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

# Used for graphing true sinewave without noise
trueX = np.linspace( 0, 1, 500, dtype = 'float32' )
trueY = np.sin( 2 * np.pi * trueX )

# Trainable Tensorflow variables
w = tf.Variable( tf.random.uniform( [M], -0.5, 0.5 ) )
mu =  tf.Variable( tf.linspace( -0.1, 1.1, [M] ) )
sig = tf.Variable( tf.repeat( 0.25, repeats = M ) )
b = tf.Variable( tf.random.uniform( [1], -0.5, 0.5 ) )

# Loss function
@tf.function
def lossFunc( x, w, mu, sig, b, y ):

    MSE = 0
    yHat = []

    for i in range( N ):
        yHat.append( estY( x[i], w, mu, sig, b ) )
```

```python
    for i in range( N ):
        MSE += 0.5 * ( y[i] - yHat[i] )**2

    return MSE


# Calculates yHat
@tf.function
def estY( x, w, mu, sig, b ):

    yHat = b

    for j in range( M ):
        yHat = yHat + w[j] * gaussian( x, mu[j], sig[j] )

    return yHat


def gaussian( x, mu, sig ):

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


def main():

    print( "Starting MSE:", lossFunc( x, w, mu, sig, b, y ).numpy() )

    # Stochastic Gradient Descent
    # Idea from https://stackoverflow.com/questions/57759563/minimize-
multivariate-function-in-tensorflow
    opt = tf.keras.optimizers.SGD()
    varList = [ w, mu, sig, b ]

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

        with tf.GradientTape() as tape:

            loss = lossFunc( x, w, mu, sig, b, y )
            print(loss)

        grads = tape.gradient( loss, varList )
        opt.apply_gradients( zip( grads, varList ) )

    print( "Final MSE:", lossFunc( x, w, mu, sig, b, y ).numpy() )
```

```python
    # Calculate y from training data
    yPred = np.zeros( len( trueX ) )
    for i in range( len( trueX ) ):
        yPred[i] = estY( trueX[i], w, mu, sig, b ).numpy()

    # First 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 sinewave
    plt.xlabel( 'x' )
    plt.ylabel( 'y', rotation = 0 )
    plt.title( "Fit 1" )
    plt.show()

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

    for j in range( M ):
        plt.plot( trueX, gaussian( trueX, mu[j], sig[j] ) )

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


main()
```

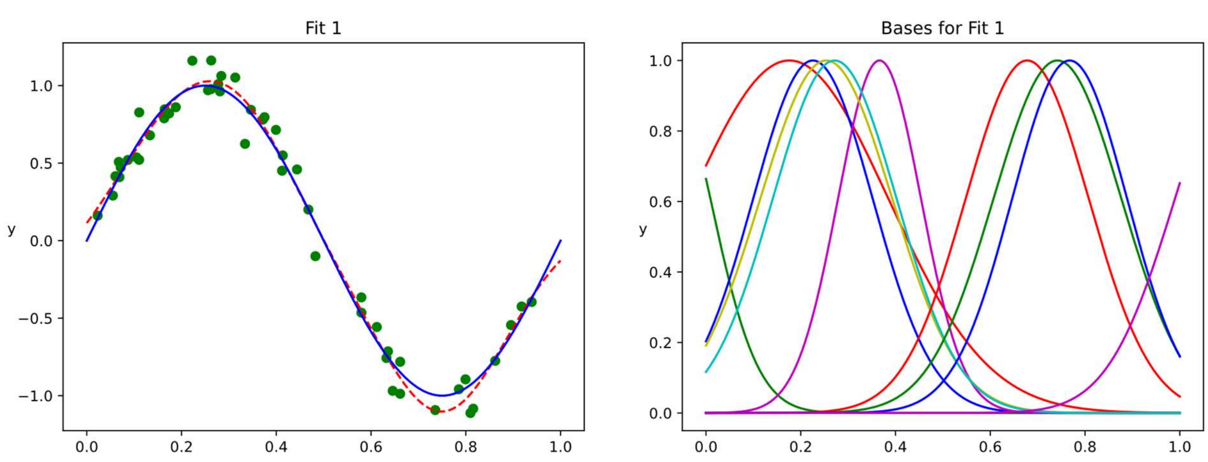

Figure 1: Example plots for linear regression of a noisy sinewave using a set of 10 Gaussian basis functions