Biologically Inspired Artificial Intelligence

Project report

Michał Zięba,
SSI,
GKiO2

1. Introduction

The goal of the project was to write a neural network program that will be able to recognize handwritten letters and digits.

2. Project documentation

2.1. Main assumptions

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

2.2. Dataset

The program uses the EMNIST database that consists 140.000 handwritten images of letters. The size of each image is 28x28 pixels.

2.3. The program

The neural network uses the 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 a1 for the input layer.

```
99      # Step 1.
100     activation = x
101     activations = [x] #list to store all the activations, layer by layer
102     zs = [] # list to store all the z vectors, layer by layer
103
```

2. Feedforward

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

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

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

```
104         # Step 2.
105         for b, w in zip(self.biases, self.weights):
106             z = np.dot(w, activation)+b     #w0*a0 + w1*a1 + ... wn*an + b
107             zs.append(z)                    #add z to the list
108             activation = sigmoid(z)         #calculate the activation aj(L)
109             activations.append(activation)  #add activation to list
110
```

3. Output the error

   Compute the vector:

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

```
111         # backward pass
112         # Step 3.
113         delta = self.cost_derivative(activations[-1], y) * sigmoid_prime(zs[-1])
114
```

4. Backpropagate the error

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

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

```
125         # Step 4.
126         for l in xrange(2, self.num_layers):
127             z = zs[-l]
128             sp = sigmoid_prime(z)
129             delta = np.dot(self.weights[-l+1].transpose(), delta) * sp
130
131             #Step 5.
132             sum_bias[-l] = delta
133             sum_weight[-l] = np.dot(delta, activations[-l-1].transpose())
```

5. Output costs:

   Count the cost function with:

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

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

3. Digits results

For the first try I used only digits from the EMNIST database. So the network had 784 inputs(one input for each pixel) and 10 outputs. First results were very impressive. For the first try I used the following setup.

Inputs	Hidden_1	Outputs	Epochs	Learning_rate
784	100	10	30	3.0

The program was learning for the 30 epochs and gave us the result of ~94%.

4. Letters results

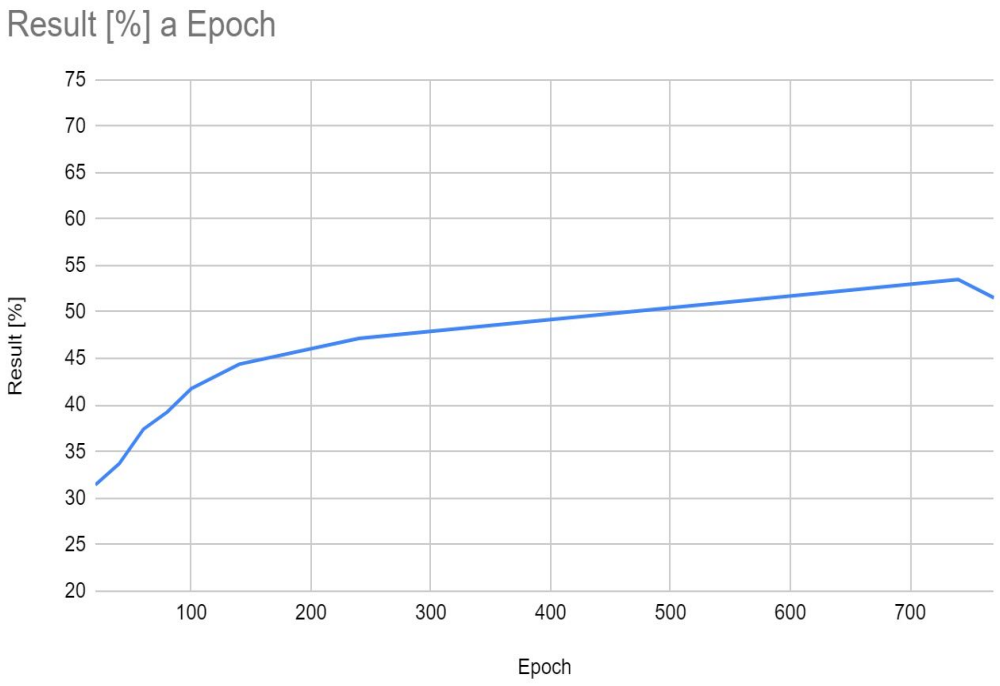
4.1. One hidden layer

Then I moved to 26 outputs. After first learning session with 20 epochs I received only 31% of positive results. The further learning sessions are shown below in the table.

Each line represents one learning session. 'First_result' and 'Last_result' means what was the result at the beginning and the end of the learning session.

	Hidden_neurons	Epochs	Batch_size	Learning_rate	First_result [%]	Last_result [%]	Increase/ Decrease
1	80	20	30	1	9	31,4277	22,4277
2	80	20	30	1	31,4277	33,7117	2,284
3	80	20	30	1	33,7117	37,3944	3,6827
4	80	20	40	1	37,3944	39,2594	1,865
5	80	20	80	1	39,2594	41,773	2,5136
6	80	40	200	1	41,773	44,3881	2,6151
7	80	100	200	1	44,3881	47,1653	2,7772
8	80	500	200	1	47,1653	53,5036	6,3383
9	80	30	200	3	53,5036	51,5305	-1,9731

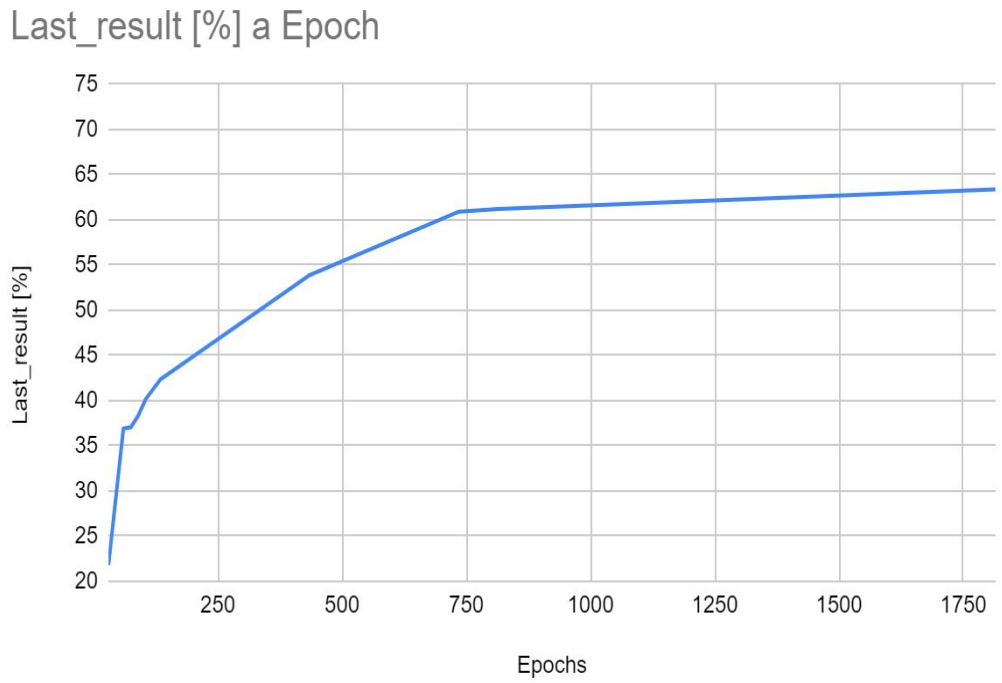
There is diagram below that shows the process of the learning. As we can see the upward trend stop around the 750 epoch and it starts to fall down.



In the next table there are results of the 120 neurons in the hidden layer.

	H_neurons	Epochs	Batch_size	Learning_rate	First_result [%]	Last_result [%]	Increase/ Decrease
1	120	30	130	3	9	21,8055	12,8055
2	120	30	130	3	21,8055	36,9011	15,0956
3	120	15	32	0,001	36,9011	37,0633	0,1622
4	120	15	32	0,05	37,0633	38,3674	1,3041
5	120	15	32	0,09	38,3674	40,1783	1,8109
6	120	30	32	0,09	40,1783	42,3677	2,1894
7	120	300	32	0,095	42,3677	53,8887	11,521
8	120	300	32	0,095	53,8887	60,9027	7,014
9	120	80	32	0,095	60,9027	61,2068	0,3041
10	120	1000	32	0,1	61,2068	63,3826	2,1758

As it is shown below, we get better results - around 60%. The upward trend is kept for the longer time, but the learning process became really slow so I decided to stop the test and try another setup.

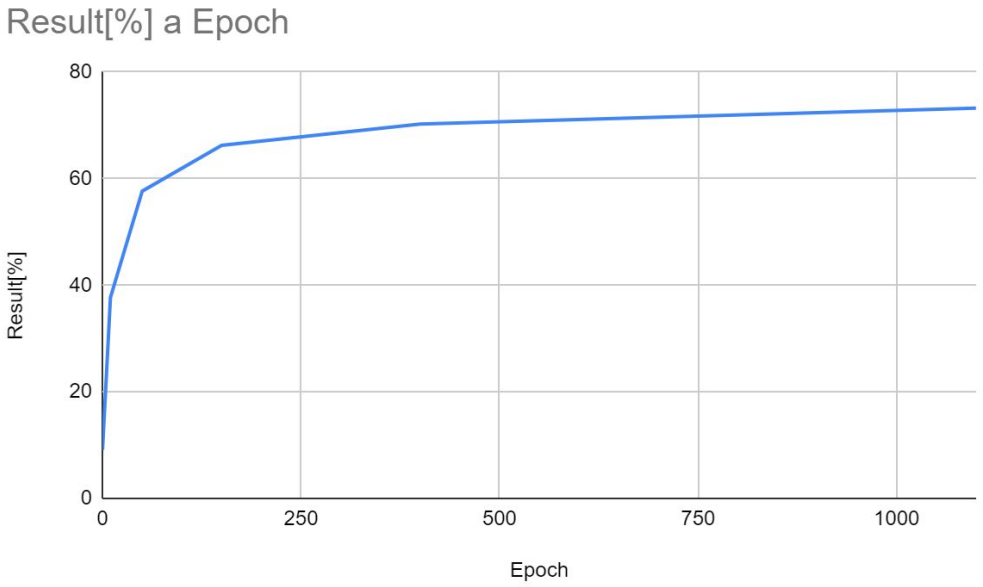


4.2. Two hidden layers

After that I tested the two hidden layers setup. The table consists the setup that I used, number of epochs, batch size and the best result that I managed to get.

	Hidden_neurons	Epochs sum	Batch_size	Learning_rate	Max_result [%]
1	50-35	1100	30	0,3	73,2374
2	60-30	1100	30	0,3	72,8098
3	30-60	1100	30	0,3	70,3899

The chart below represents the 50-35 hidden neurons setup. It stopped learning at 1000 epoch.

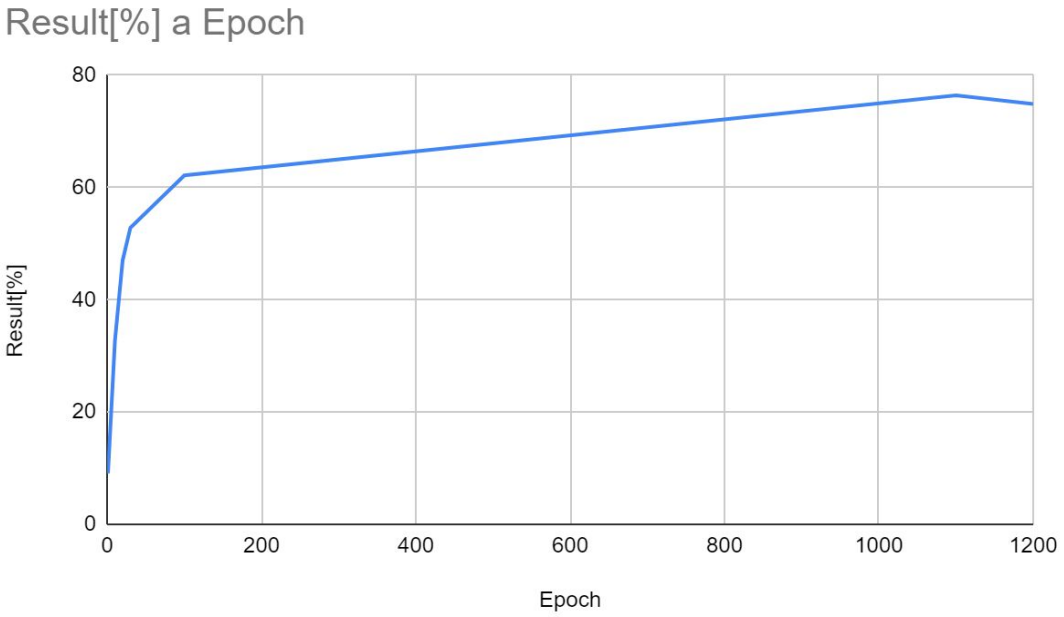


4.3. Three hidden layers

There is table that sums up my results of three hidden layers setups.

	Hidden_neurons	Epochs sum	Batch_size	Learning_rate	Max_result [%]
1	40-30-30	1100	30	0,3	71,3356
2	40-40-40	1100	30	0,3	70,8609
3	50-35-32	1200	30	0,3	74,8834
4	60-30-30	1300	30	0,3	73,0049
5	80-30-30	2800	30	0,3	71,1602

The chart below represents the 50-35-32 hidden neurons setup. As we can see, it stopped learning around 1100 epoch.



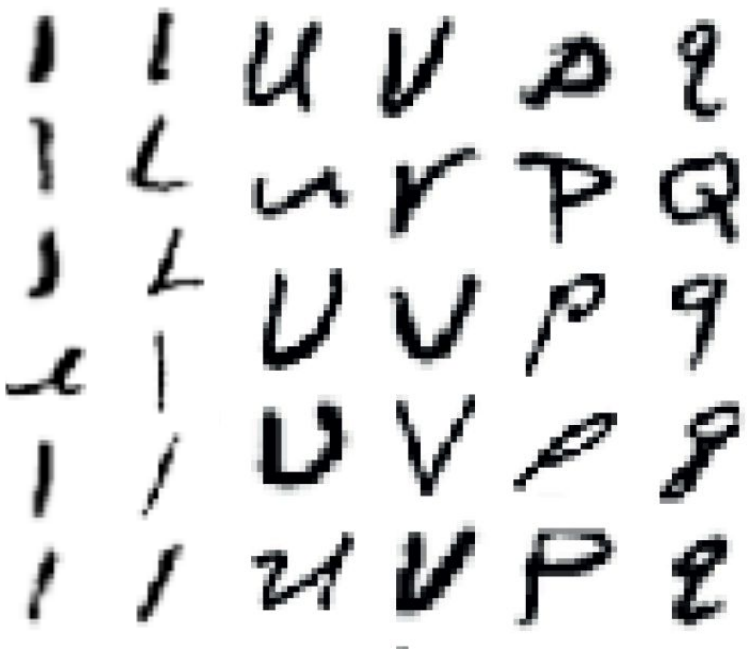
5. Conclusions

It was the first time that I was writing and learning about neural networks. I managed to get the ~90% results for the numbers (10 outputs) quite fast, but the real problem for me was to get the same result with letters (26 outputs). I couldn't get the right hidden neurons setup to achieve the 80-90% point.

The NN used to stop learning at 70-74% and became to oscillate close to these variables. The results with two and three hidden layers we similar - around 73%. The main difference was that the NN with two hidden layers was learning much faster. I also tried using 4 hidden layers, but I got even worse results than above and I quicked this idea after 100 epochs.

There are two capabilities that may cause that problem. The first is the number of hidden neurons. I was not able to find the right number that would gave the better results.

The second is the database that was used for the learning. It turned out, the NN had problems with specific letters while learning. When we look closer at the images we see that they are actually very similar.



The most mistakes had the above letters: ‘i’, ‘l’, ‘u’, ‘v’, ‘p’, ‘q’. When we compare the each pair we notice that some of them look the same and even the human might have problem to distinguish them.

The full source code with dataset and the presentation can be found at:
https://github.com/michalziebaa/biai_project