Solve Differential Equation Using Neural Network

Presenter:
Mahdieh Sadat Benis
Hosna Oyarhoseini

Mentor: Amir Hosein Hadian

October-2022

Targets:

- Scientific ML
- Method for solving DE
- Training a neural network in order to his solution satisfies the conditions required by a differential equation

Road map

1 Introduction

2 First step (Solve Simple ODE)

3 Second Step (Solve Simple PDE)

4 Final Step (Solve Our Equation!)

5 Result and Comparison

Introduction

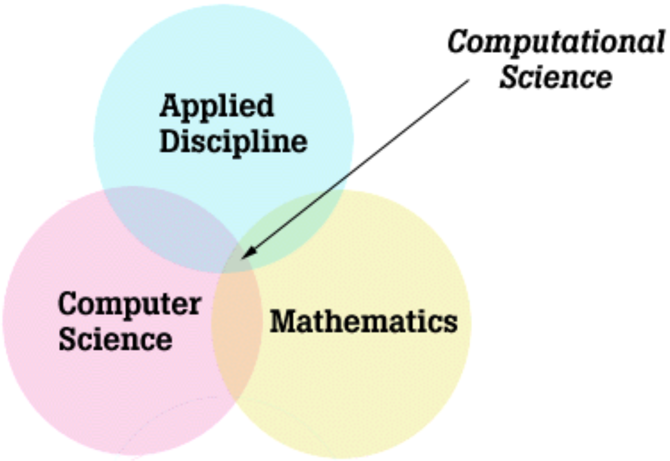
Where is our subject in ML world?

ML

Scientific ML

Solving Equations

- PDE, ODE, Fuzzy, IE, our equation

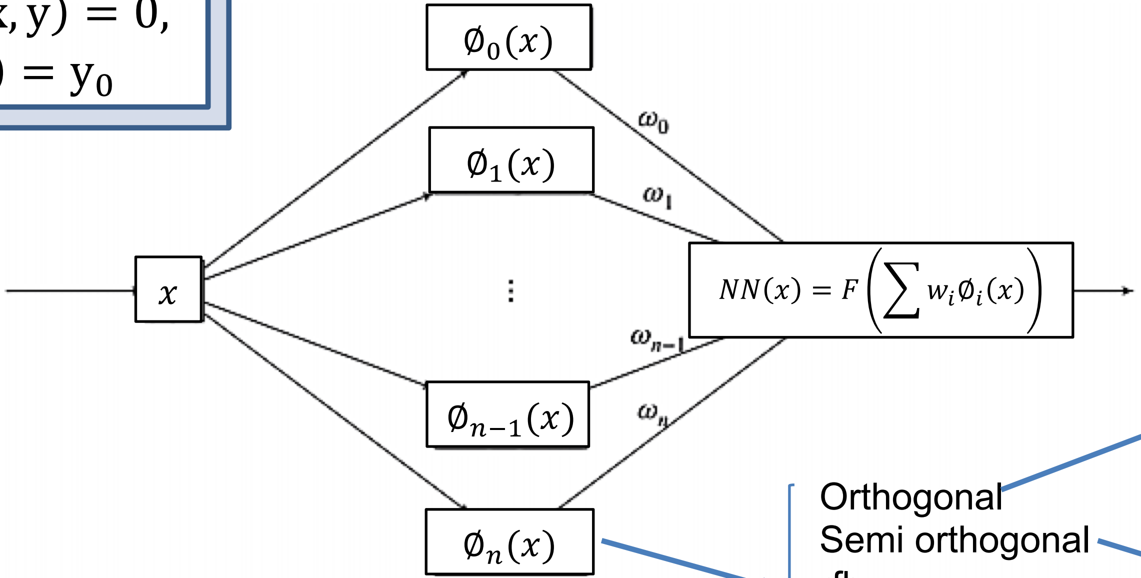


First Step

solve simple ODE

NN Structure:

$$y' - f(x, y) = 0,$$
$$y(0) = y_0$$



- Orthogonal
  - Chebyshev : [-1, 1]
  - Jacobi: [-1, 1]
  - Legendre
  - …
- Semi orthogonal
  - Bessel
  - Euler
- rfb
  - Gaussian
- Sigmoid
- …

Mathematical Foundations:

- Loss function:

$$y' - f(y, x) = 0$$

$$NN'(x) - f(NN(x), x) \approx 0$$

$$loss = NN'(x) - f(NN(x), x) - 0$$

- Initial condition?$(\mathrm{y}(0) = \mathrm{y}_0)$

$$g(x) = y(0) + xNN(x) \cong y(x)$$

- Final cost function:

Cost= $\sum_{n=1}^{M} |g'(x) - f(g(x), x)|^2$

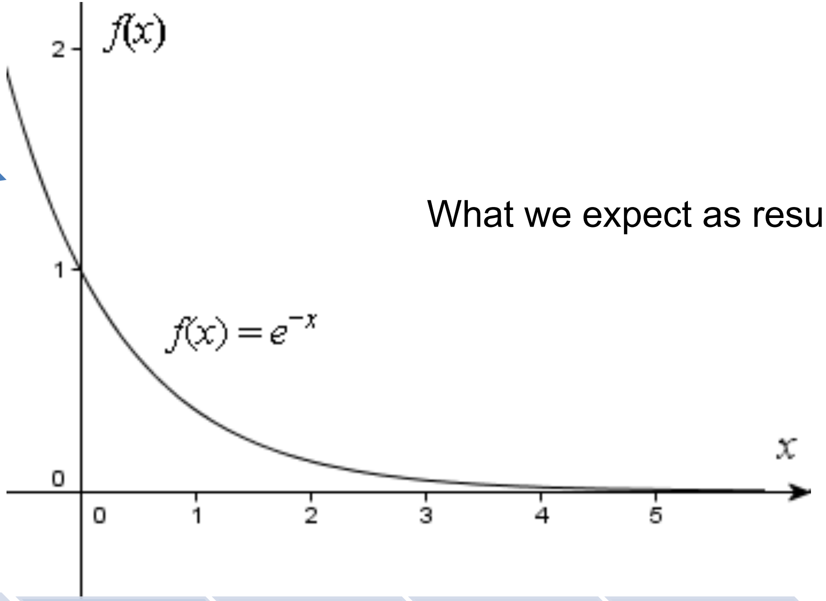
$$\min(\sum_{n=1}^{M} |Res(x_n)|^2) = GDS \Rightarrow W^*$$

Step1 (solve simple ODE using NN):

Equation: $y' + y = 0, \qquad y(0) = 1$

Domain: [0, 1]

Solution: $y(x) = e^{-x}$



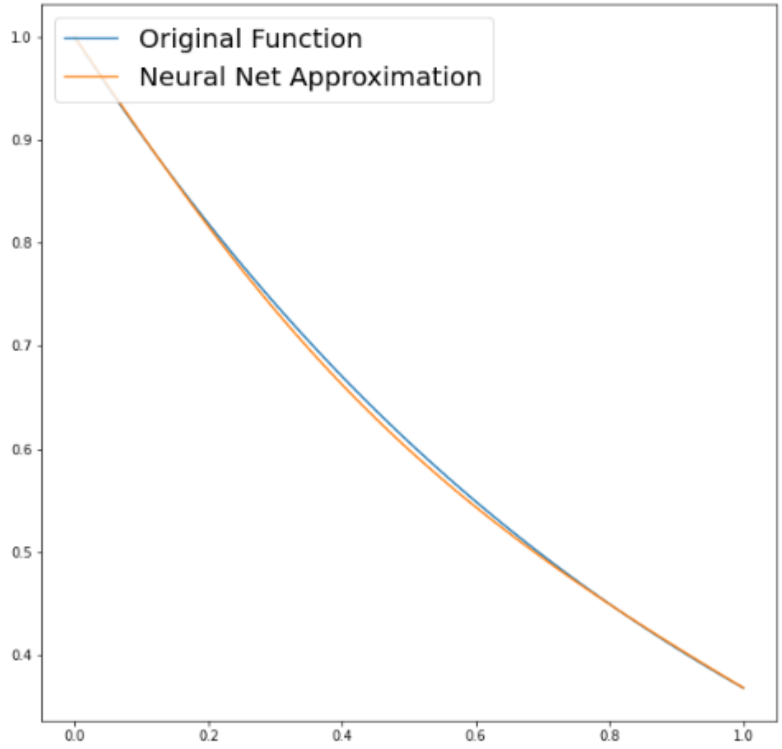
What we expect as result!

Implementation:

```
# Create model
def multilayer_perceptron(x):
    x = np.array([[[x]]],  dtype='float32')
    # Hidden fully connected layer with 32 neurons
    layer_1 = tf.add(tf.matmul(x, weights['w1']), biases['b1'])
    layer_1 = tf.nn.sigmoid(layer_1)
    # Output fully connected layer
    output = tf.matmul(layer_1, weights['w2']) + biases['b2']
    return output

# Universal Approximator
def g(x):
    return x * multilayer_perceptron(x) + f0

# Custom loss function to approximate the derivatives
def custom_loss():
    summation = []
    for x in np.linspace(0,1,10):
        dg_dx = (g(x+inf_s)-g(x))/inf_s
        summation.append((dg_dx + g(x))**2)
    return tf.reduce_sum(tf.abs(summation))

# Stochastic gradient descent optimizer.
optimizer = tf.optimizers.SGD(learning_rate)
```

$$L = \sum_i \left(\underbrace{\frac{dg(x_i)}{dx}}_{g'(x)} \underbrace{- f(y, x_i)}_{-\,g(x)} \right)^2$$

Final Output:

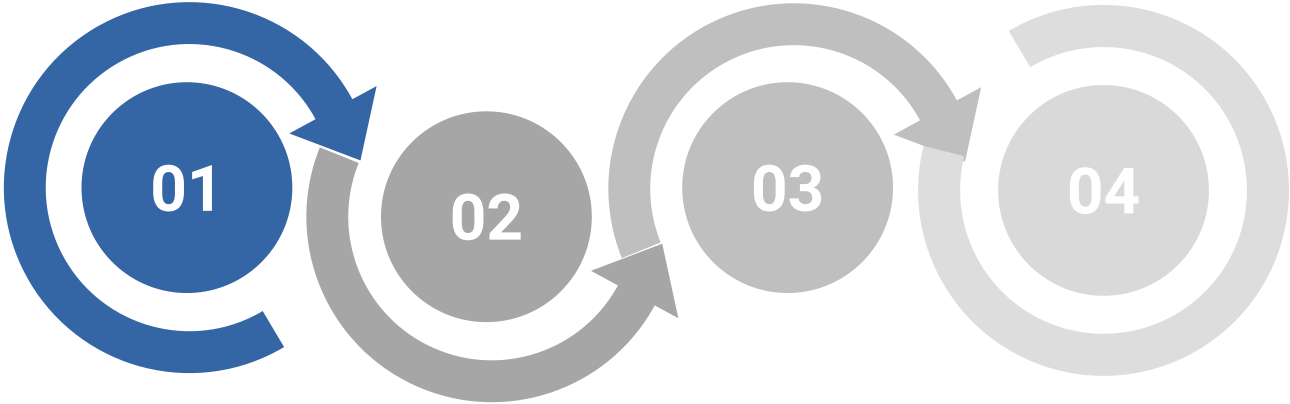



```
loss: 1293.932861
loss: 0.087478
loss: 0.018777
loss: 0.015615
loss: 0.013349
loss: 0.011632
loss: 0.010364
loss: 0.009332
loss: 0.008524
loss: 0.007919
```

Future Tasks:

Step 01
Solve Simple ODE

Step 03
Solve our equation



Step 02
Solve Simple PDE

Step 04
Testing and Results

Thanks!

Any questions?

Contact Us:
Mahdiehsadat20@gmail.com

hosna.hoseini@gmail.com