Neural Geodesic Flows

Learning Dynamics as Geodesics on a Latent Manifold

Mihir Talati (after J. Bürge, ETH Zurich 2025)

December 11, 2025

Motivation

IS THERE A DOCTOR ON BOARD?
I HAVE A DOCTOR'S DEGREE...
COME QUICKLY! A PASSENGER IS BLEEDING TO DEATH!
IN PHILOSOPHY!
WARANDPEAS.COM

Presentation Outline

Presentation Outline

- Neural Ordinary Differential Equations
  - From ResNets to ODEs
  - Neural ODEs in practice
  - ODE-based models for dynamics
- Motivation and Approach
  - Manifold hypothesis & autoencoders
  - Why bring in geometry?
  - High-level idea of Neural Geodesic Flows
- Differential Geometry in 10 Minutes
  - Manifolds and charts
  - Tangent vectors and the tangent bundle
  - Riemannian metric and geodesics
- Neural Geodesic Flows Model
  - Data and mapping assumptions
  - Architecture: autoencoder + geodesic ODE
  - Forward pass and loss functions
  - Implementation sketch (JAX)
  - Relation to other models
- Case Studies and Results
  - Toy example: geodesics on the sphere
  - Two-body problem

Neural Ordinary Differential Equations

ResNet as an Euler step

- A (very) simplified ResNet block:

$$h_{k+1} = h_k + f_\theta(h_k)$$

  where k is the layer index.
- If you think of k as discrete time and f_θ as a velocity field, this is exactly a forward Euler step for the ODE

$$\frac{dh}{dt} = f_\theta(h(t)).$$

- Idea of *Neural ODEs*: instead of stacking many discrete layers, treat depth as continuous time and let an ODE solver play the role of the network.
- Forward pass $\Rightarrow$ solve an ODE; backward pass $\Rightarrow$ differentiate *through* the ODE solver (adjoint method, automatic differentiation, . . .).

Neural ODEs as continuous-depth networks

- Define a dynamics model
$$\frac{dh}{dt} = f_\theta(h(t), t),$$
where f_θ is a neural network.
- Given initial state $h(0)$, an ODE solver gives
$$h(T) = \mathrm{ODESolve}(f_\theta, h(0), T).$$
- Neural ODEs are good for:
  - Modeling continuous-time data (trajectories, physical systems, time series).
  - Evaluating the model at arbitrary times (interpolation, extrapolation).
  - Sharing parameters across “depth”.
- Many variants: neural PDEs, Hamiltonian / Lagrangian NNs, neural manifold dynamics, . . .

NODEs, LNNs, HNNs (very briefly)

- **Neural ODEs (NODEs)**: learn a generic ODE f_θ that fits observed trajectories.
- **Lagrangian NN (LNN)**:
  - Learn a Lagrangian $L_\theta(q, \dot{q})$.
  - Dynamics come from Euler–Lagrange equations.
  - Good at conserving (approximate) energy.
- **Hamiltonian NN (HNN)**:
  - Learn Hamiltonian $H_\theta(q, p)$.
  - Dynamics come from Hamilton's equations.
  - Built-in symplectic structure and energy conservation.
- These models learn an *underlying law* of motion, not just a black-box predictor.

Motivation and Approach

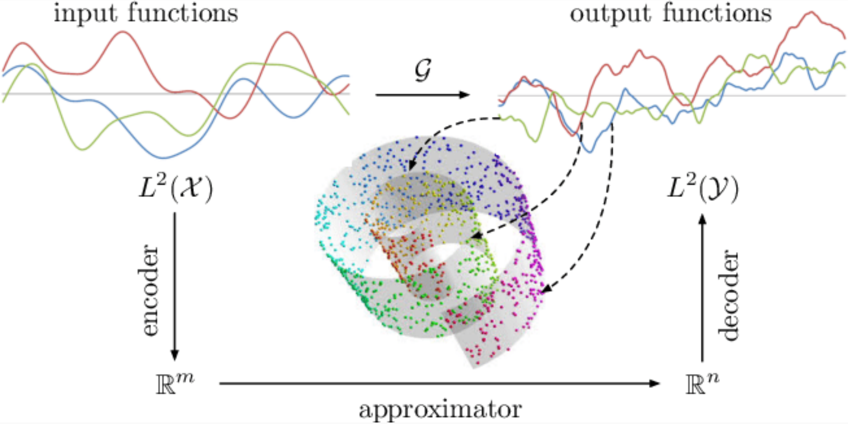
The manifold hypothesis

- Real-world data often live in a high-dimensional ambient space $\mathbb{R}^n$ (e.g. 784-D MNIST images, fluid simulations, video frames).
- **Manifold hypothesis**: data actually lie near a much lower-dimensional *manifold* embedded in $\mathbb{R}^n$.
  - Example: images of a rotating object $\approx$ points on a low-dimensional surface.
- **Autoencoders**:

$$\text{data manifold } \subset \mathbb{R}^n \longleftrightarrow \text{ latent manifold } \subset \mathbb{R}^{\tilde{d}}$$

  with encoder ψ and decoder ϕ.
- Manifold learning + dynamics:
  - Learn a good latent representation *and*
  - learn dynamics on that latent manifold.

Manifold hypothesis



Why Riemannian geometry for dynamics?

- A **Riemannian manifold** (M, g) is a manifold M equipped with a *metric tensor* g:
  - At each point, g defines dot products, lengths, angles, distances.
  - Think: curved generalization of Euclidean space.
- Once we have a metric g, we get:
  - Intrinsic distances and shortest paths (**geodesics**).
  - Curvature, areas, volumes, covariant derivatives, ...
- For dynamics data, learning a Riemannian metric means:
  - The geometry of the latent space encodes the "nature" of the dynamics.
  - We can analyze trajectories using curvature, geodesic distances, etc.
- **Neural geodesic flows** (NGFs) combine:
  - Manifold learning (autoencoder to a latent manifold),
  - Riemannian geometry (learn a metric),
  - and ODE-based dynamics (geodesic flow).

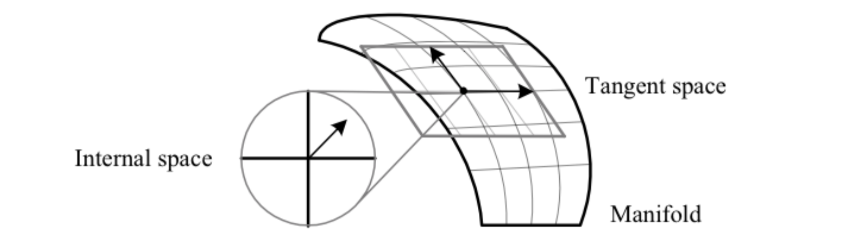
Core idea

- Assume data are trajectories $y(t)$ living on some unknown manifold embedded in $\mathbb{R}^n$.
- NGFs try to learn:
  1. A low-dimensional latent **Riemannian manifold** (M, g).
  2. A mapping of data into its **tangent bundle** TM (states + velocities).
  3. A **geodesic flow** on TM that matches the observed dynamics.
- The forward pass:

$$\text{data} \xrightarrow{\psi_\theta} z_0 \in TM \xrightarrow{\text{geodesic ODE w.r.t. } g_\theta} z_T \xrightarrow{\phi_\theta} \text{predicted data.}$$

- We learn:
  - the encoder ψ_θ (chart + diffeomorphism to TM),
  - the decoder ϕ_θ (parametrization back to data space),
  - and the metric g_θ (a neural net).

Dynamics



Differential Geometry in 10 Minutes

What is a manifold? (intuitive version)

- A **manifold** is a space that *locally* looks like $\mathbb{R}^d$, but can be globally curved or have complicated topology.
- Examples:
  - Circle S^1 (1D manifold).
  - Sphere S^2 (2D manifold).
  - Torus (donut surface, 2D manifold).
- To do calculus we use **charts**:
  - A chart is a local coordinate map
  $$\varphi : U \subset M \to \mathbb{R}^d.$$
  - One chart usually does not cover the whole manifold (e.g. longitude/latitude singularities on a sphere).
  - An **atlas** is a collection of charts that covers M.
- In NGFs we mostly work with a *single* learned chart for the latent manifold.

Tangent spaces and tangent bundle

- At each point $x \in M$ we have a **tangent space** $T_x M$: the space of velocities of curves going through x.
- Think: all possible directions you can move if you are standing at x.
- The **tangent bundle** TM collects all tangent spaces:

$$TM = \bigsqcup_{x \in M} T_x M.$$

  A point in TM is a pair (x, v) with $v \in T_x M$.
- For dynamics, (x, v) is a natural state: position x and velocity v.
- NGFs learn dynamics as a flow on TM.

Riemannian metric and geodesics

- A **Riemannian metric** g assigns to every $x \in M$ an inner product $g_x(\cdot,\cdot)$ on T_xM.
  - Generalizes the dot product.
  - Lets us define lengths, angles, energies, curvature.
- The length of a curve $\gamma(t)$ is

$$L(\gamma) = \int \sqrt{g_{\gamma(t)}(\dot{\gamma}(t), \dot{\gamma}(t))}\, dt.$$

- **Geodesics** are curves that locally minimize length (or equivalently, energy).
  - In flat space: straight lines.
  - On a sphere: great circles.
- In coordinates, geodesics satisfy the *geodesic equation*

$$\ddot{x}^k + \Gamma^k_{ab}(x)\,\dot{x}^a\dot{x}^b = 0,$$

where Γ^k_{ab} are the (Levi–Civita) connection coefficients computed from g.

Neural Geodesic Flows Model

Setup: data and latent space

- Data live in $\mathbb{R}^n$, but actually lie on some unknown submanifold $\tilde{N} \subset \mathbb{R}^n$.
- There is some (unknown) diffeomorphism

$$F : \tilde{N} \to N = TM,$$

  where N is the tangent bundle of a latent Riemannian manifold (M, g).
- On N we assume the dynamics are simply the **geodesic flow** (moving along geodesics of (M, g)).
- Roughly:
  - Data trajectories in $\mathbb{R}^n \approx$ trajectories in TM that are geodesics of some unknown metric g.
  - Our job: learn both the manifold geometry and the mapping that makes this true *approximately*.

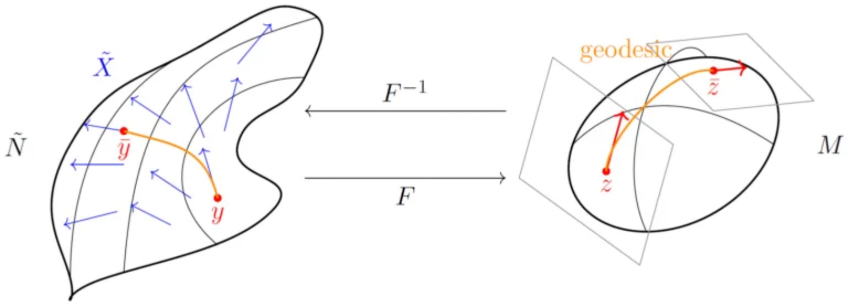
Encoder, metric network, decoder

- In practice we do not know $\tilde{N}$, M, F, or g.
- We learn:
  - Encoder $\psi_\theta : \mathbb{R}^n \to U \subset \mathbb{R}^{2m}$
    (a single chart of TM; outputs (x, v)-coordinates).
  - Decoder $\phi_\theta : U \to \mathbb{R}^n$
    (maps latent state back to data space).
  - Metric network $g_\theta : \mathbb{R}^m \to \mathbb{R}^{m \times m}$
    (given a position x, outputs a positive-definite matrix $g_\theta(x)$).
- Typical metric parametrization:

$$g_\theta(x) = I + L_\theta(x) L_\theta(x)^\top$$

(ensures positive definiteness).

Manifold hypothesis



Geodesic ODE and exponential map

- Once g_θ is known, we can compute connection coefficients $\Gamma^k_{ab}(x)$ by differentiating g_θ.
- This gives the geodesic ODE in coordinates:

$$\ddot{x}^k = -\Gamma^k_{ab}(x)\,\dot{x}^a\dot{x}^b.$$

- We package this into a vector field X on TM and define the **exponential map**:

$$\exp_{g_\theta}(z_0, t)$$

  $=$ the result of integrating the geodesic ODE starting at z_0 for time t.
- In code, this is implemented as an ODE solver (e.g. RK4) over the vector field defined by g_θ.

Forward pass of an NGF

- Given an input $y_0 \in \mathbb{R}^n$ and a time t:

$$z_0 = \psi_\theta(y_0) \quad \text{(encode to } TM\text{)}$$
$$z_t = \exp_{g_\theta}(z_0, t) \quad \text{(geodesic ODE solve)}$$
$$\hat{y}_t = \phi_\theta(z_t) \quad \text{(decode back to data).}$$

- This is a **neural differential equation** where the ODE is *specifically* the geodesic ODE of a learned metric.
- At the same time, it is:
  - a NODE (because we learn an ODE in latent space),
  - a Lagrangian NN (geodesic motion minimizes an energy functional),
  - and a Hamiltonian NN (there is an associated geodesic Hamiltonian).

Training objectives

- We have trajectories or pairs $(\text{input}, \text{target}, \Delta t)$.
- Typical losses combine:
  1. **Reconstruction loss**: make $\phi_\theta(\psi_\theta(y)) \approx y$ for all observed data points.
  2. **Data-space prediction loss**:

$$\phi_\theta\big(\exp_{g_\theta}(\psi_\theta(y_0), \Delta t)\big) \approx y_{\text{target}}.$$

  3. **Latent-space prediction loss**:

$$\exp_{g_\theta}(\psi_\theta(y_0), \Delta t) \approx \psi_\theta(y_{\text{target}}).$$

- Intuition:
  - Autoencoder should be approximately invertible on the data manifold.
  - Geodesic flow in latent space should match the observed dynamics, both before and after decoding.

Implementation ingredients (JAX / Equinox)

- JAX is used for:
  - automatic differentiation (needed for metric $\to$ Christoffel).
  - vectorization over batches.
- A `TangentBundle` class encapsulates:
  - ψ: encoder (diffeomorphism + chart).
  - ϕ: decoder (inverse chart + inverse diffeomorphism).
  - g: metric network.
  - ODE solver (`exp`, `exp_return_trajectory`).
- Training uses an optimizer (e.g. Adam via Optax), which differentiates through:
  - the encoder and decoder, and
  - the numerical ODE solver for geodesics.
- Once trained, the same class can also compute geometric quantities: scalar products, curvature, sectional curvature, ...

How NGFs relate to NODE, LNN, HNN, NMD

- **As a NODE**: the latent geodesic vector field is an ODE, so NGFs are special Neural ODEs.
- **As LNN/HNN**:
  - A geodesic flow is a special case of both Lagrangian and Hamiltonian dynamics.
  - NGFs are LNNs/HNNs where the Lagrangian/Hamiltonian has a fixed geometric form induced by g_θ.
- **Compared to Neural Manifold Dynamics (NMD)**:
  - NMD learns an atlas and a generic NODE on the latent manifold.
  - NGFs learn a *metric* and enforce the ODE to be geodesic, giving more geometric structure (at the cost of more constraints).
- The hope: this additional structure leads to better energy conservation, interpretability, and insights into the underlying dynamics.

Case Studies and Results

Toy problem: geodesic flow on S^2

- Goal: sanity check that NGFs can recover known geometry and dynamics.
- Data: trajectories that are *true* geodesics on the 2D sphere S^2 with its standard spherical metric.
- Setup:
  - Choose $N = TS^2$ as ground truth.
  - Sample geodesic curves on S^2 and embed them into $\mathbb{R}^n$.
  - Train NGF model to learn both:
    - the latent S^2-like manifold and
    - its geodesic flow.
- Results (qualitative summary):
  - Latent space looks spherical.
  - Learned metric has nearly constant positive curvature (like a sphere).
  - Geodesic predictions match the ground truth geodesics well.

Classical mechanics: two-body problem

- Dynamics: two masses interacting via Newtonian gravity in 2D.
  - Phase space: positions and momenta.
  - True system has conserved physical energy and angular momentum.
- NGF setup:
  - Autoencode the state of the system into a latent TM.
  - Use geodesic flow as the latent dynamics.
  - Compare with a Hamiltonian NN baseline.
- Results (high-level):
  - NGFs can predict orbits for a reasonably long time horizon.
  - The model conserves a learned "geodesic energy" and approximately conserves the true physical energy.
  - Performance is in the same ballpark as HNNs, sometimes slightly worse, but with richer geometric structure.

Navier–Stokes: first experiments

- Navier–Stokes: nonlinear PDE describing incompressible fluid flow.
- Idea: treat snapshots of the velocity field as points on an unknown low-dimensional manifold; try to learn a geodesic flow that approximates the dynamics.
- Practical difficulties:
  - High-dimensional input space and complex dynamics.
  - Single-chart architecture may not be expressive enough.
  - Numerical stiffness and long time horizons make ODE solving harder.
- Outcome:
  - Current NGF implementation struggled to learn good dynamics here.
  - Indicates that more expressive architectures (multiple charts, better encoders, specialized regularization) are needed for complex PDEs.

Limitations & possible extensions

- **Single chart**: current implementation learns only one global chart for the latent manifold.
  - Not suitable for manifolds that cannot be covered by a single chart (e.g. full sphere).
  - Extension: atlas of charts + chart-switching during integration.
- **Expressivity vs. constraints**:
  - Enforcing geodesic dynamics may make some tasks harder to fit.
  - But it gives geometric interpretability and energy conservation.
- **Interpretability**:
  - Metric g_θ and curvature could reveal hidden structure of the dynamics (e.g. conserved quantities, effective dimensions).
- **Future directions**:
  - Atlas-based NGFs (multiple charts).
  - Stronger priors on g_θ for known physics (symmetries, invariances).
  - Symbolic regression on learned geometric quantities.

Takeaways

- Neural geodesic flows:
  - Learn a latent Riemannian manifold and its geodesic flow.
  - Combine manifold learning, differential geometry, and neural ODEs.
- For ML students:
  - You can think of NGFs as “NODEs with geometry built in”.
  - The metric g_θ is a learned notion of distance / energy on the latent space.
  - Trajectories become shortest paths with respect to this learned geometry.
- Big picture:
  - Using geometry can make models more interpretable and physically faithful.
  - There is still a lot to explore: better architectures, richer manifolds, and new applications.