COMPUTATIONAL DIFFERENTIAL GEOMETRY

DIFFERENTIAL GEOMETRY

A *differential manifold* is a collection of points that are connected to each other in a smooth fashion such that the neighborhood of each point looks like the neighborhood of an m-dimensional Cartesian space. m is the dimensionality of the manifold.

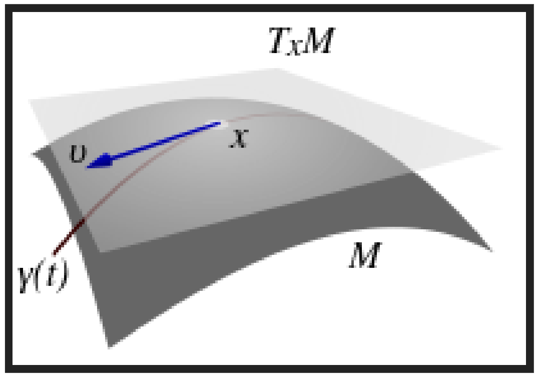
It is customary to use "manifold" to mean "differentiable manifold."

But what is a point? Any type of object you want.

In our case, points are probability distributions distinguished by their predictions.

TANGENT SPACE

- At each point, x, on the manifold we attach a vector space T_x called the tangent space.
- Vectors live in a tangent space.
- Comparing vectors in nearby tangent spaces (to do calculus, for example) requires a *Connection*.



INDEX GYMNASTICS

- Vectors and Components: (Implied sum over repeated indices.)

$$\mathbf{v} = v^{\alpha}\mathbf{e}_{\alpha} = v_{\alpha}\epsilon^{\alpha}$$

  - Basis Vectors: $\mathbf{e}_{\alpha}$
  - Contravariant Components: v^{α}
  - Dual Basis Vectors: ϵ^{α}
  - Covariant Components: v_{α}
  - $\mathbf{e}_{\alpha}(\epsilon^{\beta}) = \delta_{\alpha}^{\beta}$
- Basis vectors are coordinate derivatives:
  - $\mathbf{e}_{\alpha} = \partial\mathbf{y}/\partial\theta_{\alpha}$
- Metric is the inner product of basis vectors:
  - $g_{\alpha\beta} = \mathbf{e}_{\alpha} \cdot \mathbf{e}_{\beta} = (J^{T}J)_{\alpha\beta}$

INDEX GYMANSTICS

Raising/Lowering Indices:

- $g^{\alpha\beta} \equiv (g^{-1})^{\alpha\beta}$
- $g_{\alpha\gamma} g^{\gamma\beta} = \delta_{\alpha}^{\beta}$
- $v_{\alpha} = g_{\alpha\beta} v^{\beta}$
- $v^{\alpha} = g^{\alpha\beta} v_{\alpha}$

Transformations (Reparameterization):

Let θ and ϕ be two coordinate maps on the manifold: $\theta = \theta(\phi)$

- $\theta = \theta(\phi)$
- Denote coordinates in ϕ-basis with tick mark:
  '
- $v^{\alpha} = \frac{\partial \theta^{\alpha}}{\partial \phi^{\alpha'}} v^{\alpha'}$

CONNECTION

Goal: Tensor Calculus.

$$
\begin{aligned}
\frac{\partial}{\partial\theta^\alpha}\mathbf{v} &\equiv \partial_\alpha \mathbf{v} \\
&= \partial_\alpha \left(v^\beta \mathbf{e}_\beta\right) \\
&= \left(\partial_\alpha v^\beta\right)\mathbf{e}_\beta + v^\gamma \left(\partial_\alpha \mathbf{e}_\gamma\right) \\
&= \left(\partial_\alpha v^\beta\right)\mathbf{e}_\beta + v^\gamma \Gamma^\beta_{\alpha\gamma}\mathbf{e}_\beta
\end{aligned}
$$

- Γ are the connection coefficients that describe how nearby tangent spaces are connected.
- Γ is not a tensor. (Transforms differently)

CONNECTION

$$
\Gamma^\alpha_{\beta\gamma} = \Gamma^{\alpha'}_{\beta'\gamma'} \frac{\partial\theta^\alpha}{\partial\phi^{\alpha'}} \frac{\partial\phi^{\beta'}}{\partial\theta^\beta} \frac{\partial\phi^{\gamma'}}{\partial\theta^\gamma} + \frac{\partial\theta^\alpha}{\partial\phi^{\mu'}} \frac{\partial^2\phi^{\mu'}}{\partial\theta^\beta\partial\theta^\gamma}
$$

- In general, there may be many possible connections.
- An important result of Information Geometry is defining a family of connections known at the α-connections
- Fundamental Theorem of Riemannian Geometry
  - There is a unique, torsion-free connection that preserves the metric under parallel transport.
  - $\Gamma^{\alpha}_{\beta\gamma} = \frac{1}{2} g^{\alpha\delta} \left(\partial_\beta g_{\gamma\delta} + \partial_\gamma g_{\beta\delta} - \partial_\delta g_{\beta\gamma} \right)$

TENSOR CALCULUS

Covariant Derivative:

$$\nabla_\beta v^\alpha = \partial_\beta v^\alpha + \Gamma^{\alpha}_{\beta\gamma} v^\gamma$$

Parallel Transport:

A vector field v^α is parallel transported along the direction $\mathbf{u}$ if it satisfies

$$u^\beta \nabla_\beta v^\alpha = 0$$

GEODESICS

- Geodesics are curves that parallel transport their own tangent vector.

$$u^\beta \nabla_\beta u^\alpha = 0$$

- Let τ be a parameterization of the geodesic.
- Denote the geodesic by $\theta(\tau)$

- Tangent vector: $u^{\alpha} = \frac{d}{d\tau}\theta^{\alpha}$
- Geodesic Equation:

$$\frac{d^2}{d\tau^2}\theta^{\alpha} = -\Gamma^{\alpha}_{\beta\gamma}\frac{d\theta^{\beta}}{d\tau}\frac{d\theta^{\gamma}}{d\tau}$$

- Second order, nonlinear

EMBEDDING SPACE

- In general, calculating Γ is tedious, but is not difficult.
- Calculating Γ is necessary to find the geodesic equation on a given manifold.
- If we have expression for the manifold in an embedding space $\mathbf{y}(\theta)$, then the connection takes a nice form amenable to numerical methods:

$$\Gamma^{\alpha}_{\beta\gamma} = g^{\alpha\delta}\partial_{\delta}\mathbf{y} \cdot \partial_{\beta}\partial_{\gamma}\mathbf{y}$$

- The geodesic equation becomes:

$$\frac{d^2}{d\tau^2}\theta^{\alpha} = -g^{\alpha\delta}\partial_{\delta}\mathbf{y} \cdot \partial_{\beta}\partial_{\gamma}\mathbf{y}\frac{d\theta^{\beta}}{d\tau}\frac{d\theta^{\gamma}}{d\tau}$$

- Notice the *directional second derivative*

$$\partial_{\beta}\partial_{\gamma}\mathbf{y}\frac{d\theta^{\beta}}{d\tau}\frac{d\theta^{\gamma}}{d\tau}$$

COMPUTATIONAL METHODS

Given sloppy model $\mathbf{y}(\theta)$, we would like to explore it numerically using computational differential geometry.
We will need:

- Derivatives: $\partial_\alpha \mathbf{y}$
- Inverse Metric
    - Ill-conditioned
    - Derivatives must be calculated as accurately as possible (no finite-differences)
- Directional Second Derivatives: $v^\alpha v^\beta \partial_\alpha \partial_\beta \mathbf{y}$

Also, we would like do this for models as large as possible, so computation time is a concern.

SENSITIVITY EQUATIONS

Many of our models are ODEs: $\frac{d}{dt}\mathbf{x} = f(\mathbf{x}, \theta)$.

The *sensitivity* of $\mathbf{x}$ to a change in parameters $\mathbf{w} = \frac{\partial \mathbf{x}}{\partial \theta}$ satisfies the equation:

$$\frac{d}{dt}\mathbf{w} = \frac{\partial f}{\partial \mathbf{x}}\mathbf{w} + \frac{\partial f}{\partial \theta}$$

which is linear, but is solved simultaneously with that for $\mathbf{x}$.

These sensitivities can be derivatives along arbitrary directions of parameter space.

A similar equation exists for the second order sensitivities.

OUR (FIRST) APPROACH

We developed a modeling environment in Python/C.

Key features included:

1. A python script to define the ODE with several lists of strings
   - List of Parameters, θ
   - Dynamical Variables, $\mathbf{x}$
   - List of ODE equations, $f(\mathbf{x}, \theta)$
2. Sympy was used to calculate the directional derivative along an arbitrary direction v.
3. Automatically created and compiled c code to evaluate the model and first and second order directional derivatives.
4. All scripting was done in Python.

OUR (FIRST) APPROACH

- Our first approach worked reasonably well.
- The bottleneck ended up being compile time.

    - For models with ~50 Dynamical Variables and 100s of parameters, model generation/compile time could be about an hour.
    - This was fine until we started doing model reduction (more on that to come).
- When doing model reduction, we create many models and do few calculations with each, so the large overhead in model generation/compiling became a problem.

OUR CURRENT APPROACH

To overcome these problems, our new approach is based in Julia.

Julia is a high-level scripting language designed for scientific computing. (www.julialang.org)

It was specifically motivated to remove the need for multiple-language approach that we had used.

Downside of using Julia: Young and actively developed (current version 0.5) so language core and library APIs are subject to change.

OUR CURRENT APPROACH

In our current environment we write one julia function that defines our model.

Automatic differentiation (Dual numbers) is used to calculate directional derivatives (to arbitrary order).

In our experience, this approach is fast, stable, and ideal for working with large, sloppy models.

Downside: we are in the process of refactoring our code to take advantage of new librarie.s