Projet AI :
Solving Differential Equations with Deep Learning

Mohammed El Hadi Zerga
Abdelkader Metakalard
Aissam Kadri

February 23, 2024

1 Introduction

In this project, we aim to numerically solve ordinary differential equations using neural networks. We will elucidate the feasibility of this approach and showcase the power of the universal approximation theorem of networks, which underlies our ability to model and efficiently solve complex problems through these neural structures.

2 differential equation

First order ODE :

$$\begin{aligned} \frac{dy}{dx} + p(x) \cdot y &= f(x) \\ x &\in [0,1] \\ y(0) &= A \end{aligned} \tag{1}$$

The problem we have just presented is a Cauchy problem, which, with regularity conditions on p and f, admits a unique **continuous** solution over our interval $[0,1]$. Specifying that the solution is **continuous** is important to justify the use of neural networks in estimating the solution to our equation.

3 Universal Approximation Theorem for Neural Networks

The universal approximation theorem is one of the most fundamental concepts in applied mathematics and machine learning. It states that under certain assumptions, a neural network can approximate any continuous function to any desired precision. This property is crucial for a neural network's ability to learn complex patterns and generalize to new data.

For example, a neural network with input $\mathbf{x}$, a single hidden layer with N neurons and a sigmoid activation function σ, and an output layer with 1 neuron using the identity activation function $id(x) = x$.So, the set of all neural networks thus defined can approximate any continuous function (FIGURE 1-).

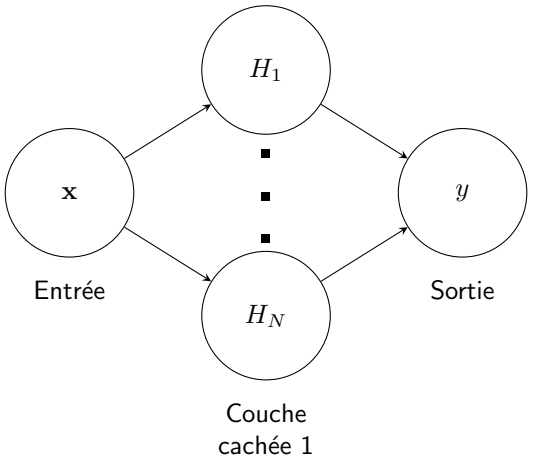

FIGURE 1 –

4 Solving differential equations with Neural Networks

We have observed that neural networks can approximate any continious function. We can assume that the solution to our problem can be estimated as follows :

$$y(x) \approx N(x)$$

where N is a neural network. Our goal is to estimate the parameters of this network to make this approximation as accurate as possible.
Let's substitute into our equation (1) :

$$\begin{gathered} \frac{dN}{dx} + p(x) \cdot N = f(x) \\ x \in [0,1] \\ N(0) = A \end{gathered} \tag{2}$$

which can be rewritten as

$$\begin{gathered} \frac{dN}{dx} + p(x) \cdot N - f(x) = 0 \\ x \in [0,1] \\ N(0) - A = 0 \end{gathered} \tag{3}$$

Now, if we define the Loss function as follows,

$$MSE = MSE_f + MSE_u$$

where

$$MSE_f = \frac{1}{N_f}\sum_{i=1}^{N_f}\left(\frac{dN(x_i)}{dx} - f\left(x_i\right)\right)^2$$

and

$$MSE_u = \left(N\left(0\right) - A\right)^2$$

where $x_i \in [0,1]$ and $0 < i \leq N_f$. It is now evident that by minimizing our loss function, our neural network N will satisfy equation (2) with a certain precision. However, since the equation has only one solution, our neural network will necessarily approximate the solution of the equation.

5 Examples

We will now exemplify the approach through a few instances. To compute the derivatives of our neural network, we will leverage the capabilities of PyTorch's automatic differentiation and perform computations on the GPU.
We will use a neural network to estimate our functions. This network consists of an input layer with one neuron, followed by three hidden layers. The first hidden layer has 20 neurons, the second also has 20, and the third has 30. Lastly, there is an output layer with one neuron (Figure 2).

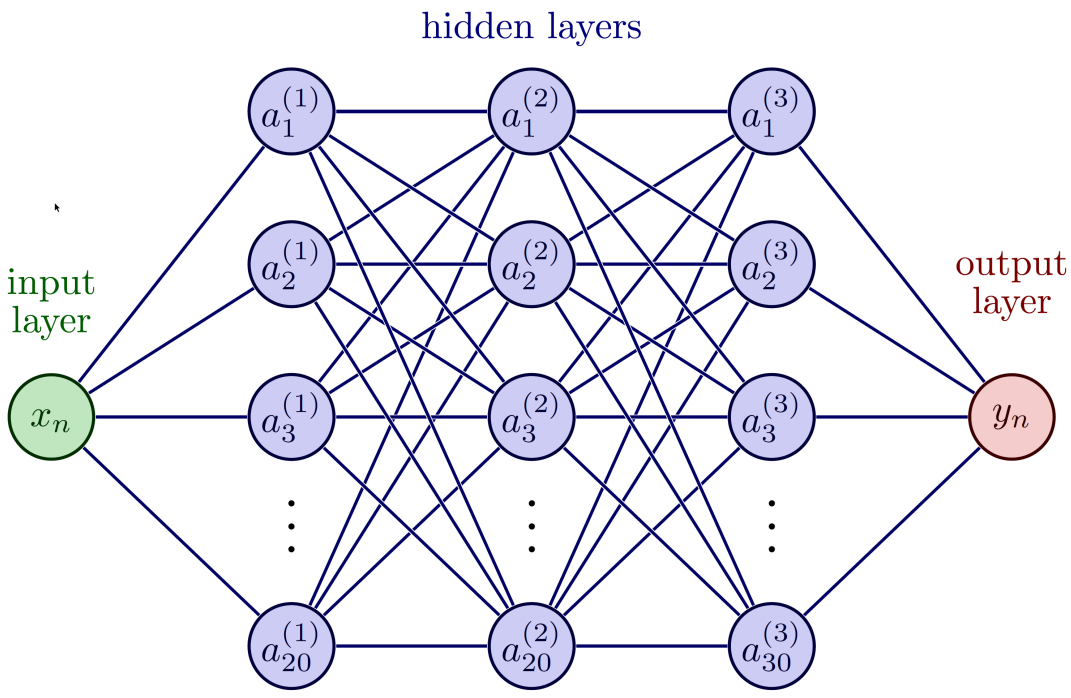

Figure 2 –

We will train our neural network and estimate its parameters for each example to approximate our functions.

We start by defining the neural network that is supposed to estimate our solution.

```
class Network(nn.Module):
    def __init__(self):
        super().__init__()
        self.hidden_layer1 = nn.Linear(1, 20)
        self.hidden_layer2 = nn.Linear(20, 20)
        self.hidden_layer3 = nn.Linear(20, 30)
        self.output_layer = nn.Linear(30, 1)

    def forward(self, x):
        layer1_out = torch.sigmoid(self.hidden_layer1(x))
        layer2_out = torch.sigmoid(self.hidden_layer2(
    layer1_out))
        layer3_out = torch.sigmoid(self.hidden_layer3(
    layer2_out))
        output = self.output_layer(layer3_out)
        return output
```

5.1 Example 1

$$\begin{cases} f(x) = e^x \\ p(x) = 0 \end{cases} \quad .$$
$$y(0) = 1$$

Exact Solution

$$y = e^x$$

We now define the function f and the loss function

```
def f(x):
    return torch.exp(x)

def loss(x):
    x.requires_grad = True
    y = N(x)
    dy_dx = torch.autograd.grad(y.sum(), x,
                        create_graph=True)[0]

    return torch.mean( (dy_dx - f(x))**2 ) +
                            (y[0, 0] - 1.)**2
```

We can now train our model and visualize the results.

```

optimizer = torch.optim.LBFGS(N.parameters())

x = torch.linspace(0, 1, 100)[:, None]

def closure():
    optimizer.zero_grad()
    l = loss(x)
    l.backward()

    return l

epochs = 10
for i in range(epochs):
    optimizer.step(closure)



xx = torch.linspace(0, 1, 100)[:, None]
with torch.no_grad():
    yy = N(xx)

plt.figure(figsize=(10, 6))
plt.plot(xx, yy, label="Predicted")
plt.plot(xx, torch.exp(xx), '--', label="Exact")
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.grid()
```

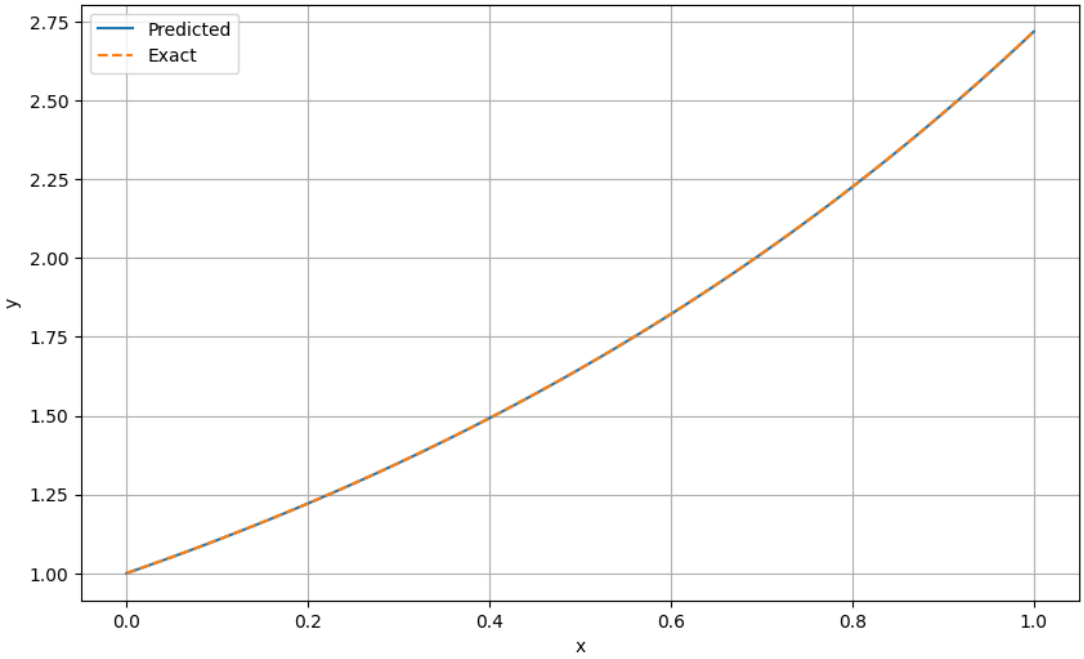
Predicted
Exact
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
y
0.0
0.2
0.4
0.6
0.8
1.0
x

FIGURE 3 –

5.2 Example 2

Example periodic solution

$$\begin{cases} f(x) = cos(x) \\ p(x) = 0 \end{cases} .$$
$$y(0) = 0$$

Exact Solution

$$y = sin(x)$$

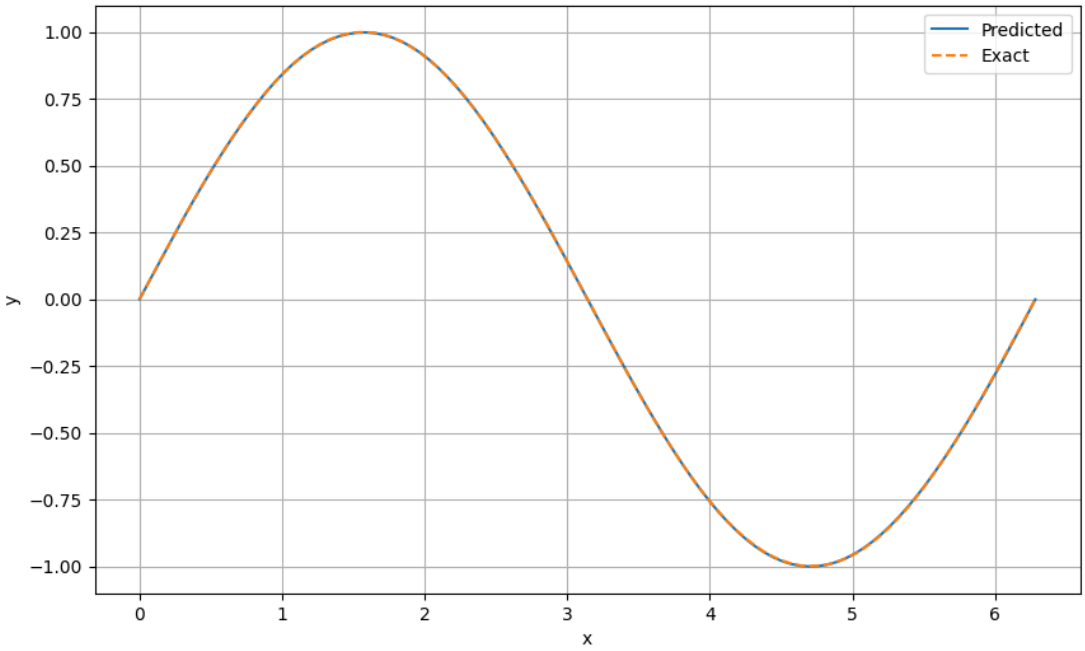

FIGURE 4 –

5.3 Example 3

We can apply the same logic to second-order equations as follows :

$$\frac{d^2y}{dx^2} + p(x)\frac{dy}{dx} + q(x)y = f(x)$$
$$x \in [0, 1]$$
$$y(0) = A$$
$$y(1) = B$$

$$\begin{cases} p(x) = 0 \\ q(x) = 0 \\ f(x) = -1 \end{cases}.$$

$$\begin{cases} y(0) = 0 \\ y(1) = 0 \end{cases}.$$

Exact Solution

$$y(x) = -\frac{1}{2}x^2 + \frac{1}{2}x$$

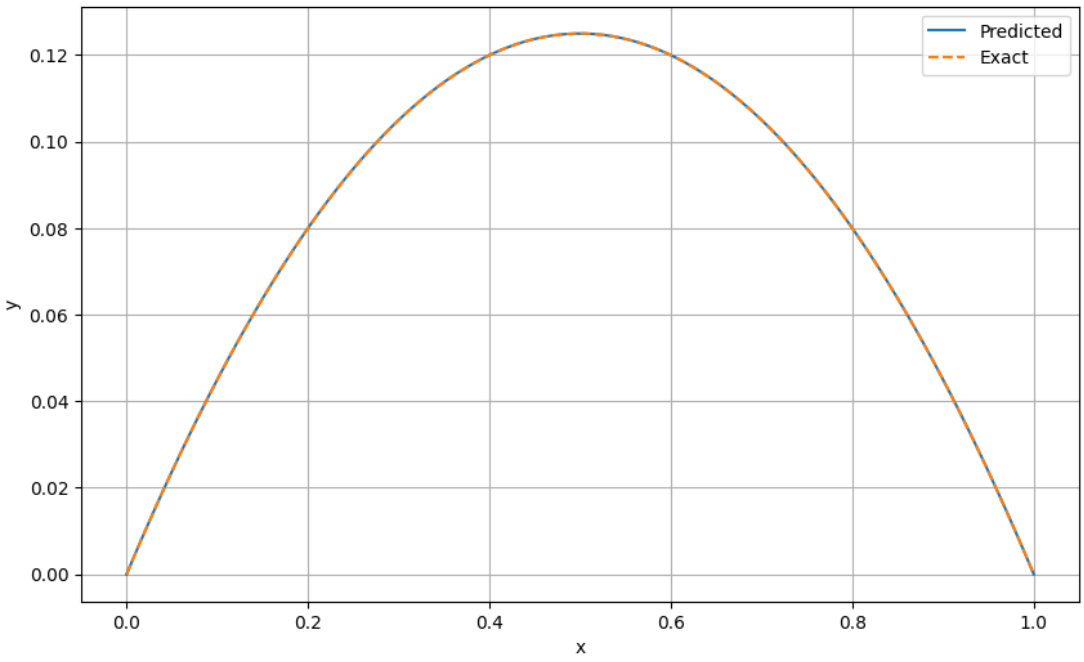

FIGURE 5 –

6 Conclusion

We introduced a numerical solution method for ordinary differential equations using neural networks and their properties, notably their universality. Subsequently, we illustrated our approach with simple examples, showing our ability to achieve precise results. This highlights the versatility and effectiveness of neural networks.