The Artificial Neuron: A Comprehensive Synthesis of Computational, Analog, and Physical Architectures

OCTA

1 Introduction: Analysis by Synthesis

The neuron is the fundamental atomic unit of cognition. In the biological brain, it is a wetware device, a microscopic cell operating on principles of ion diffusion, membrane capacitance, and neurotransmitter release and uptake. However, the *function* of the neuron—to integrate information, make a decision based on a threshold, and transmit that decision—is *substrate-independent*.

This principle of substrate independence allows engineers and scientists to "build a neuron" using materials as diverse as Python code, silicon transistors, discrete timing chips, and even fiber optic cables.

The project of building a neuron is not merely an exercise in imitation; it is an investigation through construction, often termed *analysis by synthesis*. By attempting to recreate the neuron in non-biological media, we are forced to strip away the biological complexity and identify the essential mathematical and physical principles that enable intelligence. Whether through the high-level logic of a matrix multiplication in NumPy or the low-level electron flow in a CMOS transistor, each implementation reveals a different facet of what it means to compute.

This report provides a technical manual and theoretical analysis of constructing artificial neurons across three distinct domains:

- **The Algorithmic Domain:** Implementing the perceptron and multi-layer perceptron using Python and linear algebra.
- **The Analog Electronic Domain:** Constructing spiking neurons using discrete components like the 555 timer, transistors, and Schmitt triggers.
- **The Physical Modeling Domain:** Creating macroscopic representations using fiber optics and 3D modeling materials for educational and visual analysis.

2 The Algorithmic Neuron: Mathematical Foundations and Software Implementation

In the computational domain, the neuron is an abstraction. It is stripped of its mass and energy, reduced to a set of equations that govern the transformation of data. The foundational model for this abstraction is the perceptron, developed by Frank Rosenblatt in 1957. While simple, the perceptron establishes the architectural paradigm for the massive deep learning models used today: the weighted sum followed by a non-linear activation.

2.1 The Mathematical Architecture of the Perceptron

The perceptron is a linear binary classifier. Its function is to map an input vector $\mathbf{x} \in \mathbb{R}^n$ to a binary output class $\{0, 1\}$. This operation simulates the dendritic integration and axonal firing of a biological neuron.

2.1.1 The Weighted Sum (Integration)

The dendrites of a biological neuron receive signals from presynaptic neurons. These connections vary in strength, a property known as the synaptic weight. In the perceptron, this is modeled as a dot product between the input vector $\mathbf{x}$ and a weight vector $\mathbf{w}$. Additionally, a neuron has an intrinsic excitability, modeled as a bias term b.

The mathematical formulation for the pre-activation value z is:

$$z = \mathbf{w}^\top \mathbf{x} + b = \sum_{i=1}^{n} w_i x_i + b, \tag{1}$$

where:

- $x_1, x_2, \ldots, x_n$ are the input features (e.g., pixel intensities, sensor readings),
- $w_1, w_2, \ldots, w_n$ are the learnable weights associated with each input,
- b is the bias, effectively a threshold that shifts the activation function.

This linear equation describes a hyperplane in the n-dimensional input space. The perceptron's job is to determine which side of this hyperplane a given input vector falls on.

2.1.2 The Activation Function (Firing)

Once the inputs are integrated into z, the neuron must decide whether to "fire." This is the role of the activation function $f(z)$. In the classic perceptron, this is a Heaviside step function:

$$f(z) = \begin{cases} 1, & z \geq 0, \\ 0, & z < 0. \end{cases} \tag{2}$$

If the weighted sum plus the bias is positive, the neuron outputs a 1 (fires); otherwise, it outputs 0. This binary behavior mimics the all-or-nothing nature of the biological action potential.

2.2 Implementation in Python with NumPy

To build this mathematically, we utilize Python and the NumPy library. NumPy is essential because it allows for *vectorization*—performing operations on entire arrays of numbers simultaneously, rather than looping through them individually. This mirrors the parallel nature of neural processing.

2.2.1 The Perceptron Class Structure

A robust implementation encapsulates the neuron's state (weights and bias) and its behavior (learning and prediction) within a class.

Listing 1: Minimal perceptron implementation in Python/NumPy.

```python
import numpy as np

class Perceptron:
   def __init__(self, learning_rate=0.01, epochs=1000):
      self.learning_rate = learning_rate
      self.epochs = epochs
      self.weights = None
      self.bias = None

   def fit(self, X, y):
      n_samples, n_features = X.shape
      # Weight initialization
      self.weights = np.zeros(n_features)
      self.bias = 0.0

      # Training loop
      for _ in range(self.epochs):
         for idx, x_i in enumerate(X):
            # Forward pass
            linear_output = np.dot(x_i, self.weights) + self.bias
            y_predicted = self._step_function(linear_output)

            # Perceptron learning rule
            update = self.learning_rate * (y[idx] - y_predicted)
            self.weights += update * x_i
            self.bias += update

   def predict(self, X):
      linear_output = np.dot(X, self.weights) + self.bias
      return self._step_function(linear_output)

   def _step_function(self, x):
      return np.where(x >= 0, 1, 0)
```

Analysis of the code.

- **Weight initialization.** The weights are initialized to zeros with `np.zeros`. In more complex networks, this is insufficient (leading to symmetry problems), but for a single perceptron, zero or random initialization works.
- **Forward pass.** The line `np.dot(x_i, self.weights) + self.bias` executes the core integration function $z = \mathbf{w}^\top \mathbf{x} + b$.
- **Learning rule.** The perceptron learning rule is:

$$\Delta w_i = \eta \left(y_{\text{true}} - y_{\text{pred}} \right) x_i, \tag{3}$$

$$\Delta b = \eta \left(y_{\text{true}} - y_{\text{pred}} \right), \tag{4}$$

  where η is the learning rate.

  If the prediction is correct ($y_{\text{true}} = y_{\text{pred}}$), the term $(y_{\text{true}} - y_{\text{pred}})$ is zero, and no update occurs. If the neuron fires when it should not ($0 - 1 = -1$), the weights are decreased. If it fails to fire when it should ($1 - 0 = 1$), the weights are increased.

2.3 Limitations and the Necessity of Nonlinearity

The single perceptron has a fatal flaw: it can only solve linearly separable problems. This was famously highlighted by Minsky and Papert in 1969 regarding the XOR (exclusive OR) problem.

The XOR problem. Consider:

$$(0,0) \mapsto 0,$$
$$(0,1) \mapsto 1,$$
$$(1,0) \mapsto 1,$$
$$(1,1) \mapsto 0.$$

If you plot these points on a 2D graph, it is impossible to draw a single straight line that separates the 0s from the 1s. To solve this, we must build a multi-layer perceptron (MLP). An MLP introduces *hidden layers*—neurons that process intermediate representations of the data.

However, stacking *linear* perceptrons is futile. The composition of linear functions is still a linear function. To introduce the capability to model complex curves and decision boundaries, we must use *non-linear* activation functions.

2.4 Modern Activation Functions

While the step function is historically significant, modern "software neurons" use differentiable functions to allow for training via backpropagation (calculating the gradient of the error and propagating it backward).

Activation	Equation	NumPy Implementation	Characteristics
Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$	`1 / (1 + np.exp(-x))`	Range $(0,1)$. Smooth, differentiable; used in probability outputs; suffers from vanishing gradients.
Tanh	$\tanh(x)$	`np.tanh(x)`	Range $(-1,1)$. Zero-centered, often better than sigmoid for hidden layers; still can saturate.
ReLU	$f(x) = \max(0,x)$	`np.maximum(0, x)`	Range $[0,\infty)$. Sparse activations, simple gradient for $x > 0$; enables deep networks; sometimes yields "dead ReLUs".
Step	$f(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$	`np.where(x >= 0, 1, 0)`	Non-differentiable at 0; used in basic perceptrons, not in gradient-based deep learning.

Table 1: Common activation functions for artificial neurons.

The shift to ReLU. Sigmoid functions saturate: at very high or low input values, the curve becomes flat, and the gradient (derivative) approaches zero. Since learning relies on gradients, a saturated neuron stops learning. ReLU is linear for all positive values, ensuring a constant gradient of 1, enabling faster and deeper training.

2.5 Building a Dense Neural Network

To move beyond the single neuron, we implement a *dense* (fully connected) layer. In NumPy, this involves upgrading the weight vector to a weight matrix W:

- Input X: shape $(\text{batch\_size}, \text{input\_features})$,
- Weights W: shape $(\text{input\_features}, \text{output\_neurons})$,
- Bias b: shape $(1, \text{output\_neurons})$,

with operation:

$$Z = XW + b. \tag{5}$$

This matrix multiplication effectively runs the perceptron equation for every neuron in the layer simultaneously against every sample in the batch. This is the structural foundation of libraries like TensorFlow and PyTorch.

3 The Analog Electronic Neuron: Physics in Silicon

Software neurons abstract away time. A Python function executes essentially instantaneously at the conceptual level. However, biological neurons are dynamical systems; they integrate signals over time, leak charge, and have refractory periods. To build a neuron that respects the physics of time, we turn to analog electronics.

3.1 The Leaky Integrate-and-Fire (LIF) Model

The most common analog model is the leaky integrate-and-fire (LIF) neuron. It models the cell membrane as a capacitor (storing charge) and the ion channels as a resistor (leaking charge).

A standard form of the governing differential equation is:

$$\tau_m \frac{dV_m(t)}{dt} = -\big(V_m(t) - V_{\text{rest}}\big) + R_m I(t), \tag{6}$$

where:

- $V_m(t)$ is the membrane potential,
- $\tau_m = R_m C_m$ is the membrane time constant (resistance $\times$ capacitance),
- $I(t)$ is the synaptic input current,
- V_{rest} is the resting potential.

In hardware, we build this literally: a capacitor integrates the input current, a resistor provides the leakage path to ground, and a thresholding circuit (comparator) generates the spike.

3.2 The 555 Timer Spiking Neuron

For the hobbyist or engineer prototyping macroscopic neurons, the NE555 timer IC is an ideal building block. Its internal architecture naturally mimics the soma of a neuron.

3.2.1 Internal Architecture of the 555

The 555 timer contains:

- **Voltage divider:** Three 5 kΩ resistors split the supply voltage V_{CC} into reference levels at $\frac{1}{3}V_{CC}$ and $\frac{2}{3}V_{CC}$.
- **Comparators:**
  - Threshold comparator: checks if voltage $> \frac{2}{3}V_{CC}$.
  - Trigger comparator: checks if voltage $< \frac{1}{3}V_{CC}$.
- **Flip-flop:** Stores the state (charging vs. discharging).
- **Discharge transistor:** A switch that connects Pin 7 to ground.

3.2.2 "Kepler's Neuron" Circuit Topology

A popular configuration for a spiking neuron using the 555 is sometimes called "Kepler's neuron." It operates in a modified astable mode.

Circuit construction.

- **Membrane (Pins 2 and 6):** Pins 2 (Trigger) and 6 (Threshold) are tied together and connected to a capacitor C_{mem} to ground. This capacitor represents the neuron's membrane capacitance.
- **Synaptic input:** Input signals are fed into the capacitor through resistors R_{syn}.
- **Integration phase:** When an input voltage is applied, current flows through R_{syn}, charging C_{mem}. The voltage across the capacitor rises, mimicking EPSP summation.
- **Threshold and firing:** When the capacitor voltage reaches $\frac{2}{3}V_{CC}$, the threshold comparator trips. The internal flip-flop sets, driving the output (Pin 3) low and turning on the discharge transistor at Pin 7.
- **Refractory period:** Pin 7 is connected to the capacitor via a resistor R_{dis}. When the transistor turns on, it rapidly drains the capacitor. When the voltage drops to $\frac{1}{3}V_{CC}$, the trigger comparator trips, turning off the discharge transistor and resetting the cycle.

This creates a voltage-controlled oscillator (VCO). The higher the input voltage (stimulus), the faster the capacitor charges to the threshold, and the higher the spiking frequency. This mimics the *rate coding* used by biological neurons.

3.3 Discrete Transistor Implementations

While the 555 is convenient, it consumes milliwatts of power and is physically large. To build dense neuromorphic systems, we use discrete transistors, often operating in the subthreshold region to mimic the energy efficiency of biology.

3.3.1 The Axon-Hillock Circuit

A foundational circuit in neuromorphic engineering is the axon-hillock circuit. It uses an amplifier with positive feedback to generate a sharp spike:

- **Integration:** Input current integrates onto a capacitor C_{mem}.
- **Amplification:** The voltage on C_{mem} drives the gate of a transistor or an inverter.
- **Positive feedback:** As the output begins to switch, a capacitive divider feeds some of the output voltage back to the input, causing a rapid acceleration of the voltage rise—a runaway process that creates the steep leading edge of the spike.
- **Reset:** Once the spike occurs, a reset transistor is triggered to short C_{mem} to ground, analogous to the opening of potassium channels that repolarize the cell.

3.3.2 The Thyristor (p-n-p-n) Neuron

The p-n-p-n diode (thyristor) can act as a single-device neuron.

- **Mechanism:** A thyristor blocks current until a voltage threshold (breakdown voltage) is reached. Once reached, it "latches up" and becomes highly conductive until the current drops below a holding current.
- **Operation:** A capacitor is connected in parallel with the thyristor. Input current charges the capacitor. When $V_{\text{cap}} > V_{\text{breakdown}}$, the thyristor fires, dumping the charge (spike) and resetting the capacitor.
- **Efficiency:** Because this relies on intrinsic material properties rather than complex circuitry, these neurons can operate with energy consumption on the order of femtojoules per spike, approaching biological efficiency.

3.3.3 Floating-Gate Transistors and Plasticity

A true neuron must learn. In analog hardware, this requires changing the "weight" of the connection. Floating-gate transistors (used in flash memory) enable this.

- **Principle:** A standard MOSFET has a "floating" gate insulated by oxide. Charge can be tunneled onto this gate, permanently shifting the threshold voltage of the transistor.
- **Application:** By controlling the charge on the floating gate, the transistor can be made to pass more or less current for the same input voltage. This effectively sets a synaptic weight that is non-volatile and analog, allowing for hardware implementation of learning rules such as spike-timing-dependent plasticity (STDP).

4 BEAM Robotics: The Reflexive Neuron and Nv Networks

Bridging the gap between the rigid logic of software and the component-heavy design of standard electronics is the field of BEAM robotics (Biology, Electronics, Aesthetics, Mechanics). Pioneered by Mark Tilden, BEAM focuses on "nervous networks" constructed from simple logic inverters used in analog modes. The fundamental unit here is the Nv neuron.

4.1 The 74HC14 Schmitt Trigger

The core component of BEAM neurons is the 74HC14 hex inverting Schmitt trigger IC.

- **Inverter:** Logic 1 in $\rightarrow$ Logic 0 out.
- **Schmitt trigger:** The input must rise above a high threshold V_{T+} to switch the output low, and fall below a low threshold V_{T-} to switch the output high. This *hysteresis* prevents the neuron from oscillating rapidly due to noise when the input is near the threshold and yields a clean firing event.

4.2 The Nv Neuron (Nervous Neuron)

The Nv neuron is a differentiator. It responds to a change in input (an edge) by generating a pulse of a fixed duration.

Schematic construction.

- Input signal enters through a capacitor (typically $0.1\,\mu$F to $1.0\,\mu$F).
- A resistor ($1\,\text{M}\Omega$ to $10\,\text{M}\Omega$) connects the inverter input to ground (grounded Nv) or V_{CC}.
- The capacitor–resistor junction connects to the input of one 74HC14 gate.

Operational logic.

- **Rest state:** The resistor pulls the inverter input to ground (logic 0). The inverter output is high (logic 1).
- **Stimulus:** A rising edge at the input is passed through the capacitor; the inverter input briefly spikes high.
- **Reaction:** The inverter output snaps low (active state).
- **Timing (τ):** The capacitor cannot pass DC current. The resistor slowly discharges the elevated input voltage back to ground. The time it takes to decay below the Schmitt trigger threshold V_{T-} is determined by $\tau = RC$.
- **Recovery:** Once the input drops below V_{T-}, the output snaps back high.

This creates a neuron that outputs a low pulse of specific duration in response to a stimulus. It represents a reflexive action.

4.3 The Nu Neuron (Neural Neuron)

The Nu neuron is the topological inverse of the Nv. It is an integrator.

Schematic construction.

- Input signal enters through a resistor.
- A capacitor connects the inverter input to ground.
- The 74HC14 gate acts as the non-linear threshold element.

Operational logic.

- **Delay:** When the input goes high, it must charge the capacitor through the resistor. The inverter input voltage rises slowly.
- **Threshold:** After a delay τ, the voltage crosses V_{T+}, switching the output low.
- **Function:** The Nu neuron acts as a delay line or low-pass filter. It ignores short spikes and only responds to sustained stimuli, mimicking temporal summation.

4.4 The Bicore: A Minimal Central Pattern Generator (CPG)

The most famous application of BEAM neurons is the bicore, a circuit that mimics the central pattern generators (CPGs) found in biological spinal cords (used for walking, swimming, heartbeat).

Construction. Two Nv neurons are connected in a loop:

- Output of neuron 1 $\rightarrow$ input capacitor of neuron 2.
- Output of neuron 2 $\rightarrow$ input capacitor of neuron 1.

Dynamics.

- **Grounded bicore:** Both bias resistors go to ground. The circuit is generally stable or monostable unless perturbed. Used for tactile reflexes.
- **Suspended bicore:** Bias resistors are connected to the inputs of the opposing neuron or other active signal lines, creating an unstable equilibrium and leading to continuous oscillation.
- **Motor drive:** A DC motor can be connected between the outputs of the two neurons (often via an H-bridge driver such as a 74HC240). One neuron high and the other low yields motion in one direction; the roles reversed yield motion in the opposite direction.

As the motor spins, it generates back-EMF, which feeds back into the neural circuit, altering the charging time of the capacitors. If the robot's leg hits an obstacle, the motor slows, the back-EMF changes, and the bicore automatically adjusts its timing. This creates a robot with "physical intelligence" that adapts to terrain without any software code.

5 The Physical and Educational Neuron

For educational purposes, the invisibility of electrons and code can be a barrier. Physical models are required to visualize the morphology (structure) and transmission (function) of the neuron.

5.1 3D Anatomical Modeling

Constructing a static model emphasizes the biological architecture that dictates function.

Materials and assembly.

- **Soma (cell body):** Clay, Styrofoam, or a 3D-printed sphere representing the metabolic center.
- **Dendrites:** Pipe cleaners or wire branching from the soma, indicating the receptive field.
- **Axon:** A long tube or wire extending from the soma.
- **Myelin sheath:** Beads, pasta, or clay segments wrapped around the axon, visually segmenting the transmission line and illustrating saltatory conduction.
- **Synaptic terminals:** Small bulbs at the end of the axon, representing presynaptic boutons.

5.2 Fiber Optic Functional Models

To visualize the action potential, light is a powerful proxy for electricity. This approach parallels optogenetics, a technique in neuroscience where light is used to control genetically modified neurons.

Construction of a light-based neuron.

- **Axon (fiber optics):** Side-glow fiber optic cabling for the axon. Unlike end-glow fiber used for data, side-glow emits light along its length, creating a visible "pulse" traveling down the "nerve."
- **Impulse (LEDs):** High-intensity RGB LEDs coupled to the ends of the fibers.
- **Control logic (Arduino/Raspberry Pi):** A microcontroller is programmed to mimic integrate-and-fire dynamics. Sensors (simulating dendrites) detect sound or touch; the controller integrates these and, upon threshold, triggers a PWM sequence on the LEDs, sending a pulse of light along the fiber optic axon.
- **Network visualization:** By connecting the end of one fiber-optic axon to the sensor of another node, a physical neural network can be built on a wall or 3D frame, allowing observers to see "waves" of activation propagate.

6 Synthesis and Conclusion

The construction of a neuron is a multifaceted engineering challenge that reveals the deep unity between information theory, physics, and biology.

6.1 Comparison of Substrates

6.2 The Tyranny of Numbers

We can build a single functional neuron in analog hardware that outperforms a software simulation in speed and energy efficiency. However, scaling becomes difficult: the *tyranny of numbers*. Connecting 10^{10} discrete transistors by hand is impossible. Connecting 10^{10} software neurons is computationally expensive but architecturally trivial; we change an array shape rather than a wiring harness.

Feature	Software	Analog	BEAM	Physical
Primary variable	Float value	Voltage / charge	Voltage edge	Light intensity
Time domain	Discrete steps / epochs	Continuous real time	Continuous, event-driven	Visualized timing
Connectivity	Arbitrary matrices	Limited by routing	Local neighbor-to-neighbor	Point-to-point links
Learning	Gradient descent, backprop	Floating-gate / plasticity	Back-EMF adaptation	Scripted / manual
Key advantage	Scalability, precision	Energy efficiency, speed	Reflexive robustness	Intuitive visualization

Table 2: Comparison of neuron substrates across domains.

6.3 Future Directions: Neuromorphic Convergence

The future of "building a neuron" lies in the convergence of these fields. Neuromorphic engineering chips such as Intel's Loihi and IBM's TrueNorth use the transistor physics of the analog domain but simulate the connectivity using digital routing (e.g., address-event representation). This effectively combines the physics of silicon with the scalability of code.

Whether one builds a neuron to classify data, to drive a robot, or to teach a student, the core principle remains: a neuron is a device that creates order from chaos, taking a noisy world of inputs and reducing it to a single decisive output.

7 Technical Addendum: Component Values and Tables

7.1 555 Timer "Kepler Neuron" Component Values

For builders wishing to create a visible spiking neuron (approximately 1–10 Hz):

Component	Value	Role
R1 (Input)	$10\,\mathrm{k\Omega}$ – $100\,\mathrm{k\Omega}$	Simulates synaptic weight (lower R = higher frequency).
R2 (Discharge)	$1\,\mathrm{k\Omega}$	Controls pulse width (refractory period).
C1 (Membrane)	$10\,\mu\mathrm{F}$	Membrane capacitance (larger C = slower firing).
Supply	5–9 V	Power source.
Output load	LED + $330\,\Omega$	Visual indicator of spikes.

Table 3: Example component values for a 555-based "Kepler neuron."

7.2 BEAM Nv Neuron Logic Table

Input State	Inverter Input Voltage	Inverter Output	State Description
Steady low	$\approx 0\,\mathrm{V}$ (via R to GND)	High	Rest state.
Rising edge	Rises toward V_{CC} (via C)	Low	Firing (process start).
Steady high	Decays to $0\,\mathrm{V}$ (via R)	High (after delay τ)	Recovery (process end).
Falling edge	Brief negative transient (clamped)	High	Ignored (clamped by protection).

Table 4: Qualitative state table for a BEAM Nv neuron.