Attention-Based Dynamical Art: Emergent Geometry from Iterative Projections

Vasudev Menon
North Carolina State University
vmenon2@ncsu.edu

Abstract

We explore a minimal dynamical system inspired by the mathematical structure of self-attention in neural networks [4]. Despite its simple linear-algebraic formulation, the system produces intricate and highly structured visual patterns when its parameters are varied. Each image represents the convergence landscape of this dynamical map across a two-dimensional parameter space, revealing emergent order arising from purely algebraic interactions. Our work is inspired by recent mathematical frameworks that treat self-attention as a dynamical interacting system [1].

Mathematical Formulation

Let $d = 2$ be the dimension, $n = 20$ the number of points, and $T = 100$ the number of iterations. For each parameter pair $(a, b) \in [r_{\min}, r_{\max}]^2 = [-8, 8]^2$, define

$$W_K(a,b) = \begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix}, \qquad W_Q, W_V \in \mathbb{R}^{d\times d}$$

as fixed random matrices.

We initialize $X^{(0)}(a,b) = [x_1^{(0)}, \ldots, x_n^{(0)}]^\top \in \mathbb{R}^{n\times d}$ with random unit vectors:

$$\|x_i^{(0)}\|_2 = 1 \quad \forall i.$$

At each iteration $t = 0, 1, \ldots, T-1$, the system evolves according to an attention-like mapping:

$$\begin{aligned} Q^{(t)} &= X^{(t)} W_Q^\top, & K^{(t)}(a,b) &= X^{(t)} W_K(a,b)^\top, & V^{(t)} &= X^{(t)} W_V^\top, \\ S^{(t)} &= Q^{(t)} K^{(t)\top}, & A^{(t)} &= \mathrm{softmax}(S^{(t)}), & X^{(t+1)} &= A^{(t)} V^{(t)}, \end{aligned}$$

followed by per-row normalization:

$$x_i^{(t+1)} \leftarrow \frac{x_i^{(t+1)}}{\|x_i^{(t+1)}\|_2 + \varepsilon}, \quad \varepsilon = 10^{-8}.$$

This iterative rule defines a nonlinear dynamical system on $(\mathbb{S}^{d-1})^n$, where each row of $X^{(t)}$ lies on the unit sphere. The softmax interaction couples all points through pairwise similarities determined by W_Q and $W_K(a,b)$.

Convergence Landscape

After T iterations, we measure convergence via

$$\Delta(a,b) = \max_i \left| x_{i,1}^{(T)} - x_{1,1}^{(T)} \right|,$$

and assign a binary label

$$L(a,b) = \begin{cases} 1, & \text{if } \Delta(a,b) < 10^{-5}, \\ 0, & \text{otherwise.} \end{cases}$$

Plotting $L(a,b)$ over the (a,b)-grid produces a monochrome image, where white regions correspond to parameter values leading to synchronized convergence and black regions indicate divergence or oscillatory behavior. The resulting texture visualizes the boundary between regions of stability and chaos in the parameter space of W_K.

Methods

All experiments were implemented in `PyTorch` [3] using vectorized tensor operations on a NVIDIA Tesla T4 GPU on the Google Colab platform, enabling generation of 1500×1500-resolution images in under one minute. Each image corresponds to a single random seed, which uniquely determines the random initialization of the query and value matrices W_Q and W_V, as well as the initial point configuration $X^{(0)}$. The full source code is provided in Appendix A.

Parameter Grid Sampling and Visualization

We discretize the (a,b) parameter space uniformly over $[-8,8]^2$ with 1500×1500 samples. For each grid point (a,b), the diagonal key matrix

$$W_K(a,b) = \begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix}$$

is applied to the same initialized configuration $X^{(0)}$. All grid evaluations are batched and computed in parallel. The resulting binary convergence map $L(a,b)$ is rendered using `matplotlib` [2] with bicubic interpolation and no axes, highlighting the emergent geometric structures of the dynamical system. Pixels corresponding to converged states, $L(a,b) = 1$, are displayed in white, while non-convergent regions appear in black.

Selected Artworks (Gallery)

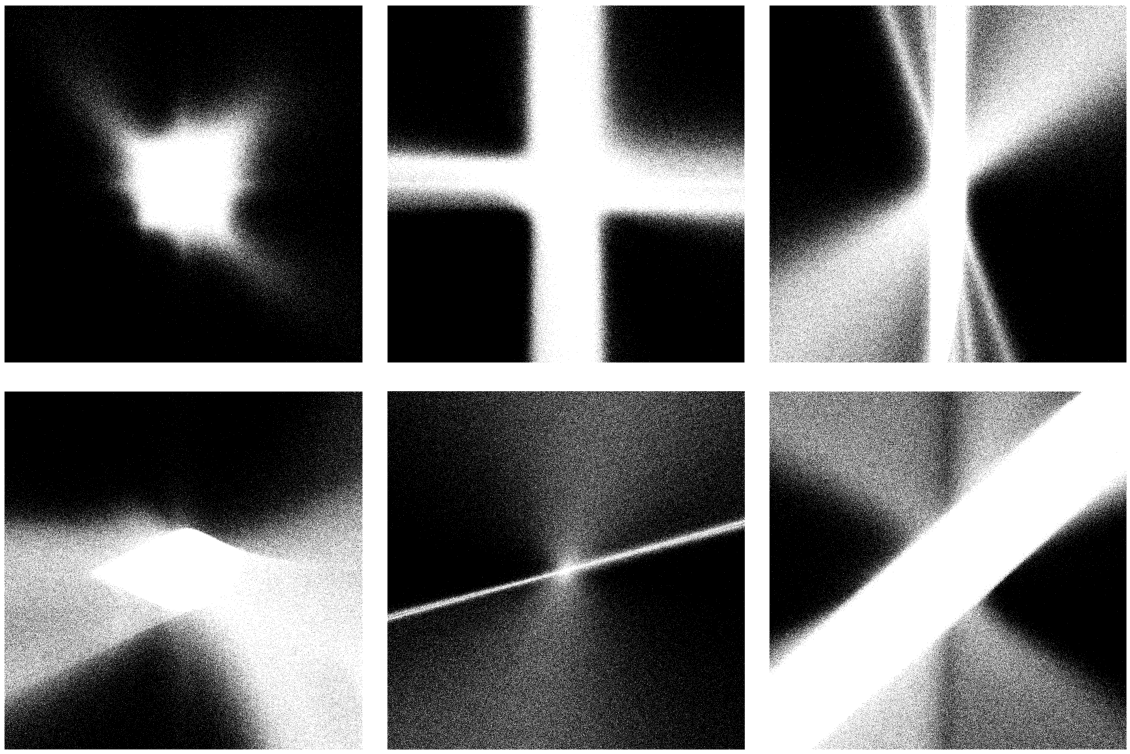

Figure 1: Gallery of convergence landscapes. Top left is generated with seed 1 and $W_K = W_Q = I$. The rest are generated with random seeds (from left to right, top to bottom): 1679700, 5227590, 6306593, 6549664, and 8012376. The corresponding weight matrices for each seed are provided in Appendix B.

References

[1] B. Geshkovski, C. Letrouit, Y. Polyanskiy, and P. Rigollet. A mathematical perspective on transformers, 2025.

[2] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[3] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2023.

A Code Listing

A minimal reference implementation is provided below for reproducibility.

```
import torch
import numpy as np
import matplotlib.pyplot as plt

seed = 1
torch.manual_seed(seed)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Running on:", device)

num_samples = 1500 # resolution
num_points = 20
num_iterations = 100
range_min, range_max = -8, 8
d = 2

WQ = torch.randn(d, d, device=device)
WV = torch.randn(d, d, device=device)

a_vals = np.linspace(range_min, range_max, num_samples)
b_vals = np.linspace(range_min, range_max, num_samples)
A, B = np.meshgrid(a_vals, b_vals)

ab_grid = torch.tensor(np.stack([A.ravel(), B.ravel()], axis=-1),
                       device=device, dtype=torch.float32)
num_grid = ab_grid.shape[0]

X_grid = torch.randn(num_grid, num_points, d, device=device)
X_grid /= X_grid.norm(dim=-1, keepdim=True)

WK_grid = torch.zeros(num_grid, d, d, device=device)
WK_grid[:, 0, 0] = ab_grid[:, 0]
WK_grid[:, 1, 1] = ab_grid[:, 1]

for _ in range(num_iterations):
    Q = X_grid @ WQ.T
    K = X_grid @ WK_grid.transpose(1, 2)
    V = X_grid @ WV.T
    scores = torch.matmul(Q, K.transpose(1, 2))
    attn = torch.softmax(scores, dim=-1)
    X_grid = torch.matmul(attn, V)
    X_grid = X_grid / (X_grid.norm(dim=-1, keepdim=True) + 1e-8)

# Check convergence
diff_grid = (X_grid[..., 0] - X_grid[:, 0, 0:1]).abs().max(dim=1).values
labels_grid = (diff_grid < 1e-5).int().cpu().numpy().reshape(num_samples, num_samples)

extent = [range_min, range_max, range_min, range_max]
```

```
plt.figure(figsize=(5,5))
plt.imshow(labels_grid.T, origin='lower', extent=extent,
        cmap='gray', interpolation='bicubic', alpha=1.0)
plt.axis('off')  # Remove axes
plt.tight_layout()
plt.show()
```

B Seeds and Matrix Parameters

For reproducibility, this appendix lists the exact random seeds and corresponding weight matrices used to generate the convergence landscapes in Figure 1.

Seed 1679700

$$W_Q = \begin{bmatrix} 1.1646 & -0.0278 \\ -0.8054 & -0.9405 \end{bmatrix}, \quad W_V = \begin{bmatrix} -1.9675 & -2.0948 \\ 0.3671 & 1.4580 \end{bmatrix}.$$

Seed 5227590

$$W_Q = \begin{bmatrix} -0.4588 & 0.7752 \\ 0.4600 & -0.6778 \end{bmatrix}, \quad W_V = \begin{bmatrix} -0.3571 & 0.0005 \\ 0.9241 & 0.7379 \end{bmatrix}.$$

Seed 6306593

$$W_Q = \begin{bmatrix} -0.4323 & -0.6186 \\ -1.1823 & -0.0825 \end{bmatrix}, \quad W_V = \begin{bmatrix} 1.7736 & -0.0223 \\ -0.1021 & 1.0466 \end{bmatrix}.$$

Seed 6549664

$$W_Q = \begin{bmatrix} -1.8000 & -2.5357 \\ 0.4014 & 0.4851 \end{bmatrix}, \quad W_V = \begin{bmatrix} 0.0223 & 1.9094 \\ 0.1081 & 2.5201 \end{bmatrix}.$$

Seed 8012376

$$W_Q = \begin{bmatrix} 0.4720 & 1.2490 \\ -0.8313 & -0.0374 \end{bmatrix}, \quad W_V = \begin{bmatrix} 0.0728 & 0.5135 \\ -0.6503 & -1.0870 \end{bmatrix}.$$