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

from tqdm import trange

import random

NUM_SAMPLES = 50
# if we increase NUM_GAUSSIANS too much, it will over-fit
NUM_GAUSSIANS = 6
NUM_FEATURES = 1
NOISE_SIGMA = 0.1
BATCH_SIZE = 32
LEARNING_RATE = 0.05
NUM_BATCHES = 500

random.seed(42)

class Data(object):
    def __init__(self, num_features=NUM_FEATURES, num_samples=NUM_SAMPLES, noise_sigma=NOISE_SIGMA):
        self.idx = np.arange(num_samples)
        self.eps = np.random.normal(loc=0.0, scale=noise_sigma, size=(num_samples, num_features))

        self.x = np.random.uniform(low=0.0, high=1.0, size=(num_samples, num_features))
        self.y = np.sin(2*np.pi*self.x)+self.eps

    def get_batch(self, batch_size=BATCH_SIZE):
        choices = np.random.choice(self.idx, size=BATCH_SIZE)
        return self.x[choices], self.y[choices]

class Model(tf.Module):
    def __init__(self, num_features=NUM_FEATURES, num_gaussains=NUM_GAUSSIANS):
        # naive weight initialization
        # self.w = tf.Variable(tf.random.normal(shape=[num_gaussains, 1]), name="w")
        # self.b = tf.Variable(tf.random.normal(shape=[1, 1]), name="b")
        # self.mu = tf.Variable(tf.random.uniform(minval=-NUM_GAUSSIANS, maxval=NUM_GAUSSIANS, shape=(1, num_gaussains)), name="mu")
        # self.sigma = tf.Variable(tf.random.uniform(shape=(1, num_gaussains)), name="sigma")

        # weight initialization is important
        self.w = tf.Variable(tf.random.normal(shape=(num_gaussains, num_features)), name="w")
        self.b = tf.Variable(tf.random.normal(shape=(1, num_features)), name="b")

        # min and max of the x is 0.01 and 0.97 (limit of the x space)
        # variance is 0.097 ~ std is 0.31
        self.mu = tf.Variable(tf.random.normal(mean=0.5, stddev=0.33, shape=(num_features, num_gaussains)), name="mu")
        self.sigma = tf.Variable(tf.math.abs(tf.random.normal(mean=0.31, stddev=0.33, shape=(num_features, num_gaussains))), name="sigma")

    def phi(self, x):
```

```python
        # broadcasting
        return tf.math.exp(-(tf.math.square(x-self.mu)/tf.math.square(self.sigma)))

    def __call__(self, x):
        return self.phi(x) @ self.w + self.b

if __name__ == "__main__":
    data = Data()

    print(f"x variance: {np.var(data.x)} x min: {np.min(data.x)} x max: {np.max(data.x)}")

    model = Model()

    optimizer = tf.optimizers.SGD(learning_rate=LEARNING_RATE)

    x = np.linspace(0, 1, 201)
    plt.subplot(4, 1, 1)
    plt.plot(x, np.sin(2*np.pi*x), '-', label="ground truth")
    plt.plot(data.x, data.y, 'o', label="sampled data")
    x = x.reshape((len(x)//NUM_FEATURES, NUM_FEATURES))
    plt.plot(x, model(x), label="untrained model")
    plt.legend()
    plt.title("untrained model")


    plt.subplot(4, 1, 2)
    plt.plot(x, model.phi(x))
    # plt.legend([f"trained gaussian{i}" for i in range(1, NUM_GAUSSIANS+1)], ncol=3)
    plt.title("untrained gaussians")

    bar = trange(NUM_BATCHES)
    for i in bar:
        # Gradient Tape records operations for autodiff
        with tf.GradientTape() as tape:
            x, y = data.get_batch()
            y_hat = model(x)
            loss = tf.reduce_mean(((y_hat-y)**2)/2)
        # Computes the gradient using operations recorded in context of this tape.
        grads = tape.gradient(loss, model.variables)
        optimizer.apply_gradients(zip(grads, model.variables))
        bar.set_description(f"Loss @ {i} => {loss.numpy():0.6f}")
        bar.refresh()

    x = np.linspace(0, 1, 201)
    plt.subplot(4, 1, 3)
    plt.plot(x, np.sin(2*np.pi*x), '-', label="ground truth")
    plt.plot(data.x, data.y, 'o', label="sampled data")
    x = x.reshape((len(x)//NUM_FEATURES, NUM_FEATURES))
    plt.plot(x, model(x), label="trained model")
    plt.legend()
    plt.title("trained model")

    plt.subplot(4, 1, 4)
    plt.plot(x, model.phi(x))
    # plt.legend([f"trained gaussian{i}" for i in range(1, NUM_GAUSSIANS+1)], ncol=3)
```

```
    plt.title("trained gaussians")
    plt.show()
```