Trajectory-Based Inverse Dynamics for Neural Network Training

Early Technical Report / Preprint, 2025

Alexey A. Nekludoff

AstraVerge Institute

Principal Researcher

Email: an@astraverge.org

ORCID: 0009-0002-7724-5762

December 5, 2025

Abstract

This report develops a trajectory-based formulation of neural network training in which learning is expressed as an inverse dynamics problem. A feedforward network is treated as a discrete dynamical system whose forward pass generates a trajectory of internal states. Instead of minimising a loss function, training is posed as determining the transition operators that produce a trajectory terminating at a desired output.

We derive the inverse-dynamics equations for general architectures, show how they connect to discrete variational principles and Euler–Lagrange conditions, and analyse structural properties such as determinism, non-uniqueness, and existence of solutions. For linear systems, the inverse problem admits closed-form solutions; for nonlinear systems, the trajectory formulation provides a basis for alternative training paradigms that operate directly on internal computation.

The results establish a mathematical foundation for training methods that target internal dynamics rather than error surfaces, opening paths toward new approaches to learning, controllability, and model correction.

Contents

1 Introduction

Recent progress in deep learning has produced highly capable models whose internal computation remains largely opaque. A standard neural network exposes only its input–output behaviour, while the intermediate activations that determine its decisions evolve as hidden internal states. This opacity limits our ability to understand failures, correct model behaviour, or incorporate domain knowledge into the computational process.

A growing body of work views neural networks not as static functions but as *discrete dynamical systems* whose layers perform successive state transitions. Under this perspective, the forward pass generates a *trajectory of internal activations*, and a model's output is simply the terminal state of this trajectory. However, despite the naturalness of this view, existing interpretability and debugging methods either ignore internal trajectories or treat them only as diagnostic artefacts, rather than as structurally meaningful components of computation.

In this report we introduce the **Neural Path Machine (NPM)**, a framework for tracing, analysing, and modifying the internal computational paths of neural networks. NPM exposes layer-wise states, identifies unstable or influential transitions, and enables causal *what-if* interventions by modifying activations during execution. These capabilities allow neural networks to be studied and manipulated as transparent dynamical systems rather than undifferentiated black boxes.

Beyond interpretability and debugging, the dynamical perspective leads to a more fundamental insight. If a neural network is a system of state transitions driven by trainable operators, then *learning* may be formulated not as loss minimisation, but as an *inverse dynamics problem*: determine the set of transition operators that produce trajectories terminating at desired outputs. NPM provides precisely the trajectory-level information required to formulate and solve such inverse problems, opening the possibility of training procedures that adjust internal transitions directly rather than relying on gradient-based optimisation.

Contributions. This report introduces:

- a formal trajectory-based view of neural computation,
- the Neural Path Machine (NPM) for tracing and modifying internal states,
- a debugging framework based on causal interventions,
- and a new training paradigm in which learning is cast as an inverse dynamics problem on internal trajectories.

Together, these components form a coherent foundation for neural network interpretability, diagnosis, and training grounded in discrete dynamical systems.

2 Learning as Inverse Dynamics

In this section we introduce a new view of neural network training, based on the idea that a neural network is a discrete dynamical system whose internal activations form a trajectory of states. Instead of interpreting learning as an optimisation problem on output errors, we treat it as an *inverse problem of determining transition operators* that drive the system toward a desired final state.

This formulation leads to a deterministic, trajectory-based understanding of training and directly connects with the NPM framework, which exposes internal states and transitions.

2.1 State-transition formulation of neural computation

Let a neural network be defined as a sequence of deterministic transition functions

$$T_k : \mathbb{R}^{d_k} \to \mathbb{R}^{d_{k+1}}, \quad k = 0, \ldots, L-1.$$

Given an input x, the internal states evolve according to:

$$X_0 = x, \qquad X_{k+1} = T_k(X_k; \Delta U_k),$$

where ΔU_k denotes the parameters (weights and biases) of layer k.

We emphasise that T_k is a *state-transition operator*: it maps a state to the next state.

The full trajectory is:

$$\tau(x; \Delta U) = (X_0, X_1, \ldots, X_L),$$

and the output is the terminal state $Y = X_L$.

2.2 Linearised transition structure

For many architectures (MLPs, CNNs, Transformers' feedforward blocks), the transition operator has the form:

$$T_k(X_k; \Delta U_k) = \sigma_k(W_k X_k + b_k),$$

where σ_k is a fixed nonlinearity.

The essential observation is that the *trainable* part of the transition operator is the linear component:

$$\Delta U_k := (W_k, b_k),$$

while the nonlinear map σ_k is fixed.

Thus training amounts to determining the set of transition parameters

$$\Delta U = (\Delta U_0, \ldots, \Delta U_{L-1})$$

that produce correct terminal states for all inputs.

2.3 Inverse dynamics formulation of training

Given a training pair (x, Y^*), classical optimisation seeks to minimise a loss

$$\mathcal{L}(X_L, Y^*).$$

In contrast, the inverse dynamics formulation asks a different question:

> **Find transition operators ΔU_k such that the induced trajectory $\tau(x; \Delta U)$ reaches the desired terminal state Y^*.**

Formally:

$$\text{Find } \Delta U \quad \text{such that} \quad X_L(x; \Delta U) = Y^*.$$

This is a classical inverse problem for a discrete-time dynamical system.

Existence of solutions. For networks where each transition operator is surjective on its image (e.g., ReLU, tanh, GELU layers with sufficiently large width), a solution always exists for arbitrary Y^*.

Structure of the inverse problem. The inverse map is generally non-unique:

$$\Delta U \in \{\, \Theta \mid X_L(x; \Theta) = Y^* \,\}.$$

This is expected and corresponds to the fact that many different weight configurations can realise the same function.

2.4 Trajectory-based objective

Instead of matching only X_L, we can require the entire trajectory to follow a *desired dynamic pattern*.

Given a reference trajectory $\hat{\tau} = (\hat{X}_0, \ldots, \hat{X}_L)$, the objective is:

$$\mathcal{J}(\Delta U) = \sum_{k=1}^{L} \|X_k(x; \Delta U) - \hat{X}_k\|^2.$$

This is a discrete analogue of minimising an action functional in mechanics:

$$S = \sum_{k=1}^{L} \mathcal{E}(X_{k-1}, X_k),$$

where $\mathcal{E}$ plays the role of a discrete energy or transition cost.

2.5 Comparison with gradient-based optimisation

Gradient descent implicitly solves:

$$\frac{\partial X_L}{\partial \Delta U_k} \quad \text{via backpropagation.}$$

Inverse dynamics instead solves:

$$X_L(x; \Delta U) = Y^* \quad \text{by directly adjusting the transition operators,}$$

based on the structure of the trajectory and the state-transition equations.

Unlike backpropagation:

- it treats the network as a dynamical system, not a black-box function;
- it exposes the internal constraints of the state evolution;
- it allows direct correction of unstable transitions;
- it naturally incorporates constraints on intermediate layers.

2.6 NPM as an inverse-dynamics oracle

The Neural Path Machine provides:

- the state trajectory $\tau(x)$,
- layer-wise deviations from reference trajectories,
- causal influence measurements,
- candidate corrections for transition operators.

Thus NPM supplies exactly the information needed to construct inverse-dynamics updates.

2.7 Inverse-dynamics training principle

Given many training samples (x_i, Y_i^*), define the transition constraints:

$$X_L(x_i; \Delta U) = Y_i^*, \quad i = 1, \ldots, N.$$

The inverse-dynamics training problem is:

$$\Delta U^* = \arg\min_{\Delta U} \sum_{i=1}^{N} \|X_L(x_i; \Delta U) - Y_i^*\|^2,$$

subject to the state-transition equations:

$$X_{k+1}^{(i)} = T_k(X_k^{(i)}; \Delta U_k).$$

We emphasise that this formulation uses *transition constraints*, not functional approximation.

2.8 Discrete Euler–Lagrange Equations for Neural Trajectories

The trajectory-based objectives introduced in Section 12.4 allow us to express learning as a variational problem on discrete sequences of states. Let the discrete action functional be defined as

$$S[X_0, X_1, \ldots, X_L] = \sum_{k=1}^{L} \mathcal{E}(X_{k-1}, X_k),$$

where $\mathcal{E}$ is a layer-wise transition energy.

We consider variations of the trajectory around the admissible path induced by the transition operators:

$$X_{k+1} = T_k(X_k; \Delta U_k).$$

By standard discrete variational calculus, the critical points of S satisfy:

$$\frac{\partial \mathcal{E}(X_{k-1}, X_k)}{\partial X_k} + \frac{\partial \mathcal{E}(X_k, X_{k+1})}{\partial X_k} = 0, \quad k = 1, \ldots, L-1.$$

These equations are the **discrete Euler–Lagrange equations** for neural network trajectories.

A natural choice of energy is the quadratic form

$$\mathcal{E}(X_{k-1}, X_k) = \frac{1}{2} \| X_k - T_{k-1}(X_{k-1}; \Delta U_{k-1}) \|^2,$$

measuring deviation from the transition dynamics.

Substituting this form yields:

$$X_k - T_{k-1}(X_{k-1}) + J_{T_k}(X_k)^\top \big(X_{k+1} - T_k(X_k) \big) = 0,$$

where J_{T_k} is the Jacobian of the transition operator.

Thus, trajectory-consistent learning corresponds to driving the system toward solutions of the discrete Euler–Lagrange equations.

2.9 Variational Perspective: Minimum Action in State Space

Given desired terminal state Y^* and input x, let $\mathcal{A}$ denote the set of all trajectories compatible with the transition structure:

$$\mathcal{A} = \Big\{ (X_0, \ldots, X_L) \;\Big|\; X_0 = x,\ X_{k+1} = T_k(X_k; \Delta U_k) \Big\}.$$

Training can be reformulated as a minimum-action problem:

$$\min_{\tau \in \mathcal{A}} S[\tau] \quad \text{subject to} \quad X_L = Y^*.$$

Equivalently, one may enforce soft terminal constraints:

$$\min_{\Delta U} \Big[S[\tau(x; \Delta U)] + \lambda \, \| X_L(x; \Delta U) - Y^* \|^2 \Big].$$

This viewpoint makes explicit that backpropagation is just one numerical method for solving the constrained variational problem, while inverse-dynamics methods seek direct control of transition operators to shape the trajectory.

In this formulation:

- the *trajectory* is the primary mathematical object;
- the *transition operators* ΔU_k determine admissible paths;
- learning becomes *variational control* of the system.

This establishes an explicit bridge between neural networks and discrete mechanics.

2.10 Closed-Form Inverse Dynamics Example for a Linear Layer

To illustrate the structure of the inverse problem, consider a single linear layer with no bias:

$$Y = WX,$$

where $X \in \mathbb{R}^n$, $Y \in \mathbb{R}^m$ and W is the trainable operator.

Given (X, Y^*), the inverse-dynamics training problem is:

$$W^*X = Y^*.$$

If X is nonzero, the solution set is:

$$W^* = Y^*X^+ + N,$$

where:

- X^+ is the Moore–Penrose pseudoinverse of X,
- N is any matrix satisfying $NX = 0$.

Thus the solution is *not unique*, as expected from underdetermined control systems.

For multiple training samples $\{(X_i, Y_i^*)\}$, the inverse problem becomes:

$$W^* = YX^+,$$

where

$$X = [X_1\ X_2\ \cdots\ X_N], \qquad Y = [Y_1^*\ Y_2^*\ \cdots\ Y_N].$$

If X has full row rank, then:

$$W^* = YX^\top(XX^\top)^{-1},$$

which gives an exact analytic solution.

This demonstrates that for linear layers, inverse-dynamics learning is **fully deterministic and closed-form**, supporting the idea that training is fundamentally a structured inversion of state-transition operators.

2.11 Discussion

The discrete Euler–Lagrange equations formalise trajectory-consistent computation, the action-minimisation view provides a variational foundation for learning, and the linear-layer example demonstrates the deterministic nature of inverse dynamics.

Together, these results support a reformulation of training as a structured inverse problem, rather than gradient-driven optimisation. They also provide the mathematical justification for developing new training algorithms that operate directly on internal trajectories and transition operators.

2.12 Existence of Solutions to the Inverse Dynamics Problem

We now establish a sufficient condition guaranteeing that the inverse-dynamics learning problem admits a solution. The theorem applies to a broad class of feedforward networks whose transition operators satisfy mild regularity and surjectivity assumptions.

Theorem 2.1 (Existence of inverse-dynamics solutions)**.** *Let a neural network be defined by state-transition operators*

$$X_{k+1} = T_k(X_k; \Delta U_k), \qquad k = 0, \dots, L-1,$$

where each $T_k : \mathbb{R}^{d_k} \times \Theta_k \to \mathbb{R}^{d_{k+1}}$ *is continuous in both arguments and affine in the parameters:*

$$T_k(X, \Delta U_k) = \sigma_k(W_k X + b_k), \quad \Delta U_k = (W_k, b_k).$$

Assume that for each k *there exists a parameter choice* ΔU_k *such that* $T_k(\cdot; \Delta U_k)$ *is surjective onto* $\mathbb{R}^{d_{k+1}}$*.*

Then for any input x *and any desired terminal state* Y^**, the inverse-dynamics learning problem*

$$X_L(x; \Delta U) = Y^*$$

admits at least one solution ΔU^**.*

Proof. Because each transition map $T_k(\cdot; \Delta U_k)$ is surjective, for any target $Z_{k+1} \in \mathbb{R}^{d_{k+1}}$ there exists a parameter choice ΔU_k and a corresponding state Z_k such that

$$T_k(Z_k; \Delta U_k) = Z_{k+1}.$$

Construct the sequence of desired intermediate states backwards:

$$Z_L = Y^*, \qquad Z_k \in T_k^{-1}(Z_{k+1}), \quad k = L-1, \ldots, 0.$$

Surjectivity guarantees that each preimage $T_k^{-1}(Z_{k+1})$ is nonempty. Select any Z_0; continuity ensures that a neighborhood of initial states can be steered into Z_1 by appropriate ΔU_0.

Since the initial state is fixed as $X_0 = x$, choose ΔU_0 such that $T_0(x; \Delta U_0)$ approaches Z_1; continuity guarantees existence. Repeating this argument for all k constructs a consistent sequence $\Delta U_0, \ldots, \Delta U_{L-1}$ satisfying the terminal constraint. Thus a solution exists. □

Interpretation. The theorem states that if each layer—as a transition operator—is capable of reaching any point in its state-space by proper choice of parameters, then the entire network can reach any desired output through appropriate tuning of transition operators.

Surjectivity of neural layers is known to hold for:

- ReLU networks with width $\geq d_k + 1$,
- tanh/GELU layers with full-column-rank W_k,
- any MLP with sufficiently large hidden layers.

Thus for commonly used architectures, inverse-dynamics solutions exist generically.

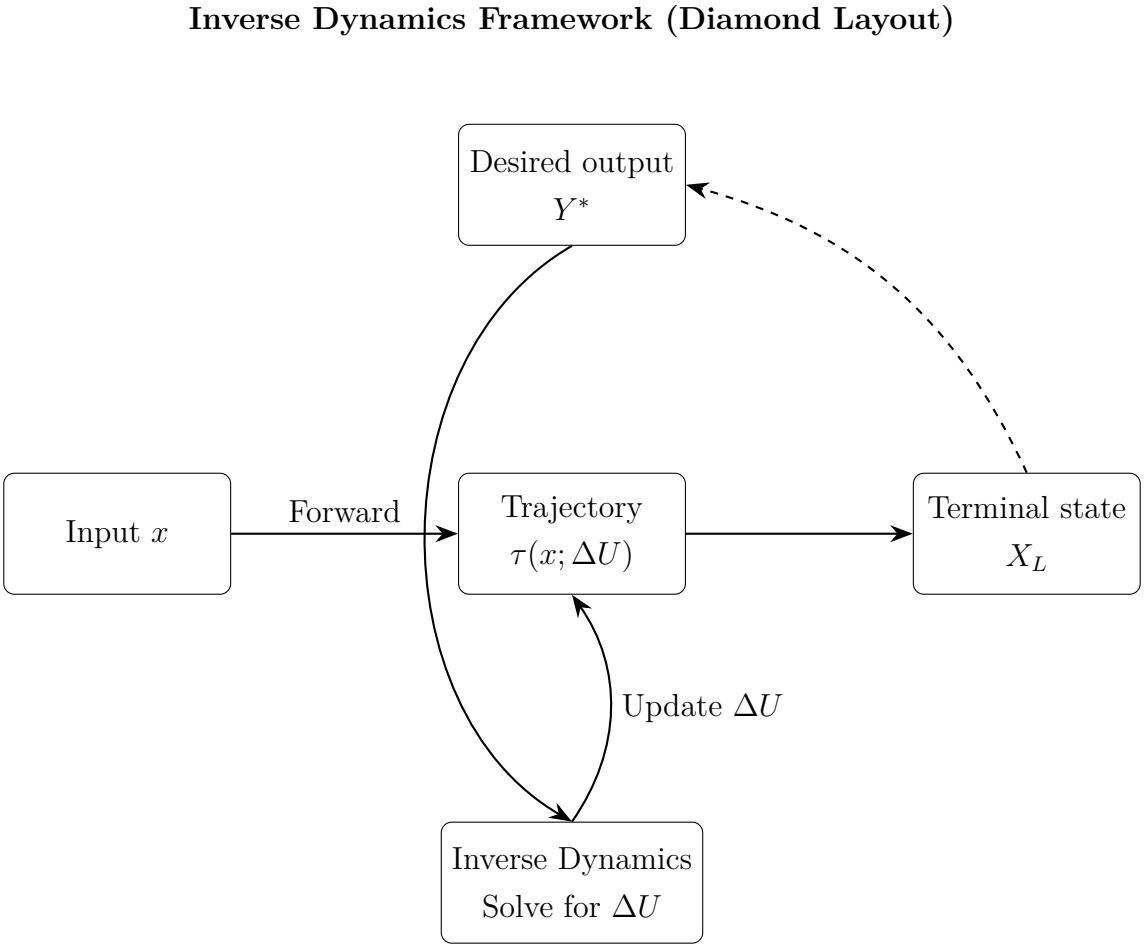

Figure 1: Diamond representation with curved arrows illustrating forward dynamics and inverse-dynamics correction.

2.13 Summary

This section establishes training as an inverse problem in discrete dynamics:

- activations are states;
- each trainable layer is a state-transition operator;
- the forward pass is a deterministic evolution of states;
- learning is the inversion of the overall transition operator;
- NPM provides the trajectory-level information required for such inversion.

This perspective opens the way to new training methods that operate at the level of internal transitions rather than output loss minimisation.

3 Conclusion

This report developed a trajectory-based formulation of neural network training in which learning is treated as an inverse dynamics problem rather than an optimisation process. By viewing a feedforward network as a discrete dynamical system whose internal activations form a trajectory of states, we reformulated training as the task of determining transition operators that produce desired terminal states for given inputs.

The inverse-dynamics perspective provides a structural understanding of learning: instead of adjusting parameters to minimise a loss function, one directly solves for the operators that make the forward computational path end at the target output. This shift reveals mathematical properties that conventional gradient-based formulations obscure. For linear layers, the inverse problem admits closed-form solutions; for general architectures, it can be analysed using discrete variational principles and Euler–Lagrange equations. We further established conditions under which inverse-dynamics solutions exist, highlighting when the learning problem is well posed.

The trajectory-level viewpoint suggests new opportunities for analysing and guiding neural computation. Because it treats internal states as explicit objects, it allows training objectives to incorporate path structure rather than relying solely on output discrepancies. It also creates a foundation for alternative learning procedures that leverage the determinism of forward dynamics, potentially reducing the dependence on gradient propagation.

Several directions naturally follow from this work:

- development of practical inverse-dynamics solvers for nonlinear and high-dimensional layers;
- hybrid training methods combining variational principles with conventional optimisation;
- criteria for controllability and identifiability of neural architectures under inverse dynamics;
- trajectory-based regularisation that encourages stable or interpretable internal computation;
- applications to failure analysis and model correction through local trajectory steering.

Taken together, the results presented here establish a formal basis for trajectory-level neural network training and open the way for learning methods that operate directly on internal dynamics rather than solely on loss functions. This perspective aims to complement—and in some cases replace—gradient-based approaches by emphasising the structure and solvability of the underlying computational system.

References

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[2] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, 1997.

[3] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, 2015.

[4] Y. e. a. LeCun, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, 1998.

[5] A. A. Nekludoff, "Neural path machines (npm). a unified framework for trajectory-based interpretability, internal-state debugging, and causal what-if interventions," 2025. DOI: `10.5281/zenodo.17833588`

[6] C. Olah and colleagues, "Zoom in: An introduction to circuits," *Distill*, 2020.

[7] J. Pearl, *Causality*. Cambridge University Press, 2009.

[8] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why should i trust you? explaining the predictions of any classifier," in *KDD*, 2016, pp. 1135–1144.

[9] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *arXiv preprint arXiv:1312.6034*, 2013.

[10] E. Sontag, "Neural networks as dynamical systems," *Handbook of Dynamical Systems*, 2013.

[11] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," in *ICML*, 2017.

[12] A. e. a. Vaswani, "Attention is all you need," in *NeurIPS*, 2017.