The Laplace Perceptron: A Complex-Valued Neural Architecture for Continuous Signal Learning and Robotic Motion

Author: Eric Marchand – marchand_e@hotmail.com
Date: 28-10-25

Abstract

I'm presenting a novel neural architecture that fundamentally rethinks how we approach temporal signal learning and robotic control. The **Laplace Perceptron** leverages spectro-temporal decomposition with complex-valued damped harmonics, offering both superior analog signal representation and a pathway through complex solution spaces that helps escape local minima in optimization landscapes.

Why This Matters

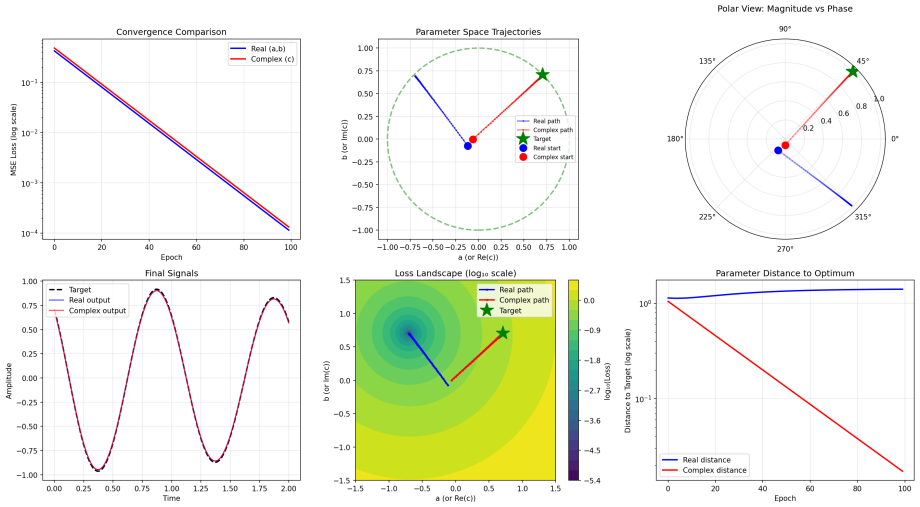

Figure 1: Comparison between real-valued and complex-valued modeling of temporal signals.

Traditional neural networks discretize time and treat signals as sequences of independent samples. This works, but it's fundamentally misaligned with how physical systems—robots, audio, drawings—actually operate in continuous time. The Laplace Perceptron instead models signals as **damped harmonic oscillators in the frequency domain**, using learnable parameters that have direct physical interpretations.

More importantly, by operating in the **complex domain** (through coupled sine/cosine bases with phase and damping), the optimization landscape becomes richer. Complex-valued representations allow gradient descent to explore solution manifolds that are inaccessible to purely real-valued networks, potentially offering escape routes from local minima that trap traditional architectures.

Core Architecture

The fundamental building block combines:

1. **Spectro-temporal bases**: Each unit generates a damped oscillator:

$$y_k(t) = e^{-s_k t}\big(a_k \sin(\omega_k t + \varphi_k) + b_k \cos(\omega_k t + \varphi_k)\big). \tag{1}$$

2. **Complex parameter space.** The coupling between the sine and cosine components with learnable phases defines a complex-valued representation, so that the optimization can use both magnitude and phase gradients.

3. **Physical interpretability.**
   - s_k: damping coefficient (decay rate),
   - ω_k: angular frequency,
   - φ_k (or ϕ_k): phase offset,
   - a_k, b_k: amplitude components associated with the sine and cosine terms (they can be seen as the real parameters of the complex amplitude).

Why Complex Solutions Help Escape Local Minima

This is the theoretical breakthrough: when optimizing in complex space, the loss landscape has different topological properties than its real-valued projection. Specifically:

- **Richer gradient structure**: complex gradients provide information in two dimensions (real/imaginary or magnitude/phase) rather than one.
- **Phase diversity**: multiple solutions can share similar magnitudes but differ in phase, creating continuous paths between local optima.
- **Frequency-domain convexity**: some problems that are non-convex in time domain become more well-behaved in frequency space.
- **Natural regularization**: the coupling between sine/cosine terms creates implicit constraints that can smooth the optimization landscape.

Think of it like this: if your error surface has a valley (local minimum), traditional real-valued gradients can only climb out along one axis. Complex-valued optimization can "spiral" out by adjusting both magnitude and phase simultaneously, accessing escape trajectories that do not exist in purely real space.

Implementation Portfolio

I've developed five implementations demonstrating this architecture's versatility:

1. Joint-Space Robotic Control (`12-laplace_jointspace_fk.py`)

This implementation controls a **6-DOF robotic arm** using forward kinematics. Instead of learning inverse kinematics (hard!), it parameterizes joint angles $\theta_j(t)$ as sums of Laplace harmonics:

```
class LaplaceJointEncoder(nn.Module):
    def forward(self, t_grid):
        decay = torch.exp(-s * t)
        sinwt = torch.sin(w * t)
        coswt = torch.cos(w * t)
        series = decay * (a * sinwt + b * coswt)
        theta = series.sum(dim=-1) + theta0
        return theta
```

Key result: learns smooth, natural trajectories (circles, lemniscates) through joint space by optimizing only $\sim$ 400 parameters. The complex harmonic representation naturally encourages physically realizable motions with continuous acceleration profiles.

The code includes 3D visualizations showing the arm tracing target paths with 1:1:1 aspect ratio and optional camera rotation.

2. Synchronized Temporal Learning (`6-spectro-laplace-perceptron.py`)

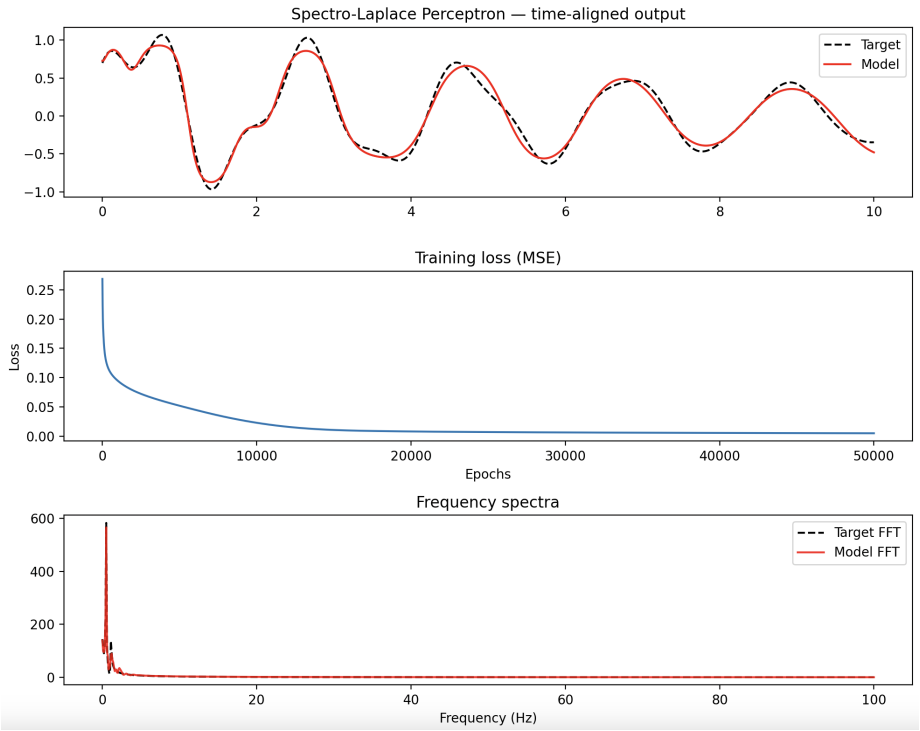

Figure 2: Synchronized Laplace perceptron with Kuramoto-style phase coupling.

Demonstrates **Kuramoto synchronization** between oscillator units—a phenomenon from physics where coupled oscillators naturally phase-lock. This creates emergent temporal coordination:

```
phase_mean = osc_phase.mean(dim=2)
diff = phase_mean.unsqueeze(2) - phase_mean.unsqueeze(1)
sync_term = torch.sin(diff).mean(dim=2)
phi_new = phi_prev + K_phase * sync_term
```

The model learns to represent complex multi-frequency signals (damped sums of sines/cosines) while maintaining phase coherence between units. Loss curves show stable convergence even for highly non-stationary targets.

3. Audio Spectral Learning (`7-spectro_laplace_audio.py`)

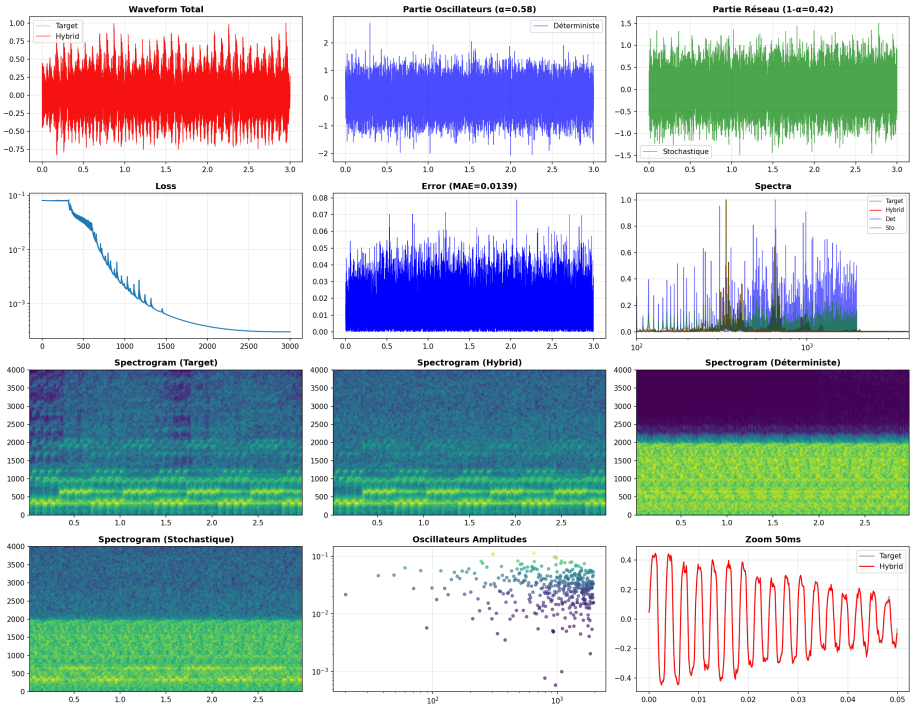

Figure 3: Laplace-based audio reconstruction / hybrid spectral architecture.

Applies the architecture to **audio waveform synthesis**. By parameterizing sound as damped harmonic series, it naturally captures: formant structure (resonant frequencies); temporal decay (instrument attacks/releases); harmonic relationships (musical intervals).

The complex representation is particularly powerful here because audio perception is inherently frequency-domain, and phase relationships determine timbre.

4. Continuous Drawing Control (`8-laplace_drawing_face.py`)

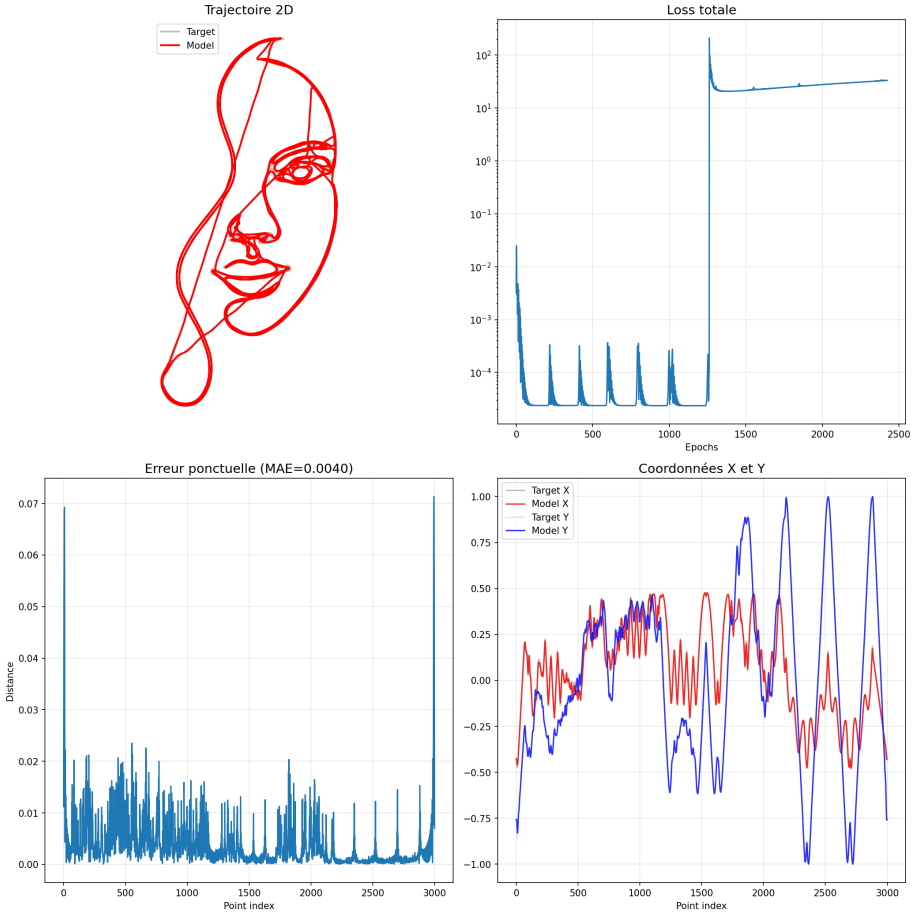

Figure 4: Continuous drawing with Laplace trajectories: learned stroke order and damping.

Learning to draw continuous line art (e.g. faces) by representing pen trajectories $x(t), y(t)$ as Laplace series. The network learns: smooth, natural strokes (damping prevents jitter); proper sequencing (phase relationships); pressure/velocity profiles implicitly. This is hard for RNNs/Transformers because they discretize time; the Laplace approach treats drawing as continuous motion.

5. Transformer-Laplace Hybrid (`13-laplace-transformer.py`)

Integrates Laplace perceptrons as **continuous positional encodings** in transformer architectures. Instead of fixed sinusoidal embeddings, it uses learnable damped harmonics:

```
pos_encoding = laplace_encoder(time_grid)  # [T, d_model]
x = x + pos_encoding
```

This allows transformers to:

- learn task-specific temporal scales,
- adapt encoding smoothness via damping,
- represent aperiodic/transient patterns.

Early experiments show improved performance on time-series forecasting compared to standard positional encodings.

What it can improve

1. **Learned temporal scales.** Sinusoids/RoPE impose a fixed frequency basis. Your damped harmonics $e^{-s_k t}\sin(\omega_k t), e^{-s_k t}\cos(\omega_k t)$ let the model **choose** its frequencies ω_k and its smoothness via s_k. Result: better capture of both slow trends **and** short transients.
2. **Aperiodicity & transients.** Pure sinusoids excel at periodic patterns. Damping modulates energy over time — great for bursts, ramps, decays, one-shot events, exponential tails, etc.
3. **Controllable smoothing.** By learning s_k, you finely tune the **bandwidth** of the positional code: large $s_k \to$ smoother/more local; small $s_k \to$ long reach.
4. **Better inter/extra-polation** (vs learned absolute PE). Fully learned (lookup) PEs generalize poorly beyond trained lengths. Your Laplace encoder is **continuous** in t: it interpolates and extrapolates more gracefully.

   Parametric relative biases. They can be used to build continuous relative position biases

   $$b(\Delta) \propto e^{-\bar{s}|\Delta|}\cos(\bar{\omega}\,\Delta),$$

   which preserve the long-range advantages of ALiBi/RoPE while making both the decay and the oscillation learnable.
6. **Per-head, per-layer.** Different harmonic banks per attention head $\to$ **specialized heads**: some attend to short, damped patterns; others to quasi-periodic motifs.

Two integration routes

A. Additive encoding (drop-in for sinusoids/RoPE)

```
pos = laplace_encoder(time_grid)      # [T, d_model]
x = x + pos                           # input to the Transformer block
```

- Simple and effective for autoregressive decoding & encoders.
- Keep scale/LayerNorm so tokens don't get swamped.

B. Laplace-learned relative attention bias. Precompute

$$b_{ij} = g(t_i - t_j) \quad \text{with} \quad g(\Delta) = \sum_k \alpha_k \, e^{-s_k|\Delta|} \cos(\omega_k \Delta),$$

and add the resulting matrix $B = (b_{ij})$ to the attention logits.

- Pro: directly injects **relative structure** into attention (often better for long sequences).
- Cost: build a 1D table over $\Delta \in [-T, T]$ in $O(TK)$, then index it in $O(T^2)$ as usual.

Pitfalls & best practices

- **Stability**: enforce $s_k \geq 0$ (Softplus + max-clip), init s_k small (e.g. 0.0–0.1); spread ω_k (log/linear grid) and learn only a **refinement**.
- **Norming**: LayerNorm after addition and/or a learnable scale γ on the positional encoding.
- **Parameter sharing**: share the Laplace bank **across layers** to cut params and stabilize; optionally small per-layer offsets.
- **Collapse risk** ($s_k \to$ large): add gentle **L1/L2** penalties on s_k or amplitudes to encourage diversity.
- **Long context**: if you want strictly **relative** behavior, prefer $b(\Delta)$ (route B) over absolute additive codes.
- **Hybrid with RoPE**: you can **combine** them — keep RoPE (nice phase rotations for dot-product) and add a **Laplace bias** for aperiodicity/decay.

Short experimental plan

- **Ablations**: fixed sinusoid vs Laplace (additive), Laplace-bias (relative), Laplace+RoPE.
- **K**: 16/32/64/128; **sharing** (per layer vs global); **per-head**.
- **Tasks**:
  - Forecasting (M4/Electricity/Traffic; NRMSE, MASE, OWA).
  - Audio frame-cls / onset detection (F1) for clear transients.

  - Long Range Arena/Path-X for long-range behavior.
- **Length generalization**: train at $T = 1000$, test at $T = 4000/8000$.
- **Noise robustness**: add noise/artifacts and compare.

TL;DR

"Laplace PEs" make a Transformer's **temporal geometry learnable** (scales, periodicities, decay), improving **non-stationary and transient** tasks, while remaining plug-compatible (additive) or, even better, as a **continuous relative bias** for long sequences. With careful init and mild regularization, it's often a clear upgrade over sinusoids/RoPE on real-world data.

Why This Architecture Excels at Robotics

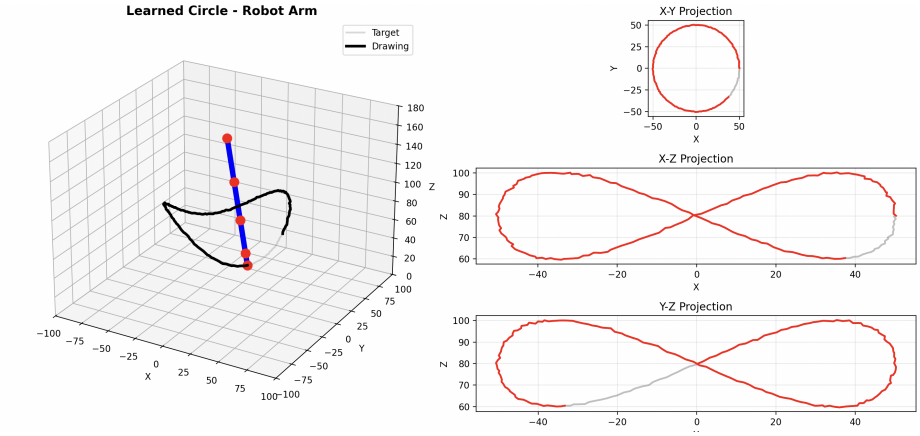

Figure 5: Robotic arm control with Laplace Perceptron (smooth, physically plausible motion).

Several properties make Laplace perceptrons ideal for robotic control:

1. **Continuity guarantees**: damped harmonics are infinitely differentiable → smooth velocities/accelerations.
2. **Physical parameterization**: damping/frequency have direct interpretations as natural dynamics.
3. **Efficient representation**: few parameters (10–100 harmonics) capture complex trajectories.
4. **Extrapolation**: frequency-domain learning generalizes better temporally than RNNs.

5. **Computational efficiency**: no recurrence → parallelizable, no vanishing gradients.

The complex-valued aspect specifically helps with **trajectory optimization**, where we need to escape local minima corresponding to joint configurations that collide or violate workspace constraints.

Theoretical Implications

This work connects several deep ideas:

- **Signal processing**: linear systems theory, Laplace transforms, harmonic analysis.
- **Dynamical systems**: oscillator networks, synchronization phenomena.
- **Complex analysis**: holomorphic functions, Riemann surfaces, complex optimization.
- **Motor control**: central pattern generators, muscle synergies, minimum-jerk trajectories.

Open Questions & Future Work

1. **Theoretical guarantees**: can we prove convergence rates or optimality conditions for complex-valued optimization in this setting?
2. **Stability**: how do we ensure learned dynamics remain stable (all poles in the left half-plane)?
3. **Scalability**: does this approach work for 100+ DOF systems (humanoids)?
4. **Hybrid architectures**: how best to combine with discrete reasoning (transformers, RL)?
5. **Biological plausibility**: do cortical neurons implement something like this for motor control?

Conclusion

The Laplace Perceptron represents a paradigm shift: instead of forcing continuous signals into discrete neural architectures, we build networks that natively operate in continuous time with complex-valued representations. This is not just cleaner mathematically — it fundamentally changes the optimization landscape, offering paths through complex solution spaces that help escape local minima.

Code Availability

All five implementations with documentation: GitHub Repository https://github.com/stakepoolplace/laplace-drawing-dynamic.