In [18]:
```
#Importing the necessary libraries
import numpy as np
import activations
import torch
from sklearn.metrics import confusion_matrix
```

In [19]:
```
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sn
```

In [1]:
```
def convert(imgf, labelf, outf, n):
    f = open(imgf, "rb")
    o = open(outf, "w")
    l = open(labelf, "rb")

    f.read(16)
    l.read(8)
    images = []

    for i in range(n):
        image = [ord(l.read(1))]
        for j in range(28*28):
            image.append(ord(f.read(1)))
        images.append(image)

    for image in images:
        o.write(",".join(str(pix) for pix in image)+"\n")
    f.close()
    o.close()
    l.close()

convert("train-images-idx3-ubyte", "train-labels-idx1-ubyte",
        "mnist_train.csv", 60000)
convert("t10k-images-idx3-ubyte", "t10k-labels-idx1-ubyte",
        "mnist_test.csv", 10000)
```

In [20]:
```
def load_data(path):
    def one_hot(y):
        table = np.zeros((y.shape[0], 10))
        for i in range(y.shape[0]):
            table[i][int(y[i][0])] = 1
        return table

    def normalize(x):
        x = x / 255
        return x

    data = np.loadtxt('{}'.format(path), delimiter = ',')
    return normalize(data[:,1:]),one_hot(data[:,:1])

train_data, train_label = load_data('mnist_train.csv')
test_data, test_label = load_data('mnist_test.csv')
```

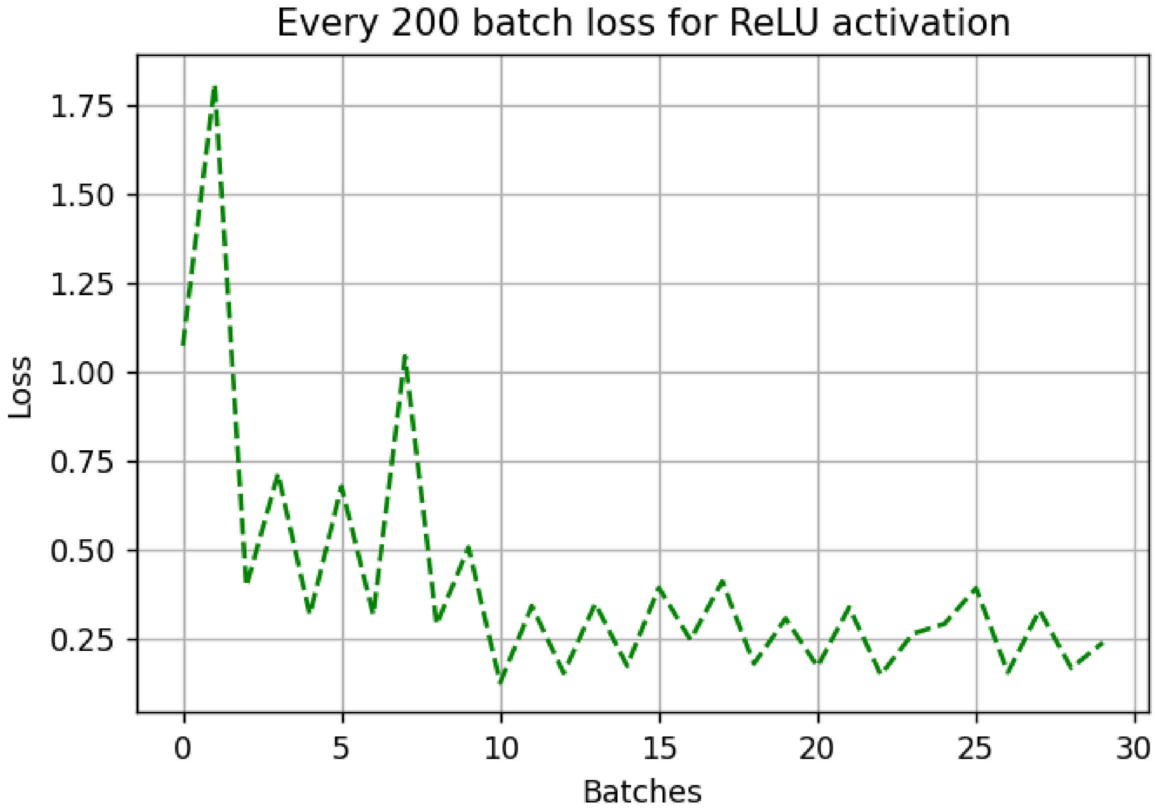
In [27]:
```
class NeuralNetwork:
    def __init__(self, X, y, batch = 64, lr = 0.01,  epochs = 15):
        self.input = X
        self.target = y
        self.batch = batch
```

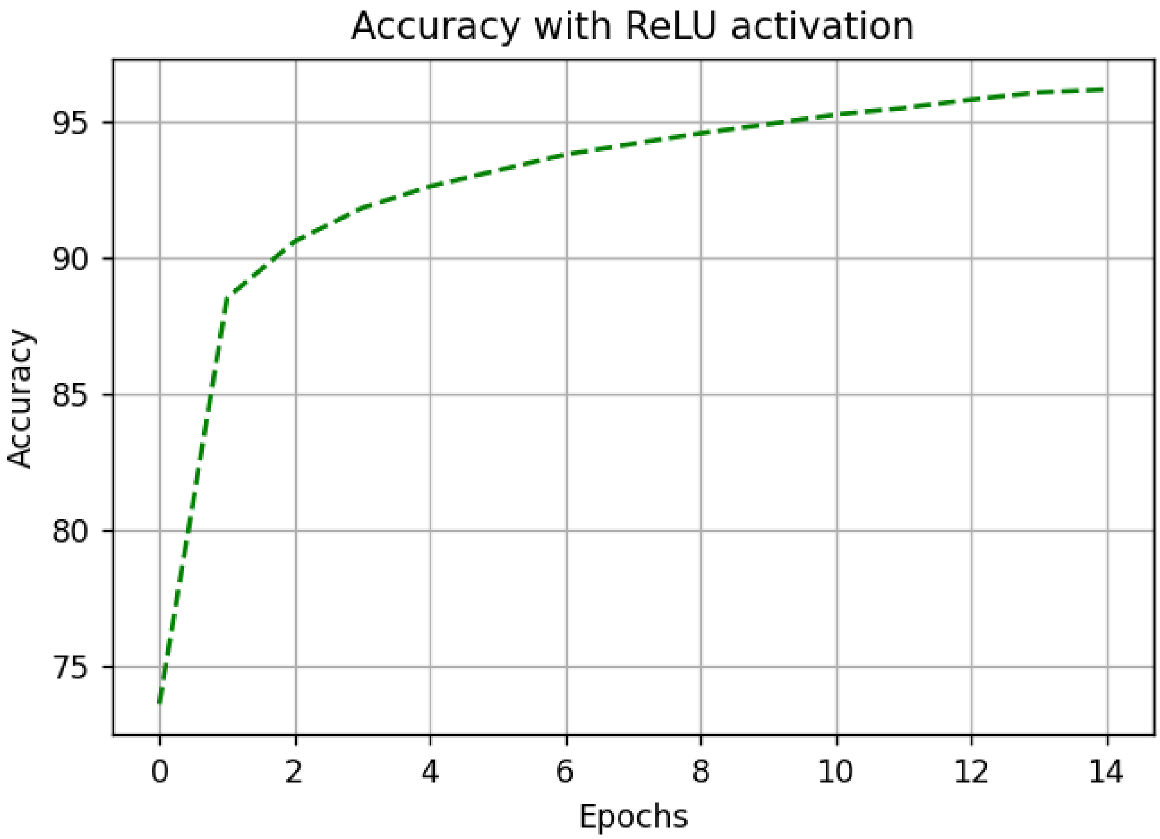
```
        self.epochs = epochs
        self.lr = lr
        self.size=[784,500,250,100,10]
        self.testerror=[]

        self.x = self.input[:self.batch]
        self.y = self.target[:self.batch]
        self.loss = []
        self.batchloss=[]
        self.acc = []

        self.init_weights()
        self.yhat=[]


    def init_weights(self):

        def glorot_initialization(inputsize, outputsize):
            wij = np.sqrt(6/(inputsize+outputsize))
            return np.random.uniform(-wij, wij, (outputsize, inputsize))

        self.W1 = glorot_initialization(500,self.size[0])
        self.W2 = glorot_initialization(250,self.size[1])
        self.W3 = glorot_initialization(100,self.size[2])
        self.W4 = glorot_initialization(self.y.shape[1],self.size[3])

        self.b1 = np.zeros(self.W1.shape[1],)
        self.b2 = np.zeros(self.W2.shape[1],)
        self.b3 = np.zeros(self.W3.shape[1],)
        self.b4 = np.zeros(self.W4.shape[1],)



    def relu(self, x):
        return np.maximum(0,x)

    def drelu(self,x):
        return 1 * (x > 0)

    def sigmoid(self,x):
        return 1/(1+np.exp(-x))

    def dsigmoid(self,x):
        return self.sigmoid(x)*(1-self.sigmoid(x))

    def tanh(self,x):
        return np.tanh(x)

    def dtanh(self,x):
        return (1 - (np.tanh(x))**2)

    def softmax(self, z):
        z = z - np.max(z, axis = 1).reshape(z.shape[0],1)
        return np.exp(z) / np.sum(np.exp(z), axis = 1).reshape(z.shape[0],1

    def shuffle(self):
        idx = [i for i in range(self.input.shape[0])]
        np.random.shuffle(idx)
        self.input = self.input[idx]
        self.target = self.target[idx]
```

```
    def feedforward(self):
        assert self.x.shape[1] == self.W1.shape[0]
        self.z1 = self.x.dot(self.W1) + self.b1
        self.a1 = self.relu(self.z1)

        assert self.a1.shape[1] == self.W2.shape[0]
        self.z2 = self.a1.dot(self.W2) + self.b2
        self.a2 = self.relu(self.z2)

        assert self.a2.shape[1] == self.W3.shape[0]
        self.z3 = self.a2.dot(self.W3) + self.b3
        self.a3 = self.relu(self.z3)

        assert self.a3.shape[1] == self.W4.shape[0]
        self.z4 = self.a3.dot(self.W4) + self.b4
        self.a4 = self.softmax(self.z4)


        self.error = self.a4 - self.y
        self.yhat.append(self.a4)


    def backprop(self):
        dz4 = (1/self.batch)*self.error
        dw4 = np.dot(self.a3.T,dz4)

        dz3 = np.matmul(self.W4,dz4.T) * self.drelu(self.z3).T
        dw3 = np.dot(dz3,self.a2).T

        dz2 = np.dot(self.W3,dz3)* self.drelu(self.z2).T
        dw2 = np.dot(dz2,self.a1).T

        dz1 = np.dot(self.W2,dz2)*self.drelu(self.z1).T
        dw1 = np.dot(dz1,self.x).T


        db4 = np.sum(dz4,axis=0)
        db3 = np.sum(np.matmul(dz4,self.W4.T) * self.drelu(self.z3),axis =
        x = np.matmul(dz4,self.W4.T) * self.drelu(self.z3)
        y = np.dot(x,self.W3.T)*self.drelu(self.z2)
        db2 = np.sum(y,axis=0)
        y1 = np.dot(y,self.W2.T) * self.drelu(self.z1)
        db1 = np.sum(y1,axis=0)


        assert dw4.shape == self.W4.shape
        assert dw3.shape == self.W3.shape
        assert dw2.shape == self.W2.shape
        assert dw1.shape == self.W1.shape

        assert db4.shape == self.b4.shape
        assert db3.shape == self.b3.shape
        assert db2.shape == self.b2.shape
        assert db1.shape == self.b1.shape

        self.W4 = self.W4 - self.lr * dw4
        self.W3 = self.W3 - self.lr * dw3
        self.W2 = self.W2 - self.lr * dw2
        self.W1 = self.W1 - self.lr * dw1

        self.b4 = self.b4 - self.lr * db4
```

```
        self.b3 = self.b3 - self.lr * db3
        self.b2 = self.b2 - self.lr * db2
        self.b1 = self.b1 - self.lr * db1


    def train(self):
        for epoch in range(self.epochs):
            l = 0
            acc = 0
            self.shuffle()
            count=0
            for batch in range(self.input.shape[0]//self.batch-1):
                start = batch*self.batch
                end = (batch+1)*self.batch
                self.x = self.input[start:end]
                self.y = self.target[start:end]
                self.feedforward()
                self.backprop()
                count+=1
                if(count%200==0):
                    for i in range(64):
                        l+=-np.sum(self.y[i,:]*np.log(self.a4[i,:]))
                    self.batchloss.append(l/64)
                acc+= np.count_nonzero(np.argmax(self.a4,axis=1) == np.argm

            self.loss.append(l/(self.input.shape[0]//self.batch))
            self.acc.append(acc*100/(self.input.shape[0]//self.batch))

    def plot(self):
        plt.figure(dpi = 125)
        plt.plot(self.batchloss,'--',color='green')
        plt.grid(True)
        plt.xlabel("Batches")
        plt.ylabel("Loss")
        plt.title("Every 200 batch loss for ReLU activation")

    def acc_plot(self):
        plt.figure(dpi = 125)
        plt.plot(self.acc,'--',color='green')
        plt.xlabel("Epochs")
        plt.ylabel("Accuracy")
        plt.grid(True)
        plt.title("Accuracy with ReLU activation")

    def confusion_matrix(self):
        k = np.argmax(self.y_hat,axis=0)
        cm=confusion_matrix(k, self.y)
        plt.figure(figsize=(8,8))
        sn.heatmap(confusion_matrix(k, self.y),annot=True,fmt='d')
        plt.xlabel('Predicted')
        plt.ylabel('truth')

    def test(self,xtest,ytest):
        self.x = xtest
        self.y = ytest
        self.feedforward()
        acc = np.count_nonzero(np.argmax(self.a4,axis=1) == np.argmax(self.
        print("Accuracy:", 100 * acc, "%")
```

```
NN = NeuralNetwork(train_data[:30000], train_label[:30000])
NN.train()
NN.plot()
NN.acc_plot()
NN.confusion_matrix()
```





In [28]:
```
NN.test(test_data,test_label)
```

```
Accuracy: 95.74000000000001 %
```