

Sep 15, 22 1:33	main.py	Page 1/4
<pre>#!/bin/env python3.8 """ Allister Liu """ import sys import matplotlib.pyplot as plt import numpy as np import tensorflow as tf from absl import flags from tqdm import trange FLAGS = flags.FLAGS flags.DEFINE_integer("sample_size", 1000, "Number of samples in dataset") flags.DEFINE_integer("batch_size", 32, "Number of samples in batch") flags.DEFINE_integer("num_iters", 300, "Number of SGD iterations") def get_spirals(size=500): """ Generate the two spirals with noise with sample size n :return: (x1, y1) => spiral 1 & (x2, y2) => spiral 2 # https://www.engineerknow.com/2021/02/spiral-graph-plotting-in-python.html """ r = np.linspace(1, 15, size, dtype="float32") t_1 = np.linspace(0, 12, size) t_2 = np.linspace(3, 15, size) x_1 = r * np.cos(t_1) + np.random.normal(scale=0.2, size=size) y_1 = r * np.sin(t_1) + np.random.normal(scale=0.2, size=size) x_2 = r * np.cos(t_2) + np.random.normal(scale=0.2, size=size) y_2 = r * np.sin(t_2) + np.random.normal(scale=0.2, size=size) return x_1, y_1, x_2, y_2 class Data(object): """ Generate datas """ def __init__(self, sam_size, x_1, y_1, x_2, y_2): """ Combine the XY coordinates of the spiral datas into one list, and label each data point with its corresponding class [0,1] :param sam_size: sample size :param x_1: x coordinates of spiral 1 :param y_1: y coordinates of spiral 1 :param x_2: x coordinates of spiral 2 :param y_2: y coordinates of spiral 2 """ self.sam_size = sam_size self.index = np.arange(sam_size) # index for get_batch() x_coor = np.concatenate([x_1, x_2]) y_coor = np.concatenate([y_1, y_2]) xy_coor = np.array(list(zip(x_coor, y_coor))[0]))</pre>		

Sep 15, 22 1:33	main.py	Page 2/4
<pre>class_labels = np.zeros(sam_size // 2, dtype="float32") class_labels = np.concatenate([class_labels, np.ones(sam_size // 2, dtype="float32")]) self.xy_coor = np.float32(xy_coor) self.class_labels = np.float32(class_labels) def get_batch(self, batch_size): """ Get a small batch of the randomly chosen datas :param batch_size: batch size :return: a small batch of randomly chosen data point coordinates with their corresponding class label """ choices = np.random.choice(self.index, size=batch_size) return self.xy_coor[choices], self.class_labels[choices].reshape((batch_size, 1)) class Model(tf.Module): def __init__(self): """ Initialize the model with 3 hidden layers: (2, 64) => (64, 32) => (32, 16) => (16, 1) weights are initialized to be a random normal distribution, and biases are initialized to zeros """ w0 = tf.Variable(tf.random.normal(shape=(2, 64)), name="w0") w1 = tf.Variable(tf.random.normal(shape=(64, 32)), name="w1") w2 = tf.Variable(tf.random.normal(shape=(32, 16)), name="w2") w3 = tf.Variable(tf.random.normal(shape=(16, 1)), name="w3") # w4 = tf.Variable(tf.random.normal(shape=(8, 1)), name="w4") self.weights = [w0, w1, w2, w3] b0 = tf.Variable(tf.zeros(64), name="b0") b1 = tf.Variable(tf.zeros(32), name="b1") b2 = tf.Variable(tf.zeros(16), name="b2") b3 = tf.Variable(tf.zeros(1), name="b3") # b4 = tf.Variable(tf.zeros(1), name="b4") self.biases = [b0, b1, b2, b3] def predict(self, datas): """ Calculate y_hat :param datas: xy coordinates of the given data points :return: y_hat """ for i, (w, b) in enumerate(zip(self.weights, self.biases)): if i == 0: self.y_h = datas @ w + b else: self.y_h = tf.nn.elu(self.y_h) @ w + b return self.y_h def loss(self, prediction, target): """ Calculate loss with L2 penalty :param prediction: y_hat :param target: y :return: loss with L2 penalty """ y_h = prediction</pre>		

Sep 15, 22 1:33	main.py	Page 3/4
	<pre> y_ = target bce = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=y_h, labels=y_)) # binary cross entropy # y_h) - (1 - y_) * tf.math.log(1 - y_h) l2 = 0 for wei in model.weights: l2 += tf.nn.l2_loss(wei) return tf.reduce_mean(bce + 0.01 * l2) if __name__ == "__main__": # Handle the flags FLAGS(sys.argv) SAMPLE_SIZE = FLAGS.sample_size BATCH_SIZE = FLAGS.batch_size NUM_ITERS = FLAGS.num_iters # Generate spirals x1, y1, x2, y2 = get_spirals(SAMPLE_SIZE // 2) # Generate data (coordinates of two noisy spirals with their class label) data = Data(sam_size=SAMPLE_SIZE, x_1=x1, y_1=y1, x_2=x2, y_2=y2) # Initialize model weights and biases model = Model() # Using Adam as optimizer with learning rate of .05 optimizer = tf.optimizers.Adam(learning_rate=0.05) losses = np.zeros(NUM_ITERS) # Train bar = trange(NUM_ITERS) for i in bar: with tf.GradientTape() as tape: # Get batch x, y = data.get_batch(BATCH_SIZE) # Get prediction y_hat y_hat = model.predict(x) # Calculate loss loss = model.loss(y_hat, y) losses[i] = loss gradients = tape.gradient(loss, model.variables) optimizer.apply_gradients(zip(gradients, model.variables)) bar.set_description(f"Loss @ {i} => {loss.numpy():0.6f}") bar.refresh() x_truth = data.xy_coor[:, 0] y_truth = data.xy_coor[:, 1] # Get evenly distributed sample points within the area of interest (-15 <= x <= 15, -15 <= y <= 15) # at 0.1 increments n = 301 x_samp = np.linspace(-15, 15, n, dtype="float32") y_samp = np.linspace(-15, 15, n, dtype="float32") (</pre>	

Sep 15, 22 1:33	main.py	Page 4/4
	<pre> x_samp_plt, y_samp_plt,) = np.meshgrid(x_samp, y_samp) # Get our model's prediction for each sample point z = np.zeros((n, n)) # decision mesh grid for i in range(n): for j in range(n): p = model.predict(np.array([[x_samp[j], y_samp[i]]])) if p >= 0.5: z[i, j] = 1 cm = plt.cm.RdBu # color map plt.contourf(x_samp, y_samp, z, cmap=cm, alpha=0.7) plt.scatter(x_truth, y_truth, c=data.class_labels, cmap=cm, alpha=1, edgecolors="black") plt.show() n, x, _ = plt.hist(losses) plt.ylim(0, 1) plt.show() """ After creating my model, I first started experimenting with hidden layer dimensions, and found (2, 64) => (64,32) => (32,16) => (16,1) would give the best prediction. Then I started playing around with the activation functions: starting with ReLU as most people, I figured that for some reason my loss often gets stuck at around 0.7. After trying different activation functions, I found that changing to ELU seems to solve this problem. Also, for my loss function, I was trying to use the formula discussed in class to implement my own bce function, but since my weights are centered around zero, it often calculates log(0), which in turn makes my loss goes to nan. HUGE SHOUT OUT to Thodoris who pointed this out, and suggested that I could either try to add some very small number like e^(-15) when taking the log, or I can just use tensorflow's builtin function for bce I then tweaked the learning rates for Adam optimizer and the beta factor for L2 penalty, found my satisfied values in the end. Finally, I figured that 5000 would be a good the number of iterations for training """ </pre>	

