Notizen zu Fermionen im gekrümmten Raum

Julian Köberle

October 16, 2023

Abstract

Das sind Notizen zu meinen zu der Arbeit Fermionen im gekrümmten Raum. Diese Notizen können etwaige Rechtschreibfehler, Grammatikfehler und stilistische Unfeinheiten beinhalten und sind deshalb nicht für die Weitergabe an Dritte bestimmt.

Contents

1 The Problem

Let's suppose one is given a set of data-points $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ on a circle with radius r. Further we will assume that those points are generated by underlying function F, such that, F produces the change of $\mathbf{x}_i$ with respect to some parameter for example time. The exact structure of F is unknown.

The goal is to find a function F' such that F' approximates F , to put it more mathematically:

$$\begin{aligned} \frac{d\mathbf{x}}{dt} &= F(\mathbf{x}(t), t) \approx F'(\mathbf{x}(t), t, \theta), \\ \mathbf{x} &: \mathbb{R} \mapsto \{\mathbf{y} \in \mathbb{R}^2 | \mathbf{y}^\top \mathbf{y} = r^2\}, \end{aligned} \tag{1}$$

where F' is a neural-network with its corresponding network parameters θ as we will see in the following sections, an algorithm to solve (1) is independent of r, so we can set $r = 1$ without loss of generality.

To enforce the S^1 constraint $\forall t$ one makes the ansatz $F' = G\mathbf{x}$ such that G is a antisymmetric matrix.

Proof. We want to preserve the norm of $\mathbf{x}$,

$$\mathbf{x}^\top \mathbf{x} = const \Rightarrow \frac{d\mathbf{x}^\top \mathbf{x}}{dt} = \dot{\mathbf{x}}^\top \mathbf{x} + \mathbf{x}^\top \dot{\mathbf{x}} = 0.$$

Inserting $\frac{d\mathbf{x}}{dt} = G\mathbf{x}$ gives,

$$\mathbf{x}^\top G^\top \mathbf{x} + \mathbf{x}^\top G \mathbf{x} = 0.$$

This needs to hold for all $\mathbf{x}$ $\Rightarrow G^\top = -G$ □

Because in $\mathbb{R}^2$ G has only one[1] free parameter, it is useful to replace $G \to Ag$, where $A = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$ and g is a scalar quantity. Hence equation (1) can be written as,

$$\frac{d\mathbf{x}(t)}{dt} = g(\mathbf{x}(t), t, \theta) A \mathbf{x}(t). \tag{2}$$

2 Forward process

To solve equation (2), an exponential integrator is useful, since it enforces the S^1 constraint for every time step. In most cases it is sufficient to solve equation (1) with a method which is of order one in time, if the time step Δt was chosen sufficiently small. Let $\mathbf{x}_0 = \mathbf{x}(t)$, $\mathbf{x}_1 = \mathbf{x}(t + \Delta t)$ and $g_0 = g(\mathbf{x}(t), t)$,

$$\mathbf{x}_1 = e^{\Delta t A g_0} \mathbf{x}_0, \tag{3}$$

$\Rightarrow$

$$\mathbf{x}_n = e^{\Delta t A g_{n-1}} e^{\Delta t A g_{n-2}} \dots e^{\Delta t A g_0} \mathbf{x}_0, \tag{4}$$

where $\mathbf{x}_n = \mathbf{x}(t + T)$, it is easy to see that (3) implements the S^1 constraint $\forall t$.

[1] in $\mathbb{R}^D$ G has $\frac{(D-1)D}{2}$ free parameters

Proof.

$$\mathbf{x}_1^\top \mathbf{x}_1 = \mathbf{x}_0^\top e^{\Delta t g_0 A^\top} e^{\Delta t A g_0} \mathbf{x}_0 \overset{(A^\top = -A)}{=} e^{-\Delta t g_0 A} e^{\Delta t A g_0} \mathbf{x}_0 \overset{([A,A]=0)}{=} \mathbf{x}_0^\top \mathbf{x}_0.$$

□

In the following and in the sections thereafter, we switch to index notation such that quantities are summed over repeated indices.

$$\mathbf{x} \to x_i^a = \begin{pmatrix} (x_1^1, x_2^1) \\ (x_1^2, x_2^2) \\ \vdots \\ (x_1^{bs}, x_2^{bs}) \end{pmatrix},$$

where bs means batch size. The indices $a, b, c \dots$ are used to notate batch components. Indices like $i, j, k \dots$ are used to notate vector components.

$$A \to A_i{}^j,$$

$$g \to \mathrm{diag}(\underbrace{g^1, g^2, \dots g^{bs}}_{g^a}) := g_b^a.$$

Note, instead of $g_b^a x_i^b$ one could write $(g * x_i)^a$ where $*$ is element-wise multiplication, but we prefer the first one, so it won't be an abuse of notation.

3 Backward process

Let's define a continuous loss function $\mathcal{L}$ which takes two parameters: the real value of x_i^a solved by F and the prediction given by the Neural-ode $\hat{x_i^a}$ solved by F' both evaluated at t.

$$\begin{aligned} \mathcal{L} : \mathbb{R}^{2\times bs} \times \mathbb{R}^{2\times bs} &\mapsto \mathbb{R}, \\ \mathcal{L} &= \mathcal{L}(x_i^a, \hat{x_i^a}), \end{aligned} \tag{5}$$

where again bs means batch-size. In the following we leave $\mathcal{L}$ arbitrary and is only constrained by (5). The overall goal is to find network parameters θ which minimizes the loss. This can be formulated as,

$$\theta_{min} = \mathrm{argmin}_\theta \, \mathcal{L}, \tag{6}$$

since $\mathcal{L}$ is at least in C^1, one needs to find the solution of $d\mathcal{L}/d\theta = 0$, this is mostly done with Stochastic Gradient Descent (SGD) or Adam [**ADAM**].

From eq. (4) one can easily see that back propagation is very memory and time-consuming, since the whole computational graph needs to be stored. A more sophisticated method is used to get $d\mathcal{L}/d\theta$. We introduce the adjoint sensitivity method, whose equation can be derived by introducing Lagrange multipliers. The benefit of this method is that only a fraction of the graph needs to be stored. In the following time-dependencies are suppressed and x_i^a is used instead of $\hat{x_i^a}$ to not overload notation. This is motivated by the fact that x_i^a is not dependent θ but so is $\hat{x_i^a}$. Since we are looking for $d\mathcal{L}/d\theta$ derivatives of x_i^a do not come into play, so the switch of notation is justified.

Let's define a scalar quantity,

$$K = \int_0^T dt\ b_a^i(\dot{x}_i^a - g_b^a A_i{}^j x_j^b). \tag{7}$$

$b_a^i = b_a^i(t)$ are the Lagrange multipliers. The quantity K is always zero and therefore one can write:

$$\frac{d\mathcal{L}}{d\theta_l} = \frac{d\mathcal{L}}{d\theta_l} + \frac{dK}{d\theta_l}. \tag{8}$$

By integration by parts, K can be written as,

$$K = b_a^i x_i^a|_0^T - \int_0^T dt\ \left(\dot{b}_a^i x_i^a + g_b^a\ b_a^i A_i{}^j x_j^b\right).$$

Taking the derivative one gets,

$$\frac{dK}{d\theta_l} = b_a^i \frac{dx_i^a}{d\theta_l}|_0^T - \int_0^T dt \left(\dot{b}_a^i \frac{dx_i^a}{d\theta_l} + \frac{dg_b^a}{d\theta_l}\ b_a^i A_i{}^j x_j^b + g_b^a\ b_a^i A_i{}^j \frac{dx_j^b}{d\theta_l}\right).$$

It is worth noting that $x_j^a(t=0)$ does not depend on θ, only $x_j^a(t=T)$ does.

Inserting the following relation

$$\frac{dg_b^a}{d\theta_l} = \frac{\partial g_b^a}{\partial x_i^c}\frac{dx_i^c}{d\theta_l} + \frac{\partial g_b^a}{\partial \theta_l} \tag{9}$$

one gets,

$$\frac{dK}{d\theta_l} = b_a^i \frac{dx_i^a}{d\theta_l}|_0^T - \int_0^T dt\ \left(\dot{b}_a^i \frac{dx_i^a}{d\theta_l} + \left(\frac{\partial g_b^a}{\partial x_k^c}\frac{dx_k^c}{d\theta_l} + \frac{\partial g_b^a}{\partial \theta_l}\right)\ b_a^i A_i{}^j x_j^b + g_b^a\ b_a^i A_i{}^j \frac{dx_j^b}{d\theta_l}\right) \tag{10}$$

collecting terms which depend on $\frac{dx_i^a}{d\theta_l}$, equation (10) reads

$$\frac{dK}{d\theta_l} = b_a^i \frac{dx_i^a}{d\theta_l}|_0^T - \int_0^T dt\ \left(\left(\dot{b}_a^i + g_a^c\ b_c^n A_n{}^i + \frac{\partial g_b^d}{\partial x_i^a} b_d^m A_m{}^j x_j^b\right)\frac{dx_i^a}{d\theta_l} + \frac{\partial g_b^a}{\partial \theta_l}\ b_a^i A_i{}^j x_j^b\right)$$

using equation (8) and

$$\frac{d\mathcal{L}}{d\theta_l} = \frac{\partial \mathcal{L}}{\partial x_i^a}\frac{dx_i^a}{d\theta_l}$$

by using $b_a^i(0) = 0$ one obtains,

$$\begin{aligned}\frac{d\mathcal{L}}{d\theta_l}|^T &= \left(\frac{\partial \mathcal{L}}{\partial x_i^a} + b_a^i\right)\frac{dx_i^a}{d\theta_l}|^T - \\ &\int_0^T dt\ (\dot{b}_a^i + g_a^c\ b_c^n A_n{}^i + \frac{\partial g_b^d}{\partial x_i^a} b_d^m A_m{}^j x_j^b)\frac{dx_i^a}{d\theta_l} + \frac{\partial g_b^a}{\partial \theta_l} b_a^i A_i{}^j x_j^b\end{aligned} \tag{11}$$

where $|^T$ means evaluated at time $t = T$. Since eq. (11) needs to hold for all $dx_i^a/d\theta_l$ follows,

$$\frac{\partial \mathcal{L}}{\partial x_i^a}|^T = -b_a^i|^T, \tag{12}$$

$$\dot{b}_a^i + g_a^c\ b_c^n A_n{}^i + \frac{\partial g_b^d}{\partial x_i^a} b_d^m A_m{}^j x_j^b = 0, \tag{13}$$

$$\frac{d\mathcal{L}}{d\theta_l}|^T = -\int_0^T dt\ \frac{\partial g_b^a}{\partial \theta_l} b_a^i A_i{}^j x_j^b, \tag{14}$$

Multiplying eq. (13) with $A_i{}^j x_j^b$ follows,

$$\dot{b}_a^i A_i{}^j x_j^b - g_a^c\ b_c^j x_j^b + \frac{\partial g_e^d}{\partial x_i^a} b_d^m A_m{}^j x_j^e A_i{}^k x_k^b = 0.$$

Using the substitution $P_a{}^b(x(t = T)) := b_a^i A_i{}^j x_j^b$ this equation can be written as,

$$\frac{dP_a{}^b}{dt} + \frac{\partial g_e^d}{\partial x_i^a} P_d{}^e A_i{}^k x_k^b = 0.$$

Since g_b^a has only diagonal entries, the expression $\partial g_b^a/\partial\theta_l\ b_a^i A_i{}^j x_j^b$ in eq. (14) reduces to $\partial g^a/\partial\theta_l\ P_a^{\text{diag}}$, where P_a^{diag} are the diagonal components of P_b^a. and g^a are the diagonal compoents of g_b^a.

Proof. by introducing

$$\delta_{ba}^c = \begin{cases} 1 & a = b = c \\ 0 & \text{else} \end{cases}$$

P^{diag} can be written in index notation as $P_c{}^b \delta_{ba}^c$, likewise $g^a = g_c^b \delta_b^{ac}$
$g_c^b = \text{diag}(g^a)_c^b = g^a \delta_{ac}^b \ \Rightarrow g_c^b P_b{}^c = g^a \delta_{ac}^b P_b{}^c = g^a P_a^{\text{diag}}$

□

Therefore, we are only interested in the diagonal entries of P, hence equation (13) can be written as,

$$\frac{dP_e^{\text{diag}}}{dt} + \frac{\partial g^d}{\partial x_i^a} P_d^{\text{diag}} A_i{}^k x_k^b \delta_{be}^a = 0, \tag{15}$$

by solving eq. (15) numerically, one needs to make sure that the indices are contracted efficient, since δ_{ba}^c carries a lot of entries, but most of them are zero anyway. With the above defined substitution equation (14) reads,

$$\frac{d\mathcal{L}}{d\theta_l}|^T = -\int_0^T dt \ \frac{\partial g^a}{\partial \theta_l} P_a^{\text{diag}} \tag{16}$$

4 Data Preprocessing

Figure 1: The solution to equation (17) is shown in blue, from this the Cartesian x and y components are extracted, shown in green and orange

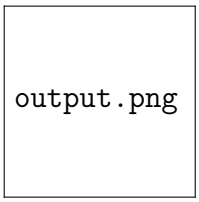

Figure 2: Randomly selected data points (x_i, y_i) are shown in blue and green. Those points define the selection of the final point $(x_f, y_f) = ((x(t_i+t_p), y(t_i+t_p))$ shown in orange and red, where t_p is the propagation time.

To test the above algorithm, a function F was chosen such that equation (1) does **not** describe a linear differential equation, otherwise the network would drop to a single layer. Further F was chosen to be smooth and not too complicated in an informal sense, additionally it is worth assuming F is bounded, since going from polar coordinates to Cartesian coordinates

discontinuities could occur. An obvious chose could be $F(\phi) = \sin(\phi)$ in polar coordinates. This enforces the S^1 constraint. From the solution of

$$\frac{d\phi}{dt} = \sin(\phi), \tag{17}$$

$x(t) = \arccos(\phi(t))$ and $y(t) = \arcsin(\phi(t))$ components are extracted. A propagation-time is defined which correlates the starting point $\mathbf{x}_i$ to the final point $\mathbf{x}_f$, as shown in figure (2).

$$(x_i^a)_{\text{init}} = \begin{pmatrix} x_0 & y_0 \\ x_1 & y_2 \\ \vdots & \vdots \\ x_{bs} & y_{bs} \end{pmatrix}_{\text{init}} \xrightarrow{F'} (\hat{x}_i^a)_{\text{final}} = \begin{pmatrix} \hat{x}_0 & \hat{y}_0 \\ \hat{x}_1 & \hat{y}_2 \\ \vdots & \vdots \\ \hat{x}_{bs} & \hat{y}_{bs} \end{pmatrix}_{\text{final}} \approx \begin{pmatrix} x_0 & y_0 \\ x_1 & y_2 \\ \vdots & \vdots \\ x_{bs} & y_{bs} \end{pmatrix}_{\text{final}}$$

5 Experiments

The experiments shows that the adjoint sensitivity method solves the S^1 toy problem perfectly, with known and unknown starting conditions. To even get better results one can increase the batch-size, the sample-size, in other words increase the data set, increase the network parameters and increase the number of training epochs. The loss function was chosen to be the usual L_2-norm averaged over the whole batch. A very interesting hyper-parameter is the propagation-time, which specifies indirectly which method to choose. For a data set where the propagation-time needs to be relatively large, the adjoint sensitivity method dominates over common back propagation method, since the computational-graph would become insanely large. For short very small propagation-time the common back propagation method is to be preferred, since the adjoint-sensitivity method carries a big overhead. In this toy problem the propagation time could be chosen to be relatively small, therefore there is no real speedup to be expected.

6 Generalization

Equation (1) can be easily generalized to $\mathbb{R}^D$ with $D \in \mathbb{N}$. Such that

$$\frac{dx_a^i}{dt} = x_b^j G_{aj}^{bi}$$

Where G_{aj}^{bi} is diagonal in a, b and antisymmetric in i, j to enforce the conserved norm on $\mathbf{x}_a^i$. $a, b = 1, \ldots, bs$ and $i, j = 1, 2 \ldots, D$.

Figure 3: Prediction versus the real values of the x and y components. The prediction was made on the same starting condition $\phi(t=0) = \pi/2$ which was chosen on the data-set. Calculations show that $x^2 + y^2 = 1$ is conserved for all time.

Figure 4: Prediction versus the real values of the x and y components. The prediction was made on a different starting condition $\phi(t=0) = \pi/4$ and different radius $R = 2$. Calculations show that $x^2 + y^2 = 4$ is conserved for all time

$$G_{aj}^{bi} = \begin{pmatrix} G_{1j}^{1i} & 0 & \dots & 0 \\ 0 & G_{2j}^{2i} & \dots & 0 \\ \vdots & \vdots & & \\ 0 & 0 & \dots & G_{bs\ j}^{bs\ i} \end{pmatrix}$$

G_{aj}^{bi} is as written here is an enormously large object, but has only $bs \times (D-1)D/2$ free parameters. Since G_{aj}^{bi} is supposed to be our neural network f. f has therefore $(D-1)D/2$ output neurons with a corresponding output shape $[bs, (D-1)D/2]$. Defining the mapping $M : \mathbb{R}^{bs\times(D-1)D/2} \mapsto \mathbb{R}^{bs\times bs\times D\times D}$, such that M is surjective helps to derive the equations of the adjoint sensitivity method, which are analogue to those defined above in the S^1, but instead of gA using G.

Another generalization to $SU(N)$ can be achieved by writing,

$$\frac{dU_a^{ik}}{dt} = i\ U_b^{ij} G_{aj}^{bk} \tag{18}$$

where $(U_a) \in SU(N) \to U_a \in \mathbb{C}^{N\times N}$ in the fundamental representation, and $(G_a) \in \mathfrak{su}(N), G_{bj}^{ai} := diag((G_1)_j^i, (G_2)_j^i, \ldots, (G_{bs})_j^i) | G_a \in \mathbb{C}^{N\times N}$, where we have used physicist convention where the generators are hermitian. Since G has $N^2 - 1$ free parameters it is convenient to implement an bijective mapping $M : \mathbb{R}^{bs\times N^2-1} \mapsto \mathbb{C}^{bs\times bs\times N\times N}$ similar to the $\mathbb{R}^D$ case above.

Proof. In the following indices i, j are suppressed. Without loss of generality, we concentrate on batch-size of order 1, therefore indices a, b can also be neglected. To see $U(t)$ stays in $SU(N)$ one can solve the above differential equation (18) and gets,

$$U(t) = \underbrace{U(0)}_{\in SU(N)} \underbrace{\mathcal{T}\exp(i \int_0^t dt' G(U(t'), t))}_{\in SU(N)},$$

follows $U(t) \in SU(N)\ \forall t$ where $\mathcal{T}$ is the time-ordering-operator. □

7 Conclusion

Although there was no real speedup in this S^1 toy problem it describes a good foundation to general cases like $SU(N)$ which is a crucial part in high energy physics to calculate Wilson-flows.