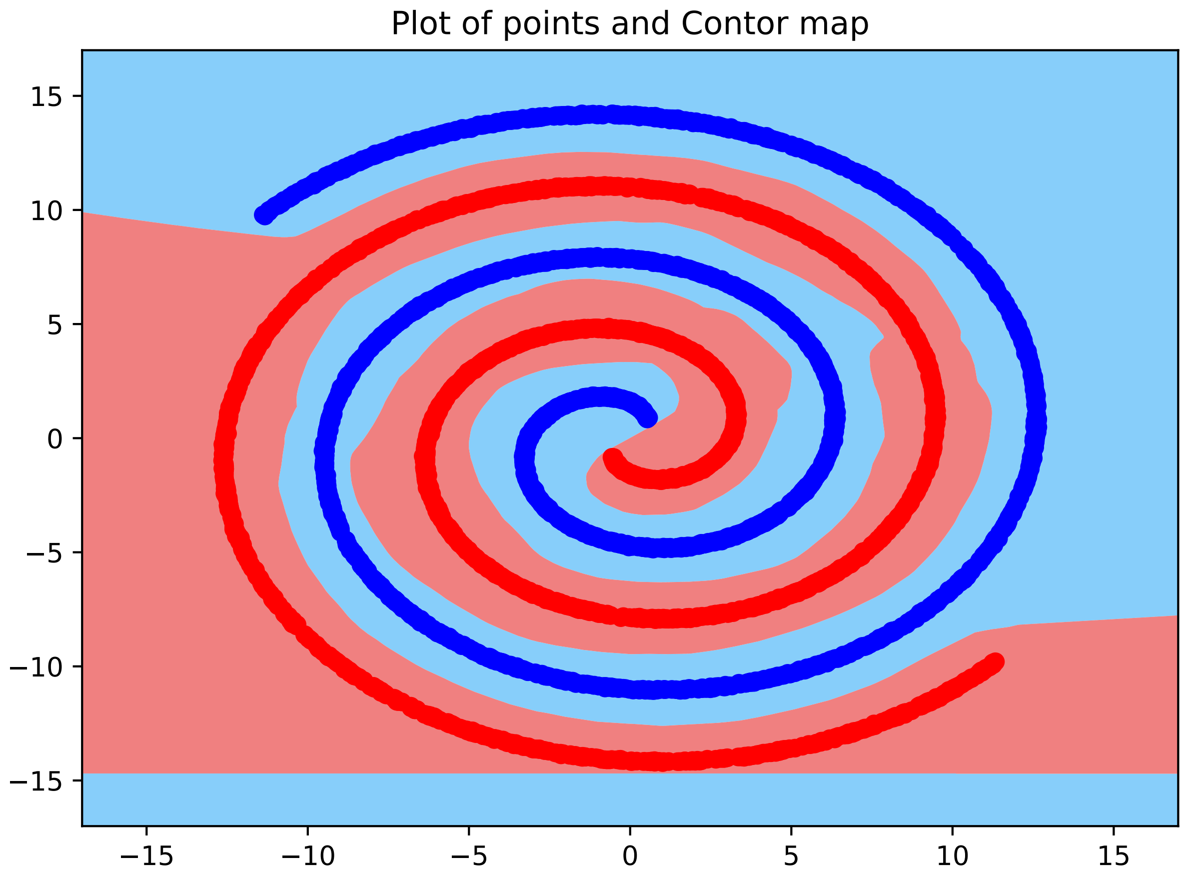
```
#!/bin/env python3.8

"""
Homework Assignment #2: Gregory Presser
"""
import os
import logging
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from dataclasses import dataclass, field, InitVar
from absl import flags,app
from tqdm import trange

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


@dataclass
class Data:
    num_samples: int
    sig: float
    range: tuple[float, float]
    x: np.ndarray = field(init=False)
    y: np.ndarray = field(init=False)
    rng: InitVar[np.random.Generator]

    def __post_init__(self, rng):
        """
        Data generation with help from
        Jacob Khalili
        """
        self.index = np.arange(self.num_samples * 2)

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

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

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

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

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

        data_1 = [x_1, y_1]
        data_2 = [x_2, y_2]
        self.x = np.concatenate([data_1, data_2], axis=1).T
        self.y = np.concatenate(([0] * self.num_samples, [1] * self.num_samples)
)

    def get_batch(self, rng, batch_size):
        """
        Select random subset of examples for training batch
        """
        choices = rng.choice(self.num_samples*2, size=batch_size)
```



```
        return self.x[choices], self.y[choices]


class Dense(tf.Module):
    def __init__(self, neurons: int, is_output: bool = False, name: str = No
        super().__init__(name=name)
        self.neurons = neurons
        self.is_output = is_output
        self.__is_built = False

    def build(self,rng, inputs: int, index: int = 0):
        self.w = tf.Variable(rng.normal(shape=[inputs, self.neurons]) * .01
e = "w" + str(index))
        self.b = tf.Variable(tf.zeros(shape=[1, self.neurons]), name = "b" +
(index))
        self.__is_built = True

    def __call__(self, x):
        if not self.__is_built:
            raise Exception("Model was never build")
        v = x @ self.w + self.b
        return tf.nn.sigmoid(v) if self.is_output else tf.nn.relu(v)


class Model(tf.Module):
    def __init__(self, rng, inputs: int, points: int, nodes: list[Dense], na
one):
        super().__init__(name=name)
        self.layers = []
        with self.name_scope:
            for (i, node) in enumerate(nodes):
                node.build(rng, inputs, i)
                self.layers.append(node)
                inputs = node.neurons

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


def loss(y, y_hat):
    EPS = 1e-15
    return tf.reduce_mean(-y * tf.math.log(y_hat + EPS) - (1-y) * tf.math.lo
y_hat + EPS))


FLAGS = flags.FLAGS
flags.DEFINE_integer("num_points", 3000, "Number of points in each spiral")
flags.DEFINE_integer("batch_size", 128, "Number of samples in batch")
flags.DEFINE_integer("random_seed", 31415, "Random seed")
flags.DEFINE_float("learning_rate", 0.001, "Learning rate")
flags.DEFINE_integer("num_iters", 5000, "Number of iterations")


def convert_to_color(val: float):
    return "blue" if val <= .5 else "red"


def main(_):
```

```
    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)

    d = Data(FLAGS.num_points, sig=.02, range=(1,15),rng=np_rng)

    model = Model(tf_rng,
                  inputs=2,
                  points=FLAGS.batch_size,
                  nodes=[
                    Dense(128),
                    Dense(128),
                    Dense(128),
                    Dense(1, True)
                  ])

    optimizer = tf.keras.optimizers.Adam(learning_rate=FLAGS.learning_rate, beta
_1 = .9, beta_2=.999, epsilon=1e-07, name="Adam")

    bar = trange(FLAGS.num_iters)
    for i in bar:
        with tf.GradientTape() as tape:
            x_train, y_train = d.get_batch(np_rng, FLAGS.batch_size)
            y_train = y_train.reshape(FLAGS.batch_size, 1)
            y_hat = model(x_train)
            ls = loss(y_train, y_hat)

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

        bar.set_description(f"Loss @ {i} => {ls.numpy():0.6f}")
        bar.refresh()

    # true_colors = [convert_to_color(y) for y in d.y]
    predictions = [convert_to_color(y) for y in model(d.x).numpy()]
    plt.scatter(d.x[::, 0], d.x[::, 1], color=predictions, zorder=10)

    x = np.linspace(-17,17,FLAGS.num_points)
    y = x
    l = len(x)
    [X,Y] = np.meshgrid(x,y)
    cords = np.vstack([X.ravel(), Y.ravel()]).T
    Z = model(cords).numpy().reshape(l,l)
    plt.contourf(X,Y,Z,[0,0.5,1], colors=["lightskyblue","lightcoral"])
    plt.title("Plot of points and Contor map")

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


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


Plot of points and Contor map
15
10
5
0
−5
−10
−15
−15
−10
−5
0
5
10
15