```python
#!/bin/env python3.8

import os

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

# import math

from tqdm import trange


script_path = os.path.dirname(os.path.realpath(__file__))
rng = np.random.default_rng(seed=69)


class Data:
    def __init__(self, ns, sig, range):

        self.index = np.arange(2 * ns)
        self.ns = ns
        self.sig = sig
        self.range = range

        self.r_1 = rng.uniform(range[0], range[1], size=ns)
        self.r_2 = rng.uniform(range[0], range[1], size=ns)

        self.x_1 = self.r_1 * tf.math.cos(self.r_1)
        self.y_1 = self.r_1 * tf.math.sin(self.r_1)

        self.x_2 = -self.r_2 * tf.math.cos(self.r_2)
        self.y_2 = -self.r_2 * tf.math.sin(self.r_2)

        self.x_1 += rng.normal(0, sig, (ns))
        self.y_1 += rng.normal(0, sig, (ns))

        self.x_2 += rng.normal(0, sig, (ns))
        self.y_2 += rng.normal(0, sig, (ns))

        self.data_1 = [self.x_1, self.y_1]
        self.data_2 = [self.x_2, self.y_2]
        self.data_in = np.concatenate([self.data_1, self.data_2], axis=1).T
        self.data_out = np.concatenate(([0] * ns, [1] * ns))

    def plot(self):
        plt.scatter(self.x_1, self.y_1, label="1")
        plt.scatter(self.x_2, self.y_2, label="2")
        plt.legend()
        plt.savefig(f"{script_path}/fit1.pdf")

    def get_batch(self, batch_size):
        choices = np.array(rng.choice(self.index, size=batch_size))
        return self.data_in[choices], self.data_out[choices]


class DenseLayer(tf.Module):
    def __init__(self, n_out, activation=tf.nn.leaky_relu, name=None):
        super().__init__(name=name)
        self.n_out = n_out
        self.activation = activation
        self.__is_built = False
```



```python
    def build(self, n_in, index=0):
        self.w = tf.Variable(
            tf.random.normal([n_in, self.n_out]) * 0.01, name="w" + str(index)
        )
        self.b = tf.Variable(tf.zeros([1, self.n_out]), name="b" + str(index))
        self.__is_built = True

    def __call__(self, x):
        if not self.__is_built:
            raise ValueError("Dense layer not built")
        return self.activation((x @ self.w) + self.b)


# def loss(y_true, y_pred):
#     return tf.reduce_mean(
#         y_true * tf.math.log(y_pred) + (1 - y_true) * tf.math.log(1 - y_pred)
#     )


class Model(tf.Module):
    def __init__(self, input, layers, name=None) -> None:
        super().__init__(name=name)
        self.layers = []
        with self.name_scope:
            for (i, node) in enumerate(layers):
                node.build(input, i)
                self.layers.append(node)
                input = node.n_out

    @tf.Module.with_name_scope
    def __call__(self, x):
        value = x
        for node in self.layers:
            value = node(value)
        return value


# model.build(input_shape=(2, 16))

data = Data(500, 0.01, (0.75, 5))
data.plot()

print("model")
model = Model(
    2,
    [
        DenseLayer(96),
        DenseLayer(96),
        DenseLayer(96),
        DenseLayer(1, activation=tf.nn.sigmoid),
    ],
)
# print(model.trainable_variables)


loss = tf.keras.losses.BinaryCrossentropy(from_logits=False)
optimizer = tf.keras.optimizers.Adam(
    learning_rate=0.01,
    beta_1=0.9,
    beta_2=0.999,
    epsilon=1e-07,
```

```
    amsgrad=True,
    name="Adam",
)
bar = trange(10000)
x, y = data.get_batch(16)

for i in bar:
    with tf.GradientTape() as tape:
        # x, y = data.get_batch(16)
        y_hat = model(x)
        loss_value = loss(y, y_hat)

    grads = tape.gradient(loss_value, model.trainable_variables)
    print(tf.reduce_mean(tf.abs(grads[0])))
    optimizer.apply_gradients(zip(grads, model.trainable_variables))

    bar.set_description(f"Loss @ {i} => {loss_value.numpy():0.8f}")
    bar.refresh()

fig, ax = plt.subplots(1, 2, figsize=(11, 4), dpi=200)

# ax[0].set_title("Linear Combination of Gaussians")
# ax[0].set_xlabel("x")
# ax[0].set_ylim(np.amin(data.y) * 1.5, np.amax(data.y) * 1.5)
# h = ax[0].set_ylabel("y", labelpad=10)
# h.set_rotation(0)

# xs = np.linspace(0, 2, 100)
# xs = xs[:, np.newaxis]
# print(tf.shape(np.squeeze(xs)))
# print(tf.shape(model(np.squeeze(xs))))
# ax[0].plot(xs, np.squeeze(model(xs)), "--", label="model")
# ax[0].plot(np.squeeze(data.x), data.y, "o", label="training data")
# ax[0].plot(xs, np.squeeze((tf.math.sin(2 * math.pi * xs))), label="training da
ta")
# ax[0].plot()

# ax[1].set_title("Linear Combination of Gaussians")
# ax[1].set_xlabel("x")
# ax[1].set_ylim(np.amin(data.y) * 1.5, np.amax(data.y) * 1.5)

# for mu_i in range(tf.shape(model.mu)[0]):
#     theta_j = tf.math.exp(-((xs - model.mu[mu_i]) ** 2) / (model.sigma[mu_i])
** 2)
#     ax[1].plot(xs, theta_j)
# ax[0].plot()
# ax[1].plot()


# plt.tight_layout()
# plt.savefig(f"{script_path}/fit.pdf")
```