Custom Training Loop

Hans C. Suganda

26^{th} August 2021

Custom Training Loop

Suppose the common ordinary differential equation problem,

$$\frac{dx}{dt} = kx$$

Expressing the derivative in terms of limits,

$$\frac{dx}{dt} = \lim_{\Delta t \to 0} \left[\frac{x(t+\Delta t) - x(t)}{\Delta t} \right]$$

Substituting the derivative of x with respect to t,

$$kx = \lim_{\Delta t \to 0} \left[\frac{x(t+\Delta t) - x(t)}{\Delta t} \right]$$

Subtracting all terms to form zero in one of the sides,

$$0 = kx - \lim_{\Delta t \to 0} \left[\frac{x(t+\Delta t) - x(t)}{\Delta t} \right]$$

Approximating the limits as a finite difference where Δx is sufficiently small suppose $\Delta x = 1 \times 10^{-6}$,

$$0 = kx - \frac{x(t+\Delta t) - x(t)}{\Delta t}$$

Suppose the loss function L of the neural network is defined as,

$$L = kx - \frac{x(t+\Delta t) - x(t)}{\Delta t}$$

Suppose a neural network of arbitrary hidden layers but one input node and one output node is used to model the function $x(t)$. As the optimizer iterates through the various weights of the neural network, the loss function would be minized, ideally approaching zero. After training, the neural network represents the function $x(t)$ which ordinary differential equation. The algorithmic implementation of this is shown below,

```
import numpy as np
import keras
import tensorflow as tf
import matplotlib.pyplot as plt

#Problem Specific Variables
delta_t = 0.000000000001 #Resolution of Numerical Method
```

```
k = 10 #k-value in ODE problem
Range = np.linspace(-10,10,100)
Iteration = np.linspace(0,10,100)

#Defining Inputs of Neural Network
inputs = tf.keras.Input(shape=(1,))

#Some Arbitrary Neural Network Architecture
layer1 = tf.keras.layers.Dense(4, activation='sigmoid')(inputs)
layer2 = tf.keras.layers.Dense(4, activation='sigmoid')(layer1)

#Defining Outputs of Neural Network
outputs = tf.keras.layers.Dense(1, activation='sigmoid')(layer2)

#Defining Model used
model = tf.keras.Model(inputs = inputs, outputs = outputs)

#Defining Loss Function
def diff_loss(ahead, current):
    loss = tf.subtract(tf.divide((tf.subtract(ahead,current)),delta_t),tf.
    multiply(k,current))
    return loss

# Instantiate an optimizer.
optimizer = tf.keras.optimizers.Adam()

#Iterate over the domain
for t in Range:
    for i in Iteration:
        # Open a GradientTape.
        with tf.GradientTape() as tape:
            #Forward Pass of f(t+dt)
            ftdt = model(tf.constant([t + delta_t]))
            #Forward Pass of f(t)
            ft = model(tf.constant([t]))
            # Loss value for batch
            loss = diff_loss(ftdt, ft)

        # Get gradients of loss wrt the weights.
        gradients = tape.gradient(loss, model.trainable_weights)

        # Update the weights of the model.
        optimizer.apply_gradients(zip(gradients, model.trainable_weights))

#Testing the model
t_test = np.linspace(-10,10,100)
result = model(t_test)

#Showing Results
```

```python
print(result)
plt.plot(t_test, result)
plt.grid()
plt.show()

#Customary End
print('Leaves Blow in the Wind...')
```