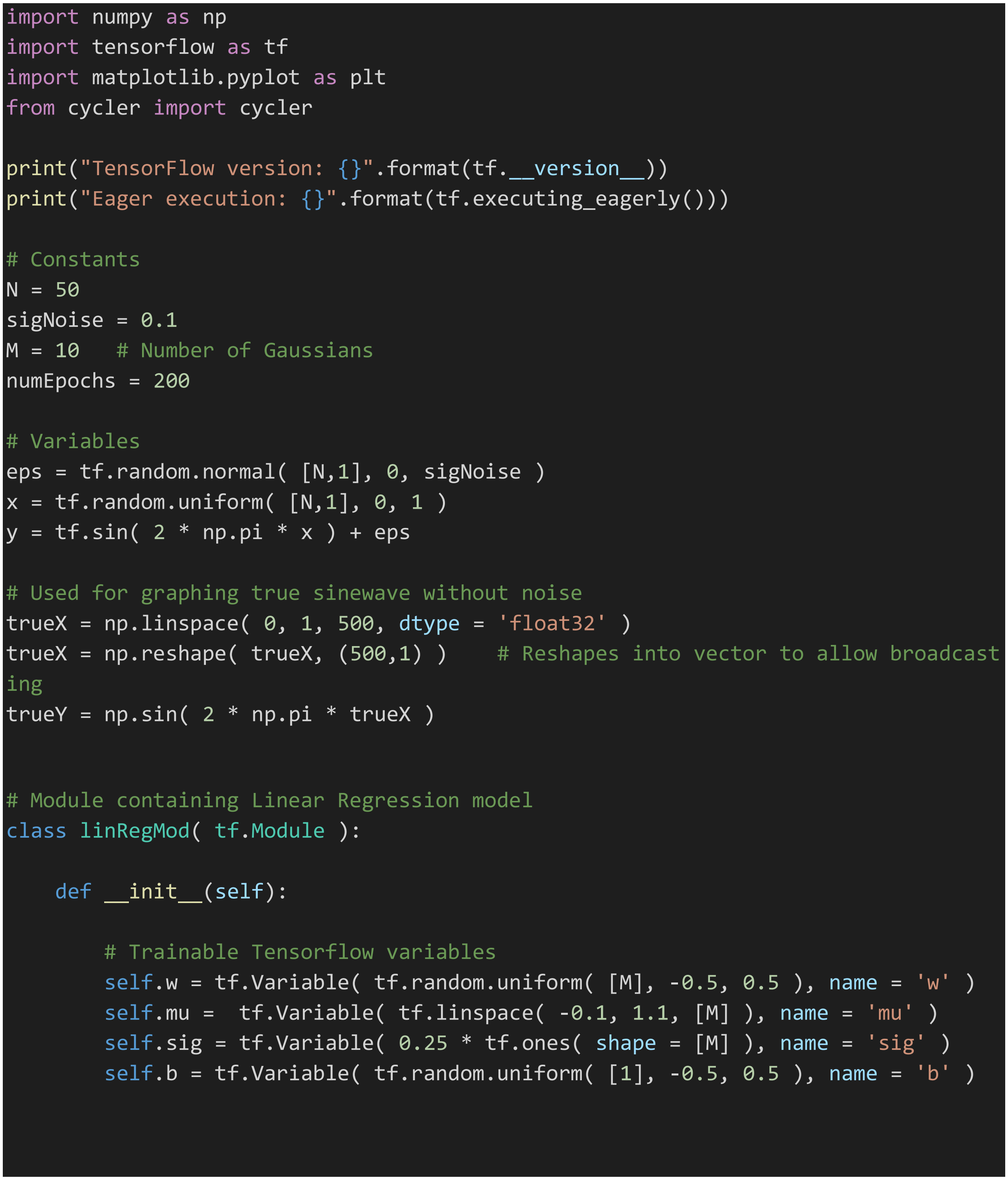
Derek Lee — Deep Learning Fall 2020

Professor Curro — Assignment #1



```python
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from cycler import cycler

print("TensorFlow version: {}".format(tf.__version__))
print("Eager execution: {}".format(tf.executing_eagerly()))

# Constants
N = 50
sigNoise = 0.1
M = 10   # Number of Gaussians
numEpochs = 200

# Variables
eps = tf.random.normal( [N,1], 0, sigNoise )
x = tf.random.uniform( [N,1], 0, 1 )
y = tf.sin( 2 * np.pi * x ) + eps

# Used for graphing true sinewave without noise
trueX = np.linspace( 0, 1, 500, dtype = 'float32' )
trueX = np.reshape( trueX, (500,1) )    # Reshapes into vector to allow broadcasting
trueY = np.sin( 2 * np.pi * trueX )


# Module containing Linear Regression model
class linRegMod( tf.Module ):

    def __init__(self):

        # Trainable Tensorflow variables
        self.w = tf.Variable( tf.random.uniform( [M], -0.5, 0.5 ), name = 'w' )
        self.mu =  tf.Variable( tf.linspace( -0.1, 1.1, [M] ), name = 'mu' )
        self.sig = tf.Variable( 0.25 * tf.ones( shape = [M] ), name = 'sig' )
        self.b = tf.Variable( tf.random.uniform( [1], -0.5, 0.5 ), name = 'b' )
    # Loss function
    @tf.function
    def lossFunc( self, x, y ):

        yHat = self.estY( x )   # Applies estY elementwise to x
        MSE = tf.reduce_sum( 0.5 * ( y - yHat )**2 )

        return MSE


    # Calculates yHat given x
    @tf.function
    def estY( self, x ):

        return tf.reduce_sum( self.w * self.gaussian( x ), 1, keepdims = True ) + self.b   # Sums along rows to get N x 1

    # Generates N x M matrix
    def gaussian( self, x ):

        return tf.math.exp( -( x - self.mu )**2 / self.sig**2 )

    def train( self ):

        # Stochastic Gradient Descent
        # Idea from https://stackoverflow.com/questions/57759563/minimize-multivariate-function-in-tensorflow
        opt = tf.keras.optimizers.SGD()

        # Iterate through epochs
        for _ in range( numEpochs ):

            with tf.GradientTape() as tape:

                loss = self.lossFunc( x, y )
                print(loss)

            grads = tape.gradient( loss, self.variables )
            opt.apply_gradients( zip( grads, self.variables ) )
    # First plot
    def plotFit( self, x, y, trueX, trueY ):

        # Calculate manifold from parameters
        yPred = self.estY( trueX ).numpy()

        # Plot
        plt.figure()
        plt.scatter( x, y, color = 'g' )                        # Noisy data
        plt.plot( trueX, yPred, color = 'r', linestyle = '--
' ) # Regression manifold
        plt.plot( trueX, trueY, color = 'b' )                   # Noiseless sinew
ave
        plt.xlabel( 'x' )
        plt.ylabel( 'y', rotation = 0 )
        plt.title( "Fit 1" )
        plt.show()

    # Second plot
    def plotBases( self, trueX ):

        plt.figure()
        plt.rc( 'axes', prop_cycle = ( cycler ('color', ['r', 'g', 'b', 'm', 'y',
 'c']) ) )

        # Plot each Gaussian
        plt.plot( trueX, self.gaussian( trueX ) )

        plt.xlabel( 'x' )
        plt.ylabel( 'y', rotation = 0 )
        plt.title( "Bases for Fit 1" )
        plt.show()

def main():

    model = linRegMod()

    print( "Starting MSE:", model.lossFunc( x, y ).numpy() )
    model.train()
    print( "Final MSE:", model.lossFunc( x, y ).numpy() )

    model.plotFit( x, y, trueX, trueY )
    model.plotBases( trueX )

main()
```

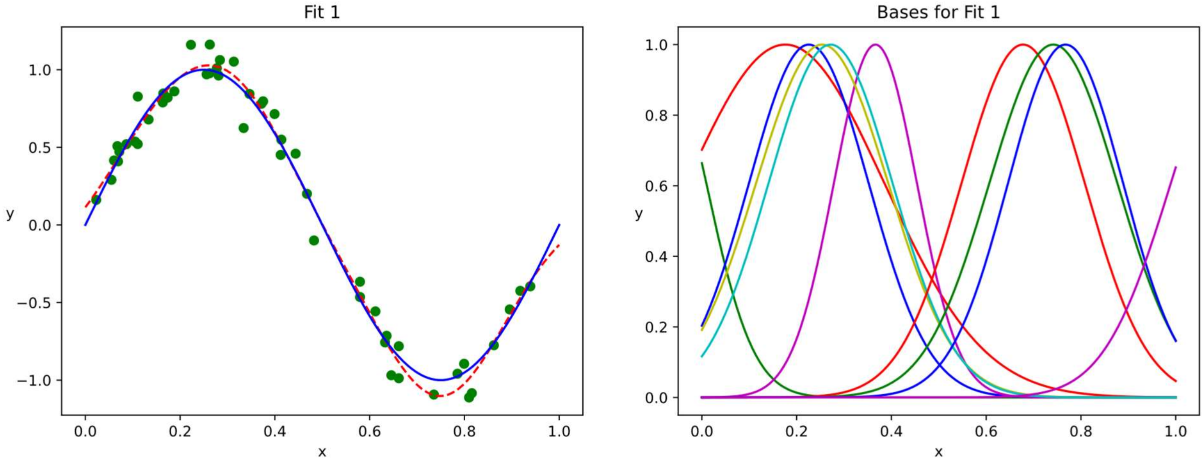


Figure 1: Example plots for linear regression of a noisy sinewave using a set of 10 Gaussian basis functions