Technische Universität Braunschweig

iRMB KI 4 ALL

Physics Informed Neural Networks

Solving a Steady-State Heat Equation

1 ECTS Microcredit Research Project

January 24, 2023

Contents

1 Overview

Neural networks are a exciting technique to solve a variety of scientific problems. They are usually used in the data-driven regime. Less known is their applicability to partial differential equations (PDE), where they can be used to obtain solutions to boundary value problems directly without any data. This approach is called physics informed neural networks (PINN). In this small project, you will familiarize yourself with this approach and solve a simple steady-state heat equation.

2 Reading assignments

1. Try out the TensorFlow guides and tutorials to advance your deep learning skills: `https://www.tensorflow.org/`
2. You can find additional information on machine learning in the following book: `http://themlbook.com/`
3. Read the original PINN paper: `https://www.brown.edu/research/projects/crunch/sites/brown.edu.research.projects.crunch/files/uploads/Physics-informed%20neural%20networks_A%20deep%20learning%20framwork%20fir%20solving%20forward%20and%20inverse%20probelms%20involving%20nonlin 20partial%20differential%20equations.pdf`

4. Get more familiar with the PINN approach watching the following videos:
   - `https://www.youtube.com/watch?v=pbt3Ztkwwz8`
   - `https://www.youtube.com/watch?v=LQ33-GeD-4Y`
   - `https://www.youtube.com/watch?v=dK9pjhRI6c8`

3 Programming task

1. Take a look at the PINN implementation for an elastic rod in section 5. Play around with the hyperparameters and try to understand the implementation. Here, the one-dimensional balance equation is solved

$$\frac{\partial \sigma(x)}{\partial x} = 0, \quad x \in [-1, 1], \quad \sigma(1) = \bar{\sigma}, \quad u(-1) = 0, \tag{1}$$

   using Hooke's material law

$$\sigma = E\varepsilon \tag{2}$$

   and small strain assumption

$$\varepsilon = \frac{\partial u(x)}{\partial x}. \tag{3}$$

2. Reformulate the PINN implementation to solve a one-dimensional steady-state heat equation

$$\frac{\partial^2 T(x)}{\partial x^2} = 0, \quad x \in [0, 1], \quad T(0) = 30, \quad T(1) = 15. \tag{4}$$

3. Derive an analytical solution for the problem and check against your implementation. (Hint: Integration!)

4 Report

Write an report of your work, including a brief literature review about PINNs, the theoretical background of the physical background, neural networks and PINNs as well as your numerical examples. The report must be created using LaTeX. For example, you can use the template provided here: `https://de.overleaf.com/edu/tu-braunschweig`

5 Code

```
"""A simple PINN for linear elasticity."""
import tensorflow as tf
import matplotlib.pyplot as plt

print(tf.__version__)


class PINN(tf.keras.Model):
    """PINN for linear elasticity.

    A PINN solving a one-dimensional homogeneous rod with one side fixed and
    the other side subjected to a uniaxial, constant force. The material law
    is Hooke's law, thus small strain theory applies.

    Args:
        youngs_modulus: The Young's modulus for the one-dimensional
        Hooke's law.
        force: The force acting on the rod.
        area: The cross-section area of the rod.
    """

    def __init__(
            self,
            youngs_modulus=210.000,
            force=100.0,
            area=20.0,
    ):
        super(PINN, self).__init__()
        self.E = youngs_modulus
        self.Force = force
        self.area = area
        self.sig = self.Force / self.area

        self.build(input_shape=1)

    def build(self, input_shape):
        """Builds the artificial neural network.

        Hard boundary conditions are used to automatically enforce the
        boundary conditions. At the respective boundary, the neural network
        term vanishes and the prescribed boundary condition is automatically
        fullfilled. Here, 'D_' is a distance function to the boundary
        coordinate and 'G_' is the value of the specific boundary condition.
        """
        input_layer = tf.keras.layers.Input(
            shape=input_shape,
```

```
            name="input_layer")

        hidden_layer = tf.keras.layers.Dense(
            units=32,
            activation="tanh",
            name="hidden_layer")(input_layer)

        output_layer = tf.keras.layers.Dense(
            units=2,
            name="output_layer")(hidden_layer)

        G_u = 0.0
        D_u = -1.0 - input_layer[:, 0]
        u = G_u + D_u * output_layer[:, 0]
        u = tf.reshape(u, (-1, 1))

        G_sig = self.sig
        D_sig = 1.0 - input_layer[:, 0]
        sig = G_sig + D_sig * output_layer[:, 1]
        sig = tf.reshape(sig, (-1, 1))

        self.ann = tf.keras.Model(
            inputs=[input_layer],
            outputs=[u, sig],
            name="ANN",
        )

        self.built = True

    def residual(self, inputs):
        """Calculate residual of governing equations.

        The following governing equations exist in linear elasticity:

            - Balance law of linear momentum: sig_x = 0
            - Hooke's law: sig - E * eps = 0
            - Kinematic relation: eps = u_x.

        Plugging in the kinematic relation into Hooke's law, one needs to
        consider two residuals, namely the material residual and the balance
        residual:

            - material residual: sig - E * u_x = residual_material
            - balance residual: sig_x = residual_balance.

        Args:
            inputs: Input coordinates.
        """
```

```
        with tf.GradientTape(persistent=True) as tape:
            tape.watch(inputs)
            u = self.ann(inputs)[0]
            sig = self.ann(inputs)[1]

        u_x = tape.gradient(u, inputs)
        sig_x = tape.gradient(sig, inputs)

        residual_material = self.E * u_x - sig
        residual_balance = sig_x

        return {
            "residual_material": residual_material,
            "residual_balance": residual_balance,
        }

    def call(self, inputs):
        """Forward pass.

        The forward pass of the PINN to calculate the residuals at every
        coordinate point of the rod.

        Args:
             inputs: Input coordinates.
        """
        inputs = tf.reshape(inputs, (-1, 1))
        residual = self.residual(inputs=inputs)
        r_mat = residual["residual_material"]
        r_bal = residual["residual_balance"]
        loss = tf.reduce_mean(tf.square(r_mat) + tf.square(r_bal))
        self.add_loss(loss)
        return loss

    def predict(self, inputs):
        """Prediction of displacement and stress.

        Args:
            inputs: Input coordinates.
        """
        u = self.ann(inputs)[0]
        sig = self.ann(inputs)[1]
        return {"disp": u, "sig": sig}


if __name__ == "__main__":
    pinn = PINN()
    pinn.compile(
        optimizer=tf.keras.optimizers.Adam(),
```

```
    run_eagerly=False)
pinn.summary()
coords = tf.linspace(-1.0, 1.0, 100)
history = pinn.fit(coords, epochs=500, verbose=2)

plt.figure(0)
plt.semilogy(history.history["loss"])

prediction = pinn.predict(coords)
displacement = prediction["disp"]
stress = prediction["sig"]

plt.figure(1)
plt.plot(displacement)
plt.title("displacement")

plt.figure(2)
plt.plot(stress)
plt.title("stress")

plt.show()
```