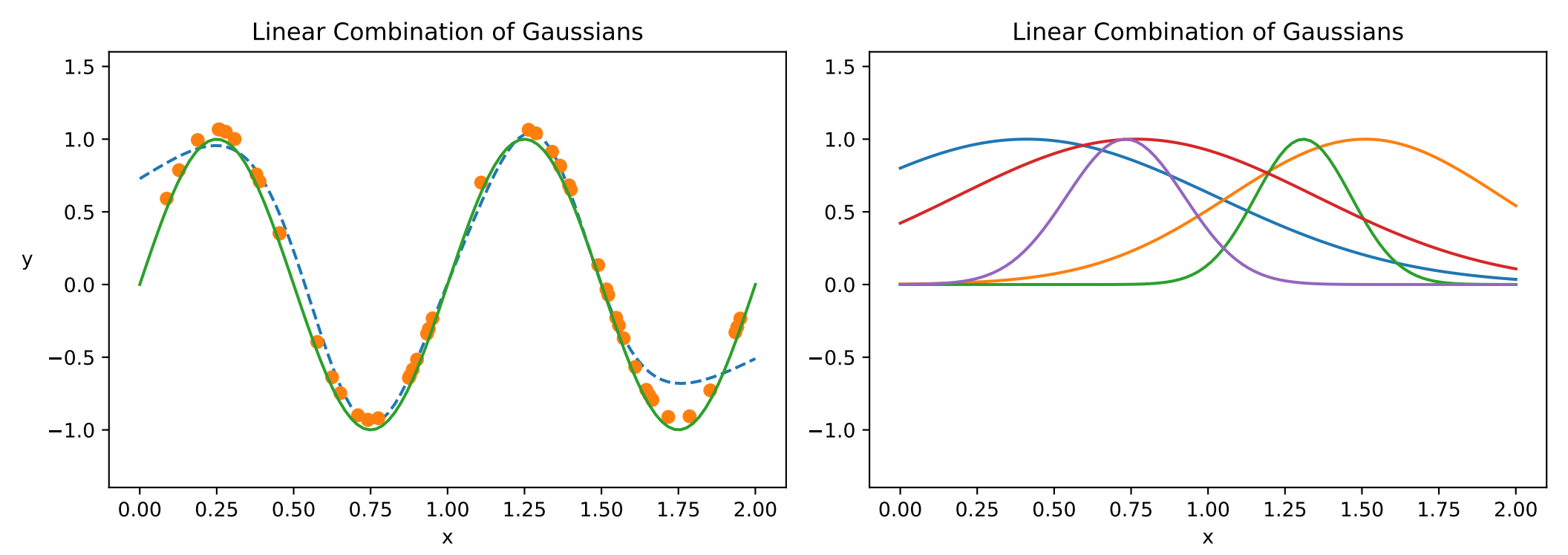
```
#!/bin/env python3.8

"""
Example assignment. Author: Chris Curro
"""
import os

# from turtle import shape
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=42)


class Data:
    def __init__(self, ns, sig, range):
        self.rng = np.random.default_rng(seed=42)
        self.index = np.arange(ns)
        self.num_samples = ns
        self.sigma = sig
        self.range = range
        self.x = np.array(self.rng.uniform(0, 2, self.num_samples))
        clean_y = tf.math.sin(2 * math.pi * self.x)
        self.y = np.array(clean_y + self.rng.normal(0, self.sigma))

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


class Model(tf.Module):
    def __init__(self, M):
        self.M = M
        self.w = tf.Variable(rng.normal(0, 1, M))
        self.mu = tf.Variable(rng.normal(1, 0.5, M))
        self.sigma = tf.Variable(rng.normal(0, 1, M))
        self.b = tf.Variable(np.array([[0.0]]))

    def __call__(self, x):
        num = tf.shape(x)[0]
        x = tf.squeeze(x)
        x = tf.transpose(tf.broadcast_to(x, [self.M, num]))
        mu = tf.broadcast_to(self.mu, [num, self.M])
        sigma = tf.broadcast_to(self.sigma, [num, self.M])
        theta = tf.math.exp(-((x - mu) ** 2) / (sigma) ** 2)

        y = tf.broadcast_to(self.w, [1, self.M]) @ tf.transpose(theta)

        return tf.cast(y, dtype="float32") + tf.cast(self.b, dtype="float32")


data = Data(50, 0.1, (0, 2))
model = Model(5)


optimizer = tf.optimizers.SGD(learning_rate=0.1)
```



```
bar = trange(500)
for i in bar:
    with tf.GradientTape() as tape:
        x, y = data.get_batch(16)
        y_hat = model(x)
        loss = 0.5 * tf.reduce_mean((y_hat - y) ** 2)

    grads = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(grads, model.trainable_variables))

    bar.set_description(f"Loss @ {i} => {loss.numpy():0.6f}")
    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 data")
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")
```


Linear Combination of Gaussians
y
x
1.5
1.0
0.5
0.0
−0.5
−1.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Linear Combination of Gaussians
x