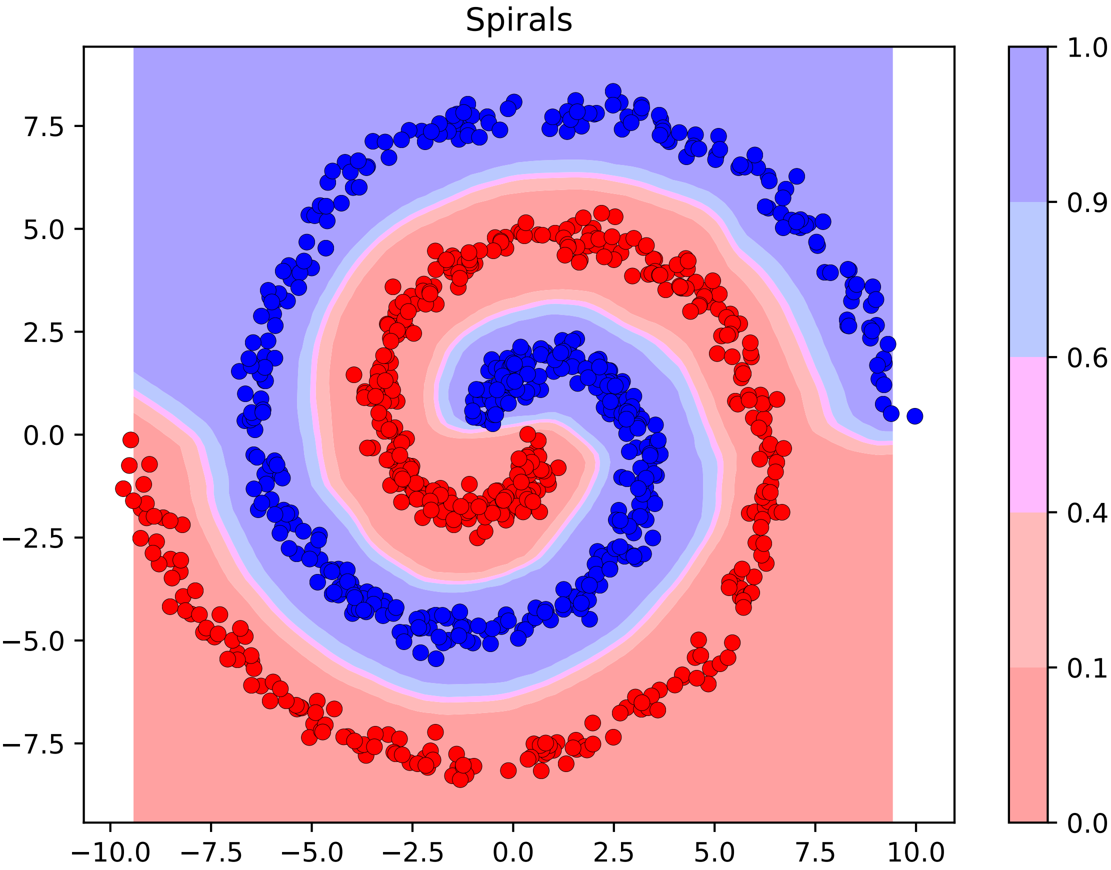
```
#!/bin/env python3.8

# Gavri Kepets
# Help received: Husam Almanakhly pointed me to this link: https://www.tensorflo
w.org/guide/core/mlp_core#multilayer_perceptron_mlp_overview

import os
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from tqdm import trange
from absl import app
from absl import flags

script_path = os.path.dirname(os.path.realpath(__file__))

FLAGS = flags.FLAGS
flags.DEFINE_integer("num_samples", default=500, help="Number of samples in dataset")
flags.DEFINE_integer("batch_size", default=16, help="Number of samples in batch")
flags.DEFINE_integer("num_iters", default=1500, help="Number of SGD iterations")
flags.DEFINE_integer("random_seed", default=31415, help="Random seed")
flags.DEFINE_float("spiral_rotations", default=1.5, help="Random seed")


class Data:
    def __init__(self, rng, num_samples, sigma):
        self.num_samples = num_samples
        self.sigma = sigma
        d = rng.uniform(
            np.pi / 4, 2 * FLAGS.spiral_rotations * np.pi, size=(self.num_sample
s, 1)
        )

        x = d * np.cos(d)
        y = d * np.sin(d)

        x1 = x + self.noise(rng)
        x2 = -x + self.noise(rng)

        y1 = -y + self.noise(rng)
        y2 = y + self.noise(rng)

        self.x = np.concatenate((x1.flatten(), x2.flatten()))
        self.y = np.concatenate((y1.flatten(), y2.flatten()))
        self.type = np.concatenate(
            (np.zeros(self.num_samples), np.ones(self.num_samples))
        )

    def noise(self, rng):
        return rng.normal(loc=0, scale=self.sigma, size=(self.num_samples, 1))

    def get_batch(self, rng, batch_size):
        choices = rng.choice(np.arange(2 * self.num_samples), size=batch_size)
        return self.x[choices], self.y[choices], self.type[choices]


# Initialize the weights with the xavier scheme, as per the TF docs
def xavier_init(shape, rng):
    in_dim, out_dim = shape
    xavier_lim = tf.sqrt(6.0) / tf.sqrt(tf.cast(in_dim + out_dim, tf.float32))
    weight_vals = rng.uniform(
        shape=(in_dim, out_dim), minval=-xavier_lim, maxval=xavier_lim
```

```
    )
    return weight_vals


class MultiLayerPerceptron(tf.Module):
    def __init__(self, layer_dims, rng):
        self.rng = rng
        self.layers = []
        for i in range(len(layer_dims)):
            self.layers.append(
                layer(
                    layer_dims[i],
                    activation=tf.nn.relu if layer_dims[i] != 1 else tf.nn.sigmo
id,
                    in_dim=2 if i == 0 else layer_dims[i - 1],
                    rng=self.rng,
                )
            )

    def __call__(self, x):
        for layer in self.layers:
            x = layer(x, self.rng)

        return tf.squeeze(x)


class layer(tf.Module):
    def __init__(self, out_dim, in_dim, rng, activation):
        self.out_dim = out_dim
        self.in_dim = in_dim
        self.activation = activation
        self.w = tf.Variable(xavier_init((self.in_dim, self.out_dim), rng))
        self.b = tf.Variable(tf.zeros(shape=(self.out_dim,)))

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


def loss(y, yh):
    return tf.reduce_mean(-y * tf.math.log(yh) - (1 - y) * tf.math.log(1 - yh))


def main(a):
    seed_sequence = np.random.SeedSequence(FLAGS.random_seed)
    np_seed, tf_seed = seed_sequence.spawn(2)
    np_rng = np.random.default_rng(np_seed)
    tf_rng = tf.random.Generator.from_seed(tf_seed.entropy)

    data = Data(np_rng, FLAGS.num_samples, 0.25)
    model = MultiLayerPerceptron([128, 64, 16, 1], tf_rng)

    optimizer = tf.optimizers.Adam(learning_rate=float(0.0025))

    bar = trange(int(FLAGS.num_iters))
    for i in bar:
        with tf.GradientTape() as tape:
            xs, ys, types = data.get_batch(np_rng, FLAGS.batch_size)
            points = np.concatenate((xs, ys)).reshape(2, FLAGS.batch_size).T
            total_loss = loss(types, model(points)) + 0.001 * tf.reduce_mean(
                [tf.nn.l2_loss(w) for w in model.trainable_variables]
            )
        grads = tape.gradient(total_loss, model.trainable_variables)
        optimizer.apply_gradients(zip(grads, model.trainable_variables))
        bar.set_description(f"Loss @ {i} => {total_loss.numpy():0.6f}")
        bar.refresh()

    fig1, ax = plt.subplots()
    # Plot data points
    for i in range(FLAGS.num_samples * 2):
        ax.plot(
            data.x[i],
            data.y[i],
            marker="o",
            color="r" if data.type[i] < 0.5 else "b",
            markeredgecolor="black",
            markeredgewidth=0.1,
        )

    # generate meshgrid for countour
    test_samples_amount = 100
    domain = np.linspace(
        -np.pi * 2 * FLAGS.spiral_rotations,
        np.pi * 2 * FLAGS.spiral_rotations,
        test_samples_amount,
    )
    xx, yy = np.meshgrid(domain, domain)

    points = np.vstack([xx.flatten(), yy.flatten()]).T
    values = np.zeros(points.shape[0])

    for i in trange(0, points.shape[0], FLAGS.batch_size):
        values[i : i + FLAGS.batch_size] = model(points[i : i + FLAGS.batch_size
])

    cs = ax.contourf(
        xx,
        yy,
        values.reshape(test_samples_amount, test_samples_amount),
        [0, 0.1, 0.4, 0.6, 0.9, 1],
        colors=["#ffa1a1", "#ffbaba", "#ffbaff", "#bac9ff", "#aaa1ff"],
    )

    fig1.colorbar(cs)

    ax.set_title("Spirals")
    plt.tight_layout()
    plt.savefig(f"{script_path}/fit.pdf")


if __name__ == "__main__":
    app.run(main)
```


Spirals
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
1.0
0.9
0.6
0.4
0.1
0.0