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

"""
Rosemary Cho
Based on Example assignment. Author: Chris Curro
"""
import os
import logging
import matplotlib
import numpy as np
import pandas as pd
from numpy import pi
import matplotlib.pyplot as plt
from pdb import set_trace
import tensorflow as tf
from absl import app
from absl import flags
from tqdm import trange
import csv

from dataclasses import dataclass, field, InitVar

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


@dataclass
class Data:

    rng: InitVar[np.random.Generator]
    num_samples: int
    noise: int
    x: np.ndarray = field(init=False)
    y: np.ndarray = field(init=False)

    def __post_init__(self, rng):
        # Reference: https://gist.github.com/45deg/e731d9e7f478de134def5668324c44c5
        self.theta = (np.random.rand(self.num_samples)) * 3.5 * pi
        self.index = np.arange(self.num_samples)
        a = 1 * self.theta + 1
        data_a = np.array([-np.cos(self.theta) * a, np.sin(self.theta) * a]).T
        x_a = data_a + np.random.randn(self.num_samples, 1) * self.noise

        b = -1 * self.theta - 1
        data_b = np.array([-np.cos(self.theta) * b, np.sin(self.theta) * b]).T
        x_b = data_b + np.random.randn(self.num_samples, 1) * self.noise

        class_a = np.append(x_a, np.zeros((self.num_samples, 1)), axis=1)
        class_b = np.append(x_b, np.ones((self.num_samples, 1)), axis=1)

        res = np.append(class_a, class_b, axis=0)

        np.savetxt(
            "initialdata.csv",
            res,
            delimiter=",",
            header="x,y,label",
            comments="",
            fmt="%.5f",
        )

        df = pd.read_csv("initialdata.csv")
```



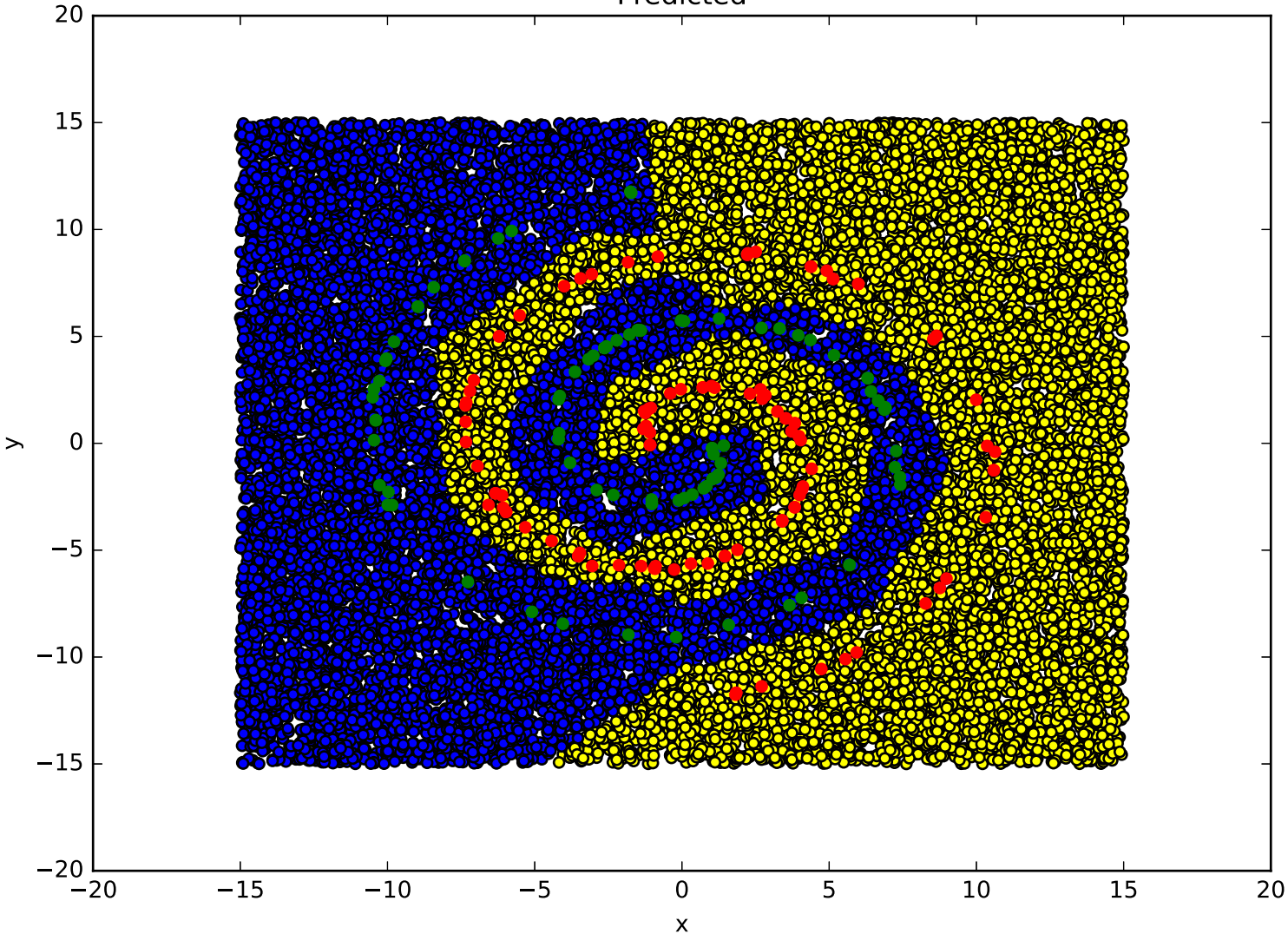
```
        df = df.sample(frac=1)
        feature = list(df.columns)
        feature.remove("label")

        self.x = df.loc[:, feature]
        self.y = df.loc[:, ["label"]]
        self.x = self.x.values
        self.y = self.y.values.flatten()

    def get_batch(self, rng, batch_size):
        """
    Select random subset of examples for training batch
    """

        choices = rng.choice(self.index, size=batch_size)
        x = self.x
        y = self.y

        return x[choices], y[choices]


font = {
    # "family": "Adobe Caslon Pro",
    "size": 10,
}

matplotlib.style.use("classic")
matplotlib.rc("font", **font)

FLAGS = flags.FLAGS
flags.DEFINE_integer("num_features", 1, "Number of features in record")
flags.DEFINE_integer("num_samples", 50, "Number of samples in dataset")
flags.DEFINE_integer("batch_size", 16, "Number of samples in batch")
flags.DEFINE_integer("num_iters", 300, "Number of SGD iterations")
flags.DEFINE_float("learning_rate", 0.1, "Learning rate / step size for SGD")
flags.DEFINE_integer("random_seed", 31415, "Random seed")
flags.DEFINE_float("noise", 0.1, "Standard deviation of noise random variable")
flags.DEFINE_bool("debug", False, "Set logging level to debug")


class Model(tf.Module):
    def __init__(self, rng, num_features, batch_size):
        """
    A plain linear regression model with a bias term
    """
        self.num_features = num_features
        self.batch_size = batch_size

        first = 16
        second = 8
        third = 4

        self.w0 = tf.Variable(rng.normal(shape=[2, first]), name="wo")
        self.w1 = tf.Variable(rng.normal(shape=[first, second]), name="w1")
        self.w2 = tf.Variable(rng.normal(shape=[second, third]), name="w2")
        self.w3 = tf.Variable(rng.normal(shape=[third, 1]), name="w3")
        self.b0 = tf.Variable(rng.normal(shape=[1, first]), name="bo")
        self.b1 = tf.Variable(rng.normal(shape=[1, second]), name="b1")
        self.b2 = tf.Variable(rng.normal(shape=[1, third]), name="b2")
        self.b3 = tf.Variable(rng.normal(shape=[1, 1]), name="b3")

    def __call__(self, x):
```

```
        relu = tf.keras.layers.ReLU()
        sigmoid = tf.keras.activations.sigmoid
        x = relu(x @ self.w0 + self.b0)
        x = relu(x @ self.w1 + self.b1)
        x = relu(x @ self.w2 + self.b2)
        x = sigmoid(x @ self.w3 + self.b3)
        result = tf.squeeze(x)
        return result


def main(a):
    logging.basicConfig()

    if FLAGS.debug:
        logging.getLogger().setLevel(logging.DEBUG)

    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,
        FLAGS.noise,
    )
    model = Model(tf_rng, FLAGS.num_features, FLAGS.batch_size)
    optimizer = tf.optimizers.SGD(learning_rate=FLAGS.learning_rate)

    loss_log = []
    bar = trange(FLAGS.num_iters)
    for i in bar:
        with tf.GradientTape() as tape:
            x, y = data.get_batch(np_rng, FLAGS.batch_size)
            y_hat = model(x)
            bce = tf.keras.losses.BinaryCrossentropy()
            lambdaa = 0.005
            loss = bce(y, y_hat) + lambdaa * tf.reduce_sum(
                (
                    tf.sqrt(tf.reduce_sum(model.trainable_variables[4] ** 2, [0,
 1]))
                    + tf.sqrt(tf.reduce_sum(model.trainable_variables[5] ** 2, [
0, 1]))
                    + tf.sqrt(tf.reduce_sum(model.trainable_variables[6] ** 2, [
0, 1]))
                    + tf.sqrt(tf.reduce_sum(model.trainable_variables[7] ** 2, [
0, 1]))
                )
            )
        grads = tape.gradient(
            loss,
            model.trainable_variables,
            unconnected_gradients=tf.UnconnectedGradients.ZERO,
        )

        optimizer.apply_gradients(zip(grads, model.trainable_variables))
        bar.set_description(f"Loss @ {i} => {loss.numpy():0.6f}")
        bar.refresh()
        loss_log.append(loss.numpy())

    result = model(x)
```



```
    class_ = np.array([])
    for i in range(0, result.shape[0]):
        if result[i] > 0.5:
            class_ = np.append(class_, 1)
        else:
            class_ = np.append(class_, 0)

    class_ = np.array([class_])
    class_ = np.reshape(class_, (-1, 1))

    y = np.reshape(y, (x.shape[0], 1))
    res = np.append(x, y, axis=1)
    res = np.append(res, class_, axis=1)
    res = np.savetxt(
        "Predicted.csv",
        res,
        delimiter=",",
        header="x,y,label,predicted_label",
        comments="",
        fmt="%.5f",
    )

    x_a = []
    y_a = []
    label = []
    predicted_labl = []

    with open("Predicted.csv", "r") as csvfile:

        plots = csv.reader(csvfile, delimiter=",")
        next(plots)
        for row in plots:
            x_a.append(float(row[0]))
            y_a.append(float(row[1]))
            label.append(float(row[2]))
            predicted_labl.append(float(row[3]))

    num = 100
    a = np.random.uniform(-15, 15, size=(FLAGS.batch_size * num, 2))
    result = []

    for i in range(0, num):
        new = np.array_split(a, num)
        res = model(new[i])
        res = res.numpy().flatten()
        result.append(res)

    array = np.array(a)
    x = array[:, 0]
    y = array[:, 1]
    result = np.array(result)
    result = result.flatten()

    color = ["blue" if 0.5 < l else "yellow" for l in result]
    pred_color = ["red" if l == 0 else "green" for l in predicted_labl]
    plt.scatter(x, y, c=color)
    plt.scatter(x_a, y_a, color=pred_color)

    plt.xlabel("x")
    plt.ylabel("y")
    plt.title("Predicted")
```

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


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

# ./tf.sh  gs -sDEVICE=pdfwrite -dNOPAUSE -dBATCH -dSAFER -sOutputFile=nonlinear
.pdf nonlinear/mainn.py.pdf nonlinear/nonlinearfit.pdf
#  ./tf.sh python nonlinear/mainn.py --num_features 1 --num_samples 600 --batch_
size 200 --num_iters 5000 --random_seed 398729765279 --debug


# After adding L2 regularization and checking that the model's loss is decreasin
g with epochs,
# I tried to improve the model by trying various L2 regularization lambda values
 and learning rates;
# I found that altering just one of them worsens the model performance.
# This is due to the coupling of L2 regularization and learning rate, which mean
s that the âM-^@M-^\two hyperparameters are interdependent
# and need to be changed simultaneously, while only changing one of them might s
ubstantially worsen resultsâM-^@M-^](from âM-^@M-^\Decoupled Weight Decay Regula
rizationâM-^@M-^]).
# Since changing one value tended to worsen the model performance, finding the o
ptimal point where the loss is minimal was challenging.
# Using a model with decoupled L2 regularization term (SGDW or AdamW) and learni
ng rate would yield better model generalization performance.
```


Predicted
20
15
10
5
0
−5
−10
−15
−20
y
−20
−15
−10
−5
0
5
10
15
20
x