notes_23

January 20, 2025

1 Previous Class Definitions

[10]:
```
# imports
import matplotlib.pyplot as plt
import numpy as np
import nnfs
from nnfs.datasets import spiral_data, vertical_data
nnfs.init()
```

[11]:
```
class Layer_Dense:
    def __init__(self, n_inputs, n_neurons):
        # Initialize the weights and biases
        self.weights = 0.01 * np.random.randn(n_inputs, n_neurons)  # Normal␣
 ↪distribution of weights
        self.biases = np.zeros((1, n_neurons))

    def forward(self, inputs):
        # Calculate the output values from inputs, weights, and biases
        self.inputs = inputs
        self.output = np.dot(inputs, self.weights) + self.biases        #␣
 ↪Weights are already transposed

    def backward(self, dvalues):
        '''Calculated the gradient of the loss with respect to the weights and␣
 ↪biases of this layer.
        dvalues is equiavelent to a transposed dl_dZ. It is the gradient
        of the loss with respect to the outputs of this layer.'''
        self.dweights = np.dot(self.inputs.T, dvalues)
        self.dbiases = np.sum(dvalues, axis=0, keepdims=0)
        self.dinputs = np.dot(dvalues, self.weights.T)

class Activation_ReLU:
    def forward(self, inputs):
        self.inputs = inputs
        self.output = np.maximum(0, inputs)

    def backward(self, dvalues):
```

```
        '''Calculated the gradient of the loss with respect to this layer's␣
 ↪activation function
        dvalues is equiavelent to a transposed dl_dZ. It is the gradient
        of the loss with respect to the outputs of this layer.'''
        self.dinputs = dvalues.copy()
        self.dinputs[self.inputs <= 0] = 0

class Activation_Softmax:
    def forward(self, inputs):
        # Get the unnormalized probabilities
        # Subtract max from the row to prevent larger numbers
        exp_values = np.exp(inputs - np.max(inputs, axis=1, keepdims=True))

        # Normalize the probabilities with element wise division
        probabilities = exp_values / np.sum(exp_values, axis=1,keepdims=True)
        self.output = probabilities

# Base class for Loss functions
class Loss:
    '''Calculates the data and regularization losses given
    model output and ground truth values'''
    def calculate(self, output, y):
        sample_losses = self.forward(output, y)
        data_loss = np.average(sample_losses)
        return data_loss

class Loss_CategoricalCrossEntropy(Loss):
    def forward(self, y_pred, y_true):
        '''y_pred is the neural network output
        y_true is the ideal output of the neural network'''
        samples = len(y_pred)
        # Bound the predicted values
        y_pred_clipped = np.clip(y_pred, 1e-7, 1-1e-7)

        if len(y_true.shape) == 1:      # Categorically labeled
            correct_confidences = y_pred_clipped[range(samples), y_true]
        elif len(y_true.shape) == 2:    # One hot encoded
            correct_confidences = np.sum(y_pred_clipped*y_true, axis=1)

        # Calculate the losses
        negative_log_likelihoods = -np.log(correct_confidences)
        return negative_log_likelihoods

    def backward(self, dvalues, y_true):
        samples = len(dvalues)

        # Number of lables in each sample
```

```
        labels = len(dvalues[0])

        # if the labels are sparse, turn them into a one-hot vector
        if len(y_true.shape) == 1:
            y_true = np.eye(labels)[y_true]

        # Calculate the gradient then normalize
        self.dinputs = -y_true / dvalues
        self.dinputs = self.dinputs / samples

class Activation_Softmax_Loss_CategoricalCrossentropy():
    def __init__(self):
        self.activation = Activation_Softmax()
        self.loss = Loss_CategoricalCrossEntropy()

    def forward(self, inputs, y_true):
        self.activation.forward(inputs)
        self.output = self.activation.output
        return self.loss.calculate(self.output, y_true)

    def backward(self, dvalues, y_true):
        samples = len(dvalues)

        # if the samples are one-hot encoded, turn them into discrete values
        if len(y_true.shape) == 2:
            y_true = np.argmax(y_true, axis=1)

        # Copy so we can safely modify
        self.dinputs = dvalues.copy()

        # Calculate and normalize gradient
        self.dinputs[range(samples), y_true] -= 1
        self.dinputs = self.dinputs / samples

class Optimizer_SGD():
    def __init__(self, learning_rate=0.5):
        self.learning_rate = learning_rate

    def update_params(self, layer):
        layer.weights += -self.learning_rate * layer.dweights
        layer.biases += -self.learning_rate * layer.dbiases
```

2 Previous Notes and Notation

The previous notation is clunky and long. From here forward, we will use the following notation for a layer with n inputs and i neurons. The neruon layer has is followed by an activation layer and then fed into a final value y with a computed loss l. There can be j batches of data.

$\vec{X}_j = \begin{bmatrix} x_{1j} & x_{2j} & \cdots & x_{nj} \end{bmatrix}$ -> Row vector for the layer inputs for the j batch of data.

$\overline{\overline{W}} = \begin{bmatrix} \vec{w}_1 \\ \vec{w}_2 \\ \vdots \\ \vec{w}_i \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{i1} & w_{i2} & \cdots & w_{in} \end{bmatrix}$ -> Matrix of weight values. Each row is a neuron's weights and each column is the weights for a given input.

$\vec{B} = \begin{bmatrix} b_1 & b_2 & \cdots & b_i \end{bmatrix}$ -> Row vector for the neuron biases

$\vec{Z}_j = \begin{bmatrix} z_{1j} & z_{2j} & \cdots & z_{ij} \end{bmatrix}$ -> Row vector for the neuron outputs for the j batch of data.

$\vec{A}_j = \begin{bmatrix} a_{1j} & a_{2j} & \cdots & a_{ij} \end{bmatrix}$ -> Row vector for the activation later outputs for the j batch of data.

y_j -> Final layer output for the j batch of data if the layer is the final layer (could be summation, probability, etc).

l_j -> Loss for the j batch of data.

The j is often dropped because we typically only need to think with 1 set of input data.

2.1 Gradient Descent Using New Notation

We will look at the weight that the i neuron applies for the n input.

$\frac{\delta l}{\delta w_{in}} = \frac{\delta l}{\delta y} \frac{\delta y}{\delta a_i} \frac{\delta a_i}{\delta z_i} \frac{\delta z_i}{\delta w_{in}}$

Similarly, for the bias of the i neuron, there is

$\frac{\delta l}{\delta b_i} = \frac{\delta l}{\delta y} \frac{\delta y}{\delta a_i} \frac{\delta a_i}{\delta z_i} \frac{\delta z_i}{\delta b_i}$

For the system we are using, where $l = (y - 0)^2$ and the activation layer is ReLU, we have

$\frac{\delta l}{\delta y} = 2y$

$\frac{\delta y}{\delta a_i} = 1$

$\frac{\delta a_i}{\delta z_i} = 1$ if $z_i > 0$ else 0

$\frac{\delta z_i}{\delta w_{in}} = x_n$

$\frac{\delta z_i}{\delta b_i} = 1$

2.2 Matrix Representation of Gradient Descent

We can simplify by seeing that $\frac{\delta l}{\delta y} \frac{\delta y}{\delta a_i} \frac{\delta a_i}{\delta z_i} = \frac{\delta l}{\delta z_i}$ is a common term.

We take $\frac{\delta l}{\delta z_i}$ and turn it into a 1 x i vector that such that

$\frac{\delta l}{\delta \vec{Z}} = \begin{bmatrix} \frac{\delta l}{\delta z_1} & \frac{\delta l}{\delta z_2} & \cdots & \frac{\delta l}{\delta z_i} \end{bmatrix}$

We than can get that the gradient matrix for all weights is a i x n matrix given by

$$\frac{\delta l}{\delta \overline{\overline{W}}} = \begin{bmatrix} \frac{\delta l}{\delta w_{11}} & \frac{\delta l}{\delta w_{12}} & \cdots & \frac{\delta l}{\delta w_{1n}} \\ \frac{\delta l}{\delta w_{21}} & w\frac{\delta l}{\delta w_{22}} & \cdots & \frac{\delta l}{\delta w_{2n}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\delta l}{\delta w_{i1}} & \frac{\delta l}{\delta w_{i2}} & \cdots & \frac{\delta l}{\delta w_{in}} \end{bmatrix} = \begin{bmatrix} \frac{\delta l}{\delta z_1} \\ \frac{\delta l}{\delta z_2} \\ \vdots \\ \frac{\delta l}{\delta z_n} \end{bmatrix} \begin{bmatrix} \frac{\delta z_1}{\delta w_{i1}} & \frac{\delta z_1}{\delta w_{i1}} & \cdots & \frac{\delta z_1}{\delta w_{in}} \end{bmatrix} = \begin{bmatrix} \frac{\delta l}{\delta z_1} \\ \frac{\delta l}{\delta z_2} \\ \vdots \\ \frac{\delta l}{\delta z_n} \end{bmatrix} \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}$$

Similarly, the gradient vector for the biases is given by $\frac{\delta l}{\delta \vec{B}} = \frac{\delta l}{\delta \vec{Z}} \frac{\delta \vec{Z}}{\delta \vec{B}} = \vec{1} \begin{bmatrix} \frac{\delta l}{\delta z_1} & \frac{\delta l}{\delta z_2} & \cdots & \frac{\delta l}{\delta z_i} \end{bmatrix}$

2.3 Gradients of the Loss with Respect to Inputs

When chaining multiple layers together, we will need the partial derivatives of the loss with respect to the next layers input (ie, the output of the current layer). This involves extra summation because the output of 1 layer is fed into every neuron of the next layer, so the total loss must be found.

The gradient of the loss with respect to the n input fed into i neurons is

$$\frac{\delta l}{\delta x_n} = \frac{\delta l}{\delta z_1}\frac{\delta z_1}{\delta x_n} + \frac{\delta l}{\delta z_2}\frac{\delta z_2}{\delta x_n} + ... + \frac{\delta l}{\delta z_i}\frac{\delta z_i}{\delta x_n}$$

Noting that $\frac{\delta z_i}{\delta x_n} = w_{in}$ allows us to have

$$\frac{\delta l}{\delta \vec{X}} = \begin{bmatrix} \frac{\delta l}{\delta x_1} & \frac{\delta l}{\delta x_2} & \cdots & \frac{\delta l}{\delta x_n} \end{bmatrix} = \begin{bmatrix} \frac{\delta l}{\delta z_1} & \frac{\delta l}{\delta z_2} & \cdots & \frac{\delta l}{\delta z_n} \end{bmatrix} \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{i1} & w_{i2} & \cdots & w_{in} \end{bmatrix}$$

2.4 Note With Layer_Dense class

The Layer_Dense class has the weights stored in the transposed fashion for forward propagation. Therefore, the weight matrix must be transposed for the backpropagation.

3 Learning Rate Decay

Start with a larger learning rate and gradually decrease it.

$\alpha = \frac{\alpha_0}{1 + decay * t}$

[16]:
```
class Optimizer_SGD():
    def __init__(self, learning_rate=0.5, decay=0.0):
        self.initial_rate = learning_rate
        self.current_learning_rate = self.initial_rate
        self.decay = decay
        self.iterations = 0

    def pre_update_params(self):
        # Update the current_learning_rate before updating params
        if self.decay:
            self.current_learning_rate = self.initial_rate / (1 + self.decay *␣
 ↪self.iterations)

    def update_params(self, layer):
        layer.weights += -self.current_learning_rate * layer.dweights
```

```
        layer.biases += -self.current_learning_rate * layer.dbiases

    def post_update_params(self):
        # Update the self.iterations for use with decay
        self.iterations += 1
```

4 Testing the Learning Rate Decay

[ ]:
```
# Create dataset
X, y = spiral_data(samples=100, classes=3)

# Create Dense layer with 2 input features and 64 output values
dense1 = Layer_Dense(2, 64)

# Create ReLU activation (to be used with Dense layer)
activation1 = Activation_ReLU()

# Create second Dense layer with 64 input features (as we take output
# of previous layer here) and 3 output values (output values)
dense2 = Layer_Dense(64, 3)

# Create Softmax classifier's combined loss and activation
loss_activation = Activation_Softmax_Loss_CategoricalCrossentropy()

# Create optimizer
optimizer = Optimizer_SGD(learning_rate=1.0, decay=1e-3)

# Train in loop
for epoch in range(10001):
    # Perform a forward pass of our training data through this layer
    dense1.forward(X)

    # Perform a forward pass through activation function
    # takes the output of first dense layer here
    activation1.forward(dense1.output)

    # Perform a forward pass through second Dense layer
    # takes outputs of activation function of first layer as inputs
    dense2.forward(activation1.output)

    # Perform a forward pass through the activation/loss function
    # takes the output of second dense layer here and returns loss
    loss = loss_activation.forward(dense2.output, y)

    # Calculate accuracy from output of activation2 and targets
    # calculate values along first axis
```

```
    predictions = np.argmax(loss_activation.output, axis=1)
    if len(y.shape) == 2:
        y = np.argmax(y, axis=1)
    accuracy = np.mean(predictions == y)

    if not epoch % 100:
        print(f'epoch: {epoch}, ' +
              f'acc: {accuracy:.3f}, ' +
              f'loss: {loss:.3f}')

    # Backward pass
    loss_activation.backward(loss_activation.output, y)
    dense2.backward(loss_activation.dinputs)
    activation1.backward(dense2.dinputs)
    dense1.backward(activation1.dinputs)

    # Update weights and biases
    optimizer.pre_update_params()
    optimizer.update_params(dense1)
    optimizer.update_params(dense2)
    optimizer.post_update_params()
```

```
epoch: 0, acc: 0.353, loss: 1.099
epoch: 100, acc: 0.467, loss: 1.079
epoch: 200, acc: 0.450, loss: 1.067
epoch: 300, acc: 0.447, loss: 1.065
epoch: 400, acc: 0.433, loss: 1.064
epoch: 500, acc: 0.430, loss: 1.062
epoch: 600, acc: 0.423, loss: 1.061
epoch: 700, acc: 0.433, loss: 1.059
epoch: 800, acc: 0.453, loss: 1.056
epoch: 900, acc: 0.467, loss: 1.054
epoch: 1000, acc: 0.467, loss: 1.051
epoch: 1100, acc: 0.463, loss: 1.048
epoch: 1200, acc: 0.463, loss: 1.045
epoch: 1300, acc: 0.467, loss: 1.041
epoch: 1400, acc: 0.470, loss: 1.037
epoch: 1500, acc: 0.480, loss: 1.032
epoch: 1600, acc: 0.480, loss: 1.026
epoch: 1700, acc: 0.490, loss: 1.020
epoch: 1800, acc: 0.493, loss: 1.014
epoch: 1900, acc: 0.493, loss: 1.006
epoch: 2000, acc: 0.507, loss: 0.998
epoch: 2100, acc: 0.510, loss: 0.989
epoch: 2200, acc: 0.527, loss: 0.980
epoch: 2300, acc: 0.540, loss: 0.971
epoch: 2400, acc: 0.433, loss: 0.976
```

```
epoch: 2500, acc: 0.430, loss: 0.973
epoch: 2600, acc: 0.430, loss: 0.968
epoch: 2700, acc: 0.433, loss: 0.964
epoch: 2800, acc: 0.433, loss: 0.958
epoch: 2900, acc: 0.440, loss: 0.953
epoch: 3000, acc: 0.443, loss: 0.948
epoch: 3100, acc: 0.447, loss: 0.943
epoch: 3200, acc: 0.457, loss: 0.939
epoch: 3300, acc: 0.467, loss: 0.935
epoch: 3400, acc: 0.470, loss: 0.930
epoch: 3500, acc: 0.477, loss: 0.927
epoch: 3600, acc: 0.487, loss: 0.923
epoch: 3700, acc: 0.483, loss: 0.919
epoch: 3800, acc: 0.480, loss: 0.915
epoch: 3900, acc: 0.480, loss: 0.911
epoch: 4000, acc: 0.487, loss: 0.907
epoch: 4100, acc: 0.503, loss: 0.904
epoch: 4200, acc: 0.513, loss: 0.900
epoch: 4300, acc: 0.517, loss: 0.896
epoch: 4400, acc: 0.523, loss: 0.893
epoch: 4500, acc: 0.533, loss: 0.889
epoch: 4600, acc: 0.533, loss: 0.886
epoch: 4700, acc: 0.537, loss: 0.882
epoch: 4800, acc: 0.537, loss: 0.878
epoch: 4900, acc: 0.537, loss: 0.875
epoch: 5000, acc: 0.547, loss: 0.871
epoch: 5100, acc: 0.543, loss: 0.868
epoch: 5200, acc: 0.547, loss: 0.865
epoch: 5300, acc: 0.550, loss: 0.861
epoch: 5400, acc: 0.553, loss: 0.857
epoch: 5500, acc: 0.557, loss: 0.854
epoch: 5600, acc: 0.567, loss: 0.850
epoch: 5700, acc: 0.563, loss: 0.846
epoch: 5800, acc: 0.567, loss: 0.843
epoch: 5900, acc: 0.567, loss: 0.840
epoch: 6000, acc: 0.567, loss: 0.837
epoch: 6100, acc: 0.583, loss: 0.833
epoch: 6200, acc: 0.587, loss: 0.830
epoch: 6300, acc: 0.593, loss: 0.827
epoch: 6400, acc: 0.600, loss: 0.824
epoch: 6500, acc: 0.600, loss: 0.821
epoch: 6600, acc: 0.597, loss: 0.818
epoch: 6700, acc: 0.597, loss: 0.815
epoch: 6800, acc: 0.597, loss: 0.812
epoch: 6900, acc: 0.597, loss: 0.809
epoch: 7000, acc: 0.603, loss: 0.805
epoch: 7100, acc: 0.603, loss: 0.803
epoch: 7200, acc: 0.610, loss: 0.800
```

```
epoch: 7300, acc: 0.607, loss: 0.797
epoch: 7400, acc: 0.613, loss: 0.794
epoch: 7500, acc: 0.617, loss: 0.792
epoch: 7600, acc: 0.620, loss: 0.788
epoch: 7700, acc: 0.620, loss: 0.785
epoch: 7800, acc: 0.620, loss: 0.783
epoch: 7900, acc: 0.627, loss: 0.780
epoch: 8000, acc: 0.627, loss: 0.779
epoch: 8100, acc: 0.627, loss: 0.775
epoch: 8200, acc: 0.627, loss: 0.772
epoch: 8300, acc: 0.630, loss: 0.769
epoch: 8400, acc: 0.630, loss: 0.765
epoch: 8500, acc: 0.633, loss: 0.762
epoch: 8600, acc: 0.630, loss: 0.759
epoch: 8700, acc: 0.643, loss: 0.755
epoch: 8800, acc: 0.647, loss: 0.751
epoch: 8900, acc: 0.647, loss: 0.748
epoch: 9000, acc: 0.647, loss: 0.743
epoch: 9100, acc: 0.647, loss: 0.740
epoch: 9200, acc: 0.647, loss: 0.736
epoch: 9300, acc: 0.650, loss: 0.732
epoch: 9400, acc: 0.650, loss: 0.729
epoch: 9500, acc: 0.647, loss: 0.725
epoch: 9600, acc: 0.650, loss: 0.722
epoch: 9700, acc: 0.653, loss: 0.718
epoch: 9800, acc: 0.657, loss: 0.714
epoch: 9900, acc: 0.660, loss: 0.711
epoch: 10000, acc: 0.660, loss: 0.708
epoch: 10100, acc: 0.663, loss: 0.705
epoch: 10200, acc: 0.670, loss: 0.701
epoch: 10300, acc: 0.677, loss: 0.698
epoch: 10400, acc: 0.680, loss: 0.695
epoch: 10500, acc: 0.683, loss: 0.692
epoch: 10600, acc: 0.683, loss: 0.689
epoch: 10700, acc: 0.680, loss: 0.686
epoch: 10800, acc: 0.690, loss: 0.683
epoch: 10900, acc: 0.690, loss: 0.680
epoch: 11000, acc: 0.690, loss: 0.677
epoch: 11100, acc: 0.697, loss: 0.674
epoch: 11200, acc: 0.707, loss: 0.671
epoch: 11300, acc: 0.707, loss: 0.668
epoch: 11400, acc: 0.713, loss: 0.665
epoch: 11500, acc: 0.713, loss: 0.662
epoch: 11600, acc: 0.717, loss: 0.658
epoch: 11700, acc: 0.720, loss: 0.655
epoch: 11800, acc: 0.720, loss: 0.654
epoch: 11900, acc: 0.727, loss: 0.650
epoch: 12000, acc: 0.733, loss: 0.647
```

```
epoch: 12100, acc: 0.733, loss: 0.644
epoch: 12200, acc: 0.733, loss: 0.641
epoch: 12300, acc: 0.733, loss: 0.639
epoch: 12400, acc: 0.733, loss: 0.637
epoch: 12500, acc: 0.727, loss: 0.634
epoch: 12600, acc: 0.723, loss: 0.630
epoch: 12700, acc: 0.733, loss: 0.627
epoch: 12800, acc: 0.733, loss: 0.626
epoch: 12900, acc: 0.737, loss: 0.622
epoch: 13000, acc: 0.737, loss: 0.619
epoch: 13100, acc: 0.740, loss: 0.617
epoch: 13200, acc: 0.743, loss: 0.614
epoch: 13300, acc: 0.743, loss: 0.611
epoch: 13400, acc: 0.740, loss: 0.608
epoch: 13500, acc: 0.740, loss: 0.606
epoch: 13600, acc: 0.737, loss: 0.604
epoch: 13700, acc: 0.743, loss: 0.602
epoch: 13800, acc: 0.747, loss: 0.600
epoch: 13900, acc: 0.747, loss: 0.598
epoch: 14000, acc: 0.757, loss: 0.596
epoch: 14100, acc: 0.753, loss: 0.594
epoch: 14200, acc: 0.760, loss: 0.592
epoch: 14300, acc: 0.760, loss: 0.590
epoch: 14400, acc: 0.757, loss: 0.589
epoch: 14500, acc: 0.763, loss: 0.587
epoch: 14600, acc: 0.763, loss: 0.585
epoch: 14700, acc: 0.773, loss: 0.584
epoch: 14800, acc: 0.777, loss: 0.582
epoch: 14900, acc: 0.777, loss: 0.580
epoch: 15000, acc: 0.783, loss: 0.579
epoch: 15100, acc: 0.783, loss: 0.577
epoch: 15200, acc: 0.783, loss: 0.576
epoch: 15300, acc: 0.790, loss: 0.575
epoch: 15400, acc: 0.787, loss: 0.573
epoch: 15500, acc: 0.790, loss: 0.572
epoch: 15600, acc: 0.790, loss: 0.570
epoch: 15700, acc: 0.790, loss: 0.569
epoch: 15800, acc: 0.793, loss: 0.568
epoch: 15900, acc: 0.793, loss: 0.566
epoch: 16000, acc: 0.793, loss: 0.565
epoch: 16100, acc: 0.793, loss: 0.564
epoch: 16200, acc: 0.790, loss: 0.562
epoch: 16300, acc: 0.790, loss: 0.561
epoch: 16400, acc: 0.790, loss: 0.560
epoch: 16500, acc: 0.790, loss: 0.559
epoch: 16600, acc: 0.793, loss: 0.558
epoch: 16700, acc: 0.793, loss: 0.557
epoch: 16800, acc: 0.793, loss: 0.555
```

```
epoch: 16900, acc: 0.797, loss: 0.554
epoch: 17000, acc: 0.797, loss: 0.553
epoch: 17100, acc: 0.797, loss: 0.552
epoch: 17200, acc: 0.797, loss: 0.551
epoch: 17300, acc: 0.797, loss: 0.550
epoch: 17400, acc: 0.797, loss: 0.549
epoch: 17500, acc: 0.797, loss: 0.548
epoch: 17600, acc: 0.797, loss: 0.547
epoch: 17700, acc: 0.797, loss: 0.546
epoch: 17800, acc: 0.797, loss: 0.545
epoch: 17900, acc: 0.793, loss: 0.544
epoch: 18000, acc: 0.790, loss: 0.543
epoch: 18100, acc: 0.793, loss: 0.542
epoch: 18200, acc: 0.793, loss: 0.542
epoch: 18300, acc: 0.793, loss: 0.541
epoch: 18400, acc: 0.793, loss: 0.540
epoch: 18500, acc: 0.793, loss: 0.539
epoch: 18600, acc: 0.793, loss: 0.538
epoch: 18700, acc: 0.790, loss: 0.537
epoch: 18800, acc: 0.793, loss: 0.536
epoch: 18900, acc: 0.793, loss: 0.535
epoch: 19000, acc: 0.793, loss: 0.535
epoch: 19100, acc: 0.793, loss: 0.534
epoch: 19200, acc: 0.793, loss: 0.533
epoch: 19300, acc: 0.793, loss: 0.532
epoch: 19400, acc: 0.793, loss: 0.531
epoch: 19500, acc: 0.793, loss: 0.531
epoch: 19600, acc: 0.793, loss: 0.530
epoch: 19700, acc: 0.793, loss: 0.529
epoch: 19800, acc: 0.793, loss: 0.528
epoch: 19900, acc: 0.793, loss: 0.528
epoch: 20000, acc: 0.793, loss: 0.527
epoch: 20100, acc: 0.797, loss: 0.526
epoch: 20200, acc: 0.797, loss: 0.525
epoch: 20300, acc: 0.797, loss: 0.525
epoch: 20400, acc: 0.797, loss: 0.524
epoch: 20500, acc: 0.797, loss: 0.523
epoch: 20600, acc: 0.797, loss: 0.523
epoch: 20700, acc: 0.797, loss: 0.522
epoch: 20800, acc: 0.800, loss: 0.521
epoch: 20900, acc: 0.800, loss: 0.521
epoch: 21000, acc: 0.800, loss: 0.520
epoch: 21100, acc: 0.800, loss: 0.519
epoch: 21200, acc: 0.800, loss: 0.519
epoch: 21300, acc: 0.800, loss: 0.518
epoch: 21400, acc: 0.800, loss: 0.518
epoch: 21500, acc: 0.800, loss: 0.517
epoch: 21600, acc: 0.800, loss: 0.516
```

```
epoch: 21700, acc: 0.800, loss: 0.516
epoch: 21800, acc: 0.800, loss: 0.515
epoch: 21900, acc: 0.800, loss: 0.515
epoch: 22000, acc: 0.800, loss: 0.514
epoch: 22100, acc: 0.797, loss: 0.513
epoch: 22200, acc: 0.800, loss: 0.513
epoch: 22300, acc: 0.797, loss: 0.512
epoch: 22400, acc: 0.797, loss: 0.512
epoch: 22500, acc: 0.797, loss: 0.511
epoch: 22600, acc: 0.797, loss: 0.510
epoch: 22700, acc: 0.797, loss: 0.510
epoch: 22800, acc: 0.797, loss: 0.509
epoch: 22900, acc: 0.797, loss: 0.509
epoch: 23000, acc: 0.797, loss: 0.508
epoch: 23100, acc: 0.800, loss: 0.508
epoch: 23200, acc: 0.797, loss: 0.507
epoch: 23300, acc: 0.797, loss: 0.507
epoch: 23400, acc: 0.797, loss: 0.506
epoch: 23500, acc: 0.797, loss: 0.506
epoch: 23600, acc: 0.800, loss: 0.505
epoch: 23700, acc: 0.803, loss: 0.505
epoch: 23800, acc: 0.803, loss: 0.504
epoch: 23900, acc: 0.800, loss: 0.504
epoch: 24000, acc: 0.803, loss: 0.503
epoch: 24100, acc: 0.807, loss: 0.503
epoch: 24200, acc: 0.807, loss: 0.502
epoch: 24300, acc: 0.807, loss: 0.502
epoch: 24400, acc: 0.807, loss: 0.501
epoch: 24500, acc: 0.807, loss: 0.501
epoch: 24600, acc: 0.807, loss: 0.500
epoch: 24700, acc: 0.807, loss: 0.500
epoch: 24800, acc: 0.807, loss: 0.499
epoch: 24900, acc: 0.807, loss: 0.499
epoch: 25000, acc: 0.807, loss: 0.498
epoch: 25100, acc: 0.807, loss: 0.498
epoch: 25200, acc: 0.807, loss: 0.497
epoch: 25300, acc: 0.807, loss: 0.497
epoch: 25400, acc: 0.807, loss: 0.497
epoch: 25500, acc: 0.803, loss: 0.496
epoch: 25600, acc: 0.803, loss: 0.496
epoch: 25700, acc: 0.803, loss: 0.495
epoch: 25800, acc: 0.803, loss: 0.495
epoch: 25900, acc: 0.803, loss: 0.494
epoch: 26000, acc: 0.800, loss: 0.494
epoch: 26100, acc: 0.803, loss: 0.493
epoch: 26200, acc: 0.800, loss: 0.493
epoch: 26300, acc: 0.800, loss: 0.492
epoch: 26400, acc: 0.800, loss: 0.492
```

```
epoch: 26500, acc: 0.800, loss: 0.492
epoch: 26600, acc: 0.800, loss: 0.491
epoch: 26700, acc: 0.803, loss: 0.491
epoch: 26800, acc: 0.800, loss: 0.490
epoch: 26900, acc: 0.800, loss: 0.490
epoch: 27000, acc: 0.800, loss: 0.490
epoch: 27100, acc: 0.803, loss: 0.489
epoch: 27200, acc: 0.803, loss: 0.489
epoch: 27300, acc: 0.803, loss: 0.488
epoch: 27400, acc: 0.803, loss: 0.488
epoch: 27500, acc: 0.803, loss: 0.488
epoch: 27600, acc: 0.803, loss: 0.487
epoch: 27700, acc: 0.803, loss: 0.487
epoch: 27800, acc: 0.803, loss: 0.487
epoch: 27900, acc: 0.803, loss: 0.486
epoch: 28000, acc: 0.803, loss: 0.486
epoch: 28100, acc: 0.803, loss: 0.485
epoch: 28200, acc: 0.803, loss: 0.485
epoch: 28300, acc: 0.803, loss: 0.485
epoch: 28400, acc: 0.803, loss: 0.484
epoch: 28500, acc: 0.803, loss: 0.484
epoch: 28600, acc: 0.803, loss: 0.484
epoch: 28700, acc: 0.800, loss: 0.483
epoch: 28800, acc: 0.803, loss: 0.483
epoch: 28900, acc: 0.800, loss: 0.483
epoch: 29000, acc: 0.800, loss: 0.482
epoch: 29100, acc: 0.800, loss: 0.482
epoch: 29200, acc: 0.800, loss: 0.482
epoch: 29300, acc: 0.800, loss: 0.481
epoch: 29400, acc: 0.800, loss: 0.481
epoch: 29500, acc: 0.800, loss: 0.481
epoch: 29600, acc: 0.800, loss: 0.480
epoch: 29700, acc: 0.800, loss: 0.480
epoch: 29800, acc: 0.800, loss: 0.480
epoch: 29900, acc: 0.800, loss: 0.479
epoch: 30000, acc: 0.800, loss: 0.479
epoch: 30100, acc: 0.800, loss: 0.479
epoch: 30200, acc: 0.800, loss: 0.478
epoch: 30300, acc: 0.800, loss: 0.478
epoch: 30400, acc: 0.800, loss: 0.478
epoch: 30500, acc: 0.800, loss: 0.477
epoch: 30600, acc: 0.800, loss: 0.477
epoch: 30700, acc: 0.800, loss: 0.477
epoch: 30800, acc: 0.800, loss: 0.476
epoch: 30900, acc: 0.800, loss: 0.476
epoch: 31000, acc: 0.800, loss: 0.476
epoch: 31100, acc: 0.800, loss: 0.476
epoch: 31200, acc: 0.800, loss: 0.475
```

```
epoch: 31300, acc: 0.800, loss: 0.475
epoch: 31400, acc: 0.800, loss: 0.475
epoch: 31500, acc: 0.800, loss: 0.474
epoch: 31600, acc: 0.800, loss: 0.474
epoch: 31700, acc: 0.800, loss: 0.474
epoch: 31800, acc: 0.800, loss: 0.473
epoch: 31900, acc: 0.800, loss: 0.473
epoch: 32000, acc: 0.800, loss: 0.473
epoch: 32100, acc: 0.800, loss: 0.473
epoch: 32200, acc: 0.800, loss: 0.472
epoch: 32300, acc: 0.800, loss: 0.472
epoch: 32400, acc: 0.800, loss: 0.472
epoch: 32500, acc: 0.800, loss: 0.471
epoch: 32600, acc: 0.800, loss: 0.471
epoch: 32700, acc: 0.800, loss: 0.471
epoch: 32800, acc: 0.800, loss: 0.471
epoch: 32900, acc: 0.800, loss: 0.470
epoch: 33000, acc: 0.800, loss: 0.470
epoch: 33100, acc: 0.800, loss: 0.470
epoch: 33200, acc: 0.800, loss: 0.469
epoch: 33300, acc: 0.800, loss: 0.469
epoch: 33400, acc: 0.800, loss: 0.469
epoch: 33500, acc: 0.800, loss: 0.469
epoch: 33600, acc: 0.800, loss: 0.468
epoch: 33700, acc: 0.800, loss: 0.468
epoch: 33800, acc: 0.800, loss: 0.468
epoch: 33900, acc: 0.800, loss: 0.468
epoch: 34000, acc: 0.800, loss: 0.467
epoch: 34100, acc: 0.800, loss: 0.467
epoch: 34200, acc: 0.800, loss: 0.467
epoch: 34300, acc: 0.800, loss: 0.467
epoch: 34400, acc: 0.800, loss: 0.466
epoch: 34500, acc: 0.800, loss: 0.466
epoch: 34600, acc: 0.800, loss: 0.466
epoch: 34700, acc: 0.800, loss: 0.465
epoch: 34800, acc: 0.800, loss: 0.465
epoch: 34900, acc: 0.800, loss: 0.465
epoch: 35000, acc: 0.800, loss: 0.464
epoch: 35100, acc: 0.800, loss: 0.464
epoch: 35200, acc: 0.800, loss: 0.464
epoch: 35300, acc: 0.800, loss: 0.464
epoch: 35400, acc: 0.800, loss: 0.463
epoch: 35500, acc: 0.800, loss: 0.463
epoch: 35600, acc: 0.800, loss: 0.463
epoch: 35700, acc: 0.800, loss: 0.462
epoch: 35800, acc: 0.800, loss: 0.462
epoch: 35900, acc: 0.800, loss: 0.462
epoch: 36000, acc: 0.800, loss: 0.462
```

```
epoch: 36100, acc: 0.800, loss: 0.461
epoch: 36200, acc: 0.800, loss: 0.461
epoch: 36300, acc: 0.800, loss: 0.461
epoch: 36400, acc: 0.800, loss: 0.460
epoch: 36500, acc: 0.800, loss: 0.460
epoch: 36600, acc: 0.800, loss: 0.460
epoch: 36700, acc: 0.800, loss: 0.460
epoch: 36800, acc: 0.800, loss: 0.459
epoch: 36900, acc: 0.800, loss: 0.459
epoch: 37000, acc: 0.800, loss: 0.459
epoch: 37100, acc: 0.800, loss: 0.458
epoch: 37200, acc: 0.797, loss: 0.458
epoch: 37300, acc: 0.800, loss: 0.458
epoch: 37400, acc: 0.800, loss: 0.457
epoch: 37500, acc: 0.800, loss: 0.457
epoch: 37600, acc: 0.800, loss: 0.457
epoch: 37700, acc: 0.800, loss: 0.456
epoch: 37800, acc: 0.800, loss: 0.456
epoch: 37900, acc: 0.800, loss: 0.456
epoch: 38000, acc: 0.800, loss: 0.455
epoch: 38100, acc: 0.800, loss: 0.455
epoch: 38200, acc: 0.800, loss: 0.455
epoch: 38300, acc: 0.800, loss: 0.454
epoch: 38400, acc: 0.800, loss: 0.454
epoch: 38500, acc: 0.800, loss: 0.454
epoch: 38600, acc: 0.800, loss: 0.454
epoch: 38700, acc: 0.800, loss: 0.453
epoch: 38800, acc: 0.800, loss: 0.453
epoch: 38900, acc: 0.800, loss: 0.453
epoch: 39000, acc: 0.800, loss: 0.453
epoch: 39100, acc: 0.800, loss: 0.452
epoch: 39200, acc: 0.800, loss: 0.452
epoch: 39300, acc: 0.800, loss: 0.452
epoch: 39400, acc: 0.800, loss: 0.451
epoch: 39500, acc: 0.800, loss: 0.451
epoch: 39600, acc: 0.800, loss: 0.451
epoch: 39700, acc: 0.803, loss: 0.451
epoch: 39800, acc: 0.807, loss: 0.450
epoch: 39900, acc: 0.807, loss: 0.450
epoch: 40000, acc: 0.807, loss: 0.450
epoch: 40100, acc: 0.807, loss: 0.450
epoch: 40200, acc: 0.807, loss: 0.449
epoch: 40300, acc: 0.807, loss: 0.449
epoch: 40400, acc: 0.807, loss: 0.449
epoch: 40500, acc: 0.803, loss: 0.449
epoch: 40600, acc: 0.803, loss: 0.448
epoch: 40700, acc: 0.803, loss: 0.448
epoch: 40800, acc: 0.803, loss: 0.448
```

```
epoch: 40900, acc: 0.807, loss: 0.448
epoch: 41000, acc: 0.807, loss: 0.447
epoch: 41100, acc: 0.807, loss: 0.447
epoch: 41200, acc: 0.807, loss: 0.447
epoch: 41300, acc: 0.807, loss: 0.447
epoch: 41400, acc: 0.807, loss: 0.446
epoch: 41500, acc: 0.807, loss: 0.446
epoch: 41600, acc: 0.810, loss: 0.446
epoch: 41700, acc: 0.807, loss: 0.446
epoch: 41800, acc: 0.807, loss: 0.446
epoch: 41900, acc: 0.807, loss: 0.445
epoch: 42000, acc: 0.807, loss: 0.445
epoch: 42100, acc: 0.807, loss: 0.445
epoch: 42200, acc: 0.807, loss: 0.445
epoch: 42300, acc: 0.807, loss: 0.444
epoch: 42400, acc: 0.807, loss: 0.444
epoch: 42500, acc: 0.807, loss: 0.444
epoch: 42600, acc: 0.807, loss: 0.444
epoch: 42700, acc: 0.807, loss: 0.444
epoch: 42800, acc: 0.807, loss: 0.443
epoch: 42900, acc: 0.807, loss: 0.443
epoch: 43000, acc: 0.807, loss: 0.443
epoch: 43100, acc: 0.807, loss: 0.443
epoch: 43200, acc: 0.807, loss: 0.443
epoch: 43300, acc: 0.807, loss: 0.442
epoch: 43400, acc: 0.807, loss: 0.442
epoch: 43500, acc: 0.807, loss: 0.442
epoch: 43600, acc: 0.807, loss: 0.442
epoch: 43700, acc: 0.807, loss: 0.442
epoch: 43800, acc: 0.807, loss: 0.441
epoch: 43900, acc: 0.807, loss: 0.441
epoch: 44000, acc: 0.807, loss: 0.441
epoch: 44100, acc: 0.807, loss: 0.441
epoch: 44200, acc: 0.807, loss: 0.441
epoch: 44300, acc: 0.807, loss: 0.440
epoch: 44400, acc: 0.807, loss: 0.440
epoch: 44500, acc: 0.807, loss: 0.440
epoch: 44600, acc: 0.807, loss: 0.440
epoch: 44700, acc: 0.807, loss: 0.440
epoch: 44800, acc: 0.807, loss: 0.439
epoch: 44900, acc: 0.810, loss: 0.439
epoch: 45000, acc: 0.807, loss: 0.439
epoch: 45100, acc: 0.807, loss: 0.439
epoch: 45200, acc: 0.810, loss: 0.439
epoch: 45300, acc: 0.810, loss: 0.438
epoch: 45400, acc: 0.810, loss: 0.438
epoch: 45500, acc: 0.810, loss: 0.438
epoch: 45600, acc: 0.810, loss: 0.438
```

```
epoch: 45700, acc: 0.810, loss: 0.438
epoch: 45800, acc: 0.810, loss: 0.437
epoch: 45900, acc: 0.810, loss: 0.437
epoch: 46000, acc: 0.810, loss: 0.437
epoch: 46100, acc: 0.810, loss: 0.437
epoch: 46200, acc: 0.810, loss: 0.437
epoch: 46300, acc: 0.810, loss: 0.437
epoch: 46400, acc: 0.810, loss: 0.436
epoch: 46500, acc: 0.810, loss: 0.436
epoch: 46600, acc: 0.810, loss: 0.436
epoch: 46700, acc: 0.810, loss: 0.436
epoch: 46800, acc: 0.810, loss: 0.436
epoch: 46900, acc: 0.810, loss: 0.435
epoch: 47000, acc: 0.810, loss: 0.435
epoch: 47100, acc: 0.810, loss: 0.435
epoch: 47200, acc: 0.813, loss: 0.435
epoch: 47300, acc: 0.813, loss: 0.435
epoch: 47400, acc: 0.813, loss: 0.435
epoch: 47500, acc: 0.813, loss: 0.434
epoch: 47600, acc: 0.813, loss: 0.434
epoch: 47700, acc: 0.813, loss: 0.434
epoch: 47800, acc: 0.813, loss: 0.434
epoch: 47900, acc: 0.817, loss: 0.434
epoch: 48000, acc: 0.817, loss: 0.433
epoch: 48100, acc: 0.817, loss: 0.433
epoch: 48200, acc: 0.813, loss: 0.433
epoch: 48300, acc: 0.817, loss: 0.433
epoch: 48400, acc: 0.817, loss: 0.433
epoch: 48500, acc: 0.813, loss: 0.433
epoch: 48600, acc: 0.817, loss: 0.432
epoch: 48700, acc: 0.817, loss: 0.432
epoch: 48800, acc: 0.817, loss: 0.432
epoch: 48900, acc: 0.817, loss: 0.432
epoch: 49000, acc: 0.817, loss: 0.432
epoch: 49100, acc: 0.817, loss: 0.432
epoch: 49200, acc: 0.817, loss: 0.431
epoch: 49300, acc: 0.817, loss: 0.431
epoch: 49400, acc: 0.817, loss: 0.431
epoch: 49500, acc: 0.817, loss: 0.431
epoch: 49600, acc: 0.817, loss: 0.431
epoch: 49700, acc: 0.817, loss: 0.431
epoch: 49800, acc: 0.817, loss: 0.430
epoch: 49900, acc: 0.817, loss: 0.430
epoch: 50000, acc: 0.817, loss: 0.430
epoch: 50100, acc: 0.820, loss: 0.430
epoch: 50200, acc: 0.820, loss: 0.430
epoch: 50300, acc: 0.820, loss: 0.429
epoch: 50400, acc: 0.820, loss: 0.429
```

```
epoch: 50500, acc: 0.820, loss: 0.429
epoch: 50600, acc: 0.820, loss: 0.429
epoch: 50700, acc: 0.820, loss: 0.429
epoch: 50800, acc: 0.820, loss: 0.429
epoch: 50900, acc: 0.820, loss: 0.428
epoch: 51000, acc: 0.820, loss: 0.428
epoch: 51100, acc: 0.820, loss: 0.428
epoch: 51200, acc: 0.820, loss: 0.428
epoch: 51300, acc: 0.820, loss: 0.428
epoch: 51400, acc: 0.820, loss: 0.428
epoch: 51500, acc: 0.820, loss: 0.427
epoch: 51600, acc: 0.820, loss: 0.427
epoch: 51700, acc: 0.820, loss: 0.427
epoch: 51800, acc: 0.820, loss: 0.427
epoch: 51900, acc: 0.820, loss: 0.427
epoch: 52000, acc: 0.820, loss: 0.427
epoch: 52100, acc: 0.820, loss: 0.426
epoch: 52200, acc: 0.820, loss: 0.426
epoch: 52300, acc: 0.820, loss: 0.426
epoch: 52400, acc: 0.820, loss: 0.426
epoch: 52500, acc: 0.820, loss: 0.426
epoch: 52600, acc: 0.820, loss: 0.426
epoch: 52700, acc: 0.820, loss: 0.426
epoch: 52800, acc: 0.820, loss: 0.425
epoch: 52900, acc: 0.820, loss: 0.425
epoch: 53000, acc: 0.820, loss: 0.425
epoch: 53100, acc: 0.820, loss: 0.425
epoch: 53200, acc: 0.820, loss: 0.425
epoch: 53300, acc: 0.820, loss: 0.425
epoch: 53400, acc: 0.820, loss: 0.424
epoch: 53500, acc: 0.820, loss: 0.424
epoch: 53600, acc: 0.820, loss: 0.424
epoch: 53700, acc: 0.820, loss: 0.424
epoch: 53800, acc: 0.820, loss: 0.424
epoch: 53900, acc: 0.820, loss: 0.424
epoch: 54000, acc: 0.820, loss: 0.424
epoch: 54100, acc: 0.820, loss: 0.423
epoch: 54200, acc: 0.820, loss: 0.423
epoch: 54300, acc: 0.820, loss: 0.423
epoch: 54400, acc: 0.820, loss: 0.423
epoch: 54500, acc: 0.820, loss: 0.423
epoch: 54600, acc: 0.823, loss: 0.423
epoch: 54700, acc: 0.823, loss: 0.422
epoch: 54800, acc: 0.823, loss: 0.422
epoch: 54900, acc: 0.823, loss: 0.422
epoch: 55000, acc: 0.823, loss: 0.422
epoch: 55100, acc: 0.823, loss: 0.422
epoch: 55200, acc: 0.823, loss: 0.422
```

```
epoch: 55300, acc: 0.823, loss: 0.422
epoch: 55400, acc: 0.823, loss: 0.421
epoch: 55500, acc: 0.823, loss: 0.421
epoch: 55600, acc: 0.823, loss: 0.421
epoch: 55700, acc: 0.823, loss: 0.421
epoch: 55800, acc: 0.823, loss: 0.421
epoch: 55900, acc: 0.823, loss: 0.421
epoch: 56000, acc: 0.823, loss: 0.421
epoch: 56100, acc: 0.823, loss: 0.420
epoch: 56200, acc: 0.823, loss: 0.420
epoch: 56300, acc: 0.823, loss: 0.420
epoch: 56400, acc: 0.823, loss: 0.420
epoch: 56500, acc: 0.823, loss: 0.420
epoch: 56600, acc: 0.823, loss: 0.420
epoch: 56700, acc: 0.823, loss: 0.420
epoch: 56800, acc: 0.823, loss: 0.419
epoch: 56900, acc: 0.820, loss: 0.419
epoch: 57000, acc: 0.820, loss: 0.419
epoch: 57100, acc: 0.820, loss: 0.419
epoch: 57200, acc: 0.820, loss: 0.419
epoch: 57300, acc: 0.820, loss: 0.419
epoch: 57400, acc: 0.820, loss: 0.419
epoch: 57500, acc: 0.820, loss: 0.418
epoch: 57600, acc: 0.820, loss: 0.418
epoch: 57700, acc: 0.820, loss: 0.418
epoch: 57800, acc: 0.820, loss: 0.418
epoch: 57900, acc: 0.820, loss: 0.418
epoch: 58000, acc: 0.820, loss: 0.418
epoch: 58100, acc: 0.820, loss: 0.418
epoch: 58200, acc: 0.820, loss: 0.417
epoch: 58300, acc: 0.820, loss: 0.417
epoch: 58400, acc: 0.820, loss: 0.417
epoch: 58500, acc: 0.820, loss: 0.417
epoch: 58600, acc: 0.820, loss: 0.417
epoch: 58700, acc: 0.823, loss: 0.417
epoch: 58800, acc: 0.823, loss: 0.417
epoch: 58900, acc: 0.823, loss: 0.417
epoch: 59000, acc: 0.823, loss: 0.416
epoch: 59100, acc: 0.823, loss: 0.416
epoch: 59200, acc: 0.823, loss: 0.416
epoch: 59300, acc: 0.823, loss: 0.416
epoch: 59400, acc: 0.823, loss: 0.416
epoch: 59500, acc: 0.823, loss: 0.416
epoch: 59600, acc: 0.823, loss: 0.416
epoch: 59700, acc: 0.823, loss: 0.415
epoch: 59800, acc: 0.823, loss: 0.415
epoch: 59900, acc: 0.823, loss: 0.415
epoch: 60000, acc: 0.823, loss: 0.415
```

```
epoch: 60100, acc: 0.823, loss: 0.415
epoch: 60200, acc: 0.823, loss: 0.415
epoch: 60300, acc: 0.820, loss: 0.415
epoch: 60400, acc: 0.820, loss: 0.415
epoch: 60500, acc: 0.820, loss: 0.414
epoch: 60600, acc: 0.820, loss: 0.414
epoch: 60700, acc: 0.823, loss: 0.414
epoch: 60800, acc: 0.820, loss: 0.414
epoch: 60900, acc: 0.823, loss: 0.414
epoch: 61000, acc: 0.823, loss: 0.414
epoch: 61100, acc: 0.823, loss: 0.414
epoch: 61200, acc: 0.823, loss: 0.414
epoch: 61300, acc: 0.827, loss: 0.413
epoch: 61400, acc: 0.827, loss: 0.413
epoch: 61500, acc: 0.827, loss: 0.413
epoch: 61600, acc: 0.827, loss: 0.413
epoch: 61700, acc: 0.827, loss: 0.413
epoch: 61800, acc: 0.827, loss: 0.413
epoch: 61900, acc: 0.827, loss: 0.413
epoch: 62000, acc: 0.827, loss: 0.413
epoch: 62100, acc: 0.827, loss: 0.412
epoch: 62200, acc: 0.827, loss: 0.412
epoch: 62300, acc: 0.827, loss: 0.412
epoch: 62400, acc: 0.827, loss: 0.412
epoch: 62500, acc: 0.827, loss: 0.412
epoch: 62600, acc: 0.827, loss: 0.412
epoch: 62700, acc: 0.827, loss: 0.412
epoch: 62800, acc: 0.827, loss: 0.412
epoch: 62900, acc: 0.827, loss: 0.412
epoch: 63000, acc: 0.827, loss: 0.411
epoch: 63100, acc: 0.827, loss: 0.411
epoch: 63200, acc: 0.827, loss: 0.411
epoch: 63300, acc: 0.827, loss: 0.411
epoch: 63400, acc: 0.827, loss: 0.411
epoch: 63500, acc: 0.830, loss: 0.411
epoch: 63600, acc: 0.830, loss: 0.411
epoch: 63700, acc: 0.830, loss: 0.411
epoch: 63800, acc: 0.830, loss: 0.410
epoch: 63900, acc: 0.830, loss: 0.410
epoch: 64000, acc: 0.830, loss: 0.410
epoch: 64100, acc: 0.830, loss: 0.410
epoch: 64200, acc: 0.830, loss: 0.410
epoch: 64300, acc: 0.830, loss: 0.410
epoch: 64400, acc: 0.830, loss: 0.410
epoch: 64500, acc: 0.830, loss: 0.410
epoch: 64600, acc: 0.830, loss: 0.410
epoch: 64700, acc: 0.833, loss: 0.409
epoch: 64800, acc: 0.833, loss: 0.409
```

```
epoch: 64900, acc: 0.833, loss: 0.409
epoch: 65000, acc: 0.833, loss: 0.409
epoch: 65100, acc: 0.833, loss: 0.409
epoch: 65200, acc: 0.833, loss: 0.409
epoch: 65300, acc: 0.833, loss: 0.409
epoch: 65400, acc: 0.833, loss: 0.409
epoch: 65500, acc: 0.837, loss: 0.409
epoch: 65600, acc: 0.837, loss: 0.408
epoch: 65700, acc: 0.837, loss: 0.408
epoch: 65800, acc: 0.837, loss: 0.408
epoch: 65900, acc: 0.837, loss: 0.408
epoch: 66000, acc: 0.837, loss: 0.408
epoch: 66100, acc: 0.837, loss: 0.408
epoch: 66200, acc: 0.833, loss: 0.408
epoch: 66300, acc: 0.833, loss: 0.408
epoch: 66400, acc: 0.833, loss: 0.408
epoch: 66500, acc: 0.833, loss: 0.408
epoch: 66600, acc: 0.833, loss: 0.407
epoch: 66700, acc: 0.833, loss: 0.407
epoch: 66800, acc: 0.833, loss: 0.407
epoch: 66900, acc: 0.833, loss: 0.407
epoch: 67000, acc: 0.830, loss: 0.407
epoch: 67100, acc: 0.833, loss: 0.407
epoch: 67200, acc: 0.830, loss: 0.407
epoch: 67300, acc: 0.830, loss: 0.407
epoch: 67400, acc: 0.830, loss: 0.407
epoch: 67500, acc: 0.830, loss: 0.406
epoch: 67600, acc: 0.830, loss: 0.406
epoch: 67700, acc: 0.830, loss: 0.406
epoch: 67800, acc: 0.830, loss: 0.406
epoch: 67900, acc: 0.830, loss: 0.406
epoch: 68000, acc: 0.830, loss: 0.406
epoch: 68100, acc: 0.830, loss: 0.406
epoch: 68200, acc: 0.830, loss: 0.406
epoch: 68300, acc: 0.830, loss: 0.406
epoch: 68400, acc: 0.833, loss: 0.406
epoch: 68500, acc: 0.833, loss: 0.405
epoch: 68600, acc: 0.830, loss: 0.405
epoch: 68700, acc: 0.833, loss: 0.405
epoch: 68800, acc: 0.837, loss: 0.405
epoch: 68900, acc: 0.837, loss: 0.405
epoch: 69000, acc: 0.837, loss: 0.405
epoch: 69100, acc: 0.837, loss: 0.405
epoch: 69200, acc: 0.837, loss: 0.405
epoch: 69300, acc: 0.837, loss: 0.405
epoch: 69400, acc: 0.833, loss: 0.405
epoch: 69500, acc: 0.837, loss: 0.404
epoch: 69600, acc: 0.837, loss: 0.404
```

```
epoch: 69700, acc: 0.837, loss: 0.404
epoch: 69800, acc: 0.837, loss: 0.404
epoch: 69900, acc: 0.837, loss: 0.404
epoch: 70000, acc: 0.837, loss: 0.404
epoch: 70100, acc: 0.837, loss: 0.404
epoch: 70200, acc: 0.837, loss: 0.404
epoch: 70300, acc: 0.840, loss: 0.404
epoch: 70400, acc: 0.840, loss: 0.404
epoch: 70500, acc: 0.840, loss: 0.404
epoch: 70600, acc: 0.840, loss: 0.403
epoch: 70700, acc: 0.840, loss: 0.403
epoch: 70800, acc: 0.840, loss: 0.403
epoch: 70900, acc: 0.840, loss: 0.403
epoch: 71000, acc: 0.840, loss: 0.403
epoch: 71100, acc: 0.840, loss: 0.403
epoch: 71200, acc: 0.840, loss: 0.403
epoch: 71300, acc: 0.840, loss: 0.403
epoch: 71400, acc: 0.840, loss: 0.403
epoch: 71500, acc: 0.840, loss: 0.403
epoch: 71600, acc: 0.840, loss: 0.402
epoch: 71700, acc: 0.840, loss: 0.402
epoch: 71800, acc: 0.840, loss: 0.402
epoch: 71900, acc: 0.840, loss: 0.402
epoch: 72000, acc: 0.840, loss: 0.402
epoch: 72100, acc: 0.840, loss: 0.402
epoch: 72200, acc: 0.840, loss: 0.402
epoch: 72300, acc: 0.840, loss: 0.402
epoch: 72400, acc: 0.840, loss: 0.402
epoch: 72500, acc: 0.840, loss: 0.402
epoch: 72600, acc: 0.840, loss: 0.401
epoch: 72700, acc: 0.840, loss: 0.401
epoch: 72800, acc: 0.840, loss: 0.401
epoch: 72900, acc: 0.840, loss: 0.401
epoch: 73000, acc: 0.840, loss: 0.401
epoch: 73100, acc: 0.840, loss: 0.401
epoch: 73200, acc: 0.840, loss: 0.401
epoch: 73300, acc: 0.840, loss: 0.401
epoch: 73400, acc: 0.840, loss: 0.401
epoch: 73500, acc: 0.840, loss: 0.401
epoch: 73600, acc: 0.840, loss: 0.401
epoch: 73700, acc: 0.840, loss: 0.400
epoch: 73800, acc: 0.840, loss: 0.400
epoch: 73900, acc: 0.840, loss: 0.400
epoch: 74000, acc: 0.837, loss: 0.400
epoch: 74100, acc: 0.837, loss: 0.400
epoch: 74200, acc: 0.837, loss: 0.400
epoch: 74300, acc: 0.837, loss: 0.400
epoch: 74400, acc: 0.837, loss: 0.400
```

```
epoch: 74500, acc: 0.837, loss: 0.400
epoch: 74600, acc: 0.833, loss: 0.399
epoch: 74700, acc: 0.833, loss: 0.399
epoch: 74800, acc: 0.833, loss: 0.399
epoch: 74900, acc: 0.837, loss: 0.399
epoch: 75000, acc: 0.837, loss: 0.399
epoch: 75100, acc: 0.837, loss: 0.399
epoch: 75200, acc: 0.837, loss: 0.399
epoch: 75300, acc: 0.837, loss: 0.399
epoch: 75400, acc: 0.837, loss: 0.399
epoch: 75500, acc: 0.837, loss: 0.399
epoch: 75600, acc: 0.837, loss: 0.398
epoch: 75700, acc: 0.837, loss: 0.398
epoch: 75800, acc: 0.837, loss: 0.398
epoch: 75900, acc: 0.837, loss: 0.398
epoch: 76000, acc: 0.837, loss: 0.398
epoch: 76100, acc: 0.837, loss: 0.398
epoch: 76200, acc: 0.837, loss: 0.398
epoch: 76300, acc: 0.837, loss: 0.398
epoch: 76400, acc: 0.837, loss: 0.398
epoch: 76500, acc: 0.837, loss: 0.398
epoch: 76600, acc: 0.837, loss: 0.397
epoch: 76700, acc: 0.837, loss: 0.397
epoch: 76800, acc: 0.837, loss: 0.397
epoch: 76900, acc: 0.837, loss: 0.397
epoch: 77000, acc: 0.837, loss: 0.397
epoch: 77100, acc: 0.837, loss: 0.397
epoch: 77200, acc: 0.837, loss: 0.397
epoch: 77300, acc: 0.837, loss: 0.397
epoch: 77400, acc: 0.837, loss: 0.397
epoch: 77500, acc: 0.837, loss: 0.397
epoch: 77600, acc: 0.837, loss: 0.397
epoch: 77700, acc: 0.837, loss: 0.396
epoch: 77800, acc: 0.837, loss: 0.396
epoch: 77900, acc: 0.837, loss: 0.396
epoch: 78000, acc: 0.837, loss: 0.396
epoch: 78100, acc: 0.837, loss: 0.396
epoch: 78200, acc: 0.837, loss: 0.396
epoch: 78300, acc: 0.837, loss: 0.396
epoch: 78400, acc: 0.837, loss: 0.396
epoch: 78500, acc: 0.837, loss: 0.396
epoch: 78600, acc: 0.837, loss: 0.396
epoch: 78700, acc: 0.837, loss: 0.396
epoch: 78800, acc: 0.837, loss: 0.396
epoch: 78900, acc: 0.837, loss: 0.395
epoch: 79000, acc: 0.837, loss: 0.395
epoch: 79100, acc: 0.837, loss: 0.395
epoch: 79200, acc: 0.837, loss: 0.395
```

```
epoch: 79300, acc: 0.837, loss: 0.395
epoch: 79400, acc: 0.837, loss: 0.395
epoch: 79500, acc: 0.837, loss: 0.395
epoch: 79600, acc: 0.837, loss: 0.395
epoch: 79700, acc: 0.837, loss: 0.395
epoch: 79800, acc: 0.837, loss: 0.395
epoch: 79900, acc: 0.837, loss: 0.395
epoch: 80000, acc: 0.837, loss: 0.394
epoch: 80100, acc: 0.837, loss: 0.394
epoch: 80200, acc: 0.837, loss: 0.394
epoch: 80300, acc: 0.837, loss: 0.394
epoch: 80400, acc: 0.837, loss: 0.394
epoch: 80500, acc: 0.837, loss: 0.394
epoch: 80600, acc: 0.837, loss: 0.394
epoch: 80700, acc: 0.837, loss: 0.394
epoch: 80800, acc: 0.837, loss: 0.394
epoch: 80900, acc: 0.837, loss: 0.394
epoch: 81000, acc: 0.837, loss: 0.394
epoch: 81100, acc: 0.837, loss: 0.394
epoch: 81200, acc: 0.837, loss: 0.393
epoch: 81300, acc: 0.837, loss: 0.393
epoch: 81400, acc: 0.837, loss: 0.393
epoch: 81500, acc: 0.837, loss: 0.393
epoch: 81600, acc: 0.837, loss: 0.393
epoch: 81700, acc: 0.837, loss: 0.393
epoch: 81800, acc: 0.837, loss: 0.393
epoch: 81900, acc: 0.837, loss: 0.393
epoch: 82000, acc: 0.837, loss: 0.393
epoch: 82100, acc: 0.837, loss: 0.393
epoch: 82200, acc: 0.837, loss: 0.393
epoch: 82300, acc: 0.837, loss: 0.393
epoch: 82400, acc: 0.837, loss: 0.392
epoch: 82500, acc: 0.837, loss: 0.392
epoch: 82600, acc: 0.837, loss: 0.392
epoch: 82700, acc: 0.837, loss: 0.392
epoch: 82800, acc: 0.837, loss: 0.392
epoch: 82900, acc: 0.837, loss: 0.392
epoch: 83000, acc: 0.837, loss: 0.392
epoch: 83100, acc: 0.837, loss: 0.392
epoch: 83200, acc: 0.837, loss: 0.392
epoch: 83300, acc: 0.837, loss: 0.392
epoch: 83400, acc: 0.837, loss: 0.392
epoch: 83500, acc: 0.837, loss: 0.392
epoch: 83600, acc: 0.837, loss: 0.392
epoch: 83700, acc: 0.837, loss: 0.391
epoch: 83800, acc: 0.837, loss: 0.391
epoch: 83900, acc: 0.837, loss: 0.391
epoch: 84000, acc: 0.837, loss: 0.391
```

```
epoch: 84100, acc: 0.837, loss: 0.391
epoch: 84200, acc: 0.837, loss: 0.391
epoch: 84300, acc: 0.837, loss: 0.391
epoch: 84400, acc: 0.837, loss: 0.391
epoch: 84500, acc: 0.837, loss: 0.391
epoch: 84600, acc: 0.837, loss: 0.391
epoch: 84700, acc: 0.837, loss: 0.391
epoch: 84800, acc: 0.837, loss: 0.391
epoch: 84900, acc: 0.837, loss: 0.391
epoch: 85000, acc: 0.837, loss: 0.390
epoch: 85100, acc: 0.837, loss: 0.390
epoch: 85200, acc: 0.837, loss: 0.390
epoch: 85300, acc: 0.837, loss: 0.390
epoch: 85400, acc: 0.837, loss: 0.390
epoch: 85500, acc: 0.837, loss: 0.390
epoch: 85600, acc: 0.837, loss: 0.390
epoch: 85700, acc: 0.837, loss: 0.390
epoch: 85800, acc: 0.837, loss: 0.390
epoch: 85900, acc: 0.837, loss: 0.390
epoch: 86000, acc: 0.837, loss: 0.390
epoch: 86100, acc: 0.837, loss: 0.390
epoch: 86200, acc: 0.837, loss: 0.390
epoch: 86300, acc: 0.837, loss: 0.390
epoch: 86400, acc: 0.837, loss: 0.389
epoch: 86500, acc: 0.837, loss: 0.389
epoch: 86600, acc: 0.837, loss: 0.389
epoch: 86700, acc: 0.837, loss: 0.389
epoch: 86800, acc: 0.837, loss: 0.389
epoch: 86900, acc: 0.837, loss: 0.389
epoch: 87000, acc: 0.837, loss: 0.389
epoch: 87100, acc: 0.837, loss: 0.389
epoch: 87200, acc: 0.837, loss: 0.389
epoch: 87300, acc: 0.837, loss: 0.389
epoch: 87400, acc: 0.837, loss: 0.389
epoch: 87500, acc: 0.837, loss: 0.389
epoch: 87600, acc: 0.837, loss: 0.389
epoch: 87700, acc: 0.837, loss: 0.389
epoch: 87800, acc: 0.837, loss: 0.388
epoch: 87900, acc: 0.837, loss: 0.388
epoch: 88000, acc: 0.837, loss: 0.388
epoch: 88100, acc: 0.837, loss: 0.388
epoch: 88200, acc: 0.837, loss: 0.388
epoch: 88300, acc: 0.837, loss: 0.388
epoch: 88400, acc: 0.837, loss: 0.388
epoch: 88500, acc: 0.837, loss: 0.388
epoch: 88600, acc: 0.837, loss: 0.388
epoch: 88700, acc: 0.837, loss: 0.388
epoch: 88800, acc: 0.837, loss: 0.388
```

```
epoch: 88900, acc: 0.837, loss: 0.388
epoch: 89000, acc: 0.837, loss: 0.388
epoch: 89100, acc: 0.837, loss: 0.388
epoch: 89200, acc: 0.837, loss: 0.388
epoch: 89300, acc: 0.837, loss: 0.387
epoch: 89400, acc: 0.837, loss: 0.387
epoch: 89500, acc: 0.837, loss: 0.387
epoch: 89600, acc: 0.837, loss: 0.387
epoch: 89700, acc: 0.837, loss: 0.387
epoch: 89800, acc: 0.837, loss: 0.387
epoch: 89900, acc: 0.837, loss: 0.387
epoch: 90000, acc: 0.837, loss: 0.387
epoch: 90100, acc: 0.837, loss: 0.387
epoch: 90200, acc: 0.837, loss: 0.387
epoch: 90300, acc: 0.837, loss: 0.387
epoch: 90400, acc: 0.840, loss: 0.387
epoch: 90500, acc: 0.840, loss: 0.387
epoch: 90600, acc: 0.840, loss: 0.387
epoch: 90700, acc: 0.843, loss: 0.387
epoch: 90800, acc: 0.843, loss: 0.386
epoch: 90900, acc: 0.843, loss: 0.386
epoch: 91000, acc: 0.843, loss: 0.386
epoch: 91100, acc: 0.843, loss: 0.386
epoch: 91200, acc: 0.843, loss: 0.386
epoch: 91300, acc: 0.843, loss: 0.386
epoch: 91400, acc: 0.843, loss: 0.386
epoch: 91500, acc: 0.843, loss: 0.386
epoch: 91600, acc: 0.843, loss: 0.386
epoch: 91700, acc: 0.843, loss: 0.386
epoch: 91800, acc: 0.843, loss: 0.386
epoch: 91900, acc: 0.843, loss: 0.386
epoch: 92000, acc: 0.843, loss: 0.386
epoch: 92100, acc: 0.843, loss: 0.386
epoch: 92200, acc: 0.840, loss: 0.386
epoch: 92300, acc: 0.840, loss: 0.386
epoch: 92400, acc: 0.840, loss: 0.385
epoch: 92500, acc: 0.840, loss: 0.385
epoch: 92600, acc: 0.840, loss: 0.385
epoch: 92700, acc: 0.840, loss: 0.385
epoch: 92800, acc: 0.840, loss: 0.385
epoch: 92900, acc: 0.840, loss: 0.385
epoch: 93000, acc: 0.840, loss: 0.385
epoch: 93100, acc: 0.840, loss: 0.385
epoch: 93200, acc: 0.840, loss: 0.385
epoch: 93300, acc: 0.840, loss: 0.385
epoch: 93400, acc: 0.840, loss: 0.385
epoch: 93500, acc: 0.840, loss: 0.385
epoch: 93600, acc: 0.840, loss: 0.385
```

```
epoch: 93700, acc: 0.840, loss: 0.385
epoch: 93800, acc: 0.840, loss: 0.385
epoch: 93900, acc: 0.840, loss: 0.385
epoch: 94000, acc: 0.840, loss: 0.384
epoch: 94100, acc: 0.840, loss: 0.384
epoch: 94200, acc: 0.840, loss: 0.384
epoch: 94300, acc: 0.840, loss: 0.384
epoch: 94400, acc: 0.840, loss: 0.384
epoch: 94500, acc: 0.840, loss: 0.384
epoch: 94600, acc: 0.840, loss: 0.384
epoch: 94700, acc: 0.840, loss: 0.384
epoch: 94800, acc: 0.840, loss: 0.384
epoch: 94900, acc: 0.840, loss: 0.384
epoch: 95000, acc: 0.840, loss: 0.384
epoch: 95100, acc: 0.840, loss: 0.384
epoch: 95200, acc: 0.840, loss: 0.384
epoch: 95300, acc: 0.840, loss: 0.384
epoch: 95400, acc: 0.840, loss: 0.384
epoch: 95500, acc: 0.840, loss: 0.384
epoch: 95600, acc: 0.840, loss: 0.384
epoch: 95700, acc: 0.840, loss: 0.383
epoch: 95800, acc: 0.840, loss: 0.383
epoch: 95900, acc: 0.840, loss: 0.383
epoch: 96000, acc: 0.840, loss: 0.383
epoch: 96100, acc: 0.840, loss: 0.383
epoch: 96200, acc: 0.840, loss: 0.383
epoch: 96300, acc: 0.840, loss: 0.383
epoch: 96400, acc: 0.840, loss: 0.383
epoch: 96500, acc: 0.840, loss: 0.383
epoch: 96600, acc: 0.840, loss: 0.383
epoch: 96700, acc: 0.840, loss: 0.383
epoch: 96800, acc: 0.840, loss: 0.383
epoch: 96900, acc: 0.840, loss: 0.383
epoch: 97000, acc: 0.840, loss: 0.383
epoch: 97100, acc: 0.840, loss: 0.383
epoch: 97200, acc: 0.840, loss: 0.383
epoch: 97300, acc: 0.840, loss: 0.383
epoch: 97400, acc: 0.840, loss: 0.382
epoch: 97500, acc: 0.840, loss: 0.382
epoch: 97600, acc: 0.840, loss: 0.382
epoch: 97700, acc: 0.840, loss: 0.382
epoch: 97800, acc: 0.840, loss: 0.382
epoch: 97900, acc: 0.840, loss: 0.382
epoch: 98000, acc: 0.840, loss: 0.382
epoch: 98100, acc: 0.843, loss: 0.382
epoch: 98200, acc: 0.843, loss: 0.382
epoch: 98300, acc: 0.843, loss: 0.382
epoch: 98400, acc: 0.843, loss: 0.382
```

```
epoch: 98500, acc: 0.843, loss: 0.382
epoch: 98600, acc: 0.843, loss: 0.382
epoch: 98700, acc: 0.843, loss: 0.382
epoch: 98800, acc: 0.843, loss: 0.382
epoch: 98900, acc: 0.843, loss: 0.382
epoch: 99000, acc: 0.843, loss: 0.382
epoch: 99100, acc: 0.843, loss: 0.382
epoch: 99200, acc: 0.843, loss: 0.381
epoch: 99300, acc: 0.843, loss: 0.381
epoch: 99400, acc: 0.843, loss: 0.381
epoch: 99500, acc: 0.843, loss: 0.381
epoch: 99600, acc: 0.843, loss: 0.381
epoch: 99700, acc: 0.843, loss: 0.381
epoch: 99800, acc: 0.843, loss: 0.381
epoch: 99900, acc: 0.843, loss: 0.381
epoch: 100000, acc: 0.843, loss: 0.381
```

5 Momentum in Training Neural Networks

Using the past information on how changing the weights and biases changes the loss, we can use the large dimensional vectors to try and cancel out the overshoot.

The new weight update = some factor * last weight update + learning rate * current gradient

[20]:
```
class Optimizer_SGD():
    def __init__(self, learning_rate=0.5, decay=0.0, momentum=0.0):
        self.initial_rate = learning_rate
        self.current_learning_rate = self.initial_rate
        self.decay = decay
        self.iterations = 0
        self.momentum = momentum

    def pre_update_params(self):
        # Update the current_learning_rate before updating params
        if self.decay:
            self.current_learning_rate = self.initial_rate / (1 + self.decay *␣
 ↪self.iterations)

    def update_params(self, layer):
        if self.momentum:
            # For each layer, we need to use its last momentums

            # First check if the layer has a last momentum stored
            if not hasattr(layer, 'weight_momentums'):
                layer.weight_momentums = np.zeros_like(layer.weights)
                layer.bias_momentums = np.zeros_like(layer.biases)

            weight_updates = self.momentum * layer.weight_momentums - \
```

```
                                self.current_learning_rate * layer.dweights
            layer.weight_momentums = weight_updates

            bias_updates = self.momentum * layer.bias_momentums - \
                           self.current_learning_rate * layer.dbiases
            layer.bias_momentums = bias_updates

        # Not using momentum
        else:
            weight_updates = -self.current_learning_rate * layer.dweights
            bias_updates = -self.current_learning_rate * layer.dbiases

        layer.weights += weight_updates
        layer.biases += bias_updates

    def post_update_params(self):
        # Update the self.iterations for use with decay
        self.iterations += 1
```

6 Testing the Gradient Optimizer with Momentum

[21]:
```
# Create dataset
X, y = spiral_data(samples=100, classes=3)

# Create Dense layer with 2 input features and 64 output values
dense1 = Layer_Dense(2, 64)

# Create ReLU activation (to be used with Dense layer)
activation1 = Activation_ReLU()

# Create second Dense layer with 64 input features (as we take output
# of previous layer here) and 3 output values (output values)
dense2 = Layer_Dense(64, 3)

# Create Softmax classifier's combined loss and activation
loss_activation = Activation_Softmax_Loss_CategoricalCrossentropy()

# Create optimizer
optimizer = Optimizer_SGD(learning_rate=1.0, decay=1e-3, momentum=0.9)

# Train in loop
for epoch in range(10001):
    # Perform a forward pass of our training data through this layer
    dense1.forward(X)

    # Perform a forward pass through activation function
```

```
    # takes the output of first dense layer here
    activation1.forward(dense1.output)

    # Perform a forward pass through second Dense layer
    # takes outputs of activation function of first layer as inputs
    dense2.forward(activation1.output)

    # Perform a forward pass through the activation/loss function
    # takes the output of second dense layer here and returns loss
    loss = loss_activation.forward(dense2.output, y)

    # Calculate accuracy from output of activation2 and targets
    # calculate values along first axis
    predictions = np.argmax(loss_activation.output, axis=1)
    if len(y.shape) == 2:
        y = np.argmax(y, axis=1)
    accuracy = np.mean(predictions == y)

    if not epoch % 100:
        print(f'epoch: {epoch}, ' +
              f'acc: {accuracy:.3f}, ' +
              f'loss: {loss:.3f}')

    # Backward pass
    loss_activation.backward(loss_activation.output, y)
    dense2.backward(loss_activation.dinputs)
    activation1.backward(dense2.dinputs)
    dense1.backward(activation1.dinputs)

    # Update weights and biases
    optimizer.pre_update_params()
    optimizer.update_params(dense1)
    optimizer.update_params(dense2)
    optimizer.post_update_params()
```

```
epoch: 0, acc: 0.317, loss: 1.099
epoch: 100, acc: 0.437, loss: 1.033
epoch: 200, acc: 0.483, loss: 0.913
epoch: 300, acc: 0.750, loss: 0.622
epoch: 400, acc: 0.810, loss: 0.471
epoch: 500, acc: 0.850, loss: 0.403
epoch: 600, acc: 0.873, loss: 0.380
epoch: 700, acc: 0.843, loss: 0.378
epoch: 800, acc: 0.897, loss: 0.289
epoch: 900, acc: 0.910, loss: 0.264
epoch: 1000, acc: 0.883, loss: 0.293
epoch: 1100, acc: 0.923, loss: 0.236
```

```
epoch: 1200, acc: 0.927, loss: 0.227
epoch: 1300, acc: 0.903, loss: 0.238
epoch: 1400, acc: 0.927, loss: 0.215
epoch: 1500, acc: 0.927, loss: 0.204
epoch: 1600, acc: 0.930, loss: 0.198
epoch: 1700, acc: 0.927, loss: 0.195
epoch: 1800, acc: 0.927, loss: 0.191
epoch: 1900, acc: 0.930, loss: 0.190
epoch: 2000, acc: 0.930, loss: 0.185
epoch: 2100, acc: 0.930, loss: 0.183
epoch: 2200, acc: 0.930, loss: 0.181
epoch: 2300, acc: 0.930, loss: 0.179
epoch: 2400, acc: 0.933, loss: 0.177
epoch: 2500, acc: 0.930, loss: 0.175
epoch: 2600, acc: 0.930, loss: 0.174
epoch: 2700, acc: 0.930, loss: 0.172
epoch: 2800, acc: 0.930, loss: 0.171
epoch: 2900, acc: 0.930, loss: 0.170
epoch: 3000, acc: 0.933, loss: 0.168
epoch: 3100, acc: 0.930, loss: 0.167
epoch: 3200, acc: 0.933, loss: 0.167
epoch: 3300, acc: 0.937, loss: 0.166
epoch: 3400, acc: 0.937, loss: 0.165
epoch: 3500, acc: 0.940, loss: 0.164
epoch: 3600, acc: 0.940, loss: 0.163
epoch: 3700, acc: 0.940, loss: 0.162
epoch: 3800, acc: 0.943, loss: 0.161
epoch: 3900, acc: 0.940, loss: 0.161
epoch: 4000, acc: 0.940, loss: 0.160
epoch: 4100, acc: 0.940, loss: 0.159
epoch: 4200, acc: 0.940, loss: 0.159
epoch: 4300, acc: 0.940, loss: 0.158
epoch: 4400, acc: 0.940, loss: 0.158
epoch: 4500, acc: 0.940, loss: 0.157
epoch: 4600, acc: 0.940, loss: 0.157
epoch: 4700, acc: 0.940, loss: 0.156
epoch: 4800, acc: 0.943, loss: 0.156
epoch: 4900, acc: 0.940, loss: 0.155
epoch: 5000, acc: 0.940, loss: 0.155
epoch: 5100, acc: 0.940, loss: 0.154
epoch: 5200, acc: 0.940, loss: 0.154
epoch: 5300, acc: 0.940, loss: 0.154
epoch: 5400, acc: 0.940, loss: 0.153
epoch: 5500, acc: 0.940, loss: 0.153
epoch: 5600, acc: 0.940, loss: 0.153
epoch: 5700, acc: 0.940, loss: 0.152
epoch: 5800, acc: 0.940, loss: 0.152
epoch: 5900, acc: 0.940, loss: 0.152
```

```
epoch: 6000, acc: 0.940, loss: 0.151
epoch: 6100, acc: 0.940, loss: 0.151
epoch: 6200, acc: 0.940, loss: 0.151
epoch: 6300, acc: 0.940, loss: 0.151
epoch: 6400, acc: 0.940, loss: 0.150
epoch: 6500, acc: 0.940, loss: 0.150
epoch: 6600, acc: 0.940, loss: 0.150
epoch: 6700, acc: 0.940, loss: 0.150
epoch: 6800, acc: 0.940, loss: 0.150
epoch: 6900, acc: 0.940, loss: 0.149
epoch: 7000, acc: 0.940, loss: 0.149
epoch: 7100, acc: 0.940, loss: 0.149
epoch: 7200, acc: 0.940, loss: 0.149
epoch: 7300, acc: 0.940, loss: 0.149
epoch: 7400, acc: 0.940, loss: 0.148
epoch: 7500, acc: 0.940, loss: 0.148
epoch: 7600, acc: 0.943, loss: 0.148
epoch: 7700, acc: 0.943, loss: 0.148
epoch: 7800, acc: 0.940, loss: 0.147
epoch: 7900, acc: 0.943, loss: 0.147
epoch: 8000, acc: 0.943, loss: 0.147
epoch: 8100, acc: 0.943, loss: 0.147
epoch: 8200, acc: 0.940, loss: 0.147
epoch: 8300, acc: 0.943, loss: 0.147
epoch: 8400, acc: 0.943, loss: 0.146
epoch: 8500, acc: 0.943, loss: 0.146
epoch: 8600, acc: 0.943, loss: 0.146
epoch: 8700, acc: 0.943, loss: 0.146
epoch: 8800, acc: 0.943, loss: 0.146
epoch: 8900, acc: 0.943, loss: 0.146
epoch: 9000, acc: 0.943, loss: 0.146
epoch: 9100, acc: 0.943, loss: 0.145
epoch: 9200, acc: 0.943, loss: 0.145
epoch: 9300, acc: 0.943, loss: 0.145
epoch: 9400, acc: 0.943, loss: 0.145
epoch: 9500, acc: 0.943, loss: 0.145
epoch: 9600, acc: 0.943, loss: 0.145
epoch: 9700, acc: 0.943, loss: 0.145
epoch: 9800, acc: 0.943, loss: 0.145
epoch: 9900, acc: 0.943, loss: 0.145
epoch: 10000, acc: 0.943, loss: 0.144
```