Magda's 2026 set of technical notes

Last update: January 10, 2026

This is the 2026 set of my technical notes on various ML topics. I started writing the 1st set when beginning my PhD, continued as postdoc and occassionally even as prof. As a prof I have less time but picking up at least some of the good old habits might help in bringing me back from admin to research. All previous docs are available in my GitHub repo `https://github.com/mgswiss15/technotes`.

The general purpose of the notes is to help me understand better the selected topics by re-explaining (*re-* because these have been explained elsewhere many times), and to have a reference and possibly reusable material for later.

This is a working document not meant to be polished. There may be typos and other editing errors. Technical errors mean that I didn't quite understand something which I unfortunately cannot rule out.

## 1 Link between cosine similarity and $\ell_2$ distance

Cosine similarity of two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ is defined as:

$$S_C(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x}^\intercal \mathbf{y}}{\|\mathbf{x}\|_2 \|\mathbf{y}\|_2} \ . \tag{1}$$

$\ell_2$ distance (Euclidean distance) between the two vectors is defined as:

$$d_{\ell_2}(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2 \ . \tag{2}$$

And the relationship between the two is given by:

$$d_{\ell_2}(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2 = \sqrt{\|\mathbf{x}\|_2^2 + \|\mathbf{y}\|_2^2 - 2\mathbf{x}^\intercal \mathbf{y}} = \sqrt{\|\mathbf{x}\|_2^2 + \|\mathbf{y}\|_2^2 - 2S_C(\mathbf{x}, \mathbf{y})\|\mathbf{x}\|_2\|\mathbf{y}\|_2} \ . \tag{3}$$

This means:

$$d_{\ell_2}^2(\mathbf{x}, \mathbf{y}) = \|\mathbf{x}\|_2^2 + \|\mathbf{y}\|_2^2 - 2S_C(\mathbf{x}, \mathbf{y})\|\mathbf{x}\|_2\|\mathbf{y}\|_2 \ . \tag{4}$$

In particular, if the **vectors are normalized to unit length**, i.e., $\|\mathbf{x}\|_2 = \|\mathbf{y}\|_2 = 1$, then the relationship simplifies to:

$$d_{\ell_2}^2(\mathbf{x}, \mathbf{y}) = 2 - 2S_C(\mathbf{x}, \mathbf{y}) \Leftrightarrow 0.5\, d_{\ell_2}^2(\mathbf{x}, \mathbf{y}) = 1 - S_C(\mathbf{x}, \mathbf{y}) \ . \tag{5}$$

## 2 Volume mapping metrics

With Philipp we discuss mapping a $d$-dimensional sphere (or ball) from the data space to the $d$-dimensional latent space in a way to preserve the probability measure (volume) as much as possible.

### 2.1 Flow maps and velocity fields

This is based on Philipp's notation for flow matching models making it somewhat more explicit at places.

- Data space: $X(1) \in \mathbb{R}^d \sim \pi_1$
- Latent space: $X(0) \in \mathbb{R}^d \sim \pi_0$

We assume a flow map (unknown) wich maps points from the latent space to probability spaces along time $t \in [0, 1]$:

$$\begin{aligned} \psi : \mathbb{R}^d \times [0,1] &\mapsto \mathbb{R}^d \\ \psi(X(0), t) &= X(t) \\ \text{such that } \psi(X(0), 0) &= X(0) \quad \text{identity map at } t = 0, \\ \psi(X(0), 1) &= X(1) \quad \text{mapping to target data space at } t = 1, \\ \text{and } X(t) &\sim \pi_t \quad \text{probability distributions along time } t \ . \end{aligned} \tag{6}$$

The time derivative of the flow map gives the velocity field which tells us how points $X$ move in space as time evolves:

$$v(X(t), t) = \frac{\partial \psi(X(0), t)}{\partial t} = \frac{\mathrm{d}}{\mathrm{d}t} X(t) \ . \tag{7}$$

Note that the velocity field depends on both position $X(t)$ and time $t$.

In flow matching models we learn to approximate the velocity field $v(X(t), t)$ with a neural network $u_\theta(X(t), t)$. This can then be used to define an ordinary differential equation (ODE) whos solution is the approximate flow map $\psi_\theta(X(0), t) = X(t)$:

$$\frac{\mathrm{d}}{\mathrm{d}t} X(t) = u_\theta(X(t), t) \quad \text{with } X(0) \sim \pi_0 \ . \tag{8}$$

We usually solve this ODE numerically using an ODE solver such as Euler or Runge-Kutta methods. With simple Euler method we have:

$$\psi_\theta(X(0), 1) = X(1) = X(0) + \int_0^1 u_\theta(X(t), t)\, \mathrm{d}t \approx X(0) + \sum_{k=0}^{N-1} \frac{1}{N} u_\theta\left(X\left(\frac{k}{N}\right), \frac{k}{N}\right) \ . \tag{9}$$

We can also reverse the process and map points from the data space back to the latent space by solving the ODE backwards in time and getting the reverse flow map $\varphi_\theta(X(1), t) = X(t)$:

$$\frac{\mathrm{d}}{\mathrm{d}t} X(t) = -u_\theta(X(t), t) \quad \text{with } X(1) \sim \pi_1 \ . \tag{10}$$

Using Euler method again we have:

$$\varphi_\theta(X(1), 0) = X(0) = X(1) - \int_0^1 u_\theta(X(t), t)\, \mathrm{d}t \approx X(1) - \sum_{k=0}^{N-1} \frac{1}{N} u_\theta\left(X\left(1 - \frac{k}{N}\right), 1 - \frac{k}{N}\right) \ . \tag{11}$$

## 2.2 Volume mapping around a data point

Consider a sphere in the data space centered at a data point $x^c(1)$ with radius $r$.

$$\mathcal{B}_r(x^c(1)) = \{x(1) \in \mathbb{R}^d : \|x(1) - x^c(1)\|_2 \leq r\} \ . \tag{12}$$

We can map the center point $x^c(1)$ to the latent space using the learned reverse flow map $\varphi_\theta$:

$$x^c(0) = \varphi_\theta(x^c(1), 0) \ . \tag{13}$$

Similary, we can map any point on the sphere $x(1)$ to the corresponding point in the latent space:

$$x(0) = \varphi_\theta(x(1), 0) \ . \tag{14}$$

The distance between the center point and any point on the sphere in the data space is $r$:

$$\|x(1) - x^c(1)\|_2 = r \ . \tag{15}$$

The distance between the the mapped points in the latent space will generally not be equal to $r$ due to the distortion introduced by the flow map:

$$\|x(0) - x^c(0)\|_2 \neq r \ . \tag{16}$$

To understand how the sphere transforms under the flow map, we can sample multiple points on the sphere in the data space, map them to the latent space, and try to recover the polytope in the latent space formed by these mapped points. This, however, is both expensive and inaccurate. Instead, we can approximate the local behavior of the flow map around the center point using its Jacobian matrix.

## 2.3 Linear approximation of the flow map

The Jacobian matrix of the reverse flow map $\varphi_\theta$ at the center point $x^c(1)$ is given by:

$$J_{\varphi_\theta}(x^c(1)) = \frac{\partial \varphi_\theta(x(1), 0)}{\partial x(1)}\Big|_{x(1)=x^c(1)} \ . \tag{17}$$

Using the Jacobian, we can approximate the mapping of points near the center point using a first-order Taylor expansion:

$$\tilde{x}(0) \approx x^c(0) + J_{\varphi_\theta}(x^c(1))(x(1) - x^c(1)), \text{ where } \|x(1) - x^c(1)\|_2 = r \ . \tag{18}$$

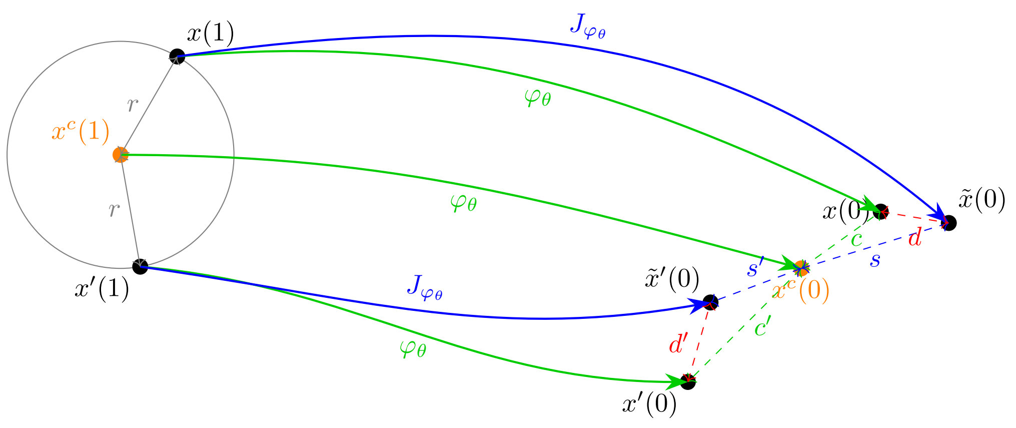

Figure 1: Mapping two points $x(1)$ and $x'(1)$ from sphere in data space to latent space. Green is the reverse flow map $\varphi_\theta$, blue is the linear approximation using the Jacobian $J_{\varphi_\theta}$. Distances to center of the exact mapping are $c, c'$ and of the linear approximation are $s, s'$. Distances between exact and approximated points are $d, d'$.

As shown in Figure 1, distances to the center in the latent space $c, c'$ are distorted by the flow map and no longer equal to $r$. The linear Jacobian mapping suffers from an approximation error and maps to different points than the exact flow map, the distances between the exact and approximated points are $d, d'$. We can compare the exact mapping $x(0)$ obtained from the flow map $\varphi_\theta$ with the linear approximation $\tilde{x}(0)$. For example we can calculate the average distance between the exact and approximated points over multiple samples on the sphere:

$$D_0 = \frac{1}{N}\sum_{i=1}^{N} \|x_i(0) - \tilde{x}_i(0)\|_2 = \frac{1}{N}\sum_{i=1}^{N} d_i \ . \tag{19}$$

Alternatively, we can compare the average distances to the center point in the latent space for both exact and approximated mappings:

$$C_0 = \frac{1}{N}\sum_{i=1}^{N} \|x_i(0) - x^c(0)\|_2 = \frac{1}{N}\sum_{i=1}^{N} c_i \quad \text{and} \quad S_0 = \frac{1}{N}\sum_{i=1}^{N} \|\tilde{x}_i(0) - x^c(0)\|_2 = \frac{1}{N}\sum_{i=1}^{N} s_i \ . \tag{20}$$

### 2.3.1 Comparisons in data space

The above comparison may be difficult to interpret since the distances are measured in the latent space. We can instead map the approximated points $\tilde{x}(0)$ back to the data space using the forward flow map $\psi_\theta$:

$$\tilde{x}(1) = \psi_\theta(\tilde{x}(0), 1) \ . \tag{21}$$

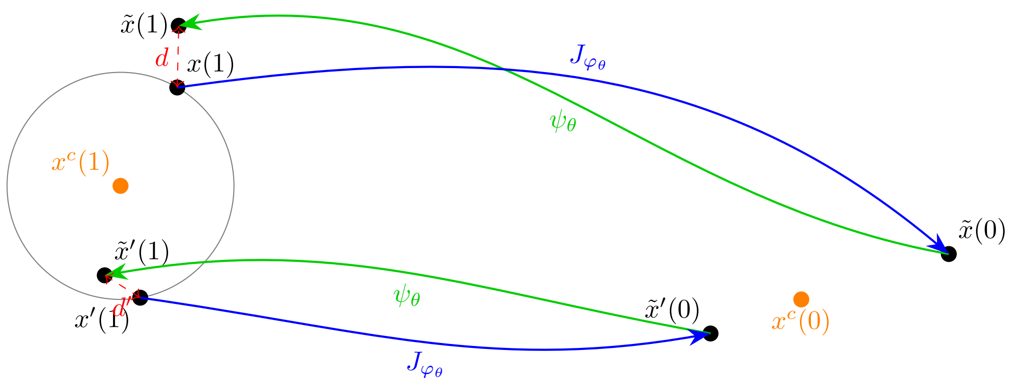

Figure 2: Approximation error of the Jacobian-based linear approximation for two points $x(1)$ and $x'(1)$ on the sphere in data space. We first map the points to the latent space by the linear approximation of the reverse flow map $J_{\varphi_\theta}$ - blue. We then brick these points by the exact flow map $\psi_\theta$ - green. The displacement between the exact points and the approximated points in data space are $d, d'$ shows the approximation error from the linearization.

We can then compare the exact points $x(1)$ with the approximated points $\tilde{x}(1)$ in the data space. For example we can calculate the average distance between the exact and approximated points over multiple samples on the sphere:

$$D_1 = \frac{1}{N}\sum_{i=1}^{N} \|x_i(1) - \tilde{x}_i(1)\|_2 \ . \tag{22}$$

Alternatively, we can compare the average distances to the center point in the data space for both exact and approximated mappings:

$$C_1 = \frac{1}{N}\sum_{i=1}^{N} \|x_i(1) - x^c(1)\|_2 = r \quad \text{and} \quad S_1 = \frac{1}{N}\sum_{i=1}^{N} \|\tilde{x}_i(1) - x^c(1)\|_2 \ . \tag{23}$$

## 2.4 How to get the Jacobian

### 2.4.1 Augmented ODE method

The Jacobian matrix $J_{\varphi_\theta}(X(1))$ for any point $X(1)$ in the data space can be computed by augmenting the initial ODE. Starting from the reverse flow ODE:

$$\frac{\mathrm{d}}{\mathrm{d}t} X(t) = -u_\theta(X(t), t) \quad \text{with } X(1) = x^c(1) \ , \tag{24}$$

the solution $X(0) = \varphi_\theta(X(1), 0)$ gives the mapped point in the latent space.

The Jacobian matrix is the derivative of the reverse flow map with respect to the input point $X(1)$.

$$J_{\varphi_\theta}(X(1)) = \frac{\mathrm{d}\varphi_\theta(X(1), 0)}{\mathrm{d}X(1)} = \frac{\mathrm{d}X(0)}{\mathrm{d}X(1)} \ . \tag{25}$$

We can define the Jacobians for all times $t$ as:

$$J(t) = \frac{\mathrm{d}X(t)}{\mathrm{d}X(1)} = \frac{\mathrm{d}\varphi_\theta(X(1), t)}{\mathrm{d}X(1)} \ . \tag{26}$$

Taking the derivative with respect to time $t$ gives:

$$\frac{\mathrm{d}}{\mathrm{d}t} J(t) = \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\mathrm{d}X(t)}{\mathrm{d}X(1)}\right) = \frac{\mathrm{d}}{\mathrm{d}X(1)}\left(\frac{\mathrm{d}X(t)}{\mathrm{d}t}\right) = \frac{\mathrm{d}}{\mathrm{d}X(1)}\left(-u_\theta(X(t), t)\right) \ . \tag{27}$$

Using the chain rule on the right side we get:

$$\frac{\mathrm{d}}{\mathrm{d}t} J(t) = -\frac{\mathrm{d}u_\theta(X(t), t)}{\mathrm{d}X(t)} \frac{\mathrm{d}X(t)}{\mathrm{d}X(1)} = -\frac{\mathrm{d}u_\theta(X(t), t)}{\mathrm{d}X(t)} J(t) \ . \tag{28}$$

We can therefore consider the augmented ODE system:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} X(t) \\ J(t) \end{bmatrix} = \begin{bmatrix} -u_\theta(X(t), t) \\ -\frac{\mathrm{d}u_\theta(X(t), t)}{\mathrm{d}X(t)} J(t) \end{bmatrix} \quad \text{with} \quad \begin{bmatrix} X(1) \\ J(1) \end{bmatrix} = \begin{bmatrix} x^c(1) \\ I \end{bmatrix} \ , \tag{29}$$

where $I$ is the $d \times d$ identity matrix $I = J(1) = \frac{\mathrm{d}X(1)}{\mathrm{d}X(1)}$.

By solving this augmented ODE backwards in time from $t = 1$ to $t = 0$, we obtain both the mapped point $x^c(0)$ and the Jacobian matrix $J_{\varphi_\theta}(x^c(1))$ at the same time:

$$\begin{bmatrix} x^c(0) \\ J_{\varphi_\theta}(x^c(1)) \end{bmatrix} = \begin{bmatrix} X(0) \\ J(0) \end{bmatrix} \ . \tag{30}$$

Using the Euler method to solve the augmented ODE numerically, we have:

$$\begin{bmatrix} X\left(t - \frac{1}{N}\right) \\ J\left(t - \frac{1}{N}\right) \end{bmatrix} \approx \begin{bmatrix} X(t) \\ J(t) \end{bmatrix} - \frac{1}{N}\begin{bmatrix} u_\theta(X(t), t) \\ \frac{\mathrm{d}u_\theta(X(t), t)}{\mathrm{d}X(t)} J(t) \end{bmatrix} \ . \tag{31}$$

This requires computing the Jacobian of the neural network $u_\theta$ with respect to its input $X(t)$ at each time step. This can be done efficiently using automatic differentiation.

The Jacobian Euler steps can also be written as:

$$J\left(t - \frac{1}{N}\right) \approx \left(I - \frac{1}{N}\frac{\mathrm{d}u_\theta(X(t), t)}{\mathrm{d}X(t)}\right) J(t) \ . \tag{32}$$

The recursions starting from $J(1) = I$ are:

$$J\left(1 - \frac{1}{N}\right) \approx \left(I - \frac{1}{N}\frac{\mathrm{d}u_\theta(X(1), 1)}{\mathrm{d}X(1)}\right) I \ , \tag{33}$$

$$J\left(1 - \frac{2}{N}\right) \approx \left(I - \frac{1}{N}\frac{\mathrm{d}u_\theta(X(1 - \frac{1}{N}), 1 - \frac{1}{N})}{\mathrm{d}X(1 - \frac{1}{N})}\right)\left(I - \frac{1}{N}\frac{\mathrm{d}u_\theta(X(1), 1)}{\mathrm{d}X(1)}\right) I \ , \tag{34}$$

and so on until $t = 0$ so that:

$$J(0) \approx \prod_{k=0}^{N-1}\left(I - \frac{1}{N}\frac{\mathrm{d}u_\theta(X(1 - \frac{k}{N}), 1 - \frac{k}{N})}{\mathrm{d}X(1 - \frac{k}{N})}\right) I \ . \tag{35}$$

### 2.4.2 Getting the points in latent space

Since in the end we only want to get the approximated points in latent space $\tilde{x}(0)$, we can avoid computing the full Jacobian matrix and instead compute the product of the Jacobian with the vector $(x(1) - x^c(1))$ directly (JVP - Jacobian-vector product). Starting from the linear approximation:

$$\tilde{x}(0) \approx x^c(0) + J_{\varphi_\theta}(x^c(1))(x(1) - x^c(1)) \ , \tag{36}$$

we can define:

$$s(t) = J(t)(x(1) - x^c(1)) \ . \tag{37}$$

Taking the derivative with respect to time $t$ gives:

$$\frac{\mathrm{d}}{\mathrm{d}t}s(t) = \frac{\mathrm{d}}{\mathrm{d}t}J(t)(x(1) - x^c(1)) = -\frac{\mathrm{d}u_\theta(X(t), t)}{\mathrm{d}X(t)}J(t)(x(1) - x^c(1)) = -\frac{\mathrm{d}u_\theta(X(t), t)}{\mathrm{d}X(t)}s(t) \ . \tag{38}$$

We can therefore consider the augmented ODE system:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} X(t) \\ s(t) \end{bmatrix} = \begin{bmatrix} -u_\theta(X(t), t) \\ -\frac{\mathrm{d}u_\theta(X(t),t)}{\mathrm{d}X(t)}s(t) \end{bmatrix} \quad \text{with} \quad \begin{bmatrix} X(1) \\ s(1) \end{bmatrix} = \begin{bmatrix} x^c(1) \\ x(1) - x^c(1) \end{bmatrix} \ , \tag{39}$$

where the initial condition for $s(1)$ is simply the vector difference $(x(1) - x^c(1))$. By solving this augmented ODE backwards in time from $t = 1$ to $t = 0$, we obtain both the mapped point $x^c(0)$ and the product of the Jacobian with the vector $(x(1) - x^c(1))$ at the same time:

$$\begin{bmatrix} x^c(0) \\ J_{\varphi_\theta}(x^c(1))(x(1) - x^c(1)) \end{bmatrix} = \begin{bmatrix} X(0) \\ s(0) \end{bmatrix} \ . \tag{40}$$

Using the Euler method to solve the augmented ODE numerically, we have:

$$\begin{bmatrix} X\left(t - \frac{1}{N}\right) \\ s\left(t - \frac{1}{N}\right) \end{bmatrix} \approx \begin{bmatrix} X(t) \\ s(t) \end{bmatrix} - \frac{1}{N}\begin{bmatrix} u_\theta(X(t), t) \\ \frac{\mathrm{d}u_\theta(X(t),t)}{\mathrm{d}X(t)}s(t) \end{bmatrix} \ . \tag{41}$$

This requires computing the Jacobian of the neural network $u_\theta$ with respect to its input $X(t)$ at each time step, but only multiplying it with the vector $s(t)$ instead of the full Jacobian matrix.

The key insight: **don't compute what you don't need.** If you only need the transformation of specific vectors, compute $J \cdot v$ directly rather than computing the full matrix $J$.

The python code to implement this is as follows - says Claude!:

```
# For each point on sphere
for v in displacement_vectors:  # v = x^(1) - x_c^(1)
    # Compute JVP using forward-mode AD
    with torch.autograd.forward_ad.dual_level():
        dual_input = torch.autograd.forward_ad.make_dual(x_c, v)
        dual_output = reverse_flow(dual_input)
        jvp = torch.autograd.forward_ad.unpack_dual(dual_output).tangent
    # Linear approximation: x(0) = x_c(0) + J·v
    x_tilde = x_c_latent + jvp
```