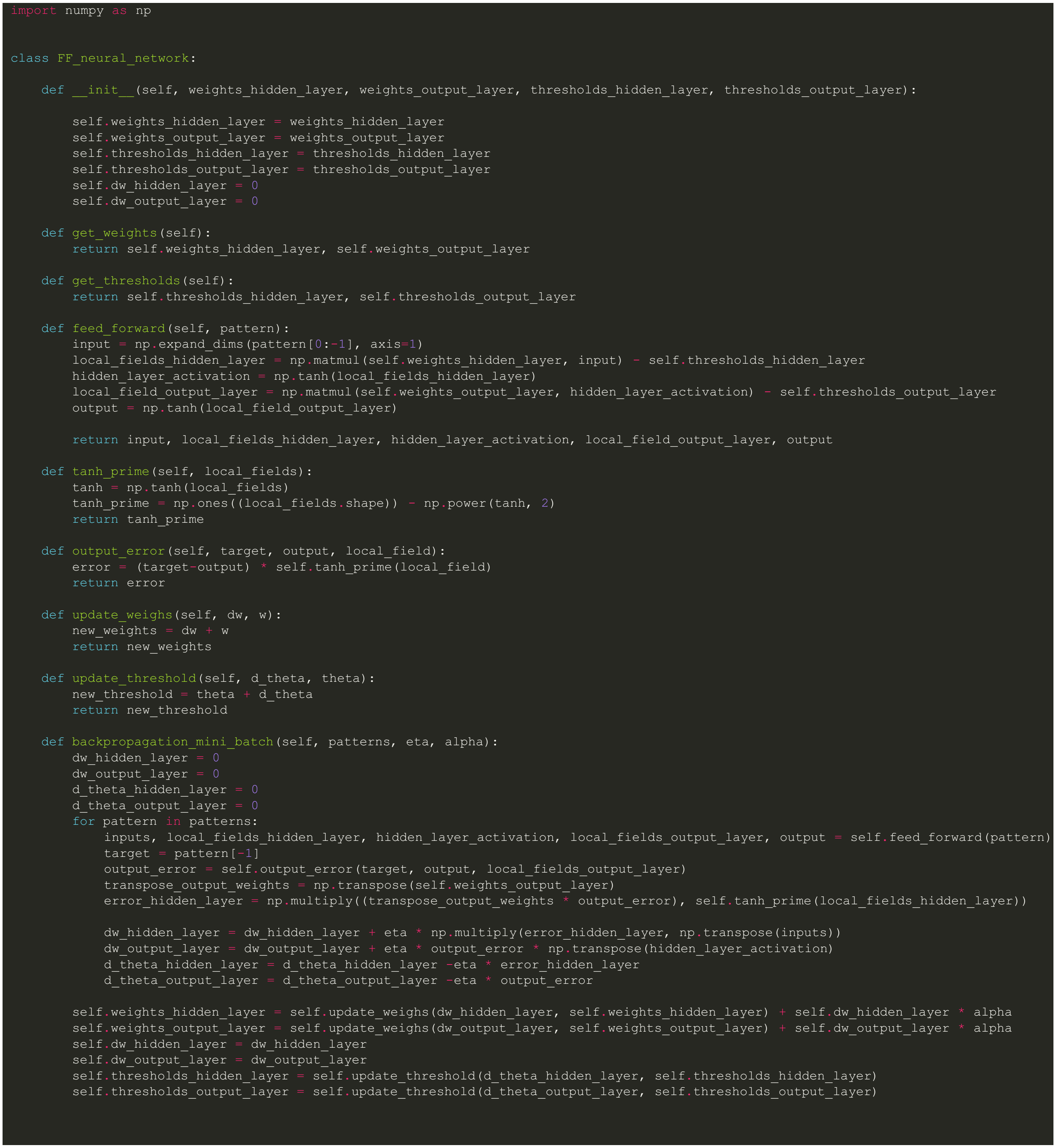
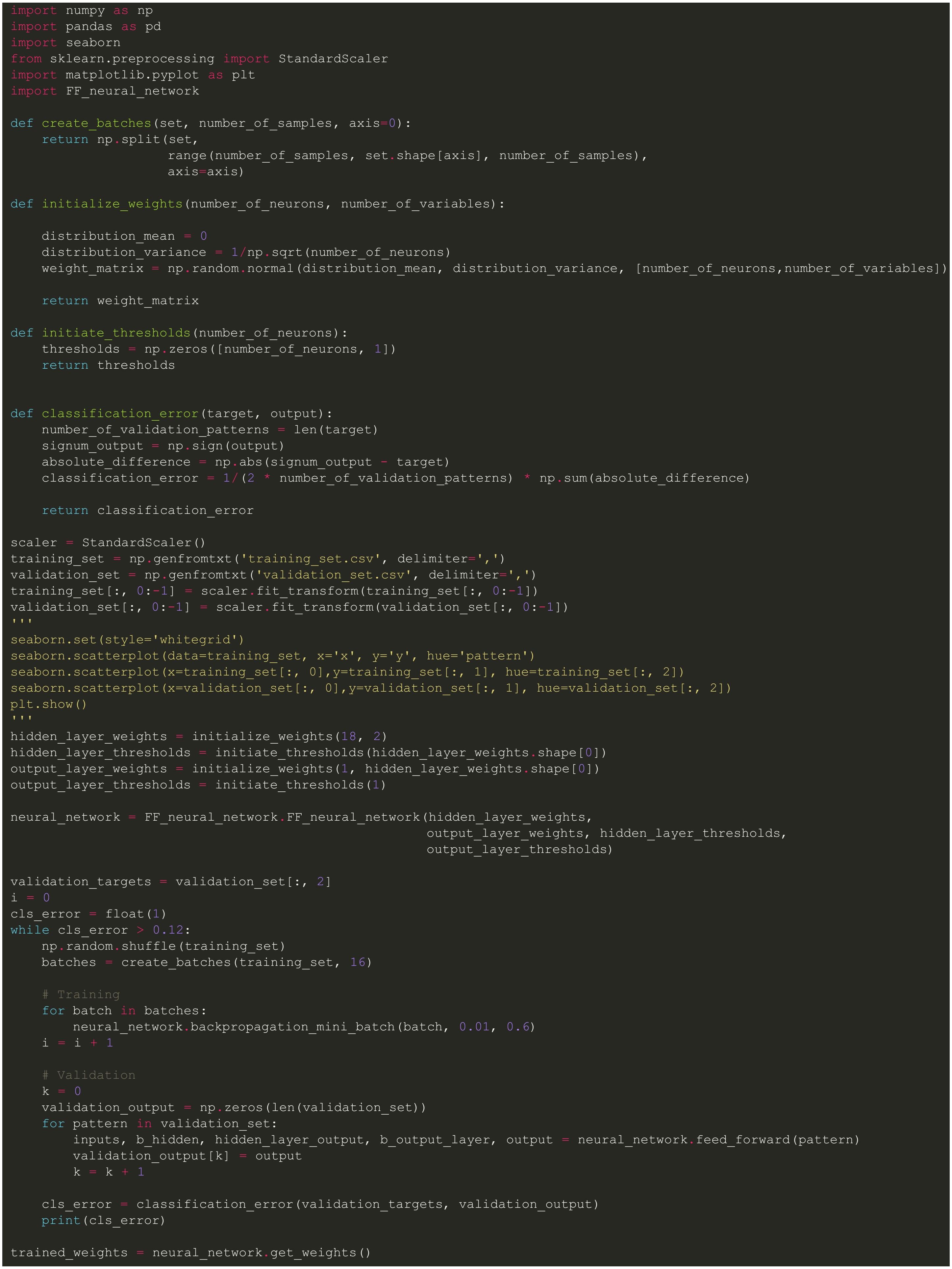


```
import numpy as np


class FF_neural_network:

    def __init__(self, weights_hidden_layer, weights_output_layer, thresholds_hidden_layer, thresholds_output_layer):

        self.weights_hidden_layer = weights_hidden_layer
        self.weights_output_layer = weights_output_layer
        self.thresholds_hidden_layer = thresholds_hidden_layer
        self.thresholds_output_layer = thresholds_output_layer
        self.dw_hidden_layer = 0
        self.dw_output_layer = 0

    def get_weights(self):
        return self.weights_hidden_layer, self.weights_output_layer

    def get_thresholds(self):
        return self.thresholds_hidden_layer, self.thresholds_output_layer

    def feed_forward(self, pattern):
        input = np.expand_dims(pattern[0:-1], axis=1)
        local_fields_hidden_layer = np.matmul(self.weights_hidden_layer, input) - self.thresholds_hidden_layer
        hidden_layer_activation = np.tanh(local_fields_hidden_layer)
        local_field_output_layer = np.matmul(self.weights_output_layer, hidden_layer_activation) - self.thresholds_output_layer
        output = np.tanh(local_field_output_layer)

        return input, local_fields_hidden_layer, hidden_layer_activation, local_field_output_layer, output

    def tanh_prime(self, local_fields):
        tanh = np.tanh(local_fields)
        tanh_prime = np.ones((local_fields.shape)) - np.power(tanh, 2)
        return tanh_prime

    def output_error(self, target, output, local_field):
        error = (target-output) * self.tanh_prime(local_field)
        return error

    def update_weighs(self, dw, w):
        new_weights = dw + w
        return new_weights

    def update_threshold(self, d_theta, theta):
        new_threshold = theta + d_theta
        return new_threshold

    def backpropagation_mini_batch(self, patterns, eta, alpha):
        dw_hidden_layer = 0
        dw_output_layer = 0
        d_theta_hidden_layer = 0
        d_theta_output_layer = 0
        for pattern in patterns:
            inputs, local_fields_hidden_layer, hidden_layer_activation, local_fields_output_layer, output = self.feed_forward(pattern)
            target = pattern[-1]
            output_error = self.output_error(target, output, local_fields_output_layer)
            transpose_output_weights = np.transpose(self.weights_output_layer)
            error_hidden_layer = np.multiply((transpose_output_weights * output_error), self.tanh_prime(local_fields_hidden_layer))

            dw_hidden_layer = dw_hidden_layer + eta * np.multiply(error_hidden_layer, np.transpose(inputs))
            dw_output_layer = dw_output_layer + eta * output_error * np.transpose(hidden_layer_activation)
            d_theta_hidden_layer = d_theta_hidden_layer -eta * error_hidden_layer
            d_theta_output_layer = d_theta_output_layer -eta * output_error

        self.weights_hidden_layer = self.update_weighs(dw_hidden_layer, self.weights_hidden_layer) + self.dw_hidden_layer * alpha
        self.weights_output_layer = self.update_weighs(dw_output_layer, self.weights_output_layer) + self.dw_output_layer * alpha
        self.dw_hidden_layer = dw_hidden_layer
        self.dw_output_layer = dw_output_layer
        self.thresholds_hidden_layer = self.update_threshold(d_theta_hidden_layer, self.thresholds_hidden_layer)
        self.thresholds_output_layer = self.update_threshold(d_theta_output_layer, self.thresholds_output_layer)
```



```
import numpy as np
import pandas as pd
import seaborn
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import FF_neural_network

def create_batches(set, number_of_samples, axis=0):
    return np.split(set,
                    range(number_of_samples, set.shape[axis], number_of_samples),
                    axis=axis)

def initialize_weights(number_of_neurons, number_of_variables):

    distribution_mean = 0
    distribution_variance = 1/np.sqrt(number_of_neurons)
    weight_matrix = np.random.normal(distribution_mean, distribution_variance, [number_of_neurons,number_of_variables])

    return weight_matrix

def initiate_thresholds(number_of_neurons):
    thresholds = np.zeros([number_of_neurons, 1])
    return thresholds


def classification_error(target, output):
    number_of_validation_patterns = len(target)
    signum_output = np.sign(output)
    absolute_difference = np.abs(signum_output - target)
    classification_error = 1/(2 * number_of_validation_patterns) * np.sum(absolute_difference)

    return classification_error

scaler = StandardScaler()
training_set = np.genfromtxt('training_set.csv', delimiter=',')
validation_set = np.genfromtxt('validation_set.csv', delimiter=',')
training_set[:, 0:-1] = scaler.fit_transform(training_set[:, 0:-1])
validation_set[:, 0:-1] = scaler.fit_transform(validation_set[:, 0:-1])
'''
seaborn.set(style='whitegrid')
seaborn.scatterplot(data=training_set, x='x', y='y', hue='pattern')
seaborn.scatterplot(x=training_set[:, 0],y=training_set[:, 1], hue=training_set[:, 2])
seaborn.scatterplot(x=validation_set[:, 0],y=validation_set[:, 1], hue=validation_set[:, 2])
plt.show()
'''
hidden_layer_weights = initialize_weights(18, 2)
hidden_layer_thresholds = initiate_thresholds(hidden_layer_weights.shape[0])
output_layer_weights = initialize_weights(1, hidden_layer_weights.shape[0])
output_layer_thresholds = initiate_thresholds(1)

neural_network = FF_neural_network.FF_neural_network(hidden_layer_weights,
                                                      output_layer_weights, hidden_layer_thresholds,
                                                      output_layer_thresholds)

validation_targets = validation_set[:, 2]
i = 0
cls_error = float(1)
while cls_error > 0.12:
    np.random.shuffle(training_set)
    batches = create_batches(training_set, 16)

    # Training
    for batch in batches:
        neural_network.backpropagation_mini_batch(batch, 0.01, 0.6)
    i = i + 1

    # Validation
    k = 0
    validation_output = np.zeros(len(validation_set))
    for pattern in validation_set:
        inputs, b_hidden, hidden_layer_output, b_output_layer, output = neural_network.feed_forward(pattern)
        validation_output[k] = output
        k = k + 1

    cls_error = classification_error(validation_targets, validation_output)
    print(cls_error)

trained_weights = neural_network.get_weights()
```

```
trained_thresholds = neural_network.get_thresholds()

w1 = pd.DataFrame(trained_weights[0])
w2 = pd.DataFrame(trained_weights[1])
t1 = pd.DataFrame(trained_thresholds[0])
t2 = pd.DataFrame(trained_thresholds[1])

w1.to_csv('w1.csv', index=False, header=False)
w2.to_csv('w2.csv', index=False, header=False)
t1.to_csv('t1.csv', index=False, header=False)
t2.to_csv('t2.csv', index=False, header=False)
```