Table of Contents

Neural Differential equations

ResNet

Let us reconsider the ResNet model. It is a neural network architecture that has direct connections of previous layers with the next layer. Formally

$$y_{l+1} = y_l + NN(y_l)$$

where y_l is the outpt of the l^{th} layer.

Euler method

Consider a differential equation

$$y' = f(y, t.)$$

Euler method is a classical method to numerically solve a differential equation. Let h be a step size, then the iteration is denided as

$$y_n = y_{n-1} + hf(y,t)$$

.

Neural Ordinary Differential Equations

The observation that ResNet is an Euler solution with a step-size 1 was noted quite early. This was redecovered in a 2018 in a paper **Neural Ordinary Differential Equations** awarded Best Papers of NeuIPS2018.

The key premise of the paper is that a discrete Residual neural network can be replaced by an architecture that is parametrised by both y and number of layers l.

In addition, since Euler method is prone of large errors, instead of doing Euler approximation as in ResNet, we can utilize some of the **advanced ODE solvers** to train the parameters. Formally it means that a layer is replaced by a ODE solver. The architecture is named as ODENet.

What about back-propogation?

The major technical challange in above idea is the following.

> How to train the ODE-Solver?

Each Neural layer is defines a loss function which can be optimised by gradient decent. Now in place of a layer we have a computer program that need to be 'differentiable' to train. They do this with Adjoint sensitivity analysis.

Equation

Thus we are modelling the system as

$$y'(x) = NN(x,t)$$

DiffEqFlux.jl

These layers are encoded in the DiffEqFlux.jl as NeuralODE function.

Example

This code is taken from documentation github repo. We will explain the code here.

We will consider a set of data points coming from some system goverened by a differntial equations. We will take 30 data points from that system and train an ODE-NET implementation in Julia.

```
(0.0, 1.5)
```

```
begin
    using Flux, DiffEqFlux, DifferentialEquations, Plots

        u0 = Float32[2.; 0.]
        datasize = 30
        tspan = (0.0f0,1.5f0)
end
```

Observations

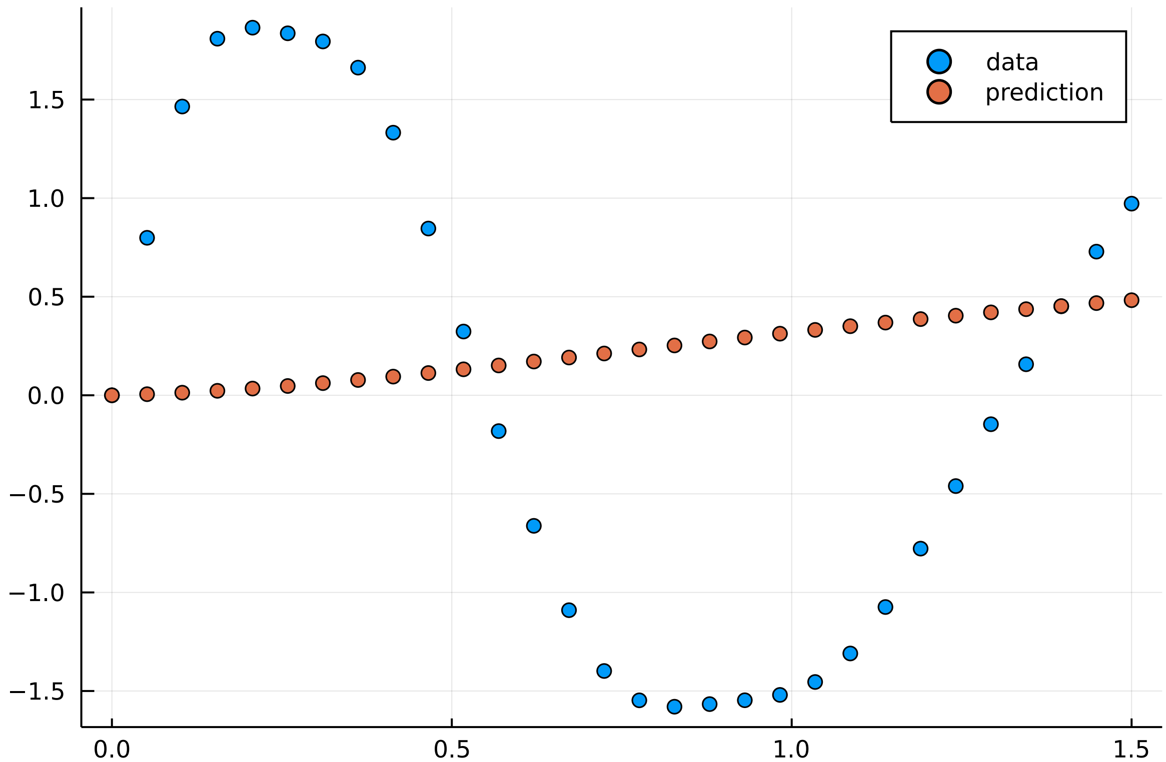
Observations are noted by using a ODE Solver for the supposed equation.

```
2×30 Matrix{Float32}:
 2.0  1.9465    1.74178  1.23837  0.577125  …  1.42516   1.40688   1.37023   1.29215
 0.0  0.798831  1.46473  1.80877  1.86465      0.157376  0.451367  0.728692  0.972095
```

```
begin
    function trueODEfunc(du,u,p,t)
            true_A = [-0.1 2.0; -2.0 -0.1]
            du .= ((u.^3)'true_A)'
        end
        true_A = [-0.1 2.0; -2.0 -0.1]
         ((u0.^3)'true_A)'
        t = range(tspan[1],tspan[2],length=datasize)
        prob = ODEProblem(trueODEfunc,u0,tspan)
        ode_data = Array(solve(prob,Tsit5(),saveat=t))
end
```

Model

The model is implemented in order to predict the time series for any given initial position.




```
begin
        dudt = Chain(x -> x.^3,
                     Dense(2,50,tanh),
                     Dense(50,2))

        n_ode = NeuralODE(dudt,tspan,Tsit5(),saveat=t,reltol=1e-7,abstol=1e-9)
        ps = Flux.params(n_ode)
        pred = n_ode(u0) # Get the prediction using the correct initial condition
        scatter(t,ode_data[1,:],label="data")
        scatter!(t,pred[1,:],label="prediction")

        scatter(t,ode_data[2,:],label="data")
        scatter!(t,pred[2,:],label="prediction")
end
```

Training the model

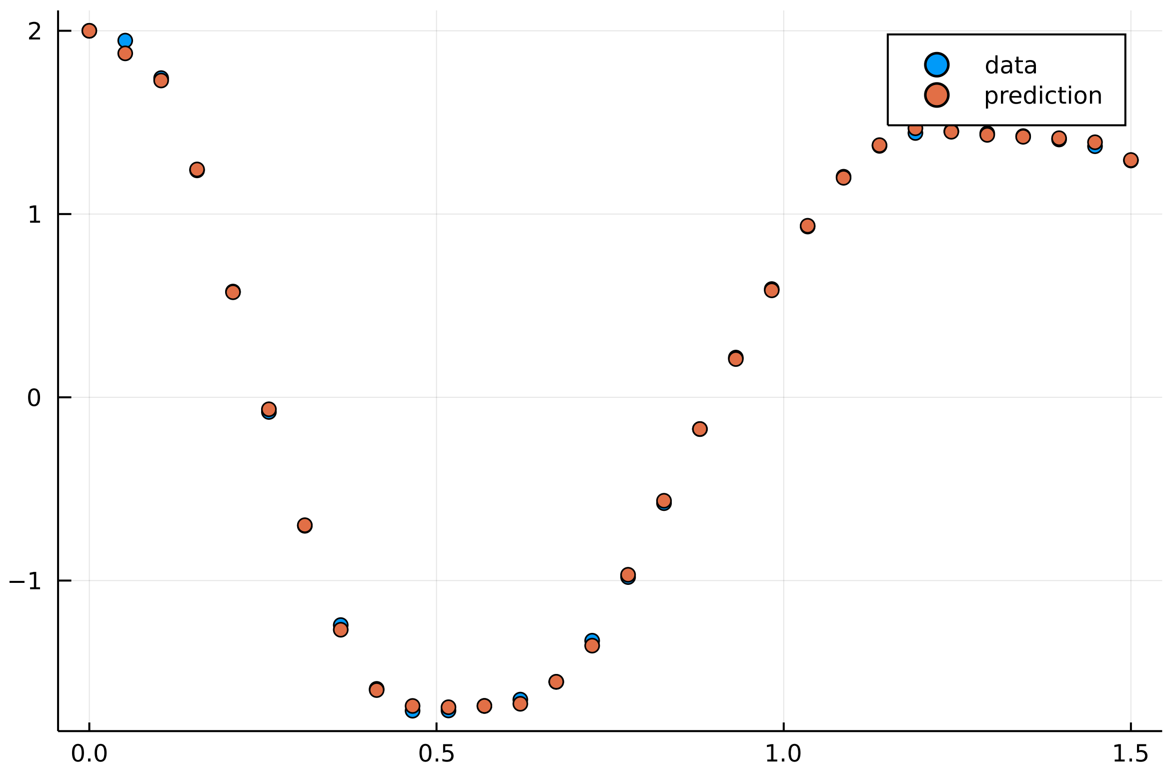
```
begin
    function predict_n_ode()
      n_ode(u0)
    end
    loss_n_ode() = sum(abs2,ode_data .- predict_n_ode())
    data = Iterators.repeated((), 1000)
    opt = ADAM(0.1)
    Flux.train!(loss_n_ode, ps, data, opt)
end
```

pred_new =

	timestamp	value1	value2
1	0.0	2.0	0.0

	timestamp	value1	value2
2	0.0517241	1.87682	0.789107
3	0.103448	1.72876	1.48022
4	0.155172	1.2434	1.82281
5	0.206897	0.573347	1.85506
6	0.258621	-0.0653149	1.82126
7	0.310345	-0.697904	1.78809
8	0.362069	-1.2683	1.68625
9	0.413793	-1.59744	1.36853

```
pred_new = n_ode(u0)
```




```
begin
    pl = scatter(t,ode_data[1,:],label="data")
      scatter!(pl,t,pred_new[1,:],label="prediction")
    plot(pl)
end
```