Neural Networks and other related topics

Marcelo Oyarzo

Contents

1 Definitions

A neural network in a very loose sense is a set of artificial neurons interconnected between each other. It receive an input –usually a set of numbers– and returns an output that is another set of numbers. An artificial neuron is a simple object defined by its number of inputs and a set of parameters and always returns a single output. There are many types of artificial neurons, the most popular is called sigmoid neuron, but to introduce the concept we define first a perceptron neuron.

A perceptron takes several *binary* inputs, let's say that takes $n \in \mathbb{N}$ inputs, $\vec{x} = (x_1, x_2, \ldots, x_n)$ with $x_i \in \{0,1\}, \forall i = 1, \ldots, n$, and returns a single output. To determine the output let us assign to each input a real number called the *weight*, all the weights of the neuron can be arranged in a vector $\vec{w} = (w_1, w_2, \ldots, w_n)$ with $\vec{w} \in \mathbb{R}^n$. Finally, we define a real number $b \in \mathbb{R}$, called the bias, that is another parameters of the neuron. We define the output of the perceptron neuron as follows

$$\text{output} = \begin{cases} 0 & \text{if } \vec{w} \cdot \vec{x} + b \leq 0 \\ 1 & \text{if } \vec{w} \cdot \vec{x} + b > 0 \end{cases} \tag{1.1}$$

In this notation a perceptron neuron with bias and weight b and $\vec{w}$ is a usual function $N_{b,\vec{w}} : \{0,1\}^n \to \{0,1\}$. We denote a neuron as a node with n lines entering to the node from the left and one output denote by a line going out of the node to the right.

In summary, to define a perceptron neuron you need to say how many input it has, what is the weight of each input and what is its bias $b \in \mathbb{R}$. You can think the weight w_i as how much the neuron like the input x_i, since $x_i \in \{0,1\}$, increasing w_i will lead a bigger number $\vec{w} \cdot \vec{x}$ which increase the possibility to cross the threshold and fire the neuron (return a 1).

A perceptron neuron essentially takes the weighted mean value of the inputs and depending on its result returns a 0 or 1. In school, professors use perceptron neurons every time when they need to decide whether you approve the course or not. So, it allows us to make decisions given certain evidence encoded in the vector $\vec{x}$. In the example, $\vec{x}$ are the grades that you got, $\vec{w}$ are the weight of each grade and the bias is related to the minimum grade required to approve the course.

One can try to concatenate perceptron neurons making the output of certain neurons as the input of other neurons, namely construct a neural network, and adjust the weights and bias such that the neural network make complicated decisions given certain inputs. It is easy to see that using only perceptron neurons the inputs will be an element of $\{0,1\}^{n_{\text{input}}}$ for some integer number n_{input} and the output will be an element of $\{0,1\}^{n_{\text{output}}}$ which are discrete sets. Then it is impossible to move the weights and the bias (that are real numbers) and make the outputs move in a continuous way. To circumvent this problem we define sigmoid neurons.

Definition 1. A sigmoid neuron is a function with n inputs $\vec{x} = (x_1, \ldots, x_n)$ with $x_i \in [0,1]$, $\forall i = 1, \ldots, n$ defined by its bias $b \in \mathbb{R}$ and n weights $\vec{w} = (w_1, \ldots, w_n)$ with $\vec{w} \in \mathbb{R}^n$. The output is defined in terms of the sigmoid function $\sigma(\vec{w} \cdot \vec{x} + b)$ that for now on we consider

$$\sigma(z) \equiv \frac{1}{1+e^{-z}}\,. \tag{1.2}$$

Remark 1. Sigmoid neurons with the sigmoid function defined as above behave like a perceptron neuron for very large values of $\vec{w} \cdot x + b$ for both positive and negative values

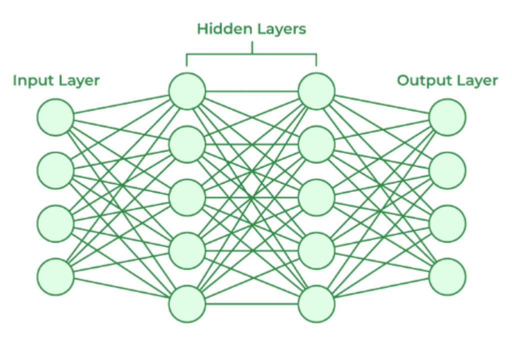
Remark 2. A sigmoid function equal to a step function will give back a perceptron neuron.

Remark 3. The fact that the output is a smooth function of $\vec{w}$ and b implies that the variation of the output under a small variation of the weights and the bias

$$\delta\texttt{output} = \sum_i \frac{\partial \sigma}{\partial w_i}\delta w_i + \frac{\partial \sigma}{\partial b}\delta b\,. \tag{1.3}$$

1.1 Neural network definition

We already introduce a single sigmoid neuron, now we introduce a neural network in a more precise way. A neural network is composed by a input layer, hidden layers and an output layer. For instance,



correspond to a neural network with 2 layers in the hidden layers. Usually, the design of the input layer and the output layer is straightforward, as it is adapted to the type of data that we want to process. The design of the hidden layer is more involved as there is no recipe to construct them, its design depends case by case.

Definition 2. A feedforward neural network are define as neural networks where the output from one layer is the input of the next layer.

All of the neural networks that we will consider are feedforward neural networks. However, one can define other type of neural networks that includes "loops". This means that there are some output of a layer j which is also an input of a layer i with $i < j$. At first sight it looks difficult to make sense of this because the argument of the sigmoid function in the layer i has a sigmoid function whose argument depends on its output. The neural networks of these type are called *recurrent neural networks*. We will not consider them here.

Let us concentrate on feedforward neural networks and set the notation for the different objects entering in the game.

Notation for feedforward neural network: Let us consider an input layer with n_1 inputs. Then, the input values can be arranged as a vector $a^1 \in [0,1]^{n_1}$ where the subscript indicates the layer. The second layer has n_2 sigmoid neurons and we denote their bias by $b^2 \in \mathbb{R}^{n_2}$, and the weights can be arrange in a matrix $W^{2,1} \in \mathbb{R}^{n_2} \otimes \mathbb{R}^{n_1}$. The output of the second layer is a vector $a^2 \in [0,1]^{n_2}$ given by

$$a^2 = \sigma(W^{2,1} a^1 + b^2)\,, \tag{1.4}$$

where σ the sigmoid function defined in (1.2) acting on the components of the vector (element-wise action). In general, the layer j has n_j sigmoid neurons and its output is

$$a^j = \sigma(W^{j,j-1} a^{j-1} + b^j)\,, \tag{1.5}$$

where

$$W^{j,j-1} \in \mathbb{R}^{n_j} \otimes \mathbb{R}^{n_{j-1}}\,, \qquad\qquad b^j \in \mathbb{R}^{n_j}\,. \tag{1.6}$$

If the neural network has L layers, then we denote the set of all weights and bias as

$$\begin{aligned} W &= \{W^{j,j-1}\}_{j=2}^{L}\,, \\ b &= \{b^j\}_{j=2}^{L}\,, \end{aligned}$$

Remark 4. The layer j in a feedforward neural network is defined by the weights matrix $W^{j,j-1} \in \mathbb{R}^{n_j} \otimes \mathbb{R}^{n_{j-1}}$ and the bias vector $b^j \in \mathbb{R}^{n_j}$.

1.2 Training the neural network

The idea is that we have our feedforward neural network with n_1 inputs and n_L outputs. Let us consider that we have a set of inputs $\mathcal{T}$ and its associated desired output through the function

$y : \mathcal{T} \to [0,1]^{n_N}$. We called $(\mathcal{T}, y)$ the *training set*[1]. The elements $x \in \mathcal{T}$ are $x \in [0,1]^{n_1}$ and $y(x) \in [0,1]^{n_N}$. We want the neural network to reproduce the results $y(x)$ for the input x. This can be achieved by tuning the weights W and the bias b of our neural network, let us define the output of our neural network for the input x by $a_{W,b}(x)$, where we emphasize that it depends on the weights and biases. Let us define the so called *cost function* as

$$C(W,b) \equiv \frac{1}{2|\mathcal{T}|} \sum_{x \in \mathcal{T}} \|y(x) - a_{W,b}(x)\|^2 , \tag{1.7}$$

where $|\mathcal{T}|$ denotes the number of elements of the training set. We want to find W, b such that this function is small.

To train the neural network we will use a method called gradient descent, that is essentially using calculus to minimize the above function. To simplify the computations let $v = (W, b)$ the set of all the parameters. The variation of the function C defined in (1.7) under a small variation $\delta v = (\delta W, \delta b)$ is given by

$$\delta C = \nabla C \cdot \delta v \,. \tag{1.8}$$

We want $\delta C < 0$ namely $C|_{v+\delta v} < C|_v$. In particular the choice

$$\delta v = -\eta \nabla C \,, \tag{1.9}$$

leads $\delta C = -\eta \|\nabla C\|^2 < 0$ for some $\eta > 0$. The parameter η is called *learning rate*. We will use this choice of δv to make the variation of the update of the parameters on each iteration. This rule defines the gradient descent algorithm, namely in each iteration we update the parameters as

$$v' = v - \eta \nabla C \,, \tag{1.10}$$

for a given learning rate η.

Stochastic gradient descent Note that usually the training set has a large number of elements and the computation of the gradient can take a lot of time. Let us simplify again the notation and define $C_x = \frac{1}{2}|y(x) - a_v(x)|^2$, then the cost function is $C = \frac{1}{|\mathcal{T}|}\sum_{x \in \mathcal{T}} C_x$. The gradient along v is

$$\nabla C = \frac{1}{|\mathcal{T}|} \sum_{x \in \mathcal{T}} \nabla C_x \,. \tag{1.11}$$

[1]Usual we abuse on the notation and we called $\mathcal{T}$ the training set. In the sense that it is not an arbitrary set, it is the set which is the pre-image of the function y.

This means that we need to compute a number $|\mathcal{T}|$ of gradients and then compute the average, which is expensive for a large set $\mathcal{T}$. To make things faster we pick a random subset $\mathfrak{t} \subset \mathcal{T}$ called mini-batch. Then we compute the gradient in this set

$$\nabla C \approx \frac{1}{|\mathfrak{t}|} \sum_{x \in \mathfrak{t}} \nabla C_x \,. \tag{1.12}$$

It so happen that the gradient in the subset is a good approximation of the gradient in full set. In this stochastic approach the update rule for the variables is

$$w_k \to w'_k = w_k - \frac{\eta}{|\mathfrak{t}|} \sum_{x \in \mathfrak{t}} \frac{\partial C_x}{\partial w_k} \,, \tag{1.13}$$

$$b_k \to b'_k = b_k - \frac{\eta}{|\mathfrak{t}|} \sum_{x \in \mathfrak{t}} \frac{\partial C_x}{\partial b_k} \,. \tag{1.14}$$

Then, we pick out another randomly chosen mini-batch and train with those. Once we exhausted the training inputs, we say that we have completed an epoch of training. After that we start over with a new training epoch by picking another set of mini-batch. In practice we reshuffle the set $\mathcal{T}$ and pick a partition of it $\bigcup_{i=1}^{m} \mathfrak{t}_i = \mathcal{T}$.

2 Learning algorithms

2.1 Initial comments: code in Python

Here I want to point out some functionalities of python that are useful. The first is the concept of *class*. In the book that we are following the author write the code to construct a feedforward neural network using a class. The first part of the code allow us to initialize the neural network by using the class functionality of python.

Let us give a simple example to show how class works. We define a class called Dog, which allow to define some functions and properties of Dog given certain information.

```
class Dog(object):
    def __init__(self,name,age):
        self.name = name
        self.age = age

    def speak(self):
        return f"{self.name} dice hola. Io sono {self.age} years old"
```

This code allow us to define a class, namely a dog, giving its name and age:

```
mi_perro = Dog("Copo",2)
```

Then by doing mi_perro.name it returns the variable name, and mi_perro.speak() execute the function that was defined inside the class. This is the essence of class, you give certain information and it defined all that you want at once. Now we proceed to breakdown and explain the code to initialize a neural network with the code of the book. Here is the code:

```
class Network(object):
    def __init__(self, sizes):
        self.num_layers = len(sizes)
        self.sizes = sizes
        self.biases = [np.random.randn(y, 1) for y in sizes[1:]]
        self.weights = [np.random.randn(y, x)
                        for x, y in zip(sizes[:-1], sizes[1:])]
```

In this case the class is called **Network** and to initialize it you need to give **sizes**, which is the size of each layer of the, in the notation that we introduce we need to give the list $[n_1, n_2, \ldots, n_L]$. Then, the variable **num_layers** is just the length of the list **sizes**, in this case is L. Then, declare a variable **sizes** to save it inside the class. In the last two lines we give a starting point for the weights and the bias by picking random real numbers. This is done by using the library numpy that was imported as **np** that inside has a package called **random** which has a function called **randn**. It takes a certain number of inputs, let's say **np.random.randn**(**a1**, **a2**, **a3**), and returns an array (**numpy.ndarray**) of that shape filled with random float numbers. In our example the array has dimensions $\mathtt{a1} \times \mathtt{a2} \times \mathtt{a3}$, and it can be controlled by asking **shape**() of the output.

For the biases we see that the line

```
self.biases = [np.random.randn(y, 1) for y in sizes[1:]]
```

creates a list [] which elements are **np.random.randn**(**y**, 1), namely, numpy arrays of $\mathtt{y} \times \mathtt{1}$ dimensions (a vector of length equal to the number of nodes), where **y** take values on **sizes** but starting from the second element to the rest because the input layer does not carry any bias. This implemented by **sizes**[1 :], recalling that python start counting from 0.

Regarding the initialization of the weights the idea is the same. The line is

```
self.weights = [np.random.randn(y, x)
                for x, y in zip(sizes[:-1], sizes[1:])]
```

weights is a list which elements are numpy array of $\mathtt{y} \times \mathtt{x}$ where **x**, **y** take values in **zip**(**sizes**[: −1], **sizes**[1 :]). Let us analyze the last command: **sizes**[: −1] is a list starting from the first element of the list **sizes** until the second to last element, the element $L-1$ in our notation, and in python we can use simply -1 to command that. More precisely

$$\mathtt{sizes}[:-1] = [n_1, n_2, \ldots, n_{L-2}, n_{L-1}]\,, \tag{2.1}$$

$$\mathtt{sizes}[1:] = [n_2, n_3, \ldots, n_{L-1}, n_L]\,. \tag{2.2}$$

Then, **zip** is a *lazy* tuples generator. Namely, it start pairing the elements of both lists. If you run the line **zip**(**sizes**[: −1], **sizes**[1 :]) its output should be something like the position of the zip in the memory, but this is enough to run the **for** cycle in the code. If you want to see the list do[2]

$$\texttt{list}(\texttt{zip}(\texttt{sizes}[: -1], \texttt{sizes}[1 :])) = [(n_1, n_2), (n_2, n_3), \ldots, (n_{L-2}, n_{L-1}), (n_{L-1}, n_L)] \,. \qquad (2.3)$$

Now it is clear why in the code we run on **x**, **y** but the matrices are generated as **np.random.randn**(**y**, **x**). This is because the matrix in the layer j must belong to $\mathbb{R}^{n_j} \otimes \mathbb{R}^{n_{j-1}}$, as we defined them.

Now we add a new function to the Class definition that allow to implement the feedforward operation. The code is

```
    def feedforward(self, a):
        for b, w in zip(self.biases, self.weights):
            a = sigmoid(np.dot(w, a)+b)
        return a
```

It is super economic and does what we want. First, **zip** create a list which elements are tuples of bias and weight of the layer. The number in the list is associated with the layer of the network. Observe that **a** is the input of the function and it enters in the first iteration of the **for** cycle. Then, it redefines **a** as the output and use it as an input for the second iteration in the **for** cycle, and so on. The result is the output of the last iteration. **sigmoid** is the sigmoid function that must be defined:

```
def sigmoid(z):
    return 1.0/(1.0+np.exp(-z))
```

[2]In Mathematica that list is obtained by considering **Transpose**[{**sizes**[: −1], **sizes**[1 :]}] where the list inside must be written appropriately in Mathematica language, but you got the idea.

2.2 Backpropagation learning

There is another piece of optimization in the learning process that is important to implement. It is called backpropagation learning. The idea is asking the question how the cost function changes under a change of a certain weight or bias at certain layer. The name backpropagation is related to the way in which the chain rule acts on a multi-variable function.

First of all let us set the notation of this section. We will use explicit index notation, therefore in a layer l we denote the output, the weight matrix and the bias vector by $a^l = (a^l_i)$, $w^{l,l-1} = (w^{l,l-1}_{ij})$ and $b^l = (b^l_i)$, respectively. The feedforward algorithm says that the output a^l of the layer l is given by

$$a^l = \sigma(z^l)\,, \qquad z^l \equiv w^{l,l-1}a^{l-1} + b^l\,, \tag{2.4}$$

where z^l is a vector of n_l components and σ acts elements-wise on the vector leading to a^l being a vector of n_l components. We consider a network with L layers. The cost function is defined by

$$C = \frac{1}{|\mathfrak{t}|}\sum_{x\in\mathfrak{t}} C_x\,, \qquad C_x \equiv \frac{1}{2}|y(x) - a^L(x)|^2\,, \tag{2.5}$$

for a mini-batch $\mathfrak{t} \subset \mathcal{T}$, inside the full training set. We will do the analysis on the function C_x with fixed x. Let us start computing the variation of the cost function from back to front. Namely, let's ask how it changes when we change the bias and the weight of the layer L.

$$\begin{tikzcd}
& \vdots \arrow[d, "\sigma"] & \\
w^{l-1,l-2} \arrow[dr] & a^{l-2} \arrow[d] & b^{l-1} \arrow[dl] \\
& z^{l-1} \arrow[d, "\sigma"] & \\
w^{l,l-1} \arrow[dr] & a^{l-1} \arrow[d] & b^{l} \arrow[dl] \\
& z^{l} \arrow[d, "\sigma"] & \\
w^{l+1,l} \arrow[dr] & a^{l} \arrow[d] & b^{l+1} \arrow[dl] \\
& z^{l+1} & \\
& \vdots \arrow[d, "\sigma"] & \\
w^{L-1,L-2} \arrow[dr] & a^{L-2} \arrow[d] & b^{L-1} \arrow[dl] \\
& z^{L-1} \arrow[d, "\sigma"] & \\
w^{L,L-1} \arrow[dr] & a^{L-1} \arrow[d] & b^{L} \arrow[dl] \\
& z^{L} \arrow[d, "\sigma"] & \\
& a^{L} &
\end{tikzcd}$$

Figure 1: Chain dependency of the network.

Definition 3. We introduce the *variation function* defined as the derivative of the function C_x with respect to z^l_j (that is the function which is almost the output of the layer l)

$$\Delta^l_j \equiv \frac{\partial C_x}{\partial z^l_j}\,. \tag{2.6}$$

Note that if we look at the last layer by definition we find

$$\Delta^L_j = \frac{\partial C_x}{\partial z^L_j} = \frac{\partial C_x}{\partial a^L_j}\frac{\partial a^L_j}{\partial z^L_j} = \frac{\partial C_x}{\partial a^L_j}\sigma'(z^L_j) \equiv \nabla_a C_x \odot \sigma'(z^L)\,. \tag{2.7}$$

note that we did not sum-up in j in the second equality as a_j^L is the only function of z_j^L, because the function σ acts element-wise. In the last line we introduce the Haddamard dot product, which take two vectors of the same size and returns another vector whose components are the product of the corresponding components of the vectors.

The variation function in an arbitrary layer l can be computed by using the chain rule where the dependency of the functions is summarize in the Figure 1.

$$\Delta_j^l = \frac{\partial C_x}{\partial z_j^l} = \frac{\partial C_x}{\partial a_j^l}\frac{\partial a_j^l}{\partial z_j^l} = \sum_k \frac{\partial C_x}{\partial z_k^{l+1}}\frac{\partial z_k^{l+1}}{\partial a_j^l}\frac{\partial a_j^l}{\partial z_j^l} = \sum_k \Delta_k^{l+1}\frac{\partial z_k^{l+1}}{\partial a_j^l}\frac{\partial a_j^l}{\partial z_j^l}\,. \tag{2.8}$$

Note that

$$\frac{\partial z_k^{l+1}}{\partial a_j^l} = \frac{\partial}{\partial a_j^l}\left(\sum_p w_{kp}^{l+1,l} a_p^l + b_k^{l+1}\right) = w_{kj}^{l+1,l}\,.$$

hence we find that

$$\Delta_j^l = \sum_k \Delta_k^{l+1} w_{kj}^{l+1,l}\sigma'(z_j^l)\,, \tag{2.9}$$

$$\Delta^l \equiv (\Delta_j^l) \equiv \left((w^{l+1,l})^T \Delta^{l+1}\right) \odot \sigma'(z^l)\,, \tag{2.10}$$

in the last line we define it in the index-free notation.

Let us compute now the derivative with respect to the bias in a layer l of the cost function

$$\frac{\partial C_x}{\partial b_j^l} = \frac{\partial C_x}{\partial z_j^l}\frac{\partial z_j^l}{\partial b_j^l} = \Delta_j^l \frac{\partial}{\partial b_j^l}\left(\sum_k w_{jk}^{l,l-1} a_k^{l-1} + b_j^l\right) = \Delta_j^l\,. \tag{2.11}$$

While the derivative with respect to the weights matrix in the layer l is given by

$$\frac{\partial C_x}{\partial w_{ij}^{l,l-1}} = \frac{\partial C_x}{\partial z_i^l}\frac{\partial z_i^l}{\partial w_{ij}^{l,l-1}} = \Delta_i^l \frac{\partial}{\partial w_{ij}^{l,l-1}}\left(\sum_k w_{ik}^{l,l-1} a_k^{l-1} + b_i^l\right) = \Delta_i^l \sum_k \delta_{kj} a_k^{l-1} = \Delta_i^l a_j^{l-1}\,. \tag{2.12}$$

In the first equality we use the fact that z_i^l depends on $w_{ij}^{l,l-1}$, for instance z_{i+1}^l does not depend on $w_{ij}^{l,l-1}$, then there is sum there.

Summarizing, we have that by introducing the variation matrix $\Delta_i^l \equiv \partial C_x/\partial z_i^l$ that can compute in any layer in a recursive manner in the of the *next* variation matrix

$$\Delta_j^l = \sum_k \Delta_k^{l+1} w_{kj}^{l+1,l}\sigma'(z_j^l) \equiv \left((w^{l+1,l})^T \Delta^{l+1}\right) \odot \sigma'(z^l)\,. \tag{2.13}$$

In particular, the last layer is given by

$$\Delta_j^L = \frac{\partial C_x}{\partial a_j^L}\sigma'(z_j^L) \equiv \nabla_a C_x \odot \sigma'(z^L)\,. \tag{2.14}$$

Having the variation matrix and the output of each layer we can compute the derivative of the cost function with respect to the coefficients that are given by

$$\frac{\partial C_x}{\partial w_{jk}^{l,l-1}} = a_k^{l-1}\Delta_j^l\,, \tag{2.15}$$

$$\frac{\partial C_x}{\partial b_j^l} = \Delta_j^l\,. \tag{2.16}$$

Then, we can run the usual gradient flow algorithm to update the parameters in the network

$$w_{jk}^{l,l-1} \to w_{jk}'^{l,l-1} = w_{jk}^{l,l-1} - \eta\frac{\partial C}{\partial w_{jk}^{l,l-1}}\,, \tag{2.17}$$

$$b_j^l \to b_j'^l = b_j^l - \eta\frac{\partial C}{\partial b_j^l}\,. \tag{2.18}$$

where C is the full cost function, or in practice it is compute in one mini-batch.

3 About neurodiffeq library

neurodiffeq is a python library that implements most of the tools to define and train neural networks to solve ordinary differential equations and partial differential equations in certain domains. Below, I will describe some of their functionalities. Before get started, we need to install the library and also install the library called **PyTorch**.

3.1 First example: Exponential ODE (with solve)

The first example uses a function **solve** which is not the best function to use, but for us, to starting familiarizing with the library it is good enough. First, we import the usual python libraries and the required **neurodiffeq** functions

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib notebook

from neurodiffeq import diff       # the differentiation operation
from neurodiffeq.ode import solve # the ANN-based solver from
neurodiffeq.conditions import IVP   # the initial condition
```

We will solve the following initial value problem

$$\frac{\mathrm{d}u}{\mathrm{d}t} + u(t) = 0\,, \qquad u(0) = 1\,, \qquad t \in [0,2]\,. \tag{3.1}$$

The analytic solution of this problem is

$$u(t) = e^{-t}\,, \qquad t \in [0,2]\,. \tag{3.2}$$

We will compare it with the solution that we will integrate using the neural network. In the following code lines we define the differential equation, the initial values and use **solve** to initialize the learning process of the neural network

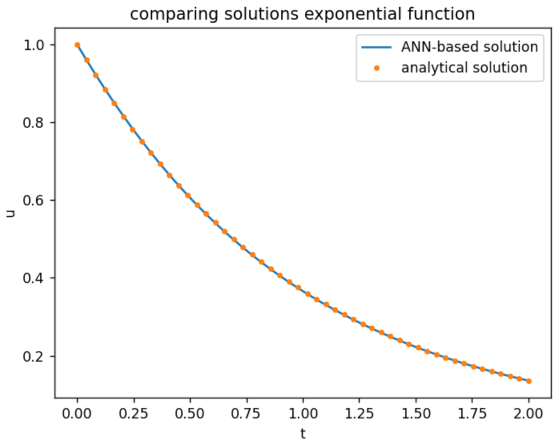
```
exponential_eq = lambda u, t: diff(u, t) + u # specify the ODE
init_val_ex = IVP(t_0=0.0, u_0=1.0)       # specify the initial conditon

solution_ex, loss_ex = solve(ode=exponential_eq, condition=init_val_ex, t_min=0.0
                                              ,
                      t_max=2.0)
```

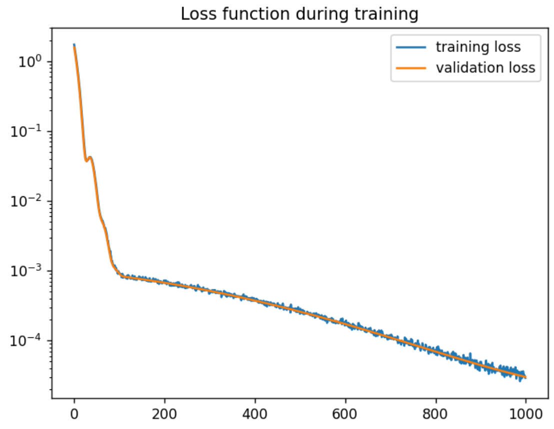
The output are **solution_ex** and **loss_ex**. **solution_ex** is a continuous function that takes one input and returns one output. Essentially, it is the function $u(t)$ that is solving the differential equation with certain precision. To see the solution and compare with the analytic solution we use

```
ts = np.linspace(0, 2.0, 50)
u_net = solution_ex(ts, to_numpy=True)
u_ana = np.exp(-ts)
plt.figure()
plt.plot(ts, u_net, label='ANN-based solution')
plt.plot(ts, u_ana, '.', label='analytical solution')
plt.ylabel('u') plt.xlabel('t')
plt.title('comparing solutions')
plt.legend()
plt.show()
```

The output is displayed in the left-panel of the figure below. Observe that, when the function **solution_ex** is evaluated, we are transforming the output (that usually is a **troch.Tensor**) to **numpy.array** by the instruction **to_numpy** = **True**. The output **loss_ex**. **solution_ex** contains the information of the training process such as the value of the loss function evaluated in both the training data and validation data. To have access to them we use **loss_ex**['**train**$_1$**oss**'] and **loss_ex**['**valid**$_1$**oss**'], respectively. The plot displayed in the right-panel of the figure below is

```
plt.figure()
plt.plot(loss_ex['train_loss'], label='training loss')
plt.plot(loss_ex['valid_loss'], label='validation loss')
plt.yscale('log') plt.title('Loss function during training')
plt.legend()
plt.show()
```





3.2 Harmonic oscillator with damping (with `solve`)

The next example that we want to discuss corresponds to the damped harmonic oscillator. The initial value problem is

$$\frac{\mathrm{d}^2u}{\mathrm{d}t^2}+\frac{1}{4}\frac{\mathrm{d}u}{\mathrm{d}t}+u(t)=0\,,\qquad u(0)=1\,,\qquad u'(0)=0\,,\qquad t\in[0,7\pi/2]\,. \tag{3.3}$$

The analytic solution reads

$$u(t)=\frac{e^{-t/8}}{21}\left[21\cos\left(\frac{3\sqrt{7}}{8}t\right)+\sqrt{7}\sin\left(\frac{3\sqrt{7}}{8}t\right)\right]\,,\qquad t\in[0,7\pi/2]\,. \tag{3.4}$$

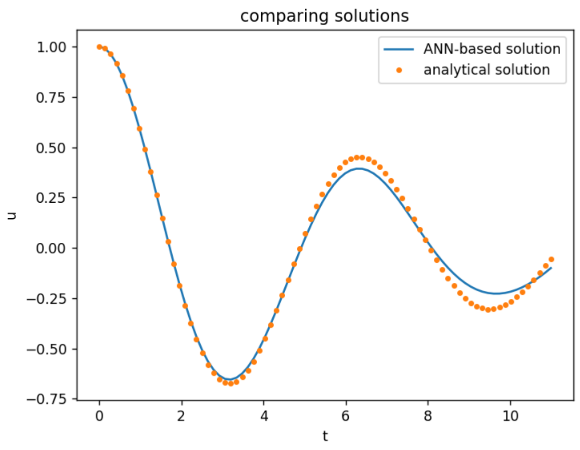
The differential equation, the initial conditions and the construction of the neural network is done by the following code.

```
harmonic_oscillator_dam = lambda u, t: diff(u, t, order=2) + u+1/4*diff(u, t,
                                           order=1)
init_val_ho = IVP(t_0=0.0, u_0=1.0, u_0_prime=0)
solution_ho, _ = solve(      ode=harmonic_oscillator_dam, condition=init_val_ho,
                                           t_min=0.0, t_max=3.5*np.pi,
                                           max_epochs=8000 )
```

Observe that now we have modified the parameter `max_epochs` which determines the number of epochs for the training process, you can play with this parameter. We put 8000 that takes few minutes in my computer to get better results. The convergence can be improved by changing the activation function of the network by using some function that has more "periodic" behavior, like trigonometric functions. The comparison between the analytic solution and the NN-based solution is done by

```
ts = np.linspace(0, 3.5*np.pi, 80)
u_net = solution_ho(ts, to_numpy=True)
u_ana =(21*np.cos((3*np.sqrt(7)*ts)/8) +    np.sqrt(7)*np.sin((3*np.sqrt(7)*ts)/
                                              8))/(21*np.exp(ts/8))
plt.figure()
plt.plot(ts, u_net,label='ANN-based solution')
plt.plot(ts, u_ana, '.', label='analytical solution')
plt.ylabel('u')
plt.xlabel('t')
plt.title('comparing solutions')
plt.legend()
plt.show()
```

leading to



3.3 System of differential equations and introducing `Solver`

Now we will do two things at once: (i) we consider a system of non-linear coupled first order differential equations and (ii) we introduce `Solver` that is the more flexible way to implement neural networks to solve differential equations.

The initial value problem that we use as example is

$$\frac{\mathrm{d}u}{\mathrm{d}t} - \alpha u(t) + \beta u(t)v(t) = 0\,, \qquad \frac{\mathrm{d}v}{\mathrm{d}t} - \delta u(t)v(t) + \gamma v(t) = 0\,, \tag{3.5}$$

$$u(0) = 1.5\,, \qquad v(0) = 1\,, \qquad t \in [0, 22] \qquad \alpha = \beta = \delta = \gamma = 1\,. \tag{3.6}$$

The differential equations are known as Lotka-Volterra system. The solutions of this system are mostly periodic, hence, as we saw in the previous example it is better to use an appropriated activation function. In this case we use sin activation function, implemented in `neurodiffeq.networks` as `SinActv`. To have more control on the NN we will define a fully connected neural network, implemented in **neurodiffeq.networks** as **FCNN**. In the following code we do that and define the initial value problem that we announced above.

```
from neurodiffeq.networks import FCNN    # fully-connect neural
network from neurodiffeq.networks import SinActv # sin activation

alpha,beta,delta,gamma=1,1,1,1
lotka_volterra = lambda u,v,t : [diff(u,t)-alpha*u+beta*u*v,
                                 diff(v,t)-delta*u*v+gamma*v]
# Initial conditions
init_vals_lv = [IVP(t_0=0.0,u_0=1.5),
                IVP(t_0=0.0,u_0=1.0)]
```

We will define to neural networks, one for each differential equation with 1 input and 1 output, two hidden layers of 32 and 32 neurons, respectively, and the activation function is **SinActv**

```
nets_lv=[
FCNN(n_input_units=1,n_output_units=1,hidden_units=(32,32),actv=SinActv),
FCNN(n_input_units=1,n_output_units=1,hidden_units=(32,32),actv=SinActv)]
```

We create a class (I think) called **mysolver** by using **Solover1D** inside **neurodiffeq.solvers** as follows

```
from neurodiffeq.solvers import Solver1D
solver=Solver1D(ode_system=lotka_volterra,conditions=init_vals_lv,
t_min=0.1,t_max=22.0,nets=nets_lv)
```

Since we implement it as a class the trying process and the extraction of the solution is done by the following two lines

```
mysolver.fit(max_epochs=5000)
newsolution_lv = mysolver.get_solution()
```

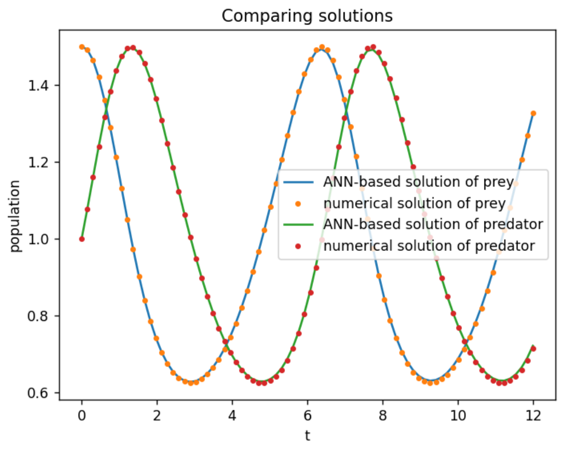
For this system of equations, there is no analytic solution. Hence, to compare with something trustable we integrate it numerically using

```
ts = np.linspace(0, 12, 200)

from scipy.integrate import odeint
def dPdt(P, t):
    return [P[0]*alpha - beta*P[0]*P[1], delta*P[0]*P[1] - gamma*P[1]]
P0 = [1.5, 1.0]
Ps = odeint(dPdt, P0, ts)
prey_num = Ps[:,0]
pred_num = Ps[:,1]
```

Then, the comparison is done by the following code

```
prey_net, pred_net = newsolution_lv(ts, to_numpy=True)
plt.figure()
plt.plot(ts, prey_net, label='ANN-based solution of prey')
plt.plot(ts, prey_num, '.', label='numerical solution of prey')
plt.plot(ts, pred_net, label='ANN-based solution of predator')
plt.plot(ts, pred_num, '.', label='numerical solution of predator')
plt.ylabel('population')
plt.xlabel('t')
plt.title('Comparing solutions')
plt.legend()
```

Leading to the following plot



3.4 Trying to integrate Schwarzschild

Einstein's equations with cosmological constant $R_{\mu\nu} - \frac{1}{2}g_{\mu\nu}R + \Lambda g_{\mu\nu} = 0$ in a spherically symmetric spacetime

$$\mathrm{d}s^2 = -f(r)\mathrm{d}t^2 + \frac{\mathrm{d}r^2}{f(r)} + r^2(\mathrm{d}\theta^2 + \sin^2\theta \mathrm{d}\varphi^2) \tag{3.7}$$

reduce to a single first order ordinary differential equation

$$-\Lambda r^2 - rf'(r) - f(r) + 1 = 0\,. \tag{3.8}$$

The only integration constant is related to the energy of the spacetime. The way to implement the fact that the metric (3.7) is a black hole is by imposing that there exist a value $r_+ \in]0, \infty[$ such that

$$f(r_+) = 0\,. \tag{3.9}$$

For "simplicity", let us set $r_+ = 1$, namely, we are fixing the mass of the black hole to a certain value. It would nice to have another input to the neural network which is related to the mass.

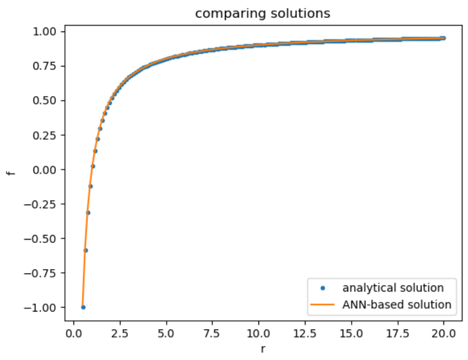
Hence, the initial value problem that we are considering is the following

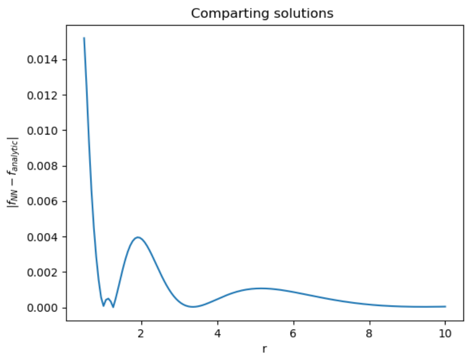
$$-\Lambda r^2 - r f'(r) - f(r) + 1 = 0\,, \qquad f(1) = 0\,, \qquad r \in [0.5, 20]\,. \tag{3.10}$$

We consider a feed-forward neural network with one input layer, one output layer, two hidden layers with 128 and 64 neurons, respectively. The activation function that we choose is tanh. Let us consider first the case $\Lambda = 0$. The code reads

```
%matplotlib inline
schw_eq = lambda u, t: [ -t*diff(u, t, order=1) - u+1 ]
init_val_schw = IVP(t_0=1.0, u_0=0.0)
net_schw = FCNN(     hidden_units=( 128, 64), actv=nn.Tanh )
import Monitor1D
monitor_callback = Monitor1D(t_min=0.5, t_max=20.0, check_every=2000).to_callback
                                                                        ()
solver = Solver1D(ode_system=schw_eq, conditions=[init_val_schw],
                                                                        t_min=0.5,t_max=20.0,nets=[net_schw])
solver.fit(max_epochs=4000, callbacks=[monitor_callback])
solution_schw = solver.get_solution()
```

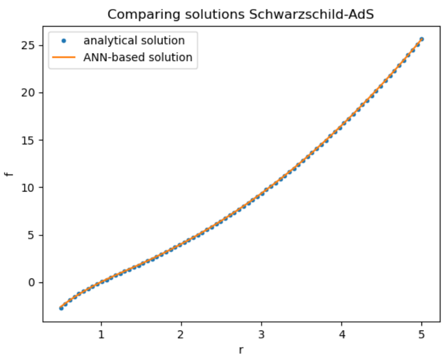
Considering 4000 epochs the results are the following

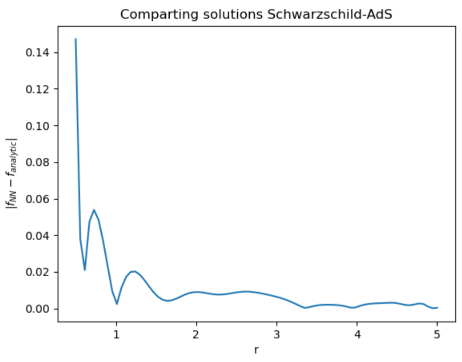




The analytic solution of the initial value problem (3.10) is $f(r) = 1 - 1/r$. The comparison between the solution generated by the neural network and the analytical solution was displayed in the above figures, it is quiet good considering that this is the first try.

Now, we implementing the same neural network architecture that we defined above, but to integrate Schwarzschild-AdS with negative cosmological constant given by $\Lambda = -3$. The analytic solution of the initial value problem with $f(1) = 0$ is $f(r) = r^2 + 1 - \frac{2}{r}$. In this case, finding a good solution for large values of r with the architecture of the neural network presented above is more involved. Hence, we consider the domain to be $[0.5, 5]$ leading the following results:





3.5 Observations 24-12-2024

File 'Integrating_Schwarzschild' I was trying to change the training set and the activation function. Using the activation function `Swish` that is defined in `torch.nn.SiLU`, an articture given by $(1, 128, 128, 1)$, and for the training using $'\texttt{log}-\texttt{spaced}'$ in the full set and $'\texttt{equally}-\texttt{spaced}'$ in the more complicated regions, gives already very good results.

3.6 Schwarzschild $f(r), g(r)$

In this section we will take note of what happen when we consider the metric

$$\mathrm{d}s^2 = -f(r)\mathrm{d}t^2 + \frac{\mathrm{d}r^2}{g(r)} + r^2(\mathrm{d}\theta^2 + \sin^2\theta \mathrm{d}\varphi^2)\,, \tag{3.11}$$

in the Einstein's equations $G_{\mu\nu} = 0$ and how the neural network will find that $f(r) = g(r)$. The set of equations are

$$rg' + g - 1 = 0\,, \tag{3.12}$$

$$rgf' + gf - f = 0\,, \tag{3.13}$$

$$rfg'f' + 2rgf''f - rgf'^2 + 2g'f^2 + 2gf'f = 0\,. \tag{3.14}$$

One can show that the first two equation imply the third equation.

3.7 Eintein-Hilbert arbitrary dimension

Considering the metric

$$\boldsymbol{g} = -e^{2A}\mathrm{d}t + e^{2B}\mathrm{d}r^2 + r^2\eta^i \otimes \eta^j \delta_{ij}$$

where $\eta^i \otimes \eta^j \delta_{ij}$ is a constant curvature $D-2$ dimensional metric. We consider that $A = A(r)$ and $B = B(r)$. A viebelin basis is

$$e^a = (\mathrm{e}^A \mathrm{d}t, \mathrm{e}^B \mathrm{d}r, r\eta^i)\,. \tag{3.15}$$

where we are expanding $a = (0, 1, i)$ with $i = 1, \ldots, D-2$. Its exterior derivative is

$$\begin{aligned}\mathrm{d}e^a &= (\mathrm{e}^A A' \mathrm{d}r \wedge \mathrm{d}t, \mathrm{e}^B B' \mathrm{d}t \wedge \mathrm{d}r, \mathrm{d}r \wedge \eta^i + r\mathrm{d}\eta^i)\,,\\ &= (\mathrm{e}^{-B} A' e^1 \wedge e^0, \mathrm{e}^{-A} B' e^0 \wedge e^1, \mathrm{e}^{-B} e^1 \wedge \eta^i + r\mathrm{d}\eta^i)\,,\end{aligned}$$

Lowering the index

$$\mathrm{d}e_a = (-\mathrm{e}^{-B} A' e^1 \wedge e^0, \mathrm{e}^{-A} B' e^0 \wedge e^1, \mathrm{e}^{-B} e^1 \wedge \eta_i + r\mathrm{d}\eta_i)$$

The contraction operator leads

$$\imath_{b_{(\mathrm{row})}} \mathrm{d}e_{a_{(\mathrm{col})}} = \left(\begin{array}{cc|c} \mathrm{e}^{-B} A' e^1 & \mathrm{e}^{-A} B' e^1 & 0 \\ \mathrm{e}^{-B} A' e^0 & -\mathrm{e}^{-A} B' e^0 & \mathrm{e}^{-B} \eta_i \\ \hline 0_j & 0_j & -\mathrm{e}^{-B} e^1 \delta_{ij} + r\imath_j \mathrm{d}\eta_i \end{array}\right)$$

contracting again

$$\imath_c \imath_b \mathrm{d}e_a = \left(\begin{array}{cc|c} \mathrm{e}^{-B} A' \delta^1_c & \mathrm{e}^{-A} B' \delta^1_c & 0_i \\ \mathrm{e}^{-B} A' \delta^0_c & -\mathrm{e}^{-A} B' \delta^0_c & \mathrm{e}^{-B} \delta^i_c \\ \hline 0_j & 0_j & -\mathrm{e}^{-B} \delta^1_c \delta_{ij} + r\imath_c \imath_j \mathrm{d}\eta_i \end{array}\right)$$

Using the following formula for the spin connection

$$\omega_{ab} = \frac{1}{2} \imath_a \imath_b \mathrm{d}e_c e^c - \imath_{[a} \mathrm{d}e_{b]}$$

Let us do an experiment

$$\mathrm{d}e_a = \begin{pmatrix} -\mathrm{e}^{-B} A' e^1 \wedge e^0 \\ \mathrm{e}^{-A} B' e^0 \wedge e^1 \\ \mathrm{e}^{-B} e^1 \wedge \eta_i + r\mathrm{d}\eta_i \end{pmatrix} \Longrightarrow \imath_b \mathrm{d}e_a = \begin{pmatrix} -\mathrm{e}^{-B} A' e^0 \delta^1_b + \mathrm{e}^{-B} A' e^1 \delta^0_b \\ \mathrm{e}^{-A} B' \delta^0_b e^1 + \mathrm{e}^{-A} B' e^0 \delta^1_b \\ \mathrm{e}^{-B} \delta^1_b \eta_i + \mathrm{e}^{-B} e^1 \imath_b \eta_i + r\imath_b \mathrm{d}\eta_i \end{pmatrix}$$

$$\begin{aligned}\imath_b \mathrm{d}e_a = {} & (-\mathrm{e}^{-B} A' e^0 \delta^1_b + \mathrm{e}^{-B} A' e^1 \delta^0_b)\delta^0_a + (\mathrm{e}^{-A} B' \delta^0_b e^1 + \mathrm{e}^{-A} B' e^0 \delta^1_b)\delta^1_a + \\ & + \sum_i (\mathrm{e}^{-B} \delta^1_b \eta_i + \mathrm{e}^{-B} e^1 \imath_b \eta_i + r\imath_b \mathrm{d}\eta_i)\delta^i_a\end{aligned}$$

Then, its antisymmetric part is

$$\begin{aligned}\imath_{[b}\mathrm{d}e_{a]} &= -\mathrm{e}^{-B}A'e^0\delta^1_{[b}\delta^0_{a]} + \mathrm{e}^{-A}B'\delta^0_{[b}\delta^1_{a]}e^1 + \\ &+ \sum_i(\mathrm{e}^{-B}\delta^i_{[a}\delta^1_{b]}\eta_i + \mathrm{e}^{-B}e^1\delta^i_{[a}\imath_{b]}\eta_i + r\imath_{[b|}\mathrm{d}\eta_i\delta^i_{|a]})\,.\end{aligned}$$

3.8 Einstein-Gauss-Bonnet-Lambda

Let us consider the following action principle

$$\begin{aligned}I &= \int \mathrm{d}^5x\sqrt{-g}(R - 2\Lambda + \alpha\mathscr{L}_{(2)}) \\ \mathscr{L}_{(2)} &= R_{\mu\nu\rho\sigma}R^{\mu\nu\rho\sigma} - 4R_{\mu\nu}R^{\mu\nu} + R^2 = \frac{1}{4}\delta^{\lambda\delta\eta\zeta}_{\mu\nu\rho\sigma}R^{\mu\nu}_{\lambda\delta}R^{\rho\sigma}_{\eta\zeta}\,.\end{aligned}$$

Let us compute the equations of motion

$$\delta I = \int \mathrm{d}^5x\sqrt{-g}\left[\left(-\frac{1}{2}\delta g^{\mu\nu}g_{\mu\nu}\right)(R + \alpha\mathscr{L}_{(2)} - 2\Lambda) + \delta g^{\mu\nu}R_{\mu\nu} + g^{\mu\nu}\delta R_{\mu\nu} + \delta\mathscr{L}_{(2)}\right]\,.$$

Note that

$$\delta\mathscr{L}_{(2)} \equiv \frac{1}{2}\delta^{\lambda\delta\eta\zeta}_{\mu\nu\rho\sigma}\delta R^{\mu\nu}{}_{\lambda\delta}R^{\rho\sigma}_{\eta\zeta} = \frac{1}{2}\delta^{\lambda\delta\eta\zeta}_{\mu\nu\rho\sigma}\delta g^{\nu\alpha}R^{\mu}{}_{\alpha\lambda\delta}R^{\rho\sigma}_{\eta\zeta} + \frac{1}{2}\delta^{\lambda\delta\eta\zeta}_{\mu\nu\rho\sigma}g^{\nu\alpha}\delta R^{\mu}{}_{\alpha\lambda\delta}R^{\rho\sigma}_{\eta\zeta}\,. \tag{3.16}$$

The variation of the Riemann tensor is

$$\delta R^{\mu}{}_{\alpha\lambda\delta} = 2\nabla_{[\lambda}\delta\Gamma^{\mu}{}_{\delta]\alpha}\,. \tag{3.17}$$

Replacin that we see that

$$\begin{aligned}\delta\mathscr{L}_{(2)} &= \frac{1}{2}\delta^{\lambda\delta\eta\zeta}_{\mu\nu\rho\sigma}\delta g^{\nu\alpha}R^{\mu}{}_{\alpha\lambda\delta}R^{\rho\sigma}_{\eta\zeta} + \delta^{\lambda\delta\eta\zeta}_{\mu\nu\rho\sigma}g^{\nu\alpha}\nabla_\lambda\delta\Gamma^{\mu}{}_{\delta\alpha}R^{\rho\sigma}_{\eta\zeta}\,, \\ &= \frac{1}{2}\delta^{\lambda\delta\eta\zeta}_{\mu\nu\rho\sigma}\delta g^{\nu\alpha}R^{\mu}{}_{\alpha\lambda\delta}R^{\rho\sigma}_{\eta\zeta} + \nabla_\lambda(\delta^{\lambda\delta\eta\zeta}_{\mu\nu\rho\sigma}g^{\nu\alpha}\delta\Gamma^{\mu}{}_{\delta\alpha}R^{\rho\sigma}_{\eta\zeta}) - \delta^{\lambda\delta\eta\zeta}_{\mu\nu\rho\sigma}g^{\nu\alpha}\delta\Gamma^{\mu}{}_{\delta\alpha}\nabla_{[\lambda}R^{\rho\sigma}_{\eta\zeta]}\,,\end{aligned}$$

The last term is zero due to the Jacobi identity and the second term is a boundary term which does not contribute to the equations of motion. Thus the variation of the action principle i

$$\delta I = \int \mathrm{d}^5x\sqrt{-g}\left[\left(-\frac{1}{2}\delta g^{\mu\nu}g_{\mu\nu}\right)(R + \alpha\mathscr{L}_{(2)} - 2\Lambda) + \delta g^{\mu\nu}R_{\mu\nu} + \alpha\frac{1}{2}\delta^{\lambda\delta\eta\zeta}_{\beta\nu\rho\sigma}\delta g^{\mu\nu}R^{\beta}{}_{\mu\lambda\delta}R^{\rho\sigma}_{\eta\zeta}\right]\,.$$

Leading to the following equations of motion

$$R_{\mu\nu} - \frac{1}{2}g_{\mu\nu}R + g_{\mu\nu}\Lambda - \frac{1}{2}\alpha g_{\mu\nu}\mathscr{L}_{(2)} + \alpha\frac{1}{2}\delta^{\lambda\delta\eta\zeta}_{\mu\beta\rho\sigma}R_{\nu}{}^{\beta}{}_{\lambda\delta}R^{\rho\sigma}_{\eta\zeta} = 0\,. \tag{3.18}$$

One can check that

$$\delta^{\lambda\delta\eta\zeta}_{\mu\beta\rho\sigma} R_{\nu}{}^{\beta}{}_{\lambda\delta} R^{\rho\sigma}_{\eta\zeta} = 4R_{\mu\nu}R - 8R_{\mu\sigma}R^{\sigma}{}_{\nu} + 8R^{\lambda}{}_{\mu\nu\sigma}R^{\sigma}{}_{\lambda} + 4R^{\lambda}{}_{\mu\rho\sigma}R^{\rho\sigma}{}_{\lambda\nu} \equiv \mathscr{G}_{(2)\mu\nu} \tag{3.19}$$

Then, the equations are

$$G_{\mu\nu} + g_{\mu\nu}\Lambda + \frac{\alpha}{2}\left(\mathscr{G}_{(2)\mu\nu} - g_{\mu\nu}\mathscr{L}_{(2)}\right) = 0\,. \tag{3.20}$$

Considering a metric of the form

$$\mathrm{d}s^2 = -f(r)\mathrm{d}t^2 + \frac{\mathrm{d}r^2}{f(r)} + r^2\mathrm{d}s^2(S^3)\,, \tag{3.21}$$

where $\mathrm{d}s^2(S^3)$ is the metric of a 3-sphere. To be concrete we are considering

$$\mathrm{d}s^2(S^3) = \mathrm{d}x^2 + \sin^2 x(\mathrm{d}y^2 + \sin^2 y\mathrm{d}z^2)\,. \tag{3.22}$$

The equations (3.20) reduces to two equations

$$(4f(r)\alpha - r^2 - 4\alpha)\frac{\mathrm{d}f}{\mathrm{d}r} = \frac{2}{3}r(r^2\Lambda + 3f(r) - 3)\,, \tag{3.23}$$

$$(r^2 - 4f(r)\alpha + 4\alpha)\frac{\mathrm{d}^2 f}{\mathrm{d}r^2} = -2r^2\Lambda + 4\alpha\left(\frac{\mathrm{d}f}{\mathrm{d}r}\right)^2 - 4r\frac{\mathrm{d}f}{\mathrm{d}r} - 2f(r) + 2\,. \tag{3.24}$$

The second order equation is a consequence of the first order equation. Hence we can solve the first order equation, leading to two solutions

$$f_{\pm}(r) = \frac{r^2}{4\alpha} + 1 \pm \frac{1}{12\alpha}\sqrt{(12\Lambda r^4 + 12\mu)\alpha + 9r^4}\,. \tag{3.25}$$

One can check that in the limit $\alpha \to 0$ is possible only by considering the branch $f_-(r)$. We will consider it from now on

$$f(r) \equiv \frac{r^2}{4\alpha} + 1 - \frac{1}{12\alpha}\sqrt{(12\Lambda r^4 + 12\mu)\alpha + 9r^4}\,. \tag{3.26}$$

In the limit $\alpha \to 0$ it becomes

$$\lim_{\alpha\to 0} f(r) = -\frac{r^2\Lambda}{6} + 1 - \frac{\mu}{6r^2}\,, \tag{3.27}$$

which is Schwarzschild-(A)ds. Let us study the location of the horizon located at r_+ such that $f(r_+) = 0$. This equation can be written as

$$-\Lambda r_0^4 + 6r_0^2 + 12\alpha - \mu = 0\,. \tag{3.28}$$

3.8.1 case $\mu = 0$

The solution is

$$f_{\pm}(r) = 1 + \left(\frac{1}{4\alpha} \pm \frac{\sqrt{12\Lambda\alpha + 9}}{12\alpha} \right) r^2 \,, \tag{3.29}$$

note that if $\Lambda = 0$

$$f_{\pm}(r) = 1 + \frac{1}{4\alpha} (1 \pm 1)\, r^2 \implies \begin{cases} f_{+}(r) = 1 + \frac{1}{2\alpha} r^2 \\ f_{-}(r) = 1 \end{cases} \tag{3.30}$$

3.8.2 $\Lambda = 0$ case

Let us consider the case without cosmological constant first. Setting $\Lambda = 0$ the solution is

$$f(r) = \frac{r^2}{4\alpha} + 1 - \frac{1}{12\alpha} \sqrt{12\mu\alpha + 9r^4} \,,$$

and the location of the horizon is

$$r_0 = \sqrt{\frac{\mu}{6} - 2\alpha} \,.$$

3.9 Kerr in Kerr-Schild coordinates

Let us consider flat spacetime in spheroidal coordinates as a seed metric

$$\mathrm{d}s^2(\mathbb{R}^{1,3}) = -\mathrm{d}t^2 + \frac{\rho^2}{r^2 + a^2} \mathrm{d}r^2 + \rho^2 \mathrm{d}\theta^2 + (r^2 + a^2) \sin^2\theta \mathrm{d}\varphi^2 \,, \tag{3.31}$$

$$\rho^2 = r^2 + a^2 \cos^2\theta \,. \tag{3.32}$$

Its four dimensional null and geodesic vector is

$$\ell_\mu \mathrm{d}x^\mu = \mathrm{d}t + \frac{\rho^2}{r^2 + a^2} \mathrm{d}r + a \sin^2\theta \mathrm{d}\varphi \,. \tag{3.33}$$

Hence, the Kerr-Schild ansatz is

$$\mathrm{d}s^2 = \mathrm{d}s^2(\mathbb{R}^{1,3}) + F(r,\theta) \ell_\mu \ell_\nu \mathrm{d}x^\mu \mathrm{d}x^\nu \,, \tag{3.34}$$

where the metric function is $F(r,\theta)$. The trace of the Einstein equations is

$$(r^2 + a^2 \cos^2\theta) \frac{\partial^2 F}{\partial r^2} + 4r \frac{\partial F}{\partial r} + 2F(r,\theta) = 0 \,, \tag{3.35}$$

whose solution is

$$F(r,\theta) = \frac{f_1(\theta)r + f_2(\theta)}{r^2 + a^2\cos^2\theta}\,. \tag{3.36}$$

Replacing into the $G_{\theta\theta} = 0$ equation that is

$$-(r^2 + a^2\cos^2\theta)^2\frac{\partial^2 F}{\partial r^2} - 2r(r^2 + a^2\cos^2\theta)\frac{\partial F}{\partial r} - 4F(r,\theta)a^2\cos^2\theta = 0\,. \tag{3.37}$$

Implies that $f_2(\theta) = 0$. The equation $G_{t\theta} = 0$ that is

$$(r^2 + a^2\cos^2\theta)^2\frac{\partial^2 F}{\partial r\partial\theta} - 2a^2\sin\theta\cos\theta\left((r^2 + a^2\cos^2\theta)\frac{\partial F}{\partial r} - 2F\right) = 0\,, \tag{3.38}$$

implies that $f_1(\theta) = C$. Then, the solution is

$$F(r,\theta) = \frac{2mr}{r^2 + a^2\cos^2\theta}\,. \tag{3.39}$$

The horizon is located where the 1-form $\mathrm{d}r$ is null. Evaluated at the solution

$$\mathrm{d}r^2 = \frac{a^2 - 2mr + r^2}{r^2 + a^2}\,. \tag{3.40}$$

The solution of the location where $\mathrm{d}r^2 = 0$ are the surfaces

$$r_\pm = m \pm \sqrt{m^2 - a^2}\,. \tag{3.41}$$

The boundary conditions for the function $F(r,\theta)$ are

$$\begin{aligned} F(r\to\infty,\theta) &= \frac{2m}{r} - \frac{2ma\cos^2\theta}{r^3} + o(r^{-5})\,,\\ F(r,\theta\to 0) &= \frac{2mr}{r^2 + a^2}\,,\\ F(r,\theta\to\pi) &= \frac{2mr}{r^2 + a^2}\,. \end{aligned} \tag{3.42}$$

Inetegration domain May be a sensible integration domain is

$$r \in [2a, R]\,, \qquad \theta \in [0,\pi]$$

where the boundary conditiones are listed above and

$$F(r\to a,\theta) = \frac{2C}{a(4 + \cos^2\theta)}$$

3.9.1 What about the vacuum?

If we consider the the possible solution with $\mu = 0$ they read

$$f_{\pm}(r) = 1 + \left(3 \pm \sqrt{12\Lambda\alpha + 9}\right) \frac{r^2}{12\alpha} \,. \tag{3.43}$$

Note that at $r = 0$ they coincide

$$f_{+}(r = 0) = f_{-}(r = 0) = 1 \,. \tag{3.44}$$

Which solution the neural network will pick?

4 Lovelock gravity

Let us consider the following action principle on d-dimensional manifold $\mathscr{M}$

$$I = \sum_{k=0}^{K} \frac{a_k}{d - 2k} \int_{\mathscr{M}} \mathcal{R}^{(k)} \,, \tag{4.1}$$

where the d-form

$$\mathcal{R}^{(k)} = \epsilon_{a_1 \ldots a_d} R^{a_1 a_2} \wedge \ldots$$

$p \in \mathscr{M}$, $\quad V(p) = V^{\mu} \frac{\partial}{\partial x^{\mu}} \in T_p\mathscr{M}$ espacio tangente, existe un espacio vectorial dual $\alpha = \alpha_{\mu} \mathrm{d}x^{\mu} \in T_p^*\mathscr{M}$ espacio co-tangente. In summary, $T_p\mathscr{M}$ dada unas coordiantes x^{μ}, hay una base $\{\frac{\partial}{\partial x^{\mu}}\}$ y en el espacio dual $T_p^*\mathscr{M}$, hay una base $\{\mathrm{d}x^{\mu}\}$

$$\mathrm{d}x^{\mu}\left(\frac{\partial}{\partial x^{\nu}}\right) = \delta^{\mu}_{\nu} \,, \qquad \frac{\partial}{\partial x^{\nu}}(\mathrm{d}x^{\mu}) = \delta^{\mu}_{\nu} \,. \tag{4.2}$$

1-forma diferencial es un elemento del espacio co-tangente $\alpha = \alpha_{\mu} \mathrm{d}x^{\mu} \in T^*\mathscr{M}$.

Uno puede definir tensores

$$A_{\mu\nu} \mathrm{d}x^{\mu} \otimes \mathrm{d}x^{\nu} \in T_p^*\mathscr{M} \otimes T_p^*\mathscr{M}$$

por ejemplo, la metrica es

$$g_{\mu\nu} \mathrm{d}x^{\mu} \otimes \mathrm{d}x^{\nu} \in T_p^*\mathscr{M} \otimes T_p^*\mathscr{M}$$

$$A_{\mu\nu} = -A_{\nu\mu} \implies A_{\mu\nu}\mathrm{d}x^\mu \otimes \mathrm{d}x^\nu \in T_p^*\mathscr{M} \otimes_{\text{antisymmetric}} T_p^*\mathscr{M}$$

wedge

$$\begin{aligned} A_{\mu\nu}\mathrm{d}x^\mu \otimes \mathrm{d}x^\nu &= \frac{1}{2}(A_{\mu\nu} + A_{\mu\nu})\mathrm{d}x^\mu \otimes \mathrm{d}x^\nu \\ &= \frac{1}{2}(A_{\mu\nu} - A_{\nu\mu})\mathrm{d}x^\mu \otimes \mathrm{d}x^\nu\,, \\ &= \frac{1}{2}(A_{\mu\nu}\mathrm{d}x^\mu \otimes \mathrm{d}x^\nu - A_{\mu\nu}\mathrm{d}x^\nu \otimes \mathrm{d}x^\mu)\,, \\ &= \frac{1}{2}A_{\mu\nu}(\mathrm{d}x^\mu \otimes \mathrm{d}x^\nu - \mathrm{d}x^\nu \otimes \mathrm{d}x^\mu)\,, \\ &\equiv \frac{1}{2}A_{\mu\nu}\mathrm{d}x^\mu \wedge \mathrm{d}x^\nu\,. \end{aligned}$$

In general a p-form is of the form

$$A_{(p)} = \frac{1}{p!}A_{\mu_1\dots\mu_p}\mathrm{d}x^{\mu_1} \wedge \cdots \wedge \mathrm{d}x^{\mu_p}\,. \tag{4.3}$$

If there is a metric in the space

$$\boldsymbol{g} = g_{\mu\nu}\mathrm{d}x^\mu \otimes \mathrm{d}x^\nu$$

we can diagonalize in any point

$$g_{\mu\nu}(x) = \eta_{ab}e^a{}_\mu(x)e^b{}_\nu(x)\,, \qquad \eta_{ab} = \mathrm{diag}(-1, 1, \dots, 1)\,. \tag{4.4}$$

where

$$e^a(x) = e^a{}_\mu \mathrm{d}x^\mu$$

is the vielbein 1-form. Replacing back

$$\boldsymbol{g} = \eta_{ab}e^a{}_\mu\mathrm{d}x^\mu \otimes e^b{}_\nu\mathrm{d}x^\nu \equiv \eta_{ab}e^a \otimes e^b\,. \tag{4.5}$$

parethesis

$$\begin{aligned}\boldsymbol{g}_{\text{Sch}} = \mathrm{d}s^2 &= -f(r)\mathrm{d}t^2 + \frac{\mathrm{d}r^2}{f(r)} + r^2(\mathrm{d}\theta^2 + \sin^2\theta \mathrm{d}\varphi^2)\,,\\ \text{existe ?} &= \eta_{ab} e^a \otimes e^b = \\ &= \eta_{00} e^0 \otimes e^0 + \eta_{11} e^1 \otimes e^1 + \eta_{22} e^2 \otimes e^2 + \eta_{33} e^3 \otimes e^3\,,\\ &= -e^0 \otimes e^0 + e^1 \otimes e^1 + e^2 \otimes e^2 + e^3 \otimes e^3\,,\\ &= -f\mathrm{d}t \otimes \mathrm{d}t + \frac{\mathrm{d}r \otimes \mathrm{d}r}{f} + r^2 \mathrm{d}\theta \otimes \mathrm{d}\theta + r^2\sin^2\theta \mathrm{d}\varphi \otimes \mathrm{d}\varphi\,,\\ &= -f\mathrm{d}t^2 + \frac{\mathrm{d}r^2}{f} + r^2\mathrm{d}\theta^2 + r^2\sin^2\theta\mathrm{d}\varphi^2\,,\end{aligned}$$

we can pick

$$e^0 = \sqrt{f}\mathrm{d}t\,, \qquad e^1 = \frac{\mathrm{d}r}{\sqrt{f}}\,, \qquad e^2 = r\mathrm{d}\theta\,, \qquad e^3 = r\sin\theta\mathrm{d}\varphi\,.$$

note

$$e^0 \wedge e^1 = \sqrt{f}\mathrm{d}t \wedge \frac{\mathrm{d}r}{\sqrt{f}} = \mathrm{d}t \wedge \mathrm{d}r\,. \tag{4.6}$$

and

$$e^0 \wedge e^1 \wedge e^2 \wedge e^3 = \sqrt{f}\mathrm{d}t \wedge \frac{\mathrm{d}r}{\sqrt{f}} \wedge r\mathrm{d}\theta \wedge r\sin\theta\mathrm{d}\varphi = r^2\sin\theta\mathrm{d}t \wedge \mathrm{d}r \wedge \mathrm{d}\theta \wedge \mathrm{d}\varphi = \sqrt{-g}\mathrm{d}^4x$$

In other words, in general

$$e^0 \wedge e^1 \wedge e^2 \wedge e^3 = \frac{1}{4!}\epsilon_{abcd} e^a \wedge e^b \wedge e^c \wedge e^d\,.$$

where ϵ_{abcd} is the symbol $\epsilon_{0123} = 1$.

Let us define the 1-form spin connection

$$\omega_{ab} = \omega_{ab\mu}\mathrm{d}x^\mu\,, \qquad \omega_{ab} = -\omega_{ba} \tag{4.7}$$

The curvature 2-form

$$R_{ab} = \mathrm{d}\omega_{ab} + \omega_{ac} \wedge \omega^c{}_b\,. \tag{4.8}$$

In components

$$R_{ab} = \frac{1}{2}R_{ab\mu\nu}\mathrm{d}x^\mu \wedge \mathrm{d}x^\nu$$

The torsion 2-form is defined by

$$T^a = \mathrm{d}e^a + \omega^a{}_b \wedge e^b \,. \tag{4.9}$$

where it is zero

$$T^a = 0 \implies \mathrm{d}e^a + \omega^a{}_b \wedge e^b = 0 \,. \tag{4.10}$$

$$L_{(\mathrm{EH})} = \frac{1}{2!} R^{ab} \wedge e^c \wedge e^b \epsilon_{abcd} \sim R \mathrm{d}^4 x \sqrt{|g|}$$

def. $\alpha = \alpha_\mu \mathrm{d}x^\mu = \alpha_a e^a$ where $\alpha_a = \alpha_\mu e_a{}^\mu$

Note that

$$\begin{aligned}
L_{(\mathrm{EH})} &= \frac{1}{2!} R^{ab} \wedge e^c \wedge e^d \epsilon_{abcd} \,, \\
&= \frac{1}{2!}\frac{1}{2!} R^{ab}{}_{\mu\nu} \mathrm{d}x^\mu \wedge \mathrm{d}x^\nu \wedge e^c \wedge e^d \epsilon_{abcd} \,, \qquad (\mathrm{d}x^\mu = e_c{}^\mu e^c = e_c{}^\mu e^c{}_\nu \mathrm{d}x^\nu = \delta^\mu_\nu \mathrm{d}x^\nu) \\
&= \frac{1}{2!}\frac{1}{2!} R^{ab}{}_{\mu\nu} e_g{}^\mu e_f{}^\nu e^g \wedge e^f \wedge e^c \wedge e^d \epsilon_{abcd} \,, \qquad (R^{ab}{}_{\mu\nu} e_c{}^\mu \equiv R^{ab}{}_{c\nu}) \\
&= \frac{1}{2!}\frac{1}{2!} R^{ab}{}_{gf} e^g \wedge e^f \wedge e^c \wedge e^d \epsilon_{abcd} \,,
\end{aligned}$$

Trick: $\delta^{gfcb}_{adei} = 4! \delta^g_{[a} \delta^f_d \delta^c_e \delta^b_{i]} =$

$$\begin{aligned}
e^g \wedge e^f \wedge e^c \wedge e^d &= \frac{1}{4!} \delta^{gfcd}_{ajei} e^a \wedge e^j \wedge e^e \wedge e^i \\
&= \frac{1}{4!} \epsilon^{gfcd} \epsilon_{ajei} e^a \wedge e^j \wedge e^e \wedge e^i \,, \\
&= \epsilon^{gfcd} \frac{1}{4!} \epsilon_{ajei} e^a \wedge e^j \wedge e^e \wedge e^i \,, \\
&= \epsilon^{gfcd} e^0 \wedge e^1 \wedge e^2 \wedge e^3 = \epsilon^{gfcb} \sqrt{|g|} \mathrm{d}^4 x
\end{aligned}$$

Replacinb back

$$\begin{aligned}
L_{(\mathrm{EH})} &= \frac{1}{2!}\frac{1}{2!}R^{ab}{}_{gf}e^g \wedge e^f \wedge e^c \wedge e^d \epsilon_{abcd}\,, \\
&= \frac{1}{2!}\frac{1}{2!}R^{ab}{}_{gf}\epsilon^{gfcd}\epsilon_{abcd}\sqrt{|g|}\mathrm{d}^4x\,, \\
&= \frac{1}{2!}\frac{1}{2!}R^{ab}{}_{gf}\delta^{gfcd}_{abcd}\sqrt{|g|}\mathrm{d}^4x\,, \\
&= \frac{1}{2!}\frac{1}{2!}R^{ab}{}_{gf}\delta^{gfcd}_{abcd}\sqrt{|g|}\mathrm{d}^4x\,, \\
&= \frac{1}{2!}\frac{1}{2!}R^{ab}{}_{gf}\frac{(4-2)!}{(4-4)!}\delta^{gf}_{ad}\sqrt{|g|}\mathrm{d}^4x\,, \\
&= \frac{1}{2!}R^{ab}{}_{gf}\delta^{gf}_{ab}\sqrt{|g|}\mathrm{d}^4x\,, \\
&= \frac{1}{2!}R^{ab}{}_{gf}(\delta^g_a\delta^f_b - \delta^g_b\delta^f_a)\sqrt{|g|}\mathrm{d}^4x\,, \\
&= R^{ab}{}_{ab}\sqrt{|g|}\mathrm{d}^4x\,, \\
&= R\sqrt{|g|}\mathrm{d}^4x
\end{aligned}$$

-We can construct a d-form. Let us consider $d = 4$.

$$\begin{aligned}
e^a \wedge e^b &= e^a{}_\mu \mathrm{d}x^\mu \wedge e^b{}_\nu \mathrm{d}x^\nu\,, \\
&= e^a{}_\mu e^b{}_\nu \mathrm{d}x^\mu \wedge \mathrm{d}x^\nu\,, \\
&= e^a{}_\mu e^b{}_\nu (\mathrm{d}x^\mu \otimes \mathrm{d}x^\nu - \mathrm{d}x^\nu \otimes \mathrm{d}x^\mu)
\end{aligned}$$

Going to Jose's paper. There is a manifold $\mathscr{M}$ of dimension d and with metric $g_{\mu\nu}\mathrm{d}x^\mu\mathrm{d}x^\nu = \eta_{ab}e^a \otimes e^b$

$$\mathcal{R}^{(k)} = \epsilon_{a_1\ldots a_d}R^{a_1a_2} \wedge R^{a_3a_4} \wedge \cdots \wedge R^{a_{2k-1}a_{2k}} \wedge e^{a_{2k+1}} \wedge e^{a_{k+2}} \wedge \cdots \wedge e^{a_d}\,. \tag{4.11}$$

for $k = 2$

$$\mathcal{R}^{(2)} = \epsilon_{a_1\ldots a_d}R^{a_1a_2} \wedge R^{a_3a_4} \wedge e^{a_5} \wedge \cdots \wedge e^{a_d}$$

for $k = 1$

$$\mathcal{R}^{(1)} = \epsilon_{a_1\ldots a_d}R^{a_1a_2} \wedge e^{a_3} \wedge \cdots \wedge e^{a_d}\,.$$

for $k = 0$

$$
\begin{aligned}
\mathcal{R}^{(0)} &= \epsilon_{a_1\dots a_d} e^{a_1} \wedge e^{a_2} \wedge \dots \wedge e^{a_d} \qquad \to (d=4) \to \mathcal{R}^{(0)} = \epsilon_{a_1a_2a_3a_4} e^{a_1} \wedge e^{a_2} \wedge e^{a_3} \wedge e^{a_4} \,. \\
\mathcal{R}^{(1)} &= \epsilon_{a_1\dots a_d} R^{a_1a_2} \wedge e^{a_3} \wedge \dots \wedge e^{a_d} \to (d=4) \to \mathcal{R}^{(1)} = \epsilon_{a_1a_2a_3a_4} R^{a_1a_2} \wedge e^{a_3} \wedge e^{a_4} \,. \\
\mathcal{R}^{(2)} &= \epsilon_{a_1\dots a_d} R^{a_1a_2} \wedge R^{a_3a_4} \wedge e^{a_5} \wedge \dots \wedge e^{a_d} \to (d=4) \to \mathcal{R}^{(2)} = \epsilon_{a_1a_2a_3a_4} R^{a_1a_2} \wedge R^{a_3a_4}
\end{aligned}
$$

$$
\frac{1}{2!} R^{ab} \wedge e^c \wedge e^d \epsilon_{abcd} \,,
$$

.with tanh or Swithc

Con f rescaled with cos works fine for spaces Asymtptiocally (A)dS.

With f without the rescaling with cos. works para asymptotically flat. In that case we dont manage to reach $\pi/2$ in the analytic solutions

5 Comments on Torch

Here I want to write a small comment on `torch` and `torch.sum`. Let us consider a tensor of 29×29

$$
\mathtt{T} = \left(\begin{array}{c|c|c|c} & & \cdots & \\ \hline \cdots & & & \cdots \\ \hline & \cdots & & \\ \hline & & \cdots & \end{array} \right)_{29\times 29}
$$

if we consider $\mathtt{torch.sum(input, dim, keepdim} = \mathtt{bol})$ where input must be `T` and $\mathtt{dim} \in \{0,1\}$, for our 29×29 tensor, will sum on the entried along that leg. Finally $\mathtt{bol} = \mathtt{True}$ will keep the dimension of the tensor, and $\mathtt{bol} = \mathtt{False}$ is the for default option and will erase the leg that we sum. For instance

$$
\mathtt{T.torch.sum(0, keepdim} = \mathtt{True}) = \left(\begin{array}{c|c|c|c} & \vdots & & \\ + & + & + & + \\ \vdots & & \vdots & \vdots \\ + & + & + & + \\ & & & \vdots \end{array} \right) = \begin{pmatrix} \bullet & \bullet & \dots & \bullet \end{pmatrix}_{27\times 1} \tag{5.1}
$$

$$
\mathtt{T.torch.sum(1, keepdim} = \mathtt{True}) = \left(\begin{array}{c} \cdots \;\; + \;\; \cdots \;\; + \\ \hline \cdots \;\; + \;\; \cdots \;\; + \\ \hline \cdots \;\; + \cdots \;\; + \\ \hline \cdots \;\; + \;\; \cdots + \end{array} \right) = \begin{pmatrix} \bullet \\ \bullet \\ \vdots \\ \bullet \end{pmatrix}_{1\times 27} \tag{5.2}
$$

The default option $\texttt{keepdim} = \texttt{True}$ will return in both cases a 27×1 array.

Now, if we want to find a probability distribution form `T`, we would like to divide `T` by certain sum. For instance, if we want that each row is a probability distribution we would like to divide each entry of a given row by the sum in (5.2). So we want to be concrete let us consider `T` being a 3 by 3 matrix. Then

$$\texttt{T} = \begin{pmatrix} t_{00} & t_{01} & t_{02} \\ t_{10} & t_{11} & t_{12} \\ t_{20} & t_{21} & t_{22} \end{pmatrix}_{3\times 3} ,$$

$$\texttt{T.torch.sum}(0, \texttt{keepdim} = \texttt{True}) = \left(\begin{array}{ccc} t_{00} & +t_{01} & +t_{02} \\ \hline t_{10} & +t_{11} & +t_{12} \\ \hline t_{20} & +t_{21} & +t_{22} \end{array}\right) \equiv \begin{pmatrix} s_0 \\ s_1 \\ s_2 \end{pmatrix}_{1\times 3}$$

Broadcasting allows us to compute what we want as a simple division

$$\begin{pmatrix} t_{00}/s_0 & t_{01}/s_0 & t_{02}/s_0 \\ t_{10}/s_1 & t_{11}/s_1 & t_{12}/s_1 \\ t_{20}/s_2 & t_{21}/s_2 & t_{22}/s_2 \end{pmatrix}_{3\times 3} = \begin{pmatrix} t_{00} & t_{01} & t_{02} \\ t_{10} & t_{11} & t_{12} \\ t_{20} & t_{21} & t_{22} \end{pmatrix}_{3\times 3} \Big/ \begin{pmatrix} s_0 \\ s_1 \\ s_2 \end{pmatrix}_{1\times 3} = \texttt{T/T.torch.sum}(0, \texttt{keepdim} = \texttt{True})$$

What the division is doing is is the following

$$\begin{pmatrix} t_{00} & t_{01} & t_{02} \\ t_{10} & t_{11} & t_{12} \\ t_{20} & t_{21} & t_{22} \end{pmatrix}_{3\times 3} \Big/ \begin{pmatrix} s_0 \\ s_1 \\ s_2 \end{pmatrix}_{1\times 3} \equiv \begin{pmatrix} t_{00} & t_{01} & t_{02} \\ t_{10} & t_{11} & t_{12} \\ t_{20} & t_{21} & t_{22} \end{pmatrix}_{3\times 3} \Big/ \begin{pmatrix} s_0 & s_0 & s_0 \\ s_1 & s_1 & s_1 \\ s_2 & s_2 & s_2 \end{pmatrix}_{3\times 3} = \begin{pmatrix} t_{00}/s_0 & t_{01}/s_0 & t_{02}/s_0 \\ t_{10}/s_1 & t_{11}/s_1 & t_{12}/s_1 \\ t_{20}/s_2 & t_{21}/s_2 & t_{22}/s_2 \end{pmatrix}$$

so, it wanted to do a element-by-element product, for that it needs to match the dimensions $\binom{3\times 3}{1\times 3} \to \binom{3\times 3}{3\times 3}$ which is done by repeating the elements into the extra direction. In the last equality, the pairwise product was performed.

Problem to be aware: If we consider

$$\texttt{T.torch.sum}(0) = \text{”} \left(\begin{array}{c|c|c|c} & \vdots & & \\ + & + & + & + \\ \vdots & & \vdots & \vdots \\ + & + & + & + \\ & & & \vdots \end{array}\right) \text{”} = [\,\bullet \quad \bullet \quad \ldots \quad \bullet\,] \equiv [v_1, v_2, v_3]_3$$

It happen that `T/T.torch.sum(0)` is broadcastable and what it is doing is the following

$$
\begin{aligned}
\texttt{T/T.torch.sum(0)} &= \begin{pmatrix} t_{00} & t_{01} & t_{02} \\ t_{10} & t_{11} & t_{12} \\ t_{20} & t_{21} & t_{22} \end{pmatrix}_{3\times 3} \Bigg/ [v_1, v_2, v_3]_3 = \\
&= \begin{pmatrix} t_{00} & t_{01} & t_{02} \\ t_{10} & t_{11} & t_{12} \\ t_{20} & t_{21} & t_{22} \end{pmatrix}_{3\times 3} \Bigg/ \begin{pmatrix} v_1 & v_2 & v_3 \\ v_1 & v_2 & v_3 \\ v_1 & v_2 & v_3 \end{pmatrix}_{3\times 3} = \begin{pmatrix} t_{00}/v_1 & t_{01}/v_2 & t_{02}/v_1 \\ t_{10}/v_1 & t_{11}/v_2 & t_{12}/v_2 \\ t_{20}/v_1 & t_{21}/v_2 & t_{22}/v_3 \end{pmatrix}
\end{aligned}
$$

therefore the *columns* are normalized now.

6 My neural network

Let us define a neural network with 1 hidden layer, one input layer and one output layer. Then the structure of the number of neurons is $\vec{N} = (1, n_2, 1)$. The activation matrices are

$$
x \quad \to \quad z^2 = W^{2,1}x + b^2 \quad \to \quad a^2 = \sigma_2(z^2) \quad \to \quad z^3 = W^{3,2}.a^2 + b^3 \quad \to \quad a^3 = \sigma_3(z^3)\,. \tag{6.1}
$$

where

$$
\begin{aligned}
W^{2,1} &\in \mathbb{R}^{n_2} \otimes \mathbb{R} \qquad & W^{3,2} &\in \mathbb{R} \otimes \mathbb{R}^{n_2}\,, \\
b^2 &\in \mathbb{R}^{n_2} & b^3 &\in \mathbb{R}\,. \\
a^2 &\in \mathbb{R}^{n_2} & a^3 &\in \mathbb{R}
\end{aligned}
$$

Then, the chain of evaluations is as follows

$$
\begin{aligned}
f(x) \equiv a^3_\square &= \sigma_3(z^3_\square) \\
z^3_\square &= \sum_j W^{3,2}_{\square j} a^2_j + b^3_\square \\
a^2_i &= \sigma_2(z^2_i) \\
z^2_i &= W^{2,1}_{i\square} x + b^2_i\,,
\end{aligned}
$$

Then

$$\begin{aligned}
f'(x) &= \sigma_3'(z_\Box^3)\frac{\mathrm{d}z_\Box^3}{\mathrm{d}x}\,,\\
\frac{\mathrm{d}z_\Box^3}{\mathrm{d}x} &= \sum_j W_{\Box j}^{3,2}\frac{\mathrm{d}a_j^2}{\mathrm{d}x}\,,\\
\frac{\mathrm{d}a_j^2}{\mathrm{d}x} &= \sigma_2'(z_j^2)\frac{\mathrm{d}z_j^2}{\mathrm{d}x}\,,\\
\frac{\mathrm{d}z_j^2}{\mathrm{d}x} &= W_{j\Box}^{2,1}
\end{aligned}$$

replacing the chain rule

$$f'(x) = \sigma_3'(z_\Box^3)\sum_j W_{\Box j}^{3,2}\sigma_2'(z_j^2)W_{j\Box}^{2,1}\,. \tag{6.2}$$

If $\sigma_3(z_\Box^3) = xz_\Box^3$ then $\frac{\mathrm{d}}{\mathrm{d}x}[\sigma_3(z_\Box^3)] = z_\Box^3 + x\frac{\mathrm{d}z_\Box^3}{\mathrm{d}x}$The

$$\begin{aligned}
f'(x) &= z_\Box^3 + x\frac{\mathrm{d}z_\Box^3}{\mathrm{d}x}\,,\\
\frac{\mathrm{d}z_\Box^3}{\mathrm{d}x} &= \sum_j W_{\Box j}^{3,2}\frac{\mathrm{d}a_j^2}{\mathrm{d}x}\,,\\
\frac{\mathrm{d}a_j^2}{\mathrm{d}x} &= \sigma_2'(z_j^2)\frac{\mathrm{d}z_j^2}{\mathrm{d}x}\,,\\
\frac{\mathrm{d}z_j^2}{\mathrm{d}x} &= W_{j\Box}^{2,1}
\end{aligned}$$

Then

$$f'(x) = z_\Box^3 + x\sum_j W_{\Box j}^{3,2}\sigma_2'(z_j^2)W_{j\Box}^{2,1}$$

6.1 Generalization

$$\begin{aligned}
f(x) &= O(z_\Box^4, x)\,, & z_\Box^4 &= \sum_i w_{\Box i}^{4,3}a_i^3 + b^4\,,\\
a_i^3 &= \sigma_3(z_i^3) & z_i^3 &= \sum_j w_{ij}^{3,2}a_j^2 + b_i^3\,,\\
a_i^2 &= \sigma_2(z_i^2) & z_i^2 &= w_{i\Box}^{2,1}a_\Box^1 + b_i^2\,,\\
a_\Box^1 &= x
\end{aligned}$$

where $O(z, x)$ is a function of two variables that will allow to impose a constraint of the form

$O(z,x)|_{x=x_0} = f_0$, where $x_0, f_0 \in \mathbb{R}$. For instance

$$O(z^N, x) = (1 - e^{x-x_0}) z^N_\square + f_0 \,. \tag{6.3}$$

recall that $z^N_\square$, depends on x in a non-linear way. The derivative of $f(x)$ is

$$\begin{aligned}
f'(x) &= O^{[1,0]}(z^4_\square, x) \frac{\mathrm{d}z^4_\square}{\mathrm{d}x} + O^{[0,1]}(z^4, x)\,, \\
\frac{\mathrm{d}z^4_\square}{\mathrm{d}x} &= \sum_i w^{4,3}_{\square i} \frac{\mathrm{d}a^3_i}{\mathrm{d}x}\,, &\qquad \frac{\mathrm{d}a^3_i}{\mathrm{d}x} &= \sigma_3'(z^3_i) \frac{\mathrm{d}z^3_i}{\mathrm{d}x}\,, \\
\frac{\mathrm{d}z^3_i}{\mathrm{d}x} &= \sum_j w^{3,2}_{ij} \frac{\mathrm{d}a^2_j}{\mathrm{d}x}\,, & \frac{\mathrm{d}a^2_j}{\mathrm{d}x} &= \sigma_2'(z^2_j) \frac{\mathrm{d}z^2_j}{\mathrm{d}x}\,, \\
\frac{\mathrm{d}z^2_j}{\mathrm{d}x} &= w^{2,1}_{j\square}
\end{aligned}$$

Plugging back

$$\begin{aligned}
f'(x) &= O^{[1,0]}(z^4_\square, x) \sum_i w^{4,3}_{\square i} \sigma_3'(z^3_i) \sum_j w^{3,2}_{ij} \sigma_2'(z^2_j) w^{2,1}_{j\square} + O^{[0,1]}(z^4, x)\,, \\
&= O^{[1,0]}(z^4_\square, x) \sum_i w^{4,3}_{\square i} \sigma_3'(z^3_i) \left(\sum_j w^{3,2}_{ij} \sigma_2'(z^2_j) w^{2,1}_{j\square} \right) + O^{[0,1]}(z^4, x)\,,
\end{aligned}$$

This can be generalized as follows

$$\begin{aligned}
f'(x) &= O^{[0,1]}(z^N, x) + \\
&+ O^{[1,0]}(z^N_\square, x) \sum_{i_1} w^{N,N_1}_{\square i_1} \sigma'_{N_1}(z^{N_1}_{i_1}) \left(\sum_{i_2} w^{N_1,N_2}_{i_1 i_2} \sigma'_{N_2}(z^{N_2}_{i_2}) \right) \left(\sum_{i_3} w^{N_2,N_3}_{i_2 i_3} \sigma'_{N_3}(z^{N_3}_{i_3}) \right) (\dots) \\
&= O^{[0,1]}(z^N_{i_{q-1}}, x) + O^{[1,0]}(z^N_{i_{q-1}}, x) \prod_{q=1}^{N-1} \sum_{i_q} w^{N_{q-1},N_q}_{i_{q-1} i_q} \sigma'_{N_q}(z^{N_q}_{i_q})\,.
\end{aligned} \tag{6.4}$$

where we identified i_{q-1}, for us runs for one index only, that we called $\square$. Also we introduced $N_q = N - q$.

We can invert the product by defining $q = N - p$, by

$$\begin{aligned}
\prod_{q=1}^{N-1} S_q &= S_1 S_2 S_3 S_4 \dots S_{N-4} S_{N-3} S_{N-2} S_{N-1}\,, \\
&= \prod_{p=1}^{N-1} S_{N-p}
\end{aligned}$$

Thus $N_q = N - q = N - (N-p) = p$, $N_{q-1} = p + 1$ and $i_q = i_{N-p} \equiv j_p$ and $i_{q-1} = i_{N-p-1} \equiv j_{p+1}$

another dummy index

$$\begin{aligned}\prod_{q=1}^{N-1}\sum_{i_q} w^{N,N_q}_{i_{q-1}i_q}\sigma'_{N_q}(z^{N_q}_{i_q}) &= \prod_{p=1}^{N-1}\sum_{j_p} w^{p+1,p}_{j_{p+1}j_p}\sigma'_p(z^p_{j_p}) \\ &= \left(\sum_{j_3} w^{4,3}_{j_4j_3}\sigma'_3(z^3_{j_3})\right)\left(\sum_{j_2} w^{3,2}_{j_3j_2}\sigma'_2(z^2_{j_2})\right)\left(\sum_{j_1} w^{2,1}_{j_2j_1}\sigma'_1(z^1_{j_1})\right)\end{aligned}$$

Therefore, we can compute the derivative of a 1-variable function as

$$f'(x) = O^{[0,1]}(z^N_{i_{q-1}}, x) + O^{[1,0]}(z^N_{i_{q-1}}, x)\mathbb{P}\,, \tag{6.5}$$

$$\mathbb{P} = \prod_{q=1}^{N-1}\sum_{i_q} w^{N,N_q}_{i_{q-1}i_q}\sigma'_{N_q}(z^{N_q}_{i_q}) = \prod_{p=1}^{N-1}\sum_{j_p} w^{p+1,p}_{j_{p+1}j_p}\sigma'_p(z^p_{j_p})\,. \tag{6.6}$$

Useful definition? We can define

$$J^{2,1}_{j_2} = \sum_{j_1} w^{2,1}_{j_2j_1}\sigma'_1(z^1_{j_1})\,, \tag{6.7}$$

$$J^{3,2}_{j_3} = \sum_{j_2} w^{3,2}_{j_3j_2}\sigma'_2(z^2_{j_2})J^{2,1}_{j_2}\,, \tag{6.8}$$

$$J^{4,3}_{j_4} = \sum_{j_3} w^{4,3}_{j_4j_3}\sigma'_3(z^3_{j_3})J^{3,2}_{j_3}\,.$$

Then

$$\mathbb{P}|_{N=4} = \left(\sum_{j_3} w^{4,3}_{j_4j_3}\sigma'_3(z^3_{j_3})\right)\left(\sum_{j_2} w^{3,2}_{j_3j_2}\sigma'_2(z^2_{j_2})J^{2,1}_{j_2}\right)\,, \tag{6.9}$$

$$= \left(\sum_{j_3} w^{4,3}_{j_4j_3}\sigma'_3(z^3_{j_3})J^{3,2}_{j_3}\right)\,, \tag{6.10}$$

$$= J^{4,3}_{j_4}$$

So the recursive definition is

$$J^{p+1,p}_{j_p} = \sum_{j_p} w^{p+1,p}_{j_{p+1}j_p}\sigma'_p(z^p_{j_p})J^{p,p-1}_{j_p}\,, \qquad p = 1,\ldots,N-1 \tag{6.11}$$

with the first defined as

$$J^{2,1}_{j_2} = \sum_{j_1} w^{2,1}_{j_2j_1}\sigma'_1(z^1_{j_1})\,. \tag{6.12}$$

Proposition 1. *The derivative of a feedfoward neural network is of the form*

$$\frac{\mathrm{d}f(x)}{\mathrm{d}x} = O^{[0,1]}(z^N_{i_{q-1}}, x) + O^{[1,0]}(z^N_{i_{q-1}}, x) J^{N,N-1}\,, \tag{6.13}$$

where $J^{N,N-1}$ is given in a recursive manner as follows

$$J^{p+1,p}_{j_{p+1}} = \sum_{j_p} w^{p+1,p}_{j_{p+1} j_p} \sigma'_p(z^p_{j_p}) J^{p,p-1}_{j_p} \equiv \frac{\mathrm{d}z^{p+1}_{j_p}}{\mathrm{d}x}\,, \qquad\qquad p = 1, \ldots, N-1\,.$$

and the last first is fixed to be

$$J^{2,1}_{j_2} = \sum_{j_1} w^{2,1}_{j_2 j_1} \sigma'_1(z^1_{j_1})\,. \tag{6.14}$$

Note that the derivative of the derivative coefficient is

$$\begin{aligned} \frac{\mathrm{d}J^{p+1,p}_{j_p}}{\mathrm{d}x} &= \sum_{j_p} w^{p+1,p}_{j_{p+1} j_p} \left(\sigma''_p(z^p_{j_p}) \frac{\mathrm{d}z^p_{j_p}}{\mathrm{d}x} J^{p,p-1}_{j_p} + \sigma'_p(z^p_{j_p}) \frac{\mathrm{d}J^{p,p-1}_{j_p}}{\mathrm{d}x} \right)\,, \\ &= \sum_{j_p} w^{p+1,p}_{j_{p+1} j_p} \left(\sigma''_p(z^p_{j_p}) J^{p,p-1}_{j_p} J^{p,p-1}_{j_p} + \sigma'_p(z^p_{j_p}) \frac{\mathrm{d}J^{p,p-1}_{j_p}}{\mathrm{d}x} \right)\,, \end{aligned} \tag{6.15}$$

So we can define another symbol for this

$$K^{p+1,p}_{j_{p+1}} \equiv \sum_{j_p} w^{p+1,p}_{j_{p+1} j_p} \left(\sigma''_p(z^p_{j_p}) (J^{p,p-1}_{j_p})^2 + \sigma'_p(z^p_{j_p}) K^{p,p-1}_{j_p} \right)\,. \tag{6.16}$$

Now for $p = 2$, namely the first. We have that

$$K^{3,2}_{j_3} = \sum_{j_2} w^{3,2}_{j_3 j_2} \left(\sigma''_2(z^2_{j_2}) (J^{2,1}_{j_2})^2 + \sigma'_2(z^2_{j_2}) K^{2,1}_{j_2} \right) \tag{6.17}$$

where

$$K^{2,1}_{j_2} = \frac{\mathrm{d}J^{2,1}_{j_2}}{\mathrm{d}x} = \sum_{j_1} w^{2,1}_{j_2 j_1} \sigma''_1(z^1_{j_1}) \frac{\mathrm{d}z^1_{j_1}}{\mathrm{d}x} = \sum_{j_1} w^{2,1}_{j_2 j_1} \sigma''_1(z^1_{j_1}). \tag{6.18}$$

Proposition 2. *The second derivative of the network is given by*

$$\frac{\mathrm{d}^2 f(x)}{\mathrm{d}x^2} = O^{[0,2]}(z^N_{\square}, x) + O^{[2,0]}(z^N_{\square}, x)(J^{N,N-1}_{\square})^2 + O^{[1,0]}(z^N_{\square}, x) K^{N,N-1}\,,$$

where $K^{N,N-1}$ is given by the following recursive expression for $p = N-1$

$$K^{p+1,p}_{j_{p+1}} \equiv \sum_{j_p} w^{p+1,p}_{j_{p+1}j_p} \left(\sigma''_p(z^p_{j_p})(J^{p,p-1}_{j_p})^2 + \sigma'_p(z^p_{j_p})K^{p,p-1}_{j_p}\right) , \qquad p = 2, \dots N-1 .u \tag{6.19}$$

subject to the condition

$$K^{2,1} = 0 , \tag{6.20}$$

if σ_1 is a linear function in the input.

we have identified x with $z^1_{j_1}$, for j_1 taking only one value.

$$K^{2,1}_{j_2} = \sum_{j_p} w^{2,1}_{j_2 j_1} \left(\sigma''_1(z^1_{j_1})(J^{1,0}_{j_1})^2 + \sigma'_1(z^1_{j_1})K^{1,0}_{j_0}\right) .$$

rIn the simplified case of $N = 4$ we have

$$\begin{aligned}
f'(x) &= O^{[1,0]}(z^4_\Box, x)\frac{\mathrm{d}z^4_\Box}{\mathrm{d}x} + O^{[0,1]}(z^4, x) , \\
\frac{\mathrm{d}z^4_\Box}{\mathrm{d}x} &= \sum_i w^{4,3}_{\Box i}\sigma'_3(z^3_i)\frac{\mathrm{d}z^3_i}{\mathrm{d}x} \equiv \sum_i w^{4,3}_{\Box i}\sigma'_3(z^3_i)J^{3,2}_i \equiv J^{4,3} , \qquad && \frac{\mathrm{d}a^3_i}{\mathrm{d}x} = \sigma'_3(z^3_i)\frac{\mathrm{d}z^3_i}{\mathrm{d}x} , \\
\frac{\mathrm{d}z^3_i}{\mathrm{d}x} &= \sum_j w^{3,2}_{ij}\sigma'_2(z^2_j)\frac{\mathrm{d}z^2_j}{\mathrm{d}x} , \qquad \frac{\mathrm{d}a^2_j}{\mathrm{d}x} = \sigma'_2(z^2_j)\frac{\mathrm{d}z^2_j}{\mathrm{d}x} , \\
\frac{\mathrm{d}z^2_j}{\mathrm{d}x} &= w^{2,1}_{j\Box}
\end{aligned}$$

6.2 Summary and proof of the previous section

Let us consider a neural network with N layers and the number of neurons per layer is given by the vector $(n_1, n_2, \dots, n_{N-1}, n_N)$. For simplicity let us consider $n_N = 1$, and the only index that it has is called $\Box$. Each layer is a vector space of that we define to be $\mathbb{V}_p \cong \mathbb{R}^{n_p}$ for $p = 1, \dots, N$. The weight matrices are $\boldsymbol{w}^{p+1,p} : \mathbb{V}_p \to \mathbb{V}_{p+1}$ are of dimension $n_{p+1} \times n_p$, and the biases are denoted by $\boldsymbol{b}^p \in \mathbb{V}_p$. The input of the layer p is given by

$$\boldsymbol{a}^p = \sigma_p(\boldsymbol{z}^p) , \qquad \boldsymbol{z}^p \equiv \boldsymbol{w}^{p,p-1}\boldsymbol{a}^{p-1} + \boldsymbol{b}^p \qquad \text{for} \qquad p = 2, \dots N . \tag{6.21}$$

subject to the condition

$$\boldsymbol{z}^2 = \boldsymbol{w}^{2,1}\boldsymbol{a}^1 + \boldsymbol{b}^1 \in \mathbb{V}_2 . \tag{6.22}$$

and $\boldsymbol{a}^1 \in \mathbb{V}_1$ being the input of the neural network. We denote the components of the indices with $i, j, k, \dots$. For instance the components of $\boldsymbol{a}^1$ are a^1_k. We will write all the sums explicitly.

Proposition 3. *The derivative of the neural network $f(\boldsymbol{a}^1) \equiv \sigma_N(\boldsymbol{z}^N)$ with respect to the input a^1_k is given by*

$$\frac{\partial f}{\partial a^1_k} = J^N_{\square|k}\,, \tag{6.23}$$

where $J^N_{\square|k}$ are the components of a vector in $\mathbb{V}_N \otimes \mathbb{V}_1$, and is defined through the following recursive formula for $p = N$,

$$J^p_{j_p|k} \equiv \sum_{j_p} w^{p,p-1}_{j_p j_{p-1}} \sigma'_{p-1}(z^{p-1}_{j_{p-1}}) J^{p-1}_{j_{p-1}|k} \equiv \frac{\partial z^{p+1}_{j_{p+1}}}{\partial a^1_k}\,, \qquad p = 2, \ldots, N\,.$$

subject to the initial function for $p = 2$ given by

$$J^{2,1}_{j_2|k} = w^{2,1}_{j_2 k}\,. \tag{6.24}$$

Proof. First, let us define the quantity

$$J^p_{j_p|k} \equiv \frac{\partial z^p_{j_p}}{\partial a^1_k}\,, \qquad p = 2, \ldots, N\,, \tag{6.25}$$

where the indices j_p and k are the components of a vector in $\mathbb{V}_p \otimes \mathbb{V}_1$. For $p = 2$ we have that

$$J^{2,1}_{j_{p+1}|k} = \frac{\partial z^2_{j_2}}{\partial a^1_k} = \frac{\partial}{\partial a^1_k}\Big(\sum_{j_1} w^{2,1}_{j_2 j_1} a^1_{j_1} + b^1_{j_1}\Big) = w^{2,1}_{j_2 k}\,. \tag{6.26}$$

Expanding the definition in (6.25) we find that

$$J^p_{j_p|k} = \frac{\partial}{\partial a^1_k}\left(\sum_{j_{p-1}} w^{p,p-1}_{j_p,j_{p-1}} \sigma_{p-1}(z^{p-1}_{j_{p-1}}) + b^p_{j_p}\right) = \sum_{j_p} w^{p,p-1}_{j_p,j_{p-1}} \sigma'_{p-1}(z^{p-1}_{j_{p-1}}) \frac{\partial z^{p-1}_{j_{p-1}}}{\partial a^1_k}\,. \tag{6.27}$$

Hence, the definition leads the following recursion

$$J^p_{j_p|k} = \sum_{j_{p-1}} w^{p,p-1}_{j_p,j_{p-1}} \sigma'_{p-1}(z^{p-1}_{j_{p-1}}) J^{p-1}_{j_{p-1}|k}\,, \qquad p = 2, \ldots, N\,. \tag{6.28}$$

Now, let us consider the derivative of the neural network with respect to a^1_k, namely the input of the neural network

$$\frac{\partial f(a^1)}{\partial a^1_k} = \sigma'_N(z^N_\square) \frac{\partial z^N_\square}{\partial a^1_k} = \sigma'_N(z^N_\square) J^N_{\square|k}\,, \tag{6.29}$$

where we already computed $J^N_{\square|k}$ in (6.28) for $p = N$. $\square$

Now we ask about the second derivative of the neural network. It can be computed as follows.

Proposition 4. *The second derivative of the Neural network defined as $f(a^1) \equiv \sigma_N(z^N)$ with respect to a^1_k and a^1_l is*

$$\frac{\partial^2 f}{\partial a^1_l \partial a^1_k} = \sigma''_N(z^N_\square) J^N_{\square|l} J^N_{\square|k} + \sigma'_N(z^N_\square) K^N_{\square|kl} \,. \tag{6.30}$$

where $J^N_{\square|k}$ and $K^N_{\square|kl}$ are defined through the following recurrences for $p = N$. The recurrences are defined by

$$J^{p+1}_{j_{p+1}|k} \equiv \sum_{j_p} w^{p+1,p}_{j_{p+1}j_p} \sigma'_p(z^p_{j_p}) J^p_{j_p|k} \,, \qquad for \qquad p = 1, \dots N-1 \,, \tag{6.31}$$

$$K^{p+1}_{j_{p+1}|kl} \equiv \sum_{j_p} w^{p+1,p}_{j_{p+1},j_p} \left(\sigma''_p(z^p_{j_p}) J^p_{j_p|l} J^p_{j_p|k} + \sigma'_p(z^p_{j_p}) K^p_{j_p|kl} \right) \,, \qquad for \qquad p = 2, \dots N-1 \tag{6.32}$$

subject to the conditions $J^2_{j_2|k} = w^{2,1}_{j_2 k}$ and

$$K^3_{j_3|kl} = \sum_{j_2} w^{3,2}_{j_3 j_2} \sigma''_2(z^2_{j_2}) J^2_{j_2|l} J^2_{j_2|k} \,.$$

The object $K^p_{j_p|kl}$ are the components of vector in $\mathbb{V}_p \otimes \mathbb{V}_1 \otimes \mathbb{V}_1$ and it is symmetric on the components along $\mathbb{V}_1 \otimes \mathbb{V}_1$, namely $K^{p+1}_{j_{p+1}|kl} = K^{p+1}_{j_{p+1}|lk}$, $\forall p$.

Proof. First we compoute the second derivative of neural network. Using the result of the previous proposition, we see that

$$\frac{\partial^2 f}{\partial a^1_l \partial a^1_k} = \frac{\partial}{\partial a^1_l} \frac{\partial f}{\partial a^1_k} = \frac{\partial}{\partial a^1_l} (\sigma'_N(z^N_\square) J^N_{\square|k}) = \sigma''_N(z^N_\square) J^N_{\square|l} J^N_{\square|k} + \sigma'_N(z^N_\square) \frac{\partial J^N_{\square|k}}{\partial a^1_l} \,. \tag{6.33}$$

The object that we need to define is the last one. In general we define the components of a tensor in $\mathbb{V}_p \otimes \mathbb{V}_1 \otimes \mathbb{V}_1$ as

$$\begin{aligned} K^p_{j_p|kl} \equiv \frac{\partial J^p_{j_p|k}}{\partial a^1_l} &= \frac{\partial}{\partial a^1_l} \left(\sum_{j_{p-1}} w^{p,p-1}_{j_p,j_{p-1}} \sigma'_{p-1}(z^{p-1}_{j_{p-1}}) J^{p-1}_{j_{p-1}|k} \right) \,, \\ &= \sum_{j_{p-1}} w^{p,p-1}_{j_p,j_{p-1}} \left(\sigma''_{p-1}(z^{p-1}_{j_{p-1}}) \frac{\partial z^{p-1}_{j_{p-1}}}{\partial a^1_l} J^{p-1}_{j_{p-1}|k} + \sigma'_{p-1}(z^{p-1}_{j_{p-1}}) \frac{\partial J^{p-1}_{j_{p-1}|k}}{\partial a^1_l} \right) \,. \end{aligned} \tag{6.34}$$

We can use the definition in the last equality, leading to a recursive definition

$$K^p_{j_p|kl} = \sum_{j_{p-1}} w^{p,p-1}_{j_p,j_{p-1}} \left(\sigma''_{p-1}(z^{p-1}_{j_{p-1}}) J^{p-1}_{j_{p-1}|l} J^{p-1}_{j_{p-1}|k} + \sigma'_{p-1}(z^{p-1}_{j_{p-1}}) K^{p-1}_{j_{p-1}|kl} \right) \,. \tag{6.35}$$

For $p = 2$ and $p = 3$ we find that

$$
\begin{aligned}
K^2_{j_2|kl} &= \frac{\partial J^2_{j_2|k}}{\partial a^1_l} = \frac{\partial w^{2,1}_{j_2k}}{\partial a^1_l} = 0\,. \\
K^3_{j_3|kl} &= \frac{\partial J^3_{j_3|k}}{\partial a^1_l} = \frac{\partial}{\partial a^1_l}(\sum_{j_2} w^{3,2}_{j_3j_2}\sigma'_2(z^2_{j_2})J^2_{j_2|k}) = \frac{\partial}{\partial a^1_l}(\sum_{j_2} w^{3,2}_{j_3j_2}\sigma'_2(z^2_{j_2})w^{2,1}_{j_2k})\,, \\
&= \sum_{j_2} w^{3,2}_{j_3j_2}\sigma''_2(z^2_{j_2})\frac{\partial z^2_{j_2}}{\partial a^1_l}w^{2,1}_{j_2k} = \sum_{j_2} w^{3,2}_{j_3j_2}\sigma''_2(z^2_{j_2})J^2_{j_2|l}J^2_{j_2|k}\,.
\end{aligned}
$$

which is the first value of the recursion for K. The symmetry of the indices in K along the $\mathbb{V}_1$ is direct from the last equation. Finally, using our definition we see that the general formula for the second derivative is

$$
\frac{\partial^2 f}{\partial a^1_l \partial a^1_k} = \sigma''_N(z^N_\square)J^N_{\square|l}J^N_{\square|k} + \sigma'_N(z^N_\square)K^N_{\square|kl}\,. \tag{6.36}
$$

□

Proposition 5. *Adding a final layer to the neural network to satisfy the boundary conditions. $\sigma_N(z^N_\square) \to O(z^N_\square, \boldsymbol{a}^1)$, leads to*

$$
\frac{\partial F(\boldsymbol{a}^1)}{\partial a^1_k} = O^{(1,0)}(z^N_\square, \boldsymbol{a}^1)J^N_{\square|k} + \frac{\partial}{\partial a^1_k}O(z^N, \boldsymbol{a}^1)\,,
$$

for the first derivative and for the second derivative the generalization is as follows

$$
\frac{\partial^2 F(\boldsymbol{a}^1)}{\partial a^1_l \partial a^1_k} = O^{(1,0)}(z^N_\square, \boldsymbol{a}^1)K^N_{\square|lk} + O^{(2,0)}(z^N_\square, \boldsymbol{a}^1)J^N_{\square|l}J^N_{\square|k} + \frac{\partial O^{(1,0)}(z^N_\square, \boldsymbol{a}^1)}{\partial a^1_l}J^N_{\square|k} + \frac{\partial^2 O(z^N, \boldsymbol{a}^1)}{\partial a^1_l \partial a^1_k}\,. \tag{6.37}
$$

Application: Let us consider the case $N = 4$ with a single input, let us call it x. Then, the second derivative is

$$
\frac{\mathrm{d}^2 f}{\mathrm{d}(a^1_x)^2} = \sigma''_N(z^4_\square)J^4_{\square|x}J^4_{\square|x} + \sigma'_N(z^4_\square)K^4_{\square|xx}
$$

with

$$
\begin{aligned}
J^2 &= w^{2,1}\,, \\
J^3 &= w^{3,2}@(\sigma'_2(z) * J^2)\,, \\
J^4_{\square|\square} &= w^{4,3}@(\sigma'_3(z^3) * J^3)\,.
\end{aligned} \tag{6.38}
$$

and

$$K^4_{\square|xx} = \sum_{j_3} w^{4,3}_{j_4,j_3} \left(\sigma''_3(z^3_{j_3}) J^3_{j_3|x} J^3_{j_3|x} + \sigma'_3(z^3_{j_3}) K^3_{j_3|xx} \right) ,$$

$$K^3_{\square|xx} = \sum_{j_2} w^{3,2}_{j_3 j_2} \sigma''_2(z^2_{j_2}) J^2_{j_2|x} J^2_{j_2|x}$$

In torch notation

$$K^4_{|xx} = w^{4,3} @ (\sigma''_3(z^3) * J^3_{|x} J^3_{|x} + \sigma'_3(z^3) * K^3_{|xx}) , \tag{6.39}$$

$$K^3_{|xx} = w^{3,2} @ (\sigma''_2(z^2) * J^2 * J^2) . \tag{6.40}$$

Observation 4-feb:

- The problem of the gradient of b_4 begin not computed was because when I program the loss function the output z4 = w32*a3+b4 was not entering in the loss, therefore b4.grad was None even after the computation of the gradient.
- I implemented the re-sampling on each iteration of the for cycle by adding to each point a small random value. This will create a "shadow" around the point that we give in the initial array.
- The second derivative in one and two variables work fine.
- Imposing boundary conditions as an extra term in the loss function works fine, but the strength of that term must be controlled case by case probably.
- There was a problem by solving the harmonic oscillator equation. I put $\vec{N} = (1, 20, 20, 1)$, but was not enough to fit the solution. Then I put $(1, 40, 20, 1)$ and it works immediately.
- I manage to impose boundary conditions by adding a term in the loss function

I proceed in the following way. First you create your training set,

$$\mathbf{x}_{\text{train}} = [x_0, x_1] \times [y_0, y_1] . \tag{6.41}$$

where the square bracket means a set of N_x and N_y points between x_0, x_1 and y_0, y_1, respectively. Then you define your contour $\mathscr{C}$ to be rectangular (we restrict ourselves to rectangular contours) and is constructed as

$$\mathscr{C} = \mathbb{I} \sqcup \mathbb{II} \sqcup \mathbb{III} \sqcup \mathbb{IV} , \tag{6.42}$$

where

$$\mathbb{I} = \{x_0\} \times [y_0, y_1]\,, \qquad \mathbb{II} = [x_0, x_1] \times \{y_0\}\,, \qquad \mathbb{III} = \{x_1\} \times [y_1, y_1]\,, \qquad \mathbb{IV} = [x_0, x_1] \times \{y_1\}\,. \tag{6.43}$$

In the end, $\mathscr{C}$ will be a set with $2N_x + 2N_y$ elements of the form $(x, y) \in \mathbb{R}^2$. Then, we define the boundary value functions

$$B_I(x_0, y)\,, \qquad B_{II}(x, y_0)\,, \qquad B_{III}(x_1, y)\,, \qquad B_{IV}(x, y_2)\,, \tag{6.44}$$

that is the expected value of the function. Then, we construct the following set

$$B(\mathscr{C}) = B_I(\mathbb{I}) \sqcup B_{II}(\mathbb{II}) \sqcup B_{III}(\mathbb{III}) \sqcup B_{IV}(\mathbb{IV})\,. \tag{6.45}$$

which is a set of $2N_x + 2N_y$ elements that are real numbers.

The training process is as follows. You append $\mathscr{C}$ to the training set $\mathbf{x}_{\text{train}}$ and create an ordered list

$$\mathbf{x}_{\text{full}} = \mathbf{x}_{\text{train}} \sqcup \mathscr{C}\,, \tag{6.46}$$

which is a list of $N_x N_y + 2N_x + 2N_y$ elements. Then, we find the values of the neural network $f(\mathbf{x}_{\text{full}})$ that has the same matricial structure. Finally we extract the last $2N_x + 2N_y$ elements that must be substracted with $B(\mathscr{C})$. Its square give a term that can enter the loss function

$$\text{Loss}_{\text{boundary}} = \text{mean}[(f(\mathscr{C}) - B(\mathscr{C}))^2]\,.$$

6.3 Generalization for any number of inputs

Let us consider a neural network with the following structure $\vec{N} = (2, n_2, \ldots, n_{N-1}, 1)$. This is function of n_1 inputs and one output. It is multivariable function $F(x, y)$,

$$F(x, y) = O(a^N_\square; x, y)\,, \tag{6.47}$$

where the output function is $O(a^N_\square; x, y)$ is an function that allow us to satisfy the boundary conditions. For instance if the domains is $x \in [0, L]$ and we want a s certain boundary condition $F(0, y) = F(L, y) = \gamma(y)$. We can consider for instance

$$O(a^N_\square; x, y) = \sin(2\pi x/L) a^N_\square + \gamma(y)\,. \tag{6.48}$$

The boundary conditions in $y \in [0, L]$, let us say $F(x, 0) = F(x, L) = \chi(x)$. Must be put in a clever way.´

Then, the derivative of F with respect to the first argument is

$$\frac{\partial F}{\partial x^k} = \frac{\partial O}{\partial x^k}(a^N_\square;\vec{x}) + O^{(1,0,0)}(a^N_\square;\vec{x})\frac{\mathrm{d}a^N_\square}{\mathrm{d}x^k}\,, \tag{6.49}$$

where

$$\frac{\mathrm{d}a^N_\square}{\mathrm{d}x^k} = J^{N,N-1}_{(k)j_p}\,, \tag{6.50}$$

and

$$J^{p+1,p}_{j_p(k)} = \sum_{j_p} w^{p+1,p}_{j_{p+1}j_p}\sigma'_p(z^p_{j_p})J^{p,p-1}_{j_p(k)}\,, \qquad\qquad p=1,\dots,N-1 \tag{6.51}$$

with the first term defined as

$$J^{2,1}_{j_2(k)} = \sum_{j_1} w^{2,1}_{j_2j_1}\delta^{j_1}_k = w^{2,1}_{j_2k}\,. \tag{6.52}$$

7 Lovelock cube

Let us consider the theory in a d-dimensional spacetime

$$\mathcal{I} = \sum_{k=0}^{3}\frac{a_k}{d-2k}\int\mathcal{R}^{(k)}\,, \tag{7.1}$$

where

$$\mathcal{R}^{(0)} = \epsilon_{f_1\cdots f_d}e^{f_1\cdots f_d}\,, \tag{7.2}$$
$$\mathcal{R}^{(1)} = \epsilon_{f_1\dots f_d}R^{f_1f_2}\wedge e^{f_3f_4\cdots f_d}\,, \tag{7.3}$$
$$\mathcal{R}^{(2)} = \epsilon_{f_1\dots f_d}R^{f_1f_2}\wedge R^{f_3f_4}\wedge e^{f_5\cdots f_d}\,, \tag{7.4}$$
$$\mathcal{R}^{(3)} = \epsilon_{f_1\dots f_d}R^{f_1f_2}\wedge R^{f_3f_4}\wedge R^{f_5f_6}\wedge e^{f_7\cdots f_d}\,. \tag{7.5}$$

Then the action expand

$$\mathcal{I} = \int\epsilon_{f_1\dots f_d}\Bigg(\frac{a_0}{d}e^{f_1\cdots f_d} + \frac{a_1}{d-2}R^{f_1f_2}\wedge e^{f_3f_4\cdots f_d} + \frac{a_2}{d-4}R^{f_1f_2}\wedge R^{f_3f_4}\wedge e^{f_5\cdots f_d} + \\ + \frac{a_3}{d-6}R^{f_1f_2}\wedge R^{f_3f_4}\wedge R^{f_5f_6}\wedge e^{f_7\cdots f_d}\Bigg)\,. \tag{7.6}$$

where we recall the notation $e^{f_1\cdots f_p} = e^{f_1}\wedge\cdots\wedge e^{f_p}$. $e^a = e^a{}_\mu\mathrm{d}x^\mu$, $g_{\mu\nu} = e^a{}_\mu e^b{}_\nu\eta_{ab}$, $\eta_{ab} = \mathrm{diag}(-1,1\dots,1)$. Then, the equations of motion are coming from the variation with respect to the vielbein

$$\epsilon_{f_1\cdots f_{d-1}a}\left(a_0 e^{f_1\cdots f_{d-1}} + a_1 R^{f_1 f_2}\wedge e^{f_3 f_4\cdots f_{d-1}} + a_2 R^{f_1 f_2}\wedge R^{f_3 f_4}\wedge e^{f_5\cdots f_{d-1}} + \right.$$
$$\left. + a_3 R^{f_1 f_2}\wedge R^{f_3 f_4}\wedge R^{f_5 f_6}\wedge e^{f_7\cdots f_{d-1}}\right) = 0 \qquad (7.7)$$

Let us postulate that this equation can be written as

$$\epsilon_{f_1\cdots f_{d-1}a}\mathcal{F}^{f_1 f_2}_{(1)}\wedge\mathcal{F}^{f_3 f_4}_{(2)}\wedge\mathcal{F}^{f_5 f_6}_{(3)}\wedge e^{f_7\cdots f_{d-1}a} = 0\,, \qquad (7.8)$$

where

$$\mathcal{F}^{ab}_{(i)} = R^{ab} - \Lambda_i e^{ab}\,. \qquad (7.9)$$

In a maximally symmetric spacetime with curvature Λ_i, the curvature 2-form is given by $R^{ab} = \Lambda_i e^{ab}$. Let's exapnd the above equation

$$\epsilon_{f_1\cdots f_{d-1}a}(R^{f_1 f_2} - \Lambda_1 e^{f_1 f_2})\wedge\mathcal{F}^{f_3 f_4}_{(2)}\wedge\mathcal{F}^{f_5 f_6}_{(3)}\wedge e^{f_7\cdots f_{d-1}a}$$

$$\begin{aligned}
&\epsilon(R-\Lambda_1)(R-\Lambda_2)(R-\Lambda_3)\\
&=\epsilon(R(R-\Lambda_2)(R-\Lambda_3)-\Lambda_1(R-\Lambda_2)(R-\Lambda_3))\\
&=\epsilon(RRR-RR\Lambda_3-R\Lambda_2R+R\Lambda_2\Lambda_3-\Lambda_1RR+\Lambda_1\Lambda_2R+\Lambda_1R\Lambda_3-\Lambda_1\Lambda_2\Lambda_3)\,,\\
&=\epsilon(RRR-RR\Lambda_3-RR\Lambda_2-RR\Lambda_1+R\Lambda_2\Lambda_3+R\Lambda_1\Lambda_2+R\Lambda_1\Lambda_3-\Lambda_1\Lambda_2\Lambda_3)\,,\\
&=\epsilon[RRR-RR(\Lambda_3+\Lambda_2+\Lambda_1)+R(\Lambda_2\Lambda_3+\Lambda_1\Lambda_2+\Lambda_1\Lambda_3)-\Lambda_1\Lambda_2\Lambda_3]\,,
\end{aligned}$$

$$\begin{aligned}
&\epsilon_{f_1\cdots f_{d-1}a}(R^{f_1 f_2}-\Lambda_1 e^{f_1 f_2})\wedge\mathcal{F}^{f_3 f_4}_{(2)}\wedge\mathcal{F}^{f_5 f_6}_{(3)}\wedge e^{f_7\cdots f_{d-1}a}\\
&=\epsilon_{f_1\cdots f_{d-1}a}[R^{f_1 f_2}R^{f_3 f_4}R^{f_5 f_6}-R^{f_1 f_2}R^{f_3 f_4}(\Lambda_3+\Lambda_2+\Lambda_1)e^{f_5 f_6}+\\
&+R^{f_1 f_2}(\Lambda_2\Lambda_3+\Lambda_1\Lambda_2+\Lambda_1\Lambda_3)e^{f_3 f_4 f_5 f_6}-\Lambda_1\Lambda_2\Lambda_3 e^{f_1\cdots f_6}]\wedge e^{f_7\cdots f_{d-1}}\,.
\end{aligned}$$

This should be compared with

$$\epsilon_{f_1\cdots f_{d-1}a}\left(a_0 e^{f_1\cdots f_6} + a_1 R^{f_1 f_2}\wedge e^{f_3\cdots f_6} + a_2 R^{f_1 f_2}\wedge R^{f_3 f_4}\wedge e^{f_5\cdots f_6} + \right.$$
$$\left. + a_3 R^{f_1 f_2}\wedge R^{f_3 f_4}\wedge R^{f_5 f_6}\right)\wedge e^{f_7\cdots f_{d-1}} = 0 \qquad (7.10)$$

Another possiblity is assume that the space is maximally symmetric, the

$$R^{ab} = \Lambda e^{ab}\,. \qquad (7.11)$$

Replacing into the equation, we have that

$$\epsilon_{f_1\ldots f_{d-1}a}\left(a_0+a_1\Lambda+a_2\Lambda^2+a_3\Lambda^3\right)e^{f_1\ldots f_{d-1}}=0\,. \tag{7.12}$$

Which is

$$\sum_{k=0}^{3}a_k\Lambda^k=0\,. \tag{7.13}$$

Let us consider a planar black hole with metric

$$\mathrm{d}s^2=-N^2f(r)\mathrm{d}t^2+\frac{\mathrm{d}r^2}{f(r)}+\frac{r^2}{L^2}\sum_{a=2}^{d-1}\mathrm{d}x^a\mathrm{d}x^a\,.$$

Considering the vielbein

$$e^0=N\sqrt{f(r)}\mathrm{d}t\,,\qquad e^1=\frac{1}{\sqrt{f(r)}}\mathrm{d}r\,,\qquad e^a=\frac{r}{L}\mathrm{d}x^a$$

The curvature 2-form are

$$\begin{aligned}R^{01}&=-\frac{1}{2}f''(r)e^0\wedge e^1\,, & R^{0a}&=-\frac{f'(r)}{2r}e^0\wedge e^a\,,\\ R^{1a}&=-\frac{f'}{2r}e^1\wedge e^a\,, & R^{ab}&=-\frac{f(r)}{r^2}e^a\wedge e^b\,.\end{aligned}$$

Replacing into the equations (SHOW!!) the should find

$$\left[\frac{\mathrm{d}}{\mathrm{d}\log r}+(d-1)\right]\left(\sum_{k=0}^{K}a_kg(r)^k\right)=0\,,$$

where $g(r)=-f(r)/r^2$. Note that $\frac{\mathrm{d}}{\mathrm{d}\log r}=r\frac{\mathrm{d}}{\mathrm{d}r}$. The solution of this simple equation is

$$\sum_{k=0}^{K}a_kg(r)^k=\frac{1}{L^2}\left(\frac{r_+}{r}\right)^{d-1}\,. \tag{7.14}$$

For $K=3$, that it what we are interested, the equation reads

$$a_0+a_1g(r)+a_2g(r)^2+a_3g(r)^3=\frac{1}{L^2}\left(\frac{r_+}{r}\right)^{d-1}\,. \tag{7.15}$$

We define $\lambda=a_2/L^2$, $\mu=3a_3/L^4$ and we set the coupling the the Einstein-Hilbert term to one $a_1=1$.

$$\epsilon_{a_1\ldots a_4}R^{a_1a_2}R^{a_3a_4}$$

Problem 1) Consider a black hole ansatz, as usual in $d = 7$

$$\mathrm{d}s^2 = -f(r)\mathrm{d}t^2 + \frac{\mathrm{d}r^2}{f(r)} + r^2(\mathrm{d}x_1^2 + \cdots + \mathrm{d}x_5^2)\,. \tag{7.16}$$

2) Compute the equation evaluated in this ansatz with the Lovelock couplings given by

$$\begin{aligned} a_0 &= \frac{1}{L^2}\,, \\ a_1 &= 1\,, \\ a_2 &= L^2\lambda\,, \\ a_3 &= \frac{L^4\mu}{3}\,. \end{aligned}$$

Note for $\lambda = 0.27, \mu = 0.0635$ there three negative roots of the "vacuum" polynomial, given by

$$\begin{aligned} \Lambda &= -6.304 \\ \Lambda &= -4.236 \\ \Lambda &= -1.615 \end{aligned}$$

3) Note that the AdS configuration in this coordinate with AdS radius L_{AdS} is

$$\mathrm{d}s^2 = -\frac{r^2}{L_{\text{AdS}}^2}\mathrm{d}t^2 + \frac{\mathrm{d}r^2}{\frac{r^2}{L_{\text{AdS}}^2}} + r^2(\mathrm{d}x_1^2 + \cdots + \mathrm{d}x_5^2)\,.$$

then

$$f(r) = \frac{r^2}{L_{\text{AdS}}^2}\,, \tag{7.17}$$

then the boundary condition is

$$f(0) = 0\,. \tag{7.18}$$

$X = [x_0, x_1]$

$$E(x) = Ae^{x-B}$$

$$\begin{aligned} E(x_0) &= x_0\,, \\ E(x_1) &= x_1 \end{aligned}$$

8 Log distribution

Let us define a interval $X = [x_0, x_1] \subset \mathbb{R}$. Let us define a equally spaced particion of it

$$\mathcal{E}_N(x_0, x_1) = \{x_0 + \Delta i \in \mathbb{R} : i = 1, \dots, n \text{ and } \Delta = (x_1 - x_0)/N\} .$$

We define the function

$$\ell(x) = x_0 e^{x - x_0} , \tag{8.1}$$

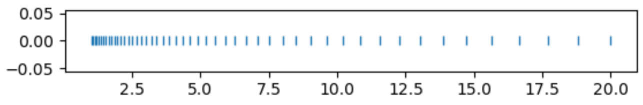
certainly it satisfy $\mathcal{L}(x_0) = x_0$. We define the Log spaced partition of the interval X as

$$\mathcal{L}_N(x_0, x_1) = \{\ell(x) \in \mathbb{R} : x \in S(x_0, x_1)\} ,$$

where the set $S(x_0, x_1)$ is the ordered set

$$S(x_0, x_1) = \{p = x_0 + \Delta i : i = 1, \dots, N \text{ and } \Delta = \log(x_1/x_0)/N\}$$

what this do is the following


0.05
0.00
−0.05
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0