VU Amsterdam - Skillf for Ai
Assignment week 4: LinAlg-Neural Network - Team 24 - 07/10/2021

Dian Liu
2700348

Xiaoyang Sun
2734554

Ágoston Szabó
2738183

1 Introduction

We used a single layer network structure, consisting of a randomly initialized 25x3 matrix NN1 (also called a filter), a sigmoid function and a softmax function. The input is a character represented as 5x5 matrix, that is first flattened to a 1x25 matrix and then multiplied with NN1, giving a 1x3 array.

A sigmoid function is then applied to the results for introducing nonlinearity and then the softmax function normalizes these values using the standard exponential function. The error is calculated using the output and the true labels for the characters, which is used for updating the matrix NN1. This is repeated for a set amount of times, called epochs.

However, to speed up processing all nine character matrices (3 variations of each letter) are concatenated into a 9x25 matrix to speed up processing. (matrix multiplication is considerably faster than looping over each character after one and other).

2 Training

Because the data set is small and the problem to be solved is fairly easy, the number of training runs (epochs) does not need to be very high. In this experiment, 10000 epochs are set, the learning rate is 0.001, taking only a few seconds to complete.

3 Results

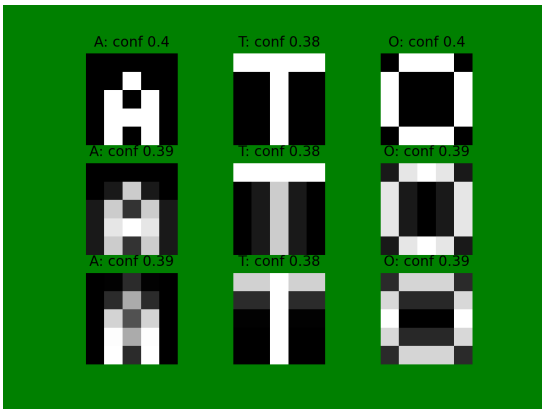

Figure 1: Results for Agoston

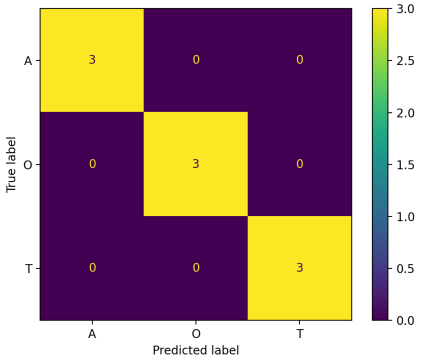

Figure 2: Confusion matrix for Agoston

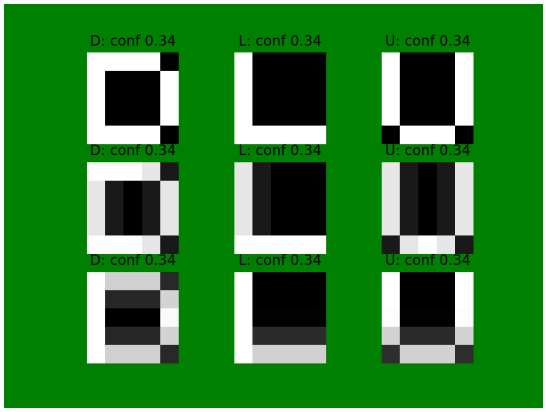

Figure 3: Results for Liu

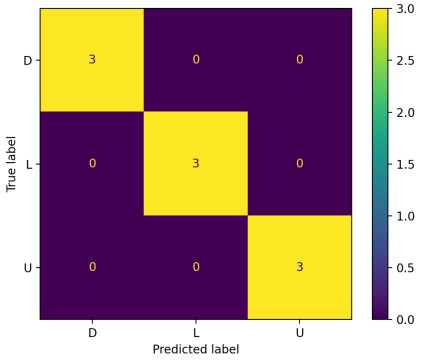

Figure 4: Confusion matrix for Liu

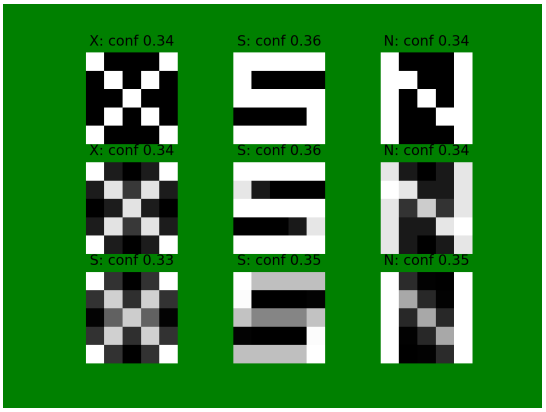

Figure 5: Results for Xiaoyang

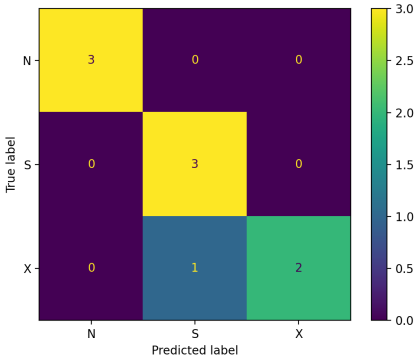

Figure 6: Confusion matrix for Xiaoyang

4 Analysis

4.1 Are you satisfied with the accuracy / performance of this algorithm? Is it good enough for actual use?

The algorithm is able to identify all of the 9 letters, apart from mistaking one letter for Xiaoyang, but the confidence of the predictions is fairly low, that is the sign for that recognising more letters using this technique is not possible. In order to identify all 26 letters of the alphabet a more complicated network structure is necessary, either using more layers or multiple matrices in 1 layer.

4.2 Can this algorithm be extended to work with all 26 characters?

In order to identify all 26 letters of the alphabet a more complicated network structure is necessary, either using more layers or multiple filters in one layer.

4.3 What are the weak spots of the algorithm? Which inputs does it not work well on? What input is likely to be confused?

First, the weak spot of the algorithm is the number of layers (only one), the number of filters and the size of the filters. Second, the algorithm in its current state always outputs a prediction, which can be seen as a weak point.

4.4 If you use it in a real world application, can you define thresholds so that we return ”Don’t Know” if the difference in scores is too low? What would the thresholds be?

A way to improve this would be to define a certain confidence threshold only over that the algorithm outputs a predictions. However, the confidence number can be used as a measure for the the probability of a correct or incorrect prediction.

4.5 Preprocessing any input is a good way to make AI algorithms more reliable. What preprocessing could be used for this problem to make the input data more uniform, so that the algorithm works well?

In addition to the clipping, blurring, and movement we use, there are many preprocessing methods that can make the network more robust. For example, the spatial domain method can be adopted, upscaling, data augmentation or normalizing the input.

Neighborhood enhancement algorithms are divided into two types: image smoothing and sharpening. Common smoothing algorithms include mean filtering and median filtering, spatial filtering. Common algorithms for sharpening include gradient operator method, second derivative operator method, high-pass filtering, mask matching method, etc.

5 Code

5.1 letterrecognition.py

```
import numpy as np
from Dataset import *
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay


dataset = Letters()
x=[]
y=[]
for i in dataset.images[:3]:
    x.append(i)
    y.append(i.target)
    #x.append(darken(i))
    #y.append(i.target)
    for b in blur(i):
        x.append(b)
        y.append(i.target)



X_train = np.asarray(x)
X_test = np.asarray(x)
Y_train = np.squeeze(y)
Y_test = np.squeeze(y)

map = 0 #0:Agoston, 1:Xiaoyang, 2:Liu

def init(x,y):
    layer = np.random.uniform(-1. ,1., size=(x,y)) / np.sqrt(x*y)
    return layer.astype(np.float32)

np.random.seed(0)
NN1 = init(5*5, 3)


#Sigmoid funstion
def sigmoid(x):
    return 1 / (np.exp(-x) + 1)

#Softmax
def softmax(x):
    exp_element = np.exp(x - x.max())
    return exp_element / np.sum(exp_element, axis=0)

#derivative of softmax
def d_softmax(x):
    exp_element = np.exp(x - x.max())
    return exp_element / np.sum(exp_element, axis=0) * (1 - exp_element /
    np.sum(exp_element, axis=0))

# forward and backward pass
def step(x, y):
    targets = np.zeros((len(y), 3), np.float32)
    targets[range(targets.shape[0]), y] = 1

    x_nn1 = x.dot(NN1)
```

```
    x_sigmoid = sigmoid(x_nn1)
    out = softmax(x_sigmoid)

    error = 2 * (out - targets) / out.shape[0] * d_softmax(x_sigmoid)
    update_nn1 = x.T @ error

    return out, update_nn1


def plot9(images):
    fig, ax1 = plt.subplots(3, 3)
    fig.set_facecolor('g')
    for i, img in enumerate(images):
        res = sigmoid(img.reshape(-1, 5 * 5).dot(NN1))
        ax1[i % 3][int(np.floor(i/3))].axis('off')
        ax1[i % 3][int(np.floor(i/3))].set_title(str(letter(np.argmax(res), map)) +
        ': conf ' + str(round(np.max(softmax(res.reshape(3,-1))),2)))
        ax1[i % 3][int(np.floor(i/3))].imshow(img, cmap='gray')
    plt.show()


epochs = 10000
lr = 0.001
batch = 9
losses, accuracies, val_accuracies = [], [], []


for i in range(epochs):
    sample = np.random.randint(0, X_train.shape[0], size=batch)
    X = X_train[sample].reshape((-1, 5 * 5))
    Y = Y_train[sample]

    out, update_nn1 = step(X, Y)
    category = np.argmax(out, axis=1)
    accuracy = (category == Y).mean()
    accuracies.append(accuracy)

    loss = ((category - Y) ** 2).mean()
    losses.append(loss.item())
    NN1 = NN1 - lr * update_nn1

    if (i % 10 == 0):
        X_test = X_test.reshape((-1, 5 * 5))
        val_out = np.argmax(softmax(sigmoid(X_test.dot(NN1))), axis=1)
        val_acc = (val_out == Y_test).mean()
        val_accuracies.append(val_acc.item())
    if (i % 500 == 0): print(f'For {i}th epoch: train accuracy: {accuracy:.3f} |
    validation accuracy:{val_acc:.3f}')


plot9(x)

predictions = []
labels = []
for img, label in zip(x, y):
    predictions.append(np.argmax(sigmoid(img.reshape(-1, 5 * 5).dot(NN1))))
    labels.append(label)
```

```
labels = [letter(label, map) for label in labels]
predictions = [letter(pred, map) for pred in predictions]


fig = ConfusionMatrixDisplay.from_predictions(labels, predictions)
fig.ax_.set_title('Confusion Matrix')
fig.plot()
```

5.2 Dataset.py

```
import numpy as np
from dataclasses import dataclass


@dataclass
class Letters:
    def __init__(self):
        self.images = []

        # Agoston Szabo
        self.images.append(np.matrix(([0,0,0,0,0], [0,0,1,0,0],
        [0,1,0,1,0], [0,1,1,1,0], [0,1,0,1,0])))
        self.images[0].target = 0 #A
        self.images.append(np.matrix(([1, 1, 1, 1, 1], [0, 0, 1, 0, 0],
        [0, 0, 1, 0, 0], [0, 0, 1, 0, 0], [0, 0, 1, 0, 0])))
        self.images[1].target = 1 #T
        self.images.append(np.matrix(([0, 1, 1, 1, 0], [1, 0, 0, 0, 1],
        [1, 0, 0, 0, 1], [1, 0, 0, 0, 1], [0, 1, 1, 1, 0])))
        self.images[2].target = 2 #O
        self.images.append(np.matrix(([1, 0, 0, 0, 1], [0, 1, 0, 1, 0],
        [0, 0, 1, 0, 0], [0, 1, 0, 1, 0], [1, 0, 0, 0, 1])))

        # Xiaoyang Sun
        self.images[3].target = 0  # X
        self.images.append(np.matrix(([1, 1, 1, 1, 1], [1, 0, 0, 0, 0],
        [1, 1, 1, 1, 1], [0, 0, 0, 0, 1], [1, 1, 1, 1, 1])))
        self.images[4].target = 1  # S
        self.images.append(np.matrix(([1, 0, 0, 0, 1], [1, 1, 0, 0, 1],
        [1, 0, 1, 0, 1], [1, 0, 0, 1, 1], [1, 0, 0, 0, 1])))
        self.images[5].target = 2  # N
        self.images.append(np.matrix(([1, 1, 1, 1, 0], [1, 0, 0, 0, 1],
        [1, 0, 0, 0, 1], [1, 0, 0, 0, 1], [1, 1, 1, 1, 0])))

        # Liu Dian
        self.images[6].target = 0  # D
        self.images.append(np.matrix(([1, 0, 0, 0, 0], [1, 0, 0, 0, 0],
        [1, 0, 0, 0, 0], [1, 0, 0, 0, 0], [1, 1, 1, 1, 1])))
        self.images[7].target = 1  # L
        self.images.append(np.matrix(([1, 0, 0, 0, 1], [1, 0, 0, 0, 1],
        [1, 0, 0, 0, 1], [1, 0, 0, 0, 1], [0, 1, 1, 1, 0])))
        self.images[8].target = 2  # U
        self.images.append(np.matrix(([1, 1, 1, 1, 1], [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1], [1, 1, 1, 1, 1], [1, 1, 1, 1, 1])))
        self.images[9].target = 0  # All 1

def flatten(image):
    return image.reshape(1, -1)

def blur(image):
```

```
    blur1 = np.array([[0.9, 0.1, 0, 0, 0], [0.1, 0.8, 0.1, 0, 0],
    [0, 0.1, 0.8, 0.1, 0], [0, 0, 0.1, 0.8, 0.1],[0, 0, 0, 0.1, 0.9]])
    return np.matmul(image, blur1), np.matmul(blur1, np.matmul(blur1, image))

def darken(image, factor = 0.9):
    return factor * image

def letter(num, map=0):
    mapping = {0: {0: 'A', 1: 'T', 2: 'O'}, 1: {0: 'X', 1: 'S', 2: 'N'},
    2: {0: 'D', 1: 'L', 2: 'U'}}
    return (mapping[map])[num]
```