```
Amaan Rahman
ECE 472: Deep Learning
Professor Curro

                                Assignment 2
                            Binary Classification

Remarks:

    Attempts to implement ReLU, Leaky-ReLU, and Sigmoid activation functions
    were made and unsuccessful; the functions themselves have been fully and
    properly implemented, however an unecessary amount time was wasted on
    integrating the "handmade" activation functions into Tensorflow. The
    realization that "handmade" activation functions require integration
    was realizing that this very reason of no integration was causing my
    model to be unable to train due to failure of gradient computation.
    The quick solution that has been used instead was to utilize the built
    in functions instead.

MultiPerceptron Design Considerations:

    One thing to note is that I don't include the input layer within my
    discussion of design considerations (only hidden layers and output layer).

    Initially, I decided on testing 8->4->2->1 setup, however my loss didn't con
verge.
    I ramped the widths up by about times 4, and it didn't converge. I then ramp
ed the
    widths by 10 fold about and then I noticed convergence over 1500 iterations
given
    a batch size of 32. This "funnel" design yielded losses to as low as 0.003 o
r possibly
    even lower. I tested out my final design, which is the "hourglass" configrua
tion:

    100->75->50->25->50->75->100->1

    This design yielded optimal convergence compared to all permutations I have
tested out
    thus far, yielding losses as low as 0.000002.

Citations:

Training function reference from Professor Curro's example
```


```
@misc{brownlee_plot_2020,
    title = {Plot a {Decision} {Surface} for {Machine} {Learning} {Algorithms} i
n {Python}},
    url = {https://machinelearningmastery.com/plot-a-decision-surface-for-machin
e-learning/},
    abstract = {Classification algorithms learn how to assign class labels to ex
amples, although their decisions can appear opaque. A popular
    diagnostic for [âM-^@¦]},
    language = {en-US},
    urldate = {2021-09-19},
    journal = {Machine Learning Mastery},
    author = {Brownlee, Jason},
    month = aug,
    year = {2020},
    file = {Snapshot:C\:\\Users\\Amaan\\Zotero\\storage\\XRTBM7XD\\plot-a-decisi
on-surface-for-machine-learning.html:text/html},
}
```



```
@misc{noauthor_archimedean_2021,
    title = {Archimedean spiral},
    copyright = {Creative Commons Attribution-ShareAlike License},
    url = {https://en.wikipedia.org/w/index.php?title=Archimedean_spiral&oldid=1
039754847},
    abstract = {The Archimedean spiral (also known as the arithmetic spiral) is
a spiral named after the 3rd-century BC Greek
    mathematician Archimedes. It is the locus corresponding to the locations ove
r time of a point moving away from a fixed point with a
    constant speed along a line that rotates with constant angular velocity. Equ
ivalently, in polar coordinates (r, Î¸) it can be described
    by the equation




        r
        =
        a
        +
        b
        âM-^KM-^E
        Î¸


    \{{\textbackslash}displaystyle r=a+b{\textbackslash}cdot {\textbackslash}the
ta \}
    with real numbers a and b. Changing the parameter a moves the centerpoint of
 the spiral outward from the origin (positive a toward Î¸ = 0 and negative a tow
ard Î¸ = ÏM-^@)
    essentially through a rotation of the spiral, while b controls the distance
between loops.
    From the above equation, it can thus be stated: the position of particle fro
m the point of start is proportional to the angle Î¸ as time elapses.
    Archimedes described such a spiral in his book On Spirals.  Conon of Samos w
as a friend of his and Pappus states that this spiral was discovered by Conon.},
    language = {en},
    urldate = {2021-09-19},
    journal = {Wikipedia},
    month = aug,
    year = {2021},
    note = {Page Version ID: 1039754847},
    file = {Snapshot:C\:\\Users\\Amaan\\Zotero\\storage\\XXALYR9E\\index.html:te
xt/html},
}

@misc{noauthor_python_nodate,
    title = {python - {Pandas} \& {MatPlotLib}: {Plot} a {Bar} {Graph} on {Exist
ing} {Scatter} {Plot} or {Vice} {Versa}},
    shorttitle = {python - {Pandas} \& {MatPlotLib}},
    url = {https://stackoverflow.com/questions/49991227/pandas-matplotlib-plot-a
-bar-graph-on-existing-scatter-plot-or-vice-versa},
    urldate = {2021-09-19},
    journal = {Stack Overflow},
    file = {Snapshot:C\:\\Users\\Amaan\\Zotero\\storage\\HS537TGY\\pandas-matplo
tlib-plot-a-bar-graph-on-existing-scatter-plot-or-vice-versa.html:text/html},
}
```

```
"""
Amaan Rahman
ECE 472: Deep Learning
Assigment 2: Binary Classification
"""

import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from tqdm import trange

# ---- Global Variables ----
NUM_SAMPLES = 500
BATCH_SIZE = 32
NUM_ITR = 2000
SEED = 1618
SIGMA_NOISE = 0.1
ROT_NUM = 2

# class for generating data
class Data(object):
    def __init__(self, num_samples, sigma, id, attr):

        # spiral attributes
        theta = np.random.uniform(attr["min"], attr["max"], size=(num_samples))
        spiral = self.Spiral(attr["center"], attr["gap"], theta, 1)

        # generate data
        factor = 1 if id == 1 else -1
        noise = sigma * np.random.normal(size=(num_samples))  # gaussian noise
        self.x = (
            factor * spiral.r * np.cos(theta) / 1.5 + noise
        )  # arbitrary scaling factor
        self.y = factor * spiral.r * np.sin(theta) + noise

        self.spiral = spiral._data((self.x, self.y, [id] * num_samples))

    def _init_input(self, data):
        self.data = tf.constant(data[0 : data.shape[0] - 1], dtype=np.float32)
        self.labels = tf.constant(
            data[data.shape[0] - 1], shape=[1, data.shape[1]], dtype=np.float32
        )

    def _batchGet(self, batch_size):
        self.index = NUM_SAMPLES * 2
        rand_ind = np.random.choice(self.index, size=batch_size)
        batch_data = tf.squeeze(tf.gather(self.data, rand_ind, axis=1))
        batch_labels = tf.squeeze(tf.gather(self.labels, rand_ind, axis=1))

        # normalize data
        return (
            batch_data,
            batch_labels,
        )

    # https://en.wikipedia.org/wiki/Archimedean_spiral
    class Spiral(object):
        def __init__(self, a, b, theta, n):
            self.r = a + b * (theta ** (1 / n))

        def _data(self, xy_dat):
            self.data = xy_dat
```

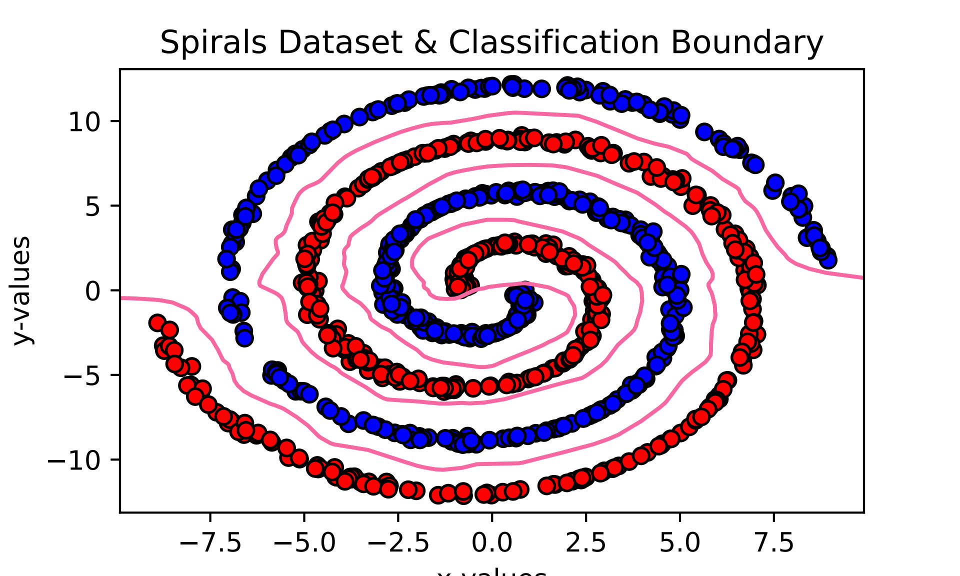


```
            return self


class MLP(tf.Module):
    def __init__(self, X_features, depth, width_arr):
        self.W = [None] * depth
        self.B = [None] * depth
        for width, k in zip(width_arr, range(1, depth + 1)):
            self.W[k - 1] = tf.Variable(
                0.2 * tf.random.normal(shape=[X_features, width]),
                name=("WEIGHTS_" + str(k)),
                dtype=np.float32,
            )
            self.B[k - 1] = tf.Variable(
                0.001 * tf.ones(shape=[width, 1]),
                name=("BIAS_" + str(k)),
                dtype=np.float32,
            )

            X_features = width

    def __call__(self, X):  # output from current layer
        X_k = X
        for W_k, B_k in zip(self.W, self.B):
            func = tf.nn.relu if W_k.shape[1] != 1 else tf.nn.sigmoid
            self.Z = tf.squeeze(func(((tf.transpose(W_k) @ X_k) + B_k)))
            X_k = tf.squeeze(self.Z)
        return self.Z  # output is the predicted probabilities for input batch


def train(data, model):
    optimizer = tf.optimizers.Adam()
    bar = trange(NUM_ITR)
    loss_dat = [0] * NUM_ITR
    for i in bar:
        with tf.GradientTape() as tape:
            X, y_true = data._batchGet(BATCH_SIZE)
            y_hat = model(X)
            loss_dat[i] = tf.losses.binary_crossentropy(y_true, y_hat)

        grads = tape.gradient(loss_dat[i], model.trainable_variables)
        optimizer.apply_gradients(zip(grads, model.trainable_variables))
        bar.set_description(f"Loss @ {i} => {loss_dat[i].numpy():0.6f}")
        bar.refresh()

    return loss_dat


# https://machinelearningmastery.com/plot-a-decision-surface-for-machine-learnin
g/
def decision_surf(data, model):
    min1, max1 = data[0, :].min() - 1, data[0, :].max() + 1
    min2, max2 = data[1, :].min() - 1, data[1, :].max() + 1

    x1grid = np.arange(min1, max1, 0.1)
    x2grid = np.arange(min2, max2, 0.1)
    X, Y = np.meshgrid(x1grid, x2grid)
    r1, r2 = X.flatten(), Y.flatten()
    r1, r2 = r1.reshape((1, len(r1))), r2.reshape((1, len(r2)))
    G = np.vstack((r1, r2))
    Z = tf.reshape(model(G), shape=X.shape)
    return (X, Y, Z)
```

```python
# very messy data object setup :/
# generating 2 seperate data objects
def main():
    np.random.seed(SEED)
    # generate 2 Archimidean spirals
    dataset = (
        Data(
            NUM_SAMPLES,
            SIGMA_NOISE,
            1,
            {"min": -ROT_NUM * 2 * np.pi + 0.1, "max": -0.1, "center": -1, "gap": 1
},
        ),
        Data(
            NUM_SAMPLES,
            SIGMA_NOISE,
            0,
            {"min": -ROT_NUM * 2 * np.pi + 0.1, "max": -0.1, "center": -1, "gap": 1
},
        ),
    )

    spiral_A = list(
        zip(
            dataset[0].spiral.data[0],
            dataset[0].spiral.data[1],
            dataset[0].spiral.data[2],
        )
    )
    spiral_B = list(
        zip(
            dataset[1].spiral.data[0],
            dataset[1].spiral.data[1],
            dataset[1].spiral.data[2],
        )
    )
    input_data = np.concatenate((spiral_A, spiral_B), axis=0)
    dataset[0]._init_input(input_data.T)
    mlp_model = MLP(dataset[0].data.shape[0], 8, [100, 75, 50, 25, 50, 75, 100,
1])
    train(dataset[0], mlp_model)
    prob_surf = decision_surf(dataset[0].data.numpy(), mlp_model)

    # https://stackoverflow.com/questions/49991227/pandas-matplotlib-plot-a-bar-
graph-on-existing-scatter-plot-or-vice-versa
    fig = plt.figure(figsize=(5, 3), dpi=200)
    ax = fig.add_subplot(111)
    ax.contour(*prob_surf, cmap="RdPu", linestyles="solid", levels=1)
    ax.scatter(
        input_data[0:NUM_SAMPLES, 0],
        input_data[0:NUM_SAMPLES, 1],
        c="r",
        edgecolors="k",
    )
    ax.scatter(
        input_data[NUM_SAMPLES:, 0], input_data[NUM_SAMPLES:, 1], c="b", edgecol
ors="k"
    )
    ax.set_title("Spirals Dataset & Classification Boundary")
    ax.set(xlabel="x-values", ylabel="y-values")
```



```python
    plt.savefig("output1.pdf")


if __name__ == "__main__":
    main()
```



Spirals Dataset & Classification Boundary
y-values
10
5
0
−5
−10
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5

```
compile:
	black bin_class.py
	flake8 --ignore=E,W bin_class.py
	python3 bin_class.py

pdf:
	a2ps bin_class.py -o bin_class.ps --pro=color
	a2ps Makefile -o Makefile.ps --pro=color
	a2ps README.txt -o README.ps --pro=color
	ps2pdf bin_class.ps
	ps2pdf Makefile.ps
	ps2pdf README.ps
	gs -dBATCH -dNOPAUSE -q -sDEVICE=pdfwrite -sOutputFile=classif_AR.pdf RE
ADME.pdf bin_class.pdf output.pdf Makefile.pdf
```