applied sciences

MDPI

Article

Neuromorphic Neural Engineering Framework-Inspired Online Continuous Learning with Analog Circuitry

Avi Hazan and Elishai Ezra Tsur *

Neuro-Biomorphic Engineering Lab, Department of Mathematics and Computer Science, The Open University of Israel, Ra'anana 4353701, Israel
* Correspondence: elishai@nbel-lab.com

Abstract: Neuromorphic hardware designs realize neural principles in electronics to provide high-performing, energy-efficient frameworks for machine learning. Here, we propose a neuromorphic analog design for continuous real-time learning. Our hardware design realizes the underlying principles of the neural engineering framework (NEF). NEF brings forth a theoretical framework for the representation and transformation of mathematical constructs with spiking neurons, thus providing efficient means for neuromorphic machine learning and the design of intricate dynamical systems. Our analog circuit design implements the neuromorphic prescribed error sensitivity (PES) learning rule with OZ neurons. OZ is an analog implementation of a spiking neuron, which was shown to have complete correspondence with NEF across firing rates, encoding vectors, and intercepts. We demonstrate PES-based neuromorphic representation of mathematical constructs with varying neuron configurations, the transformation of mathematical constructs, and the construction of a dynamical system with the design of an inducible leaky oscillator. We further designed a circuit emulator, allowing the evaluation of our electrical designs on a large scale. We used the circuit emulator in conjunction with a robot simulator to demonstrate adaptive learning-based control of a robotic arm with six degrees of freedom.

Keywords: OZ spiking neurons; NEF; Nengo; REACH; prescribed error sensitivity; neuromorphic engineering; neuromorphic computing; machine learning; online learning; real-time learning; adaptive robotics; neurorobotics; MuJoCo

Citation: Hazan, A.; Ezra Tsur, E. Neuromorphic Neural Engineering Framework-Inspired Online Continuous Learning with Analog Circuitry. *Appl. Sci.* **2022**, *12*, 4528. https://10.3390/app12094528

Academic Editor: DaeEun Kim

Received: 31 March 2022
Accepted: 27 April 2022
Published: 29 April 2022

1. Introduction

While proven incredibly valuable for numerous applications, ranging from robotics and medicine to economy and computational cognition, artificial intelligence (AI), in many ways, is nullified when compared with biological intelligence. For example, the Cockatiel Parrot can navigate and learn unknown environments at 35 km/h, manipulate objects, and use human language, with a brain consuming merely 50 mW of power [1]. Comparably, an autonomous drone with comparable mass and size consumes 5000 mW of power while being limited to pretrained flying in a known environment with limited capacity for real-time learning [2]. Deep learning with artificial neural networks (ANNs) is a predominant method in AI. ANNs, however, are limited to slow generalization with massive data, offline training, and batched optimization [3]. In contrast, biological learning is characterized by fast generalization and online incremental learning [4]. Spiking neural networks (SNNs) closely follow the computational characteristics of biological learning and stand as a new frontier of AI [5]. SNNs comprise densely connected, physically implemented "silicon neurons", which communicate with spikes [6]. SNNs were realized in various hardware designs, including IBM's TrueNorth [7], Intel's Loihi [8], the NeuroGrid [9], the SpiNNaker [10], and the BrainDrop [11].

Programming a neuromorphic system is a challenging endeavor, as it requires the ability to represent data, manipulate and retrieve it with spike-based computing. One theoretical framework designed to address these challenges is the neural engineering framework

(NEF) [12]. NEF brings forth a theoretical framework for representing high-dimensional mathematical constructs with spiking neurons for the implementation of functional large-scale neural networks. It was used to design a broad spectrum of neuromorphic systems ranging from vision processing [13] to robotic control [14]. NEF was compiled to work on each of the neuromorphic hardware designs listed above [15] via Nengo, a Python-based "neural compiler", which translates high-level descriptions to low-level neural models [16].

One of the most promising directions for neuromorphic systems is real-time continuous learning [6]. A neuromorphic continuous learning framework was recently shown to handle temporal dependencies spanning 100,000 time steps, converge rapidly, and use few internal state variables to learn complex functions spanning long time windows and outperforming state-of-the-art ANNs [17]. Neuromorphic systems, however, can realize their full potential only when deployed on neuromorphic hardware. While NEF was previously adopted for both digital [18] and hybrid (analog/digital) neuromorphic circuitry [11,19], we propose a detailed, fully analog design for NEF-based online learning.

The BrainDrop was designed to specifically tailor NEF. In this hybrid circuit, computation is held in analog circuitry and spike routing is held digitally. Learning in the BrainDrop is realized by digitally solving a least-squares optimization problem. While digital and hybrid neuromorphic designs have proven immensely important for the development of various neuromorphic applications, analog designs have been tremendously successful in vision [20] and sound [21] processing due to their unique tradeoff between performance and energy efficiency [19]. Here, we propose an entirely different approach based on the analog realization of PES learning. Thus, this study provides a full analog neuromorphic implementation of NEF-inspired learning. Our circuit design utilizes OZ, an analog implementation of the NEF-inspired spiking neuron we recently proposed [15]. OZ is a programmable spiking neuron that can support arbitrary response dynamics (Figure 1A–C). We used online learning to represent high-dimensional mathematical constructs (encoding and decoding with spiking neurons), transform one neuromorphic representation to another, and implement complex dynamical behaviors. We further designed a circuit emulator, allowing the evaluation of our electrical designs on a large scale. We used the emulator to demonstrate adaptive learning-based control of a six-degree-of-freedom robotic arm (Figure 1D). Our design supports the basic three fundamental principles of NEF (representation, transformation, and dynamics) and can therefore be of potential use for various neuromorphic systems.

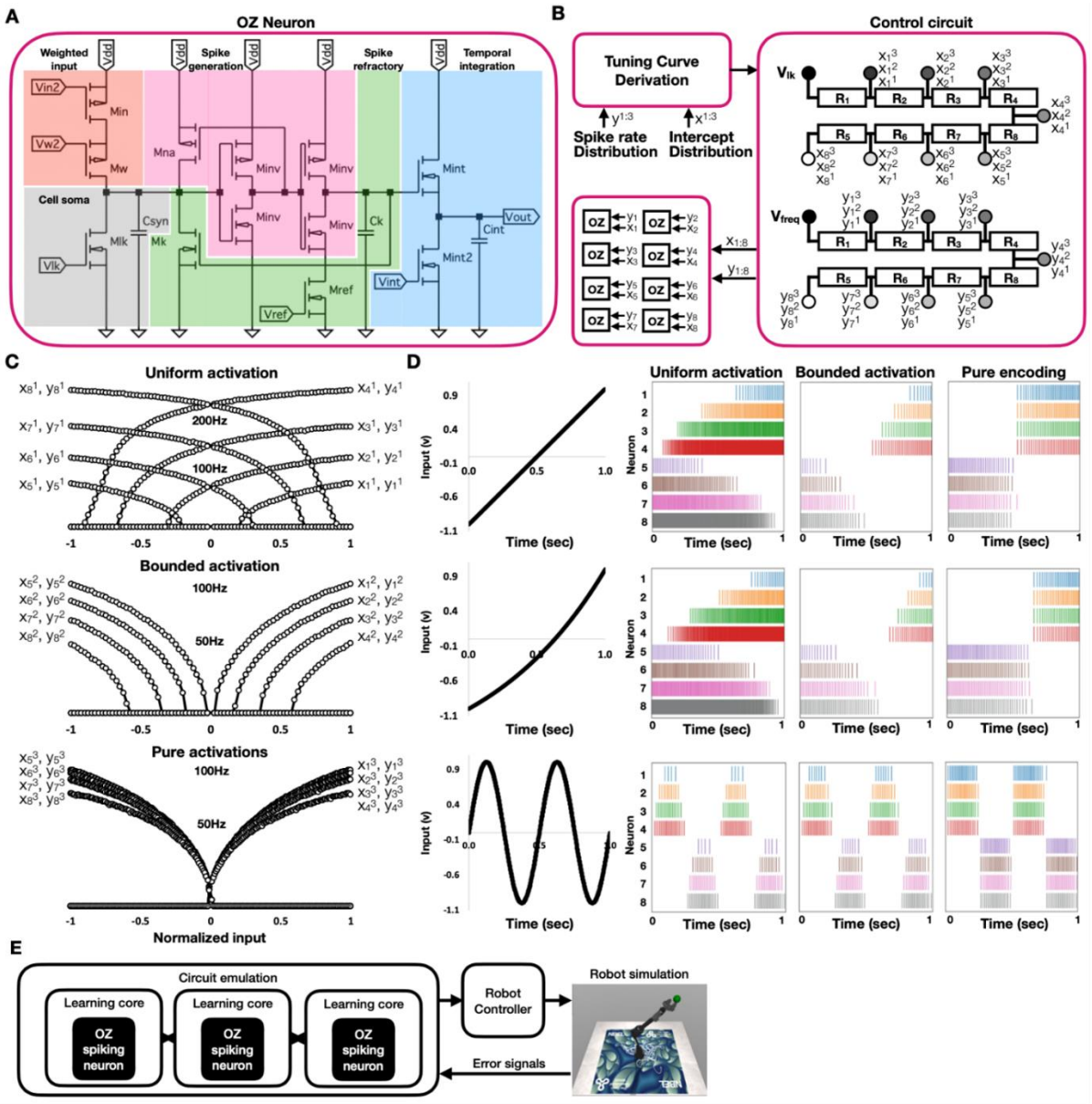

Figure 1. Neuromorphic encoding of mathematical constructs. (**A**) OZ circuit design; (**B**) A control circuit aiming at creating a distribution of intercepts ($x_1 - x_8$) and maximal firing rates ($y_1 - y_8$) for eight neurons' tuning; (**C**) Three tuning modalities: uniform (**top**), bounded (**middle**), and pure (**bottom**); (**D**) Raster plots of eight OZ neurons, defined by the tuning distribution set in (**C**) and driven by a linear (**top**), exponential (**middle**) and sinusoidal (**bottom**) input signals; (**E**) Schematic of this work's methodology.

2. Methods

NEF is based upon three fundamental principles: representation, transformation, and dynamics. Here, we show that our hardware PES-driven analog design can be used to implement these principles.

2.1. Neuromorphic Representation with NEF

NEF brings forth a theoretical framework for neuromorphic encoding of mathematical constructs with spiking neurons, allowing for the implementation of functional large-scale neural networks [12]. It provides a computational framework with which information, given in terms of vectors and functions, can be transformed into a set of interconnected

ensembles of spiking neurons. In NEF, spike train δ_i of neuron i in response to a stimulus x is defined as follows:

$$\delta_i(x) = G_i\left[\alpha_i e_i + J_i^b\right], \tag{1}$$

where G_i is a spiking neuron model, α_i is a gain term, e_i is the neuron's preferred stimulus (encoding vector), and J_i^b is a fixed background current. An ensemble of neurons can encode a high-dimensional vector, which can be linearly decoded as $\hat{x}$ according to:

$$\hat{x} = \sum_i^N a_i(x) d_i, \tag{2}$$

where N is the number of neurons, $a_i(x)$ is the postsynaptic low pass filtered response of neuron i to the stimulus x, and d_i is a linear decoder that was optimized to reconstruct x using least-squares optimization. Neuron's postsynaptic response is defined in NEF as:

$$a_i(\mathbf{x}) = \sum h_i * \delta_i\big(t - t_j(\mathrm{x})\big), \tag{3}$$

where h_i is the synaptic response function (usually an exponential with a time constant determined by the neurotransmitter type at the synapse), $*$ is the convolution operator, and $\delta_i\big(t - t_j(\mathrm{x})\big)$ is the spike train produced by neuron i in response to stimulus x, with spike times indexed by j.

Importantly, when the representation is distributively realized with spiking neurons, the number of neurons dramatically affects performance and stability. This is referred to as decoder-induced static noise S_N, and it is proportional to the number of neurons N according to:

$$S_N \sim 1/N^2 \tag{4}$$

2.2. Neuromorphic Transformation with NEF

Equations (1) and (2) describe how vectors are encoded and decoded using neural spiking activity in neuronal ensembles. Propagation of data from one ensemble to another is realized through weighted synaptic connections, formulated with a weight matrix. The resulting activity transformation is a function of x. Notably, it was shown that any function $f(x)$ could be approximated using some set of decoding weights d^{f} [12]. Defining $f(x)$ in NEF can be made by connecting the two neuronal ensembles A and B via neural connection weights $w_{ij}(x)$ using:

$$w_{ij} = d_i \otimes e_j, \tag{5}$$

where i is the neuron index in ensemble A, j is the neuron index in ensemble B, and d_i are the decoders of ensemble A, which were optimized to transform x to $f(x)$, and e_j are the encoders of ensemble B, which represents $f(x)$ and $\otimes$ is the outer product operation.

2.3. Prescribed Error Sensitivity

Connection weights, which govern the transformation between one representation to another, can also be adapted or learned in real time rather than optimized during model building. Weight adaptation in real time is of particular interest in AI, where unknown environmental perturbations can affect the error. One efficient way to implement real time learning with NEF is using the prescribed error sensitivity (PES) learning rule. PES is a biologically plausible supervised learning rule that modifies a connection's decoders d to minimize an error signal e. This error signal is calculated as the difference between the stimulus and its approximated representation: $\hat{x} - x$. The PES applies the update rule:

$$\Delta d = \lambda e \delta, \tag{6}$$

where λ is the learning rate. Notably, it was shown that when $a - \lambda\|\delta\|^2$ (denoted γ) is larger than -1, the error e goes to 0 exponentially with rate γ. PES is described at length in [22].

2.4. Neuromorphic Dynamics with NEF

We recently described neuromorphic dynamics with NEF in [13]. System dynamics is a theoretical framework concerning the nonlinear behavior of complex systems over time. Dynamics is the third fundamental principle of NEF, and it provides the framework for using SNNs to solve differential equations. It is essentially an integration of the first two NEF principles: representation and transformation, where transformation is used in a recurrent scheme. A recurrent connection (connecting a neural ensemble back to itself) is defined using $x(t) = f(x(t)) * h(t)$. A canonical description of a linear error-correcting feedback loop can be described using $\frac{dx}{dt} = Ax(t) + Bu(t)$, where $x(t)$ is a state vector, which summarizes the effect of all past inputs, $u(t)$ is the input vector, B is the input matrix, and A is the dynamic matrix. In NEF, this standard control can be realized by using:

$$\frac{dx}{dt} = A'x(t) + B'u(t), \tag{7}$$

where A' is the recurrent connection matrix, defined as $\tau A + I$, where I is the identity matrix, τ is the synapse decaying time constant, and B' is the input connection, which is defined as τB.

An oscillator is a fundamental dynamical system. A two-dimensional (2D) oscillator, which alternates the values of x_1 and x_2, at a rate r, can be defined recurrently using:

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 & r \\ -r & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = Ax. \tag{8}$$

To achieve an oscillatory dynamic in which $\frac{dx_1}{dt} = rx_2$ and $\frac{dx_2}{dt} = -rx_1$, the following recurrent connections: $x_1 = x_{1pre} + rx_2$ and $x_2 = x_{2pre} - rx_1$ are defined, achieving $x_1 = \frac{r}{\tau}x_2$ and $x_2 = \frac{-r}{\tau}x_1$. Implementing this model without inducing some initial value x_1 or x_2 will result in a silent oscillator, i.e., it will stand still at $(0, 0)$. However, when a stimulus is applied—even a very short stimulus—the oscillator is driven to oscillate indefinitely. A leaky oscillator can be defined by introducing κ as a dumping factor:

$$\begin{pmatrix} x_0 \\ x_1 \end{pmatrix} = (A - \kappa I) \begin{pmatrix} x_0 \\ x_1 \end{pmatrix}, \tag{9}$$

where I is the identity matrix.

2.5. OZ NEF-Inspired Spiking Neuron

OZ is an analog implementation of a NEF-inspired spiking neuron (Figure 1A). In this circuit design, the input voltage is transformed into a proportional current, driven into a voltage-amplifier leaky integrated and fire (LIF) neuron circuit. The neuron produces a spike train according to its response dynamic (tuning curve). Finally, the spike train is introduced into an integrator circuit to provide temporal integration. OZ neurons can be explicitly defined to feature arbitrary tuning curves following NEF neuron specifications. In NEF, a tuning curve is described with an intercept, the value for which the neuron starts to produce spikes at a high rate and a maximal firing rate. OZ response dynamic is defined by the values of V_{lk} and V_{ref}. V_{lk} controls the discharge rate of C_{mem} (via the voltage-amplifier LIF neuron), thus controlling the neuron's intercept, and V_{ref} controls the spikes' refractory period, thus controlling the neuron's firing rate. OZ design was shown to have high predictability of the produced spike trains and a complete correspondence with NEF across firing rates, intercepts, and encoders [15].

2.6. Circuit Simulation and Emulation

Circuit simulations in this work were executed using LTspice by Analog Devices [23]. The simulator is based on the open-sourced SPICE framework [24], which utilizes the numerical Newton–Raphson method to analyze nonlinear systems [25]. We used LTspice

to evaluate the performance of our circuit design on a small scale. To efficiently demonstrate our circuit design on a larger scale, we designed a scalable Python-based emulator, supporting the emulation of many OZ neurons and numerous PES-based learning circuit blocks. We further compared our results to Nengo-based simulations [16].

2.7. Robotic Arm Simulation

We recently described our robotic arm simulation in [14]. To simulate a robotic arm, we used the Multi-Joint dynamics with Contact (MuJoCo) physics simulation framework. The mechanical description of the robotic arm and its accurate joint dynamics were specified using CAD-derived specifications and inertia and mass matrices provided by Trossen Robotics (Downers Grove, IL, USA). The arm provides 6 DOF, 82 cm reach, and 1.64 m span. The simulation was developed using Python. Arm control was evaluated using Nengo, a Python package for building, testing, and deploying NEF-based neural networks and our circuit emulator described below.

3. Results

3.1. Circuit Design

3.1.1. Control Circuit

OZ neurons are configured using two control signals (V_{lk}, V_{ref}; see Figure 1A), setting the neuron's intercept and maximal firing rate. However, individually controlling each neuron is tedious and not feasible in a large-scale design. We would therefore prefer programming the neurons to exhibit different distributions of response dynamics. One way in which a distribution can be configured is with simple resistor ladders (a serially connected set of resistors) (Figure 1B). One ladder sets the neurons' intercepts, and the other sets their maximal firing rate. In this simple control design, changing the voltages that feed the ladders uniformly shifts the response of the controlled OZ ensemble in the representation space. We simulated this control circuit to generate various distributions of tuning curves, including a uniform and bounded intercept distribution (by feeding different input values to both resistor ladders) and a pure configuration in which the intercepts were set to zero (Figure 1C). Each ensemble was driven by linear, exponential, and sinusoidal inputs to highlight their different response dynamics (Figure 1D). Note that in the design of OZ, a preprocessing module generates a signal for positively encoded and negatively encoded neurons. Therefore, by assuming symmetry of activations (such as the one demonstrated in Figure 1C), only four activations should be set for the configuration of eight neurons.

3.1.2. Learning Core

We propose a novel analog circuitry with which OZ neurons could be utilized to learn neuromorphic representations, transformation, and dynamics using the realization of the PES learning rule. This circuit design aims at online learning with neuromorphic analog circuitry, and it is scalable to comprise numerous OZ learning cores. The system schematic is shown in Figure 2A. In each learning core, a normalized input signal is preprocessed to drive positively and negatively encoded neurons. The preprocessing module is slightly differently configured for each encoding modality, simplified here for clarity. See [15] for a full description.

With OZ neurons, the input signal is rate-coded (following the neuron's tuning curve) and temporally integrated. The integrated signal is driven into a learning block alongside a normalized error signal and a learning rate. The learning block (Figure 2B) processes the signal and transforms it to an optimized value by minimizing the error signal using hardware realization of PES learning. The learning block is also responsible for the maintenance of the neuron's weight. Finally, all learning blocks' outputs are summed up by an adder block achieving an estimated signal. The estimated signal is subtracted from the desired signal, providing the error signal. The error signal is divided by the number of neurons (via a voltage divider, not shown in the diagram) and sent back to the learning block.

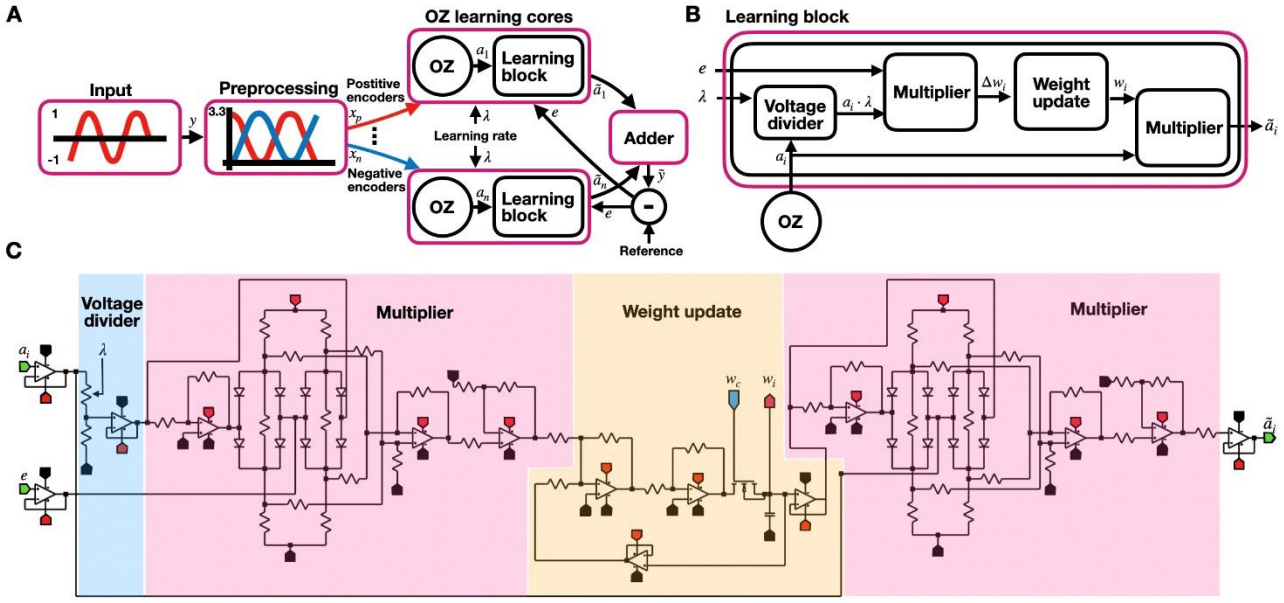

Figure 2. OZ Learning core. (**A**,**B**) Schematic of a learning core (**A**), in which a learning block (**B**) is utilized to provide PES-driven online learning; (**C**) A Circuit implementation of a learning block, utilizing a voltage divider, two multipliers, and a weight update module.

The electrical circuit constituting the learning block is shown in Figure 2C. The learning block circuit comprises a voltage divider (accounting for a learning rate, colored blue), two multipliers (colored purple), and a weight update module (colored orange). Analog multipliers were implemented by subtracting the outputs of two analog squaring circuits. One squaring circuit is driven by the summation of the two signals (x, y) and the other by their difference, following: $(x+y)^2 - (x-y)^2 = 4xy$. A differential amplifier further modulates the resulting value to factor out the constant. The diode bridge operates in an extensive frequency range, and its square law region is at the core of the squaring circuit. The left diode bridge handles $x+y$ and the right bridge handles $(x-y)$ (y is negated with an inverting op-amp). The squaring circuit's output current can be approximated with Taylor's series. As the differential output across the diode bridges is symmetric, each bridge's output comprises the even terms of the combined Taylor expansions. Odd terms are removed due to the four diode currents, as they produce frequency components outside the multiplier's passband. Therefore, the resulting output of the circuit is proportional to the square of its input.

The first multiplier multiplies the normalized error with the neuron's temporally integrated spikes, constituting a weight update. Weights are implemented with a memory cell (transistor-capacitor), allowing the maintenance of negative values at low overhead. Using a recurrently connected summing amplifier, the weight update circuit sums the updated value with its current weight value. The second multiplier multiplies the weight with the neuron's temporally integrated spikes providing the neuron's output.

3.2. Circuit Simulation

In this section, we show that our hardware PES-driven analog design can be used to implement NEF's three fundamental principles: representation, transformation, and dynamics (described above). The results below were generated using SPICE, with the exceptions of Figures 7 and 8, in which the results were generated using our Python-based emulator (described below), and Figure 3D, where the purple traces were generated using Nengo.

3.2.1. Representation

In NEF-driven representation, input signals are distributively *encoded* with neurons as spikes (following each neuron's tuning) and *decoded* by either calculating a set of decoders (Equation (2)) or learning a set of weights (Equation (5)) via PES learning (Equation (6)). In both cases, neuromorphic representation entails a reference signal (supervised learning).

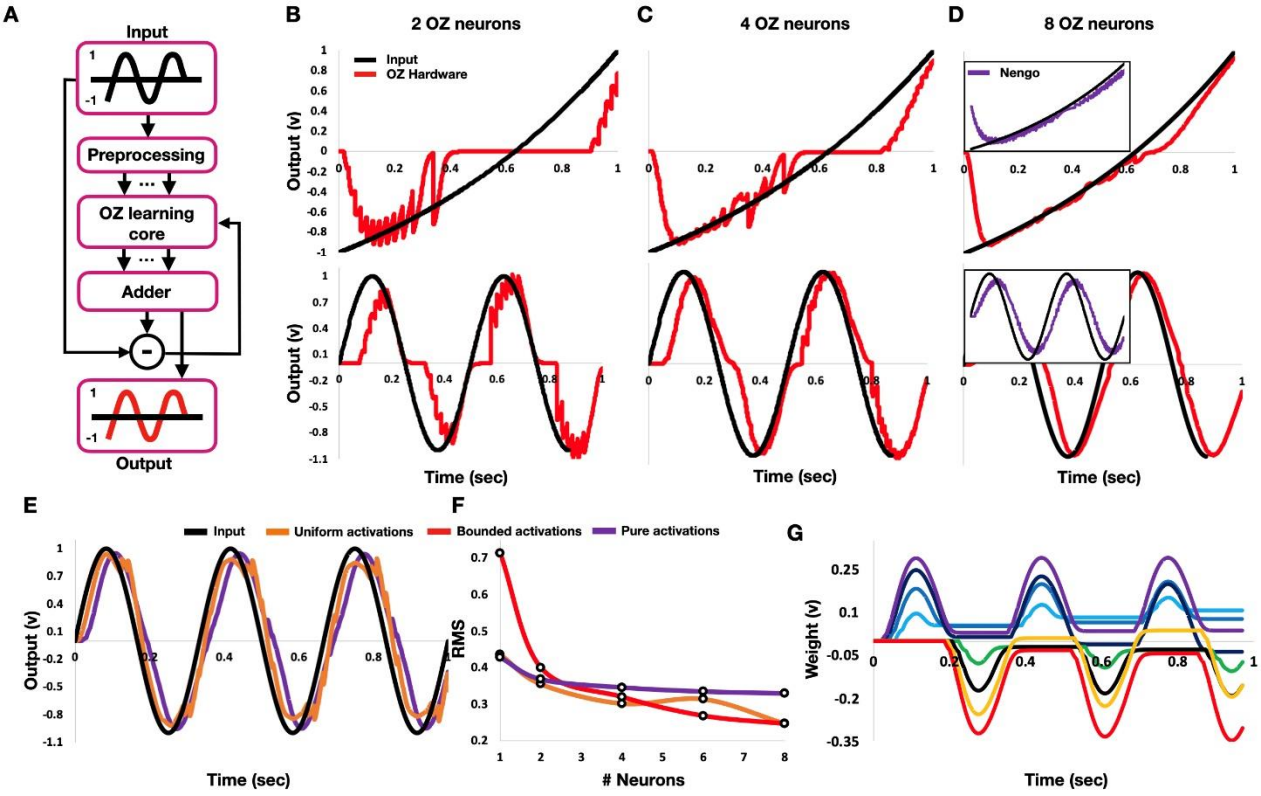

Figure 3. Neuromorphic representation. (**A**) System schematic for a neuromorphic representation of input values; (**B**–**D**) Representation of exponential (**top**) and sinusoidal (**bottom**) input signals with two, four, and eight neurons. Representation was based on neurons with a bounded distribution of activation (Figure 1C, middle); (**E**) Representation of a sinusoidal input with eight neurons featuring uniform (Figure 1C, top) and pure (Figure 1C, bottom) distributions of activation; (**F**) Representational error RMS with bounded, uniform, and pure activations with one to eight neurons; (**G**) Continual weight tuning of each of the eight neurons during representation.

We utilized our proposed learning block (Figure 2B,C) to realize neuromorphic representation with PES learning, using the input signal itself as a reference signal (Figure 3A). We used the system to encode and decode exponential and sinusoidal signals with two, four, and eight OZ neurons (Figure 3B–D). As expected, following Equation (4), as the number of neurons increases, the learning system's performance improves. Our hardware simulation-derived results (Figure 3D, red traces) closely follow Nengo's NEF-based software simulation (Figure 3D, purple traces), with a cross-correlation similarity (sliding dot product) of 0.872 ± 0.032. We show that an analog learning system comprising only 8 OZ neurons can accurately represent the input with a swift convergence toward the represented value.

As described above, representation is highly dependent on neuron tuning. The results shown in Figure 3B–D were derived using neurons with a bounded activation distribution. We further represented the sinusoidal input with neurons characterized by uniform and pure activations, following Figure 1C. The results are shown in Figure 3E. We evaluated this representation using the three activation schemes with one to eight neurons by calculating the error's root mean square (RMS). Our results demonstrate the superior performance for

a bounded distribution of neuron tuning (Figure 3F). The continually changing weights of each neuron are shown in Figure 3G, demonstrating continual online learning.

An essential characteristic of NEF is the ability to represent high-dimensional mathematical constructs with high-dimensional neurons. OZ neurons can be driven with high-dimensional signals (using few weighted inputs), featuring high-dimensional tuning [15]. The analog learning system schematic is shown in Figure 4A. By setting a similar weight for the two incoming signals, we derive the neuron's tuning shown in Figure 4B. We used the circuit described in Figure 4A, with eight 2D spiking neurons to represent a 2D signal, wherein one dimension follows an exponentially rising signal and the other, a sinusoidal wave. Representation results are shown in Figure 4C, and the error traces are shown in Figure 4D.

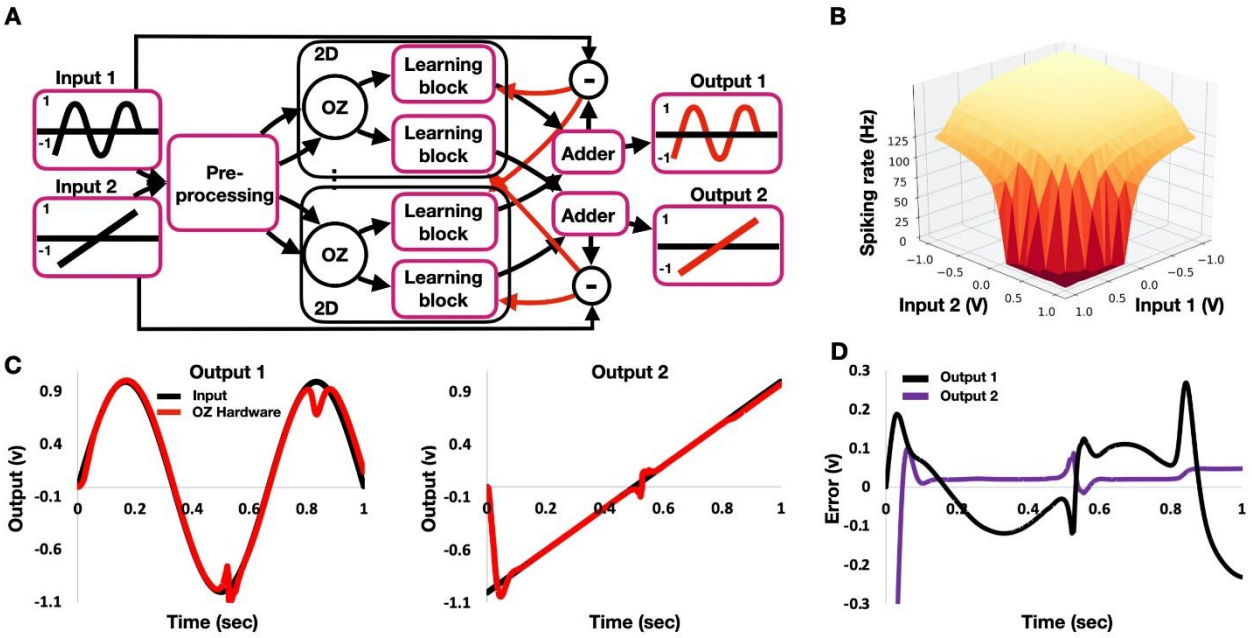

Figure 4. 2D Neuromorphic representation. (**A**) A schematic of 2D representation of exponential and sinusoidal signals using OZ learning cores; (**B**) 2D tuning curve, realized with two equal input weights; (**C**) Decoding of the two input signals; (**D**) Error traces for each decoded signal.

3.2.2. Transformation

Whereas in signal representation, we represent the input signal itself; in signal transformation, we represent some function of the input signal. Here, the system was utilized to neuromorphically perform squaring of an input sinusoidal signal (Figure 5A). The transformation results with one positively encoded neuron and eight bounded neurons, as shown in Figure 5B. Whereas one positively encoded neuron cannot account for the input signal's negative phase, it provides a temporal output only at the positive sinus phase. With eight neurons, however, the results show accurate transformation (Figure 5C). We measured transformation error with bounded, uniform, and pure activations with one to eight neurons. The results show superior performance for a uniform distribution of neuron tuning (Figure 5D). The neuromorphic system presented herein continually modulates neuronal weights. Weight tuning for each of the eight spiking neurons during transformation is shown in Figure 5E.

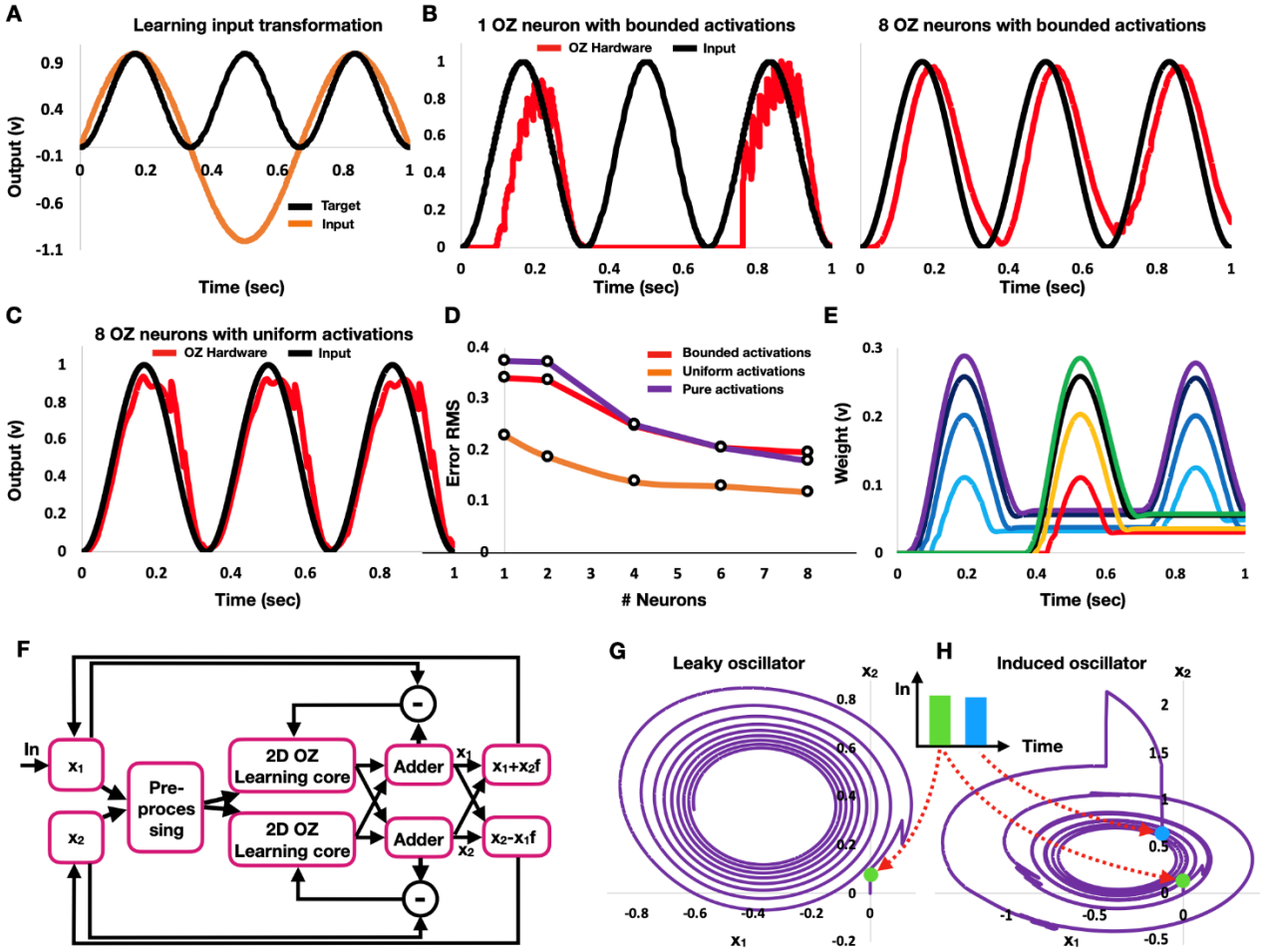

Figure 5. Neuromorphic transformation and dynamics. (**A**) Input (sinus wave) and target transformation (squared input); (**B**) Transformation with one (positively encoded) and eight OZ neurons with a bounded distribution of activations; (**C**) Transformation with eight OZ neurons with a uniform distribution of activations; (**D**) Transformation error with 1 to 8 OZ neurons with bounded, uniform, and pure distribution of activations; (**E**) Continual weight tuning of each the eight neurons during transformation; (**F**) System schematic for 2D oscillatory dynamic; (**G**) Hardware defined leaky oscillation. Oscillation is induced by a single 5 ms pulse (green); (**H**) A leaky oscillator is sequentially induced by two 5 ms pulses (green, blue), achieving maintained oscillation.

3.2.3. Dynamics

Neuromorphic representation and transformation are the first two main pillars of NEF. The third is the realization of a dynamical system. Here, we used our circuit design to implement an induced leaky oscillator (Equation (9)). The system schematic is shown in Figure 5F. This system utilizes our 2D representation scheme, described above in Figure 4. We traced the system's two dimensions (x_1 and x_2) throughout time, following induction with a single three mSec pulse (driven to x_1). The resulting oscillatory dynamic is shown in Figure 5G. Oscillation slowly converges back to zero at a rate determined by the hardware's leaky characteristic. When induced again, oscillation can be maintained, as demonstrated in Figure 5H, where two five mSec pulses are spaced by two seconds.

3.2.4. Worst-Case Analysis

Analog circuit elements (e.g., resistors, capacitors, transistors) are prone to process, voltage, and temperature (PVT) variations. "Process" in this case refers to manufacturing as a measure of the statistical variability of the physical characteristics from component to component as they come off the production line (ranging from variations in mask alignment

and etching times to doping levels). These variations affect the electrical parameters of the components, such as the sheet and contact resistance. Analog components also change in time to their endurance limit (the stress level below which an infinite number of loading cycles can be applied to a material without causing fatigue failure). Here, we used Monte Carlo-driven variations to study: (1) The way our hardware design handles a high degree of component variation; and (2) To compare the traditional variation-based spanning of a representation space with the programmed neurons' tuning approach. In each simulation run, all components in our circuit design were varied within an explicitly defined variation rate (e.g., in the 5% variation case study, the 10 nF capacitors featured in our OZ circuit design will randomly be specified in the 9.5–10.5 nF range). Transistors were similarly varied in their sizes. The level of process variation increases as the process size decreases. For example, a fabrication process that decreases from 350 nm to 90 nm will reduce chip yield from nearly 90% to a mere 50%, and with 45 nm, the yield will be approximately 30% [26]. Here, we simulate 100 Monte Carlo runs with 3, 5, and 7% variability. The resulting neurons' tuning in the bounded distribution of intercepts and firing rates and with a single setpoint (used for the variation-based spanning of representation space) are shown in Figure 6A. The results show that the intercepts are much more prone to variation than the neurons' firing rate. Importantly, we show that relying on process variation for the manifestation of neurons with heterogeneous tuning curves is inadequate compared to a predefined distribution of neuron tuning (Figure 6B). These results further demonstrate that our learning circuit design can compensate for process variation.

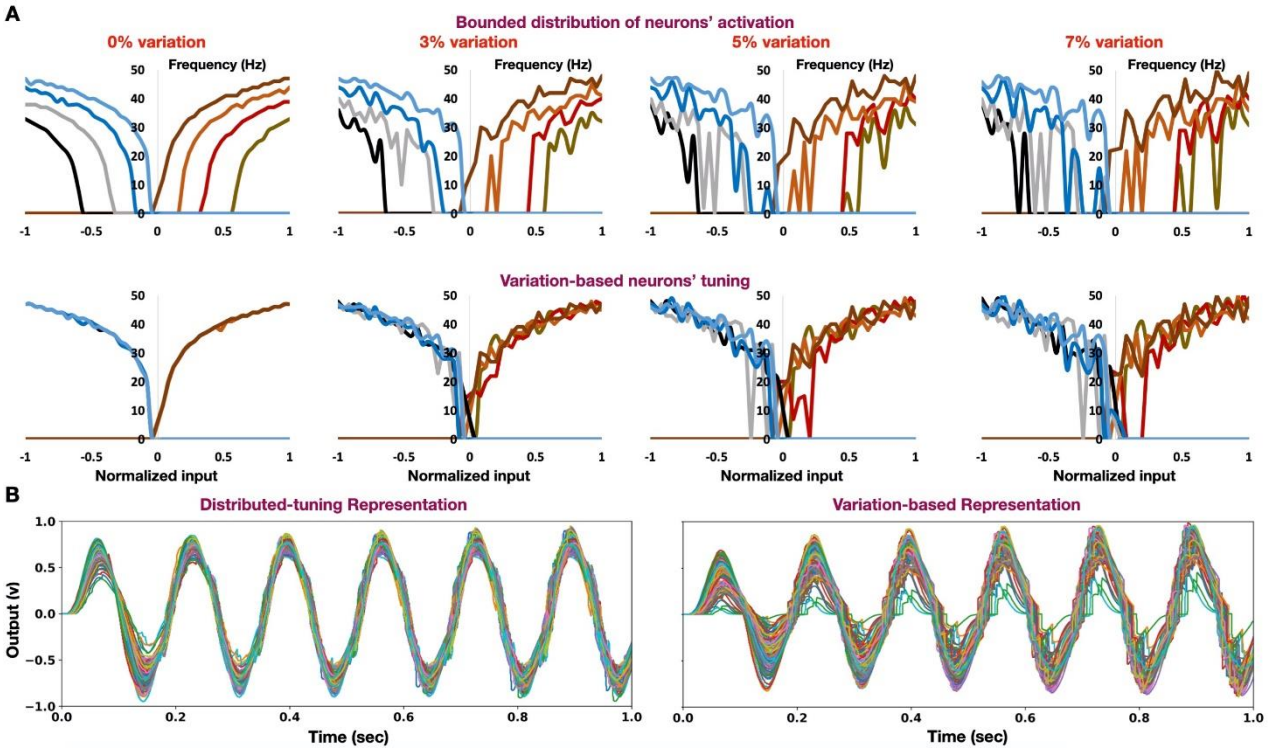

Figure 6. Worst-case analysis and variation-based tuning representation. (**A**) Bounded (**top**) and variation-based (**bottom**) tuning curves for eight OZ neurons with 0, 3, 5, and 7% transistor size variation (**left to right**); (**B**) Representation of a sinusoidal wave with 100 Monte-Carlo driven runs of eight neurons with a bounded distribution of activation (**left**) and variation-based (7%) (**right**) OZ neurons.

3.2.5. Circuit Emulator

To efficiently demonstrate our circuit design on a large scale, we designed a neural emulator. Our emulator is a scalable Python-based framework designed to support compil-

ing, testing, and deploying OZ-based SNNs, and support PES-based learning as described above. In contrast to the SPICE-driven simulations described above, this Python-based emulator realizes the SPICE-derived component behavior without simulating the actual components, allowing an efficient evaluation of the circuit. The emulator is time-based with a predefined simulation time and number of steps. At each step, the emulator's scheduler traverses a list of SimBase objects, activating them. The SimBase object structure constitutes the network design, and it is up to the user to define. Each SimBase object is aware of the simulation time step via a configuration class. Its responsibility is to process the input data received via a voltage or a current source interface object. Following each activation step, each object stores its resulting state. Each building block (learning block, error block, etc.) has a corresponding model created using its SPICE simulation with varying input signals. Blocks can be built hierarchically. For example, the OZ neuron block comprises the pulse current synapse block, which comprises a current source. The emulator is schematically shown in Figure 7A.

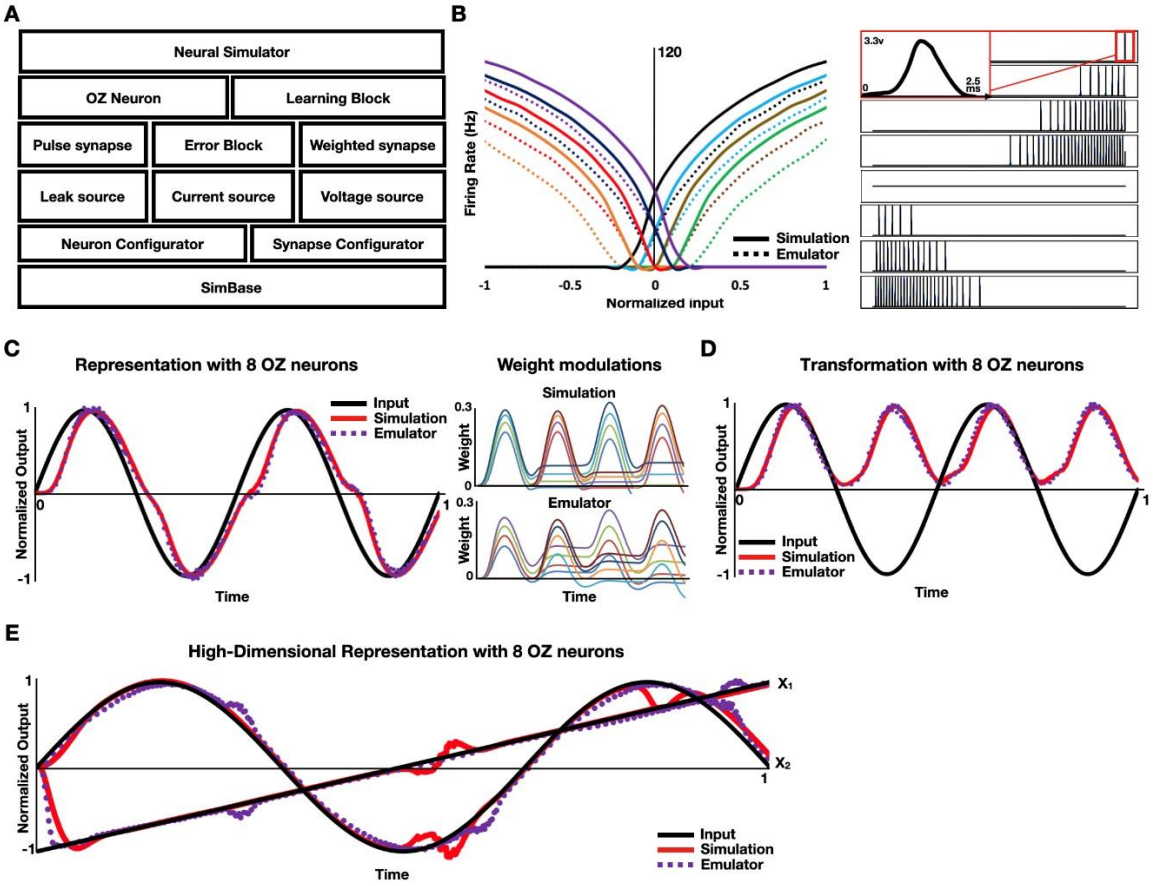

Figure 7. Circuit emulator. (**A**) Emulator schematic; (**B**) Eight uniformly distributed tuning curves for eight OZ neurons in both SPICE (lines) and OZ emulator (dashed lines). A raster plot showing the spikes generated by the eight neurons of the emulator as a response to a linearly increasing input voltage (−1 to 1 V) is shown on the right; (**C**) A representation of a sinusoidal input wave with eight neurons, in both SPICE and OZ emulator. Weight modulations are shown on the right; (**D**) A transformation of a sinusoidal input wave to its squared value with eight neurons, in both SPICE and OZ emulator; (**E**) Representation of both linear and sinusoidal voltage inputs with high-dimensional representation in both SPICE and OZ emulator.

To demonstrate the emulator, the generated neuron's tuning curves and the corresponding generated spikes are shown in Figure 7B. A SPICE-made spike was encapsulated in the emulator and iteratively generated in Python following the desired tuning curve,

specified by OZ's voltage signals described above. SPICE and emulator-generated neuron tuning show high correspondence. Similar to Figure 3D, we used our emulator to represent sinusoidal waves with eight neurons (Figure 7C). The results show a very high correlation with the SPICE circuit simulation (cross-correlation similarity (sliding dot product) of 0.904 ± 0.01), thus demonstrating the accurate emulation of the learning core. Weight modulations in both the SPICE simulation and the emulator show comparable patterns. Similarly, high-dimensional representation (Figure 4C) and transformation (Figure 5B) demonstrate the emulator's accuracy (Figure 7D,E).

3.2.6. Application in Adaptive Neurorobotics

In the past few decades, multijoint open-chain robotic arms have been utilized in a diverse set of applications, ranging from robotic surgeries [27] to space debris mitigation [28]. The control of robotic arms is currently dominated by proportional, integral, and derivative (PID)-based modeling. Such a model aims to accurately represent the controlled system. It would capture the effect of external perturbations and the system's internal dynamics on its ability to move. Thus, it provides signals for movement control, given a desired location. However, in a human collaborative-assistive setting, the controller should consider kinematic changes in the system, such as object manipulation of an unknown dimension or at an unknown gripping point. Neuromorphic systems have been shown to outperform PID-based implementation of the required nonlinear adaptation, particularly in their ability to handle high degree-of-freedom systems. One possible implementation for neuromorphic control is the recurrent error-driven adaptive control hierarchy (REACH) model proposed by DeWolf and colleagues [29]. REACH is powered by PES, implemented using NEF, realized within the Nengo development environment, and open-sourced by Applied Brain Research Inc. The model has been demonstrated to control a three-link, nonlinear arm through complex paths, including handwritten words and numbers. It can adapt to environmental changes (e.g., an unknown force field) and changes to the physical properties of the arm (e.g., tear of joints) (Figure 8A).

To demonstrate the applicability of our circuit design, we used the OZ and our learning core emulator to implement REACH on a 6 degree-of-freedom robotic arm in a physical simulator (Figure 8B). We have implemented two simulations. In the first simulated scenario, we applied an external force field on the robot's joints. The arm, therefore, cannot accurately reach the specified target points without adaptation, as the internal joint's dynamic does not consider unknown perturbations (Figure 8C). In the second simulated scenario, we used adaptive signals, allowing the arm to adjust its behavior in real time using PES. We used the classical REACH Nengo-driven model and our circuit emulator to power the arm adaptation. Both Nengo and OZ show similar adaptation patterns, allowing the arm to reach the desired target points accurately while an external force field is applied (Figure 8D).

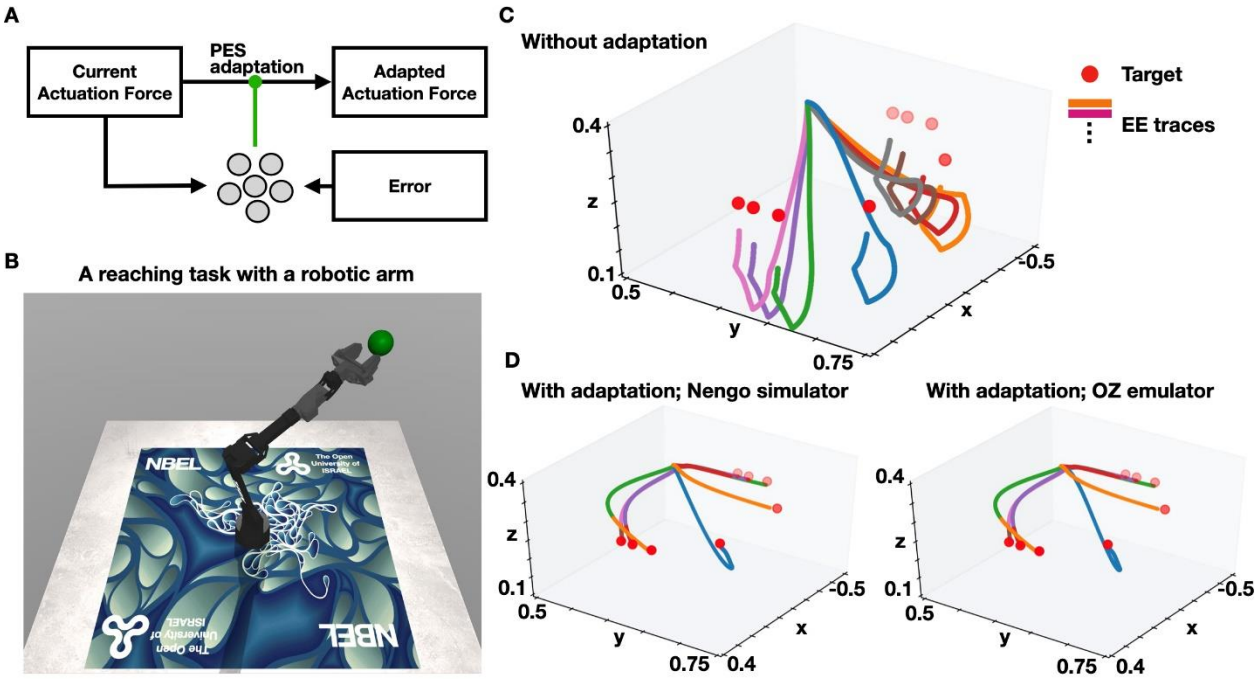

Figure 8. Application in adaptive neurorobotics. (**A**) A simplified schematic of the REACH model for adaptive robotic control; (**B**) A screenshot from the simulated arm and a designated target point in space; (**C**) Reaching eight target points while an external force is applied without adaptation; (**D**) Reaching eight target points while an external force is applied with adaptation using Nengo-based simulation (**left**) and OZ emulator (**right**).

4. Discussion

NEF is one of the most utilized frameworks in neuromorphic computing. It has been utilized to implement a wide range of neuromorphic applications, ranging from robotics [14] to the most comprehensive functional cognitive model [30]. NEF is based on three main pillars: representation of high-dimensional mathematical constructs, the transformation of one representation to another, and the two's utilization to build dynamical systems. In this work, we propose a fully analog circuit design, which realizes these three main pillars. Our design comprises the realization of PES, a neuromorphic online learning rule, and the utilization of OZ, a spiking neuron model tailored to the NEF configuration.

Previous designs utilized NEF to implement neuromorphic functional circuits. For example, Intel's Loihi can be programmed to work with the NEF-based Nengo compiler [18], realizing, among other features, PES learning [31]. Although Loihi, TrueNorth [32], and SpiNNaker [33] can be programmed with Nengo, they are digital circuits. NEF was also compiled to work on hybrid neuromorphic designs, such as the NeuroGrid [19] and its later successor, the BrainDrop [11]. While digital and hybrid neuromorphic were used to support a wide array of neuromorphic applications, analog designs have been tremendously successful in signal processing due [19]. Analog components, however, are particularly vulnerable to fabrication variability. There are several techniques to reduce the process variation; for example, adding dummy transistors to pad the operational transistors in the layout stage. Fortunately, heterogeneous neuronal tuning is desirable with NEF, as it provides the variability in activation needed for spanning a representation space. Circuit variability was shown to be essential for adequately spanning a representation space [19]. NEF's computational approach, therefore, embraces variability. However, we show that relying on process variation alone may require a large number of neurons. We demonstrated that by programming neuron tuning, we could better span a representation space per a given number of neurons. More importantly, even though a postsilicon calibration sequence can be used to compensate for process variation during neuron programming, we show

that our learning circuit can inherently compensate for it. Our circuit design can therefore learn to adapt to changes within itself.

To conclude, in this work, we propose a design of a novel analog circuitry for online learning, implementing the PES neuromorphic learning rule using NEF-specified spiking neurons (OZ). We demonstrated for the first time the representation and transformation of high-dimensional mathematical constructs using NEF-specified spiking neurons using our online learning analog circuitry. We illustrate the design of dynamical oscillatory dynamics by combining NEF-driven neuromorphic representation and transformation with analog circuitry. We further propose a new circuit emulator (written with Python), allowing us to efficiently simulate our analog circuitry on a large scale (thousands of spiking neurons and learning circuits). Finally, we demonstrate adaptive control of a robotic arm using a physics-aware robotic framework (MuJoCo) and using analog spiking neurons, featuring representation, transformation, and our circuit emulator.

The next design improvement to be further considered is our naïve control circuit. The control circuit presented herein is limited, as the resistance values governing the neurons' tuning curves are constant. Therefore, changing the voltages that feed the control circuit uniformly shifts the neurons' representation characteristics. A more advanced resistance design, via digitally controlled resistors in a hierarchical pattern or via a diffusor (used in the design of the BrainDrop), will enable dynamic nonlinear configurations. Furthermore, we aim to advance to a full VLSI large-scale design of the proposed circuit.

Author Contributions: A.H. designed the circuits and performed circuit simulation and analysis; E.E.T. conceptualized the research, supervised the circuit design, and wrote the manuscript. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Open University of Israel research grant.

Data Availability Statement: The circuit simulation files, and the emulator are available in: http://nbel-lab.com/ (accessed on 15 March 2022).

Acknowledgments: The authors would like to thank the members of the Neuro and Biomorphic Engineering Lab for the discussions.

Conflicts of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

References

1. Olkowicz, S.; Kocourek, M.; Lučan, R.K.; Porteš, M.; Fitch, W.T.; Herculano-Houzel, S.; Němec, P. Birds have primate-like numbers of neurons in the forebrain. *Proc. Natl. Acad. Sci. USA* **2016**, *113*, 7255–7260. [CrossRef] [PubMed]
2. Palossi, D.; Loquercio, A.; Conti, F.; Flamand, E.; Scaramuzza, D.; Benini, L. Ultra low power deep-learning-powered autonomous nano drones. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Madrid, Spain, 1–5 October 2018.
3. Parisi, G.I.; Kemker, R.; Part, J.L.; Kanan, C.; Wermter, S. Continual lifelong learning with neural networks: A review. *Neural Netw.* **2019**, *113*, 54–71. [CrossRef] [PubMed]
4. Kouvaris, K.; Clune, J.; Kounios, L.; Brede, M.; Watson, R.A. How evolution learns to generalise: Using the principles of learning theory to understand the evolution of developmental organisation. *PLoS Comput. Biol.* **2017**, *13*, e1005358. [CrossRef] [PubMed]
5. Tsur, E.E. *Neuromorphic Engineering: The Scientist's, Algorithm Designer's, and Computer Architect's Perspectives on Brain-Inspired Computing*; CRC Press: Boca Raton, FL, USA, 2021.
6. Marković, D.; Mizrahi, A.; Querlioz, D.; Grollier, J. Physics for neuromorphic computing. *Nat. Rev. Phys.* **2020**, *2*, 499–510. [CrossRef]
7. Debole, M.V.; Appuswamy, R.; Carlson, P.J.; Cassidy, A.S.; Datta, P.; Esser, S.K.; Garreau, G.J.; Holland, K.L.; Lekuch, S.; Mastro, M.; et al. TrueNorth: Accelerating From Zero to 64 Million Neurons in 10 Years. *Computer* **2019**, *52*, 20–29. [CrossRef]
8. Davies, M.; Srinivasa, N.; Lin, T.-H.; Chinya, G.; Cao, Y.; Choday, S.H.; Dimou, G.; Joshi, P.; Imam, N.; Jain, S.; et al. Loihi: A Neuromorphic Manycore Processor with On-Chip Learning. *IEEE Micro* **2018**, *38*, 82–99. [CrossRef]
9. Benjamin, B.V.; Gao, P.; McQuinn, E.; Choudhary, S.; Chandrasekaran, A.R.; Bussat, J.-M.; Alvarez-Icaza, R.; Arthur, J.V.; Merolla, P.A.; Boahen, K. Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations. *Proc. IEEE* **2014**, *102*, 699–716. [CrossRef]
10. Furber, S.; Galluppi, F.; Temple, S.; Plana, L.A. The SpiNNaker Project. *Proc. IEEE* **2014**, *102*, 652–665. [CrossRef]

11. Neckar, A.; Fok, S.; Benjamin, B.V.; Stewart, T.C.; Oza, N.N.; Voelker, A.R.; Eliasmith, C.; Manohar, R.; Boahen, K. Braindrop: A Mixed-Signal Neuromorphic Architecture with a Dynamical Systems-Based Programming Model. *Proc. IEEE* **2018**, *107*, 144–164. [CrossRef]
12. Eliasmith, C.; Anderson, C.H. *Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems*; MIT Press: Cambridge, MA, USA, 2004.
13. Tsur, E.E.; Rivlin-Etzion, M. Neuromorphic implementation of motion detection using oscillation interference. *Neurocomputing* **2020**, *374*, 54–63. [CrossRef]
14. Zaidel, Y.; Shalumov, A.; Volinski, A.; Supic, L.; Tsur, E.E. Neuromorphic NEF-Based Inverse Kinematics and PID Control. *Front. Neurorobot.* **2021**, *15*, 631159. [CrossRef] [PubMed]
15. Hazan, A.; Tsur, E.E. Neuromorphic Analog Implementation of Neural Engineering Framework-Inspired Spiking Neuron for High-Dimensional Representation. *Front. Neurosci.* **2021**, *15*, 109. [CrossRef] [PubMed]
16. Bekolay, T.; Bergstra, J.; Hunsberger, E.; DeWolf, T.; Stewart, T.; Rasmussen, D.; Choo, X.; Voelker, A.; Eliasmith, C. Nengo: A Python tool for building large-scale functional brain models. *Front. Neuroinform.* **2014**, *7*, 48. [CrossRef] [PubMed]
17. Voelker, A.; Kajić, I.; Eliasmith, C. Legendre Memory Units: Continuous-Time Representation in Recurrent Neural Networks. In Proceedings of the 33rd International Conference on Neural Information Processing Systems, Vancouver, BC, Canada, 8–14 December 2019.
18. Lin, C.-K.; Wild, A.; Chinya, G.N.; Cao, Y.; Davies, M.; Lavery, D.M.; Wang, H. Programming Spiking Neural Networks on Intel's Loihi. *Computer* **2018**, *51*, 52–61. [CrossRef]
19. Boahen, K. A neuromorph's prospectus. *Comput. Sci. Eng.* **2017**, *19*, 14–28. [CrossRef]
20. Indiveri, G.; Douglas, R. Neuromorphic Vision Sensors. *Science* **2000**, *288*, 1189–1190. [CrossRef]
21. Liu, S.-C.; Delbruck, T. Neuromorphic sensory systems. *Curr. Opin. Neurobiol.* **2010**, *20*, 288–295. [CrossRef]
22. Voelker, A.R. *A Solution to the Dynamics of the Prescribed Error Sensitivity Learning Rule*; Centre for Theoretical Neuroscience: Waterloo, ON, Canada, 2015.
23. Devices, A. LTspice Simulator. 2008. Available online: http://www.analog.com/en/design-center/design-tools-and-calculators/ltspice-simulator.html (accessed on 26 September 2020).
24. Nagel, L.; Pederson, D. *SPICE (Simulation Program with Integrated Circuit Emphasis)*; University of California: Berkeley, CA, USA, 1973.
25. Nichols, K.; Kazmierski, T.; Zwolinski, M.; Brown, A. Overview of SPICE-like circuit simulation algorithms. *IEE Proc. Circuits Devices Syst.* **1994**, *141*, 242–250. [CrossRef]
26. Mittal, S. A Survey of Architectural Techniques for Managing Process Variation. *ACM Comput. Surv.* **2016**, *48*, 1–29. [CrossRef]
27. Lanfranco, A.R.; Castellanos, A.E.; Desai, J.P.; Meyers, W.C. Robotic surgery: A current perspective. *Ann. Surg.* **2004**, *239*, 14. [CrossRef]
28. Nishida, S.-I.; Kawamoto, S.; Okawa, Y.; Terui, F.; Kitamura, S. Space Debris Removal System using a Small Satellite. *Acta Astronaut.* **2009**, *65*, 95–102. [CrossRef]
29. DeWolf, T.; Stewart, T.C.; Slotine, J.-J.; Eliasmith, C. A spiking neural model of adaptive arm control. *Proc. R. Soc. B Boil. Sci.* **2016**, *283*, 20162134. [CrossRef] [PubMed]
30. Eliasmith, C. *How to Build a Brain: A Neural Architecture for Biological Cognition*; Oxford University Press: Oxford, UK, 2013.
31. Applied Brain Research Inc. Available online: https://www.nengo.ai/nengo-loihi/v0.9.0/examples/learn_communication_channel.html (accessed on 26 March 2021).
32. Fischl, K.D.; Andreou, A.G.; Stewart, T.C.; Fair, K. Implementation of the Neural Engineering Framework on the TrueNorth Neurosynaptic System. In Proceedings of the 2018 IEEE Biomedical Circuits and Systems Conference (BioCAS), Cleveland, OH, USA, 17–19 October 2018.
33. Mundy, A.; Knight, J.; Stewart, T.C.; Furber, S. An efficient SpiNNaker implementation of the Neural Engineering Framework. In Proceedings of the 2015 International Joint Conference on Neural Networks (IJCNN), Killarney, Ireland, 12–16 July 2015.