Letters recognition with neural network

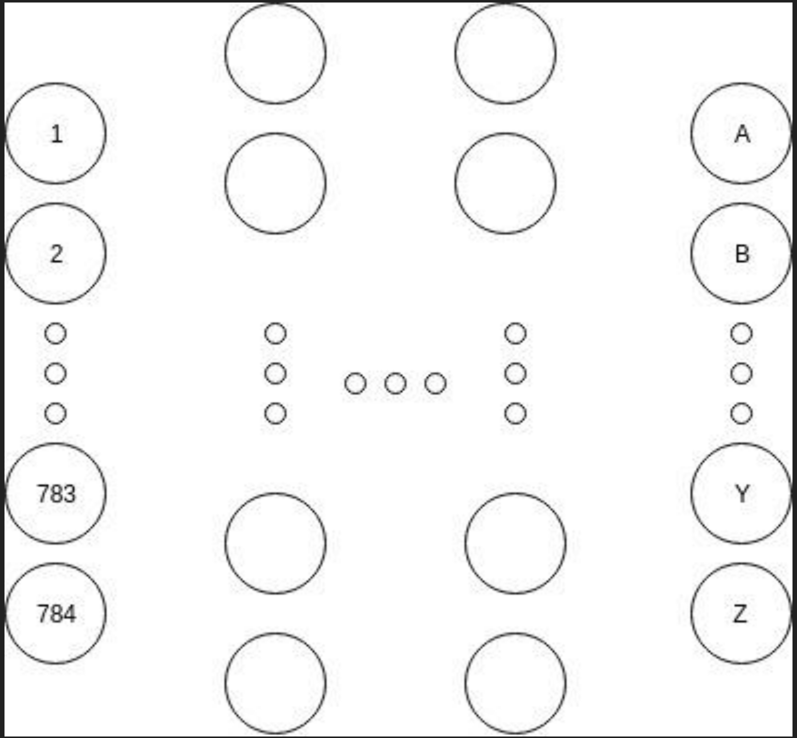
Assumptions

- program written in python language
- program can recognize handwritten letters
- program uses numpy library
- program uses EMNIST letters database

EMNIST

- 28x28
- format .csv
- database of 140 000 letters

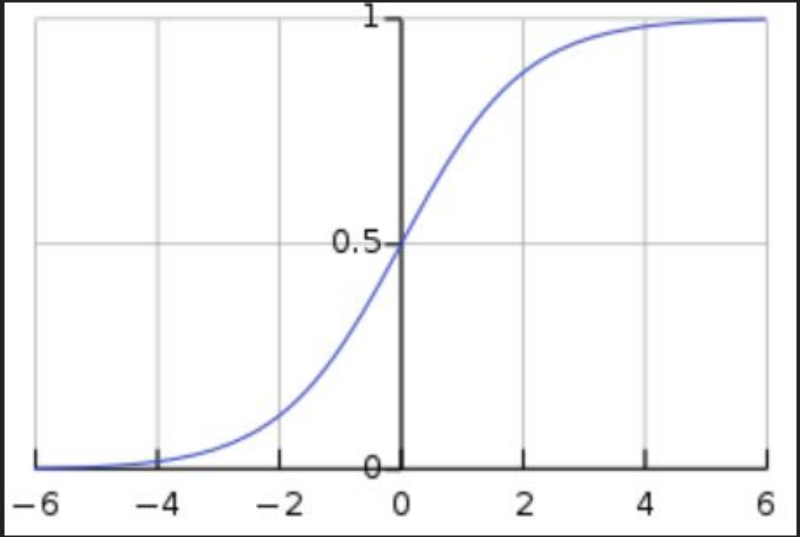
The network scheme



The function used in program

Sigmoid:

$$S(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1}.$$



The algorithm of neural network

The program uses Stochastic Gradient Descent to count the weights and biases. Before learning, program divides the examples into mini-batches to make learning quickier. The further algorithm can be divided into 5 main steps:

1. Input
2. Feedforward
3. Output error
4. Backpropagate the error
5. Output costs

1. Input

Set the corresponding activation a^1 for the input layer.

```
# Step 1.
activation = x
activations = [x] #list to store all the activations, layer by layer
zs = [] # list to store all the z vectors, layer by layer
```

2. Feedforward

For each l = 2,3,...,L compute the:

$$z^l = w^l a^{l-1} + b^l$$

$$a^l = \sigma(z^l)$$

```
# Step 2.
for b, w in zip(self.biases, self.weights):
    z = np.dot(w, activation)+b       #w0*a0 + w1*a1 + ... wn*an + b
    zs.append(z)                      #add z to the list
    activation = sigmoid(z)           #calculate the activation aj(L)
    activations.append(activation)    #add activation to list
```

3. Output error

Compute the vector:

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

```
# backward pass
# Step 3.
delta = self.cost_derivative(activations[-1], y) * sigmoid_prime(zs[-1])
```

4. Backpropagate the error

For each l = L - 1, L - 2,... compute the:

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

```
# Step 4.
for l in xrange(2, self.num_layers):
    z = zs[-l]
    sp = sigmoid_prime(z)
    delta = np.dot(self.weights[-l+1].transpose(), delta) * sp

    #Step 5.
    sum_bias[-l] = delta
    sum_weight[-l] = np.dot(delta, activations[-l-1].transpose())
```

5. Output costs

Count the cost function with:

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j$$

```
# Step 4.
for l in xrange(2, self.num_layers):
    z = zs[-l]
    sp = sigmoid_prime(z)
    delta = np.dot(self.weights[-l+1].transpose(), delta) * sp

    #Step 5.
    sum_bias[-l] = delta
    sum_weight[-l] = np.dot(delta, activations[-l-1].transpose())
```

The results can be seen in the Project report

Thank you for your attention.