and-Fire model (Abbott, 1999), we propose to use a piecewise linear parameterization to approximate its temporal evolution in the logarithmic space. The key difference between PPLNs and the bio-inspired Spiking Neural Networks (SNNs) (Eshraghian et al., 2021) in existing literature is that PPLNs are an alternative to general GPU-based temporal inference models, whereas SNNs aim at the deployment on hardware neuromorphic chips (Davies et al., 2018). Instead of real-valued membrane potentials, SNNs also propagate binary spikes, leading to reduced energy consumption and training instabilities.

PPLNs are conceptually similar to the emerging Kolmogorov–Arnold Networks (KANs) (Liu et al., 2024). Both KANs and PPLNs leverage learnable parametric functions to build a deep network. Different from KANs, which use input-independent B-splines, PPLNs exploits input-dependent piecewise linear functions. Using piecewise linear functions allows our brain-inspired design to mimic biological neural principles and the event generation model. Predicting function coefficients at inference time allows the network to better handle input heterogeneity.

We modify the network architecture in state-of-the-art event-based vision algorithms by replacing multi-layer perceptions and convolution operators with PPLN nodes and evaluate PPLNs on three applications: steering prediction, 3D human pose estimation, and motion deblurring. With fewer or a similar number of trainable parameters, PPLNs improve baselines by 30.8% in steering prediction, 11.1% in human pose estimation, and 5.6% in motion deblurring. To demonstrate the potential of PPLNs, we experiment on the conventional frame-based version of the same applications without the event input and observe consistent improvements. Additionally, we present a mathematical analysis of the convergence properties to showcase the robustness.

In summary, we make the following contributions:

- We propose Parametric Piecewise Linear Networks (PPLNs), mimicking biological principles by approximating membrane potentials as parametric mappings.
- We show how to predict a set of parametric coefficients from the input of the PPLN node and evaluate the membrane potential at any timestamp of interest.

- We present a mathematical analysis of the convergence properties of PPLNs.
- We apply PPLNs to event-based and frame-based applications and achieve state-of-the-art results.

2 Related work

Biological neurons. As shown in Figure 1 (Left), if the combined effect of small perturbations over time causes the membrane potential to exceed the threshold potential, the neuron will become excited. During the excitement, the neuron produces an output spike through the synapses (Lacinová, 2005; Maeda et al., 2009). In Figure 1 (Middle), the Leaky Integrate-and-Fire model (Abbott, 1999) suggests that the membrane potential can be approximated by a piecewise linear function as a parametric mapping from time to the logarithm of the voltage difference.

Spiking neural networks. Spiking Neural Networks (SNNs) are Recurrent Neural Networks (RNNs) that simulate biological neurons (Eshraghian et al., 2021). Each SNN node carries an internal variable corresponding to the membrane potential. The output of the SNN node is a binary signal that is zero by default and becomes one if excited (Abbott, 1999). The key advantage of SNNs is low power consumption, making them the ideal model to be deployed on hardware neuromorphic chips (Davies et al., 2018). By contrast, PPLNs are designed to be an alternative to general GPU-based temporal inference models, even though SNNs and PPLNs are motivated by similar biological principles.

Event cameras. Event cameras are neuromorphic devices that summarize the evolving environment as a stream of events (Lichtsteiner et al., 2008). Each event is analogous to a significant intensity change that exceeds the hardware threshold and excites a biological light-sensing neuron. An event is represented as a 4-tuple (x, y, t, p), where (x, y) are the pixel coordinates, t is the timestamp, and $p \in \{-1, +1\}$ is the polarity of the intensity change. Event cameras have a fast data rate, high dynamic range, low power consumption, and minimal motion blur compared to conventional cameras (Gallego et al., 2020). A recent trend is to utilize hardware that simultaneously captures event and conventional streams, where the event stream is used as an auxiliary input to enhance regular computer vision algorithms. For example, while image-to-image deblurring is a well-studied problem in the research community (Richardson, 1972; Fish et al., 1995; Krishnan & Fergus, 2009; Joshi et al., 2009; Levin et al., 2007; Kim et al., 1998; Shan et al., 2008; Fergus et al., 2006; Xu et al., 2013; Xu & Jia, 2010; Perrone & Favaro, 2014; Babacan et al., 2012; Kupyn et al., 2018, 2019) , several works in event-based vision demonstrate the possibility of converting a blurry image into a sharp video that explains the motion during the exposure interval (Pan et al., 2019, 2020; Wang et al., 2020; Song et al., 2022, 2024).

Learning activation functions. The Rectified Linear Unit (ReLU) (Nair & Hinton, 2010) represents a two-piece linear activation function, $f(x) = x$ if $x > 0$, and $f(x) = 0$ if $x \leq 0$ and has several variants. Our design is conceptually similar to the Piecewise Linear Unit (PWLU) (Zhou et al., 2021), where the activation is defined as an n-piece linear function with learnable slopes and intercepts. Kolmogorov-Arnold Networks (KANs) have also received wide attention, which build a deep model by stacking layers of parametric B-spline activation functions (Liu et al., 2024). In addition to the conceptual similarities, PPLNs are fundamentally different from PWLUs and KANs since PPLNs incorporate temporal modeling. Additionally, PPLNs allow discontinuities at segment endpoints and the learning of endpoint locations, neither of which is supported by PWLUs or KANs. Furthermore, the ReLUs are activation functions to be appended after prediction layers (*e.g.*, linear and convolution layers), whereas the PPLNs are designed to replace the prediction layers in temporal learning models.

3 Method

3.1 Overview

As shown in Figure 2 (a, b), a PPLN node implements the following mapping:

$$f : \mathbb{R}^k \times [0, 1] \to \mathbb{R} \tag{1}$$

where $\mathbf{x} \in \mathbb{R}^k$ is the k-dimensional non-temporal component of the input, and $t \in [0, 1]$ is the normalized input scalar timestamp. The mapping f converts the input $(\mathbf{x}, t)$ to a scalar in $\mathbb{R}$.

The first step in the calculation of f is to predict the linear coefficients, $\Theta = \{\mathbf{m}, \mathbf{b}, \mathbf{s}\}$, using the trainable parameters, W_m, W_b, W_s, and $\mathbf{w}_V$. Section 3.2 explains this process in detail. The predicted coefficients allow us to assemble the piecewise linear membrane potential function, as illustrated by the blue plot in the bottom-left corner of Figure 2 (a, b). To better handle the numerical instability, Section 3.3 discusses the procedure to smooth the boundaries of the predicted linear pieces. The smoothed function is then normalized by another predicted value $\overline{V}$, as explained in Section 3.4. Finally, the output of the PPLN node is given as $f(\mathbf{x}, t) = \sigma(\tilde{V}_\Theta(t))$ (*i.e.*, the smoothed, and normalized potential function evaluated at the timestamp of interest). While it is straightforward to construct a fully-connected network by stacking PPLN nodes, Section 3.5 discusses how to support convolution operations.

3.2 Coefficient prediction

Let n be a hyper-parameter denoting the number of line segments in the piecewise linear modeling. The parametric coefficients $\Theta = \{\mathbf{m}, \mathbf{b}, \mathbf{s}\}$ are given by:

$$\mathbf{m} = \tanh(W_m \cdot \mathbf{x}) \tag{2}$$

$$\mathbf{b} = W_b \cdot \mathbf{x} \tag{3}$$

$$\mathbf{s} = \text{softmax}(W_s \cdot \mathbf{x}) \tag{4}$$

where W_m, W_b, and W_s are $n \times k$ dimensional matrices containing trainable weights. $\mathbf{m} = (m_1, \ldots, m_n)^T$ and $\mathbf{b} = (b_1, \ldots, b_n)^T$ are the slopes and intercepts of n different line segments, respectively. $\mathbf{s} = (s_1, \ldots, s_n)^T$ defines the temporal interval size for each line segment. Let $t_0 = 0$ and $t_i = t_{i-1} + s_i$. With the softmax function, the temporal space $[0, 1]$ is divided into n non-overlapping intervals by $\mathbf{s}$. We predict the aforementioned coefficients and approximate the membrane potential as:

$$\tilde{V}_\Theta(t) := \begin{cases} m_1 t + b_1 & t_0 \le t < t_1 \\ m_2 t + b_2 & t_1 \le t < t_2 \\ \cdots \\ m_n t + b_n & t_{n-1} \le t \le t_n \end{cases} \tag{5}$$

Here, the hyperbolic tangent function restricts the slope, $\mathbf{m}$, preventing the exploding gradient problem commonly observed in temporal models (Pascanu et al., 2013; Bengio et al., 1994).

3.3 Smoothing

While we can build a network by stacking layers of the vanilla PPLN nodes described above, training presents a challenge for numerical stability. Equation (5) suggests that $\frac{\partial \tilde{V}_\Theta(t)}{\partial \mathbf{t}}$ is an all-zero vector. In other words, gradient-based optimizers (Bottou et al., 1991; Kingma & Ba, 2014) cannot update the values of t_i's (the location of segment endpoints).

To address this issue, we propose to smooth the segment boundaries. The key idea is to blend the linear pieces across adjacent intervals. Let $(x, \pi_i(x))$ be the point on the i^{th} predicted segment, that is, $\pi_i(x) := m_i x + b_i$.

Assuming $t_{i-1} \le t < t_i$ (*i.e.*, t belongs to the i^{th} predicted segment), the smoothed potential is:

$$\tilde{V}^T_\Theta(t) := w_l^{(i)} \pi_{i-1}(t) + (1 - w_l^{(i)} - w_r^{(i)}) \pi_i(t) + w_r^{(i)} \pi_{i+1}(t) \tag{6}$$

where the weights $w_{\leftarrow}^{(i)}$ and $w_{\rightarrow}^{(i)}$ are defined through the temperature hyper-parameter T:

$$w_l^{(i)} := \begin{cases} 0 & i = 1 \\ \Big(1 + \exp\big(T(t - t_{i-1})\big)\Big)^{-1} & i = 2, \ldots, n \end{cases}$$

$$w_r^{(i)} := \begin{cases} \Big(1 + \exp\big(T(t_i - t)\big)\Big)^{-1} & i = 1, \ldots, n-1 \\ 0 & i = n \end{cases}.$$

Importantly,

$$\lim_{T \to +\infty} \tilde{V}^T_\Theta(t) = \tilde{V}_\Theta(t).$$

The difference between the smoothed potential, $\tilde{V}^T_\Theta(t)$, and the unsmoothed potential, $\tilde{V}_\Theta(t)$, is that the smoothed gradients $\frac{\partial \tilde{V}^T_\Theta(t)}{\partial \mathbf{t}}$ do not vanish. We present a theorem with proof in the appendix

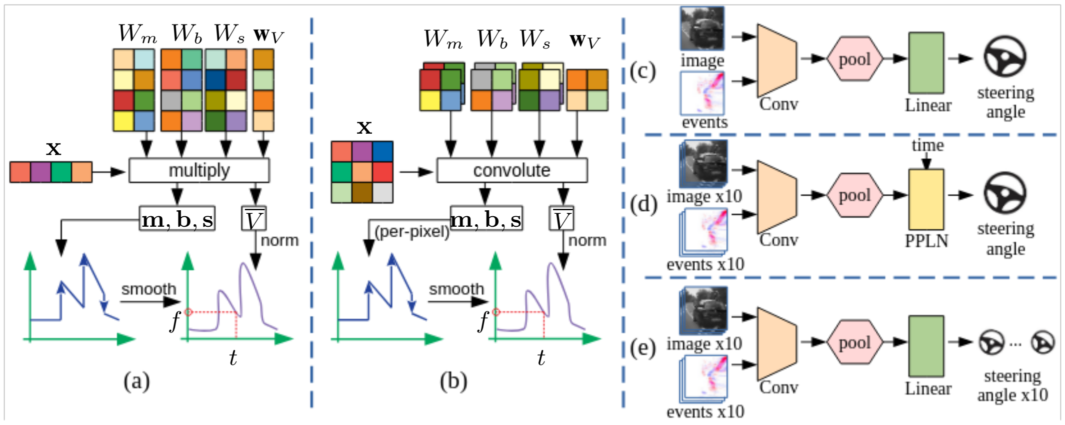

Figure 2: **(a)** A linear PPLN node, which maps the input $(\mathbf{x}, t)$ to output f. The trainable parameters are W_m, W_b, W_b, and $\mathbf{w}_V$. **(b)** A similarly structured 2D convolutional PPLN node. **(c)** The baseline architecture for steering angle prediction (Hu). **(d)** Our model. **(e)** The modified baseline (HuMod).

showing the local convergence properties of the piecewise linear model after smoothing. Assuming the underlying function to fit is indeed an n-piece piecewise linear function, the theorem states that the coefficients can be accurately learned from a set of noisy samples, provided that the noises are reasonably small, the coefficients are adequately initialized, and the temperature T is sufficiently large. For ease of discussion, the theorem uses segment endpoints $\mathbf{t} = \{t_i\}$ instead of interval lengths $\mathbf{s}$ for the parameterization. Their relation is given in Section 3.2.

Theorem 3.1. ***(Informal)*** *Consider an underlying n-segment piecewise linear function parameterized by $\Theta^\star = \{\mathbf{m}^\star, \mathbf{b}^\star, \mathbf{t}^\star\}$ as defined in (5). Let (τ_j, v_j), $j = 1, \ldots, m$ be m point samples, where $v_j = \tilde{V}_{\Theta^\star}(\tau_j) + \psi_j$ in which ψ_j is a small random noise.*

The L2 loss for the smoothed curve is defined by:

$$\mathcal{L}^T(\Theta) := \sum_{j=1}^{m} \left(\tilde{V}_\Theta^T(\tau_j) - v_j\right)^2. \tag{7}$$

Denote by $\Theta_T^\star$ the weights at which the minimum of (7) is attained at temperature T. Then we show that starting from some initial Θ_0 close to $\Theta^\star$, by applying vanilla gradient descent with appropriate temperature increase strategy and a learning rate $\eta = O(\frac{1}{T})$, Θ is guaranteed to converge to $\Theta_\infty^\star$ at a linear convergence rate.

Specifically, the error of recovered segments is bounded by:

$$\sup_{\tau_{\min} \le \tau \le \tau_{\max}} |\tilde{V}_{\Theta_\infty^\star}(\tau) - \tilde{V}_{\Theta^\star}(\tau)| < O(\max |\psi_j|), \tag{8}$$

where $\tau_{\min}$, $\tau_{\max}$ are the smallest and largest values among τ_j, for $j = 1, \ldots, m$, respectively.

In addition to the theorem above, the supplementary material uses ablation studies to discuss the practical implication of incorporating the smoothing operation.

3.4 Integral normalization

While the smoothing operator introduced above enriches temporal gradients, $\frac{\partial \tilde{V}_\Theta(t)}{\partial \mathbf{t}}$, the integral normalization operator addresses the issue that $\frac{\partial \tilde{V}_\Theta(t)}{\partial \mathbf{m}}$ and $\frac{\partial \tilde{V}_\Theta(t)}{\partial \mathbf{b}}$ are both very sparse vectors with only one non-zero entry out of all n elements. From Equation (5), we have:

$$\int_0^1 \tilde{V}_\Theta(t)dt = \frac{1}{2}\sum_{i=1}^{n} m_i(t_i^2 - t_{i-1}^2) + \sum_{i=1}^{n} b_i(t_i - t_{i-1}) \tag{9}$$

Let $\overline{V}$ be a parameter that controls the mean of $\tilde{V}_\Theta(t)$ when $0 \le t \le 1$. The integral normalization operator $\sigma(\cdot)$ is defined as:

$$\sigma(\tilde{V}_\Theta(t)) = \tilde{V}_\Theta(t) - \int_0^1 \tilde{V}_\Theta(t)dt + \overline{V} \approx V(t) \tag{10}$$