Image Recognition using Neural Networks

Aerospace Engineering

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

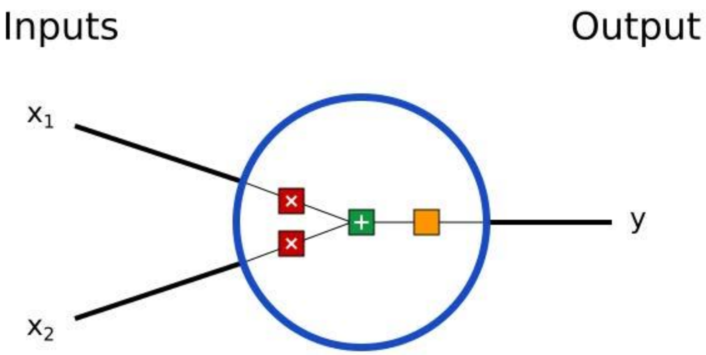
Authors:
Yi Qiang Ji Zhang

Professor/s:
Alex Ferrer Ferré

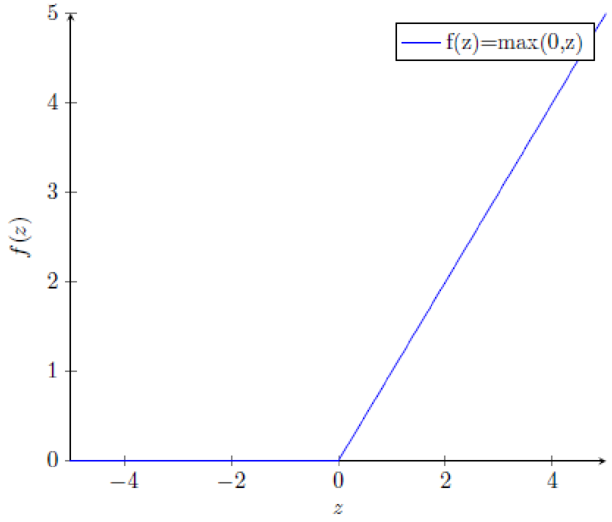
Table of contents

Introduction

- Neural Networks
- A neuron is the basic unit of the neural network. It takes inputs parameters, then does some computation with them, and produces one output.



Neuron scheme [1]



ReLU function

Each input is multiplied by a weight:

$$x_1 \longrightarrow \omega_1 \cdot x_1$$

$$x_2 \longrightarrow \omega_2 \cdot x_2$$

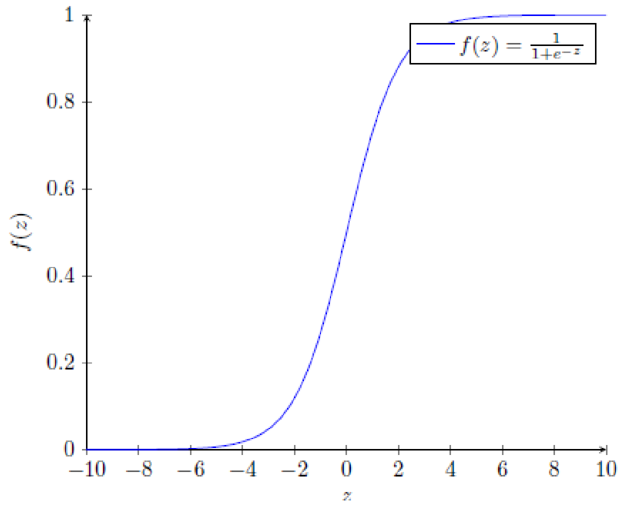
All the weight inputs are added with some bias:

$$(\omega_1 \cdot x_1) + (\omega_2 \cdot x_2) + b$$

And finally, the sum is passed with an activation function:

$$y = (\omega_1 \cdot x_1 + \omega_2 \cdot x_2 + b)$$

A commonly used activation function is the sigmoid function:

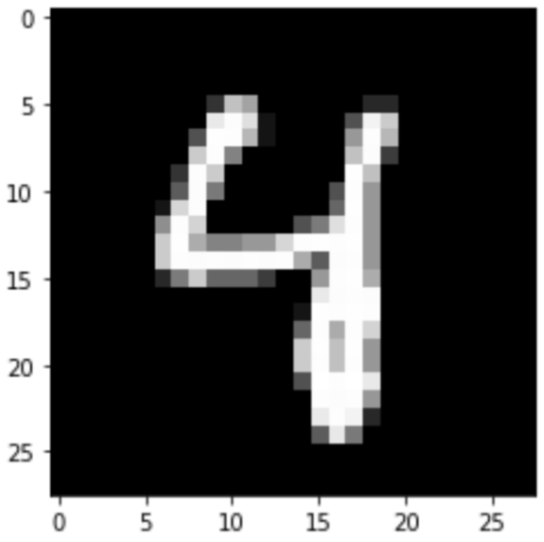


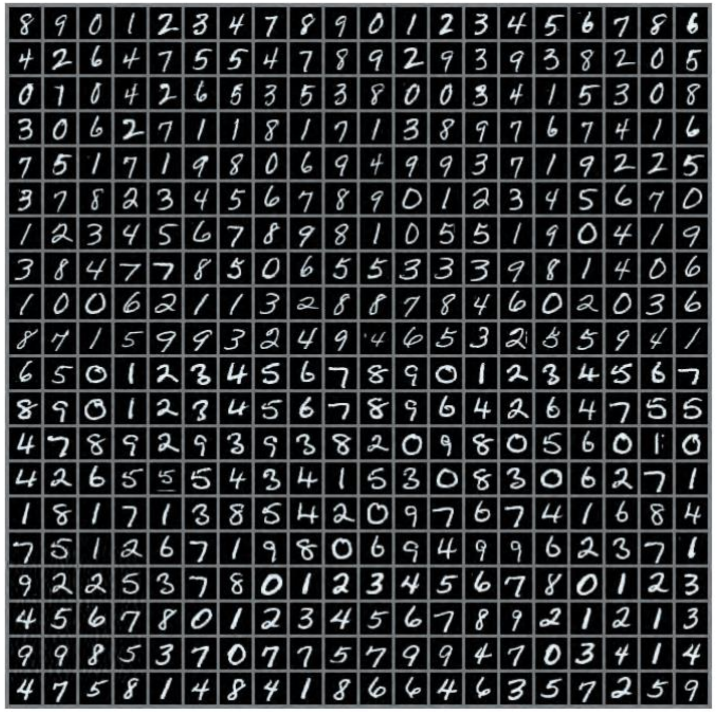
Sigmoid function

[1] https://www.researchgate.net/figure/A-Basic-sigmoid-function-with-two-parameters-c1-and-c2-as-commonly-used-for-subitizing_fig2_325868989

Problem statement

- The dataset we're using is the well-known MNIST handwritten digit dataset, which is frequently used in both ML and computer vision applications. It includes 28 × 28 grayscale photos of handwritten numerals that resemble as follows:





MNIST dataset [1] [2]

[1] https://www.researchgate.net/figure/A-Basic-sigmoid-function-with-two-parameters-c1-and-c2-as-commonly-used-for-subitizing_fig2_325868989
[2] https://www.deeplearningbook.org/

Methodology

Forward propagation

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$
$$A^{[1]} = g_{\text{ReLU}}(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$A^{[2]} = g_{\text{softmax}}(Z^{[2]})$$

Backward propagation

$$dZ^{[2]} = A^{[2]} - Y$$
$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$$
$$dB^{[2]} = \frac{1}{m} \Sigma dZ^{[2]}$$
$$dZ^{[1]} = W^{[2]T} dZ^{[2]} * g^{[1]\prime}(z^{[1]})$$
$$dW^{[1]} = \frac{1}{m} dZ^{[1]} A^{[0]T}$$
$$dB^{[1]} = \frac{1}{m} \Sigma dZ^{[1]}$$

Parameter updates

$$W^{[2]} := W^{[2]} - \alpha \, dW^{[2]}$$
$$b^{[2]} := b^{[2]} - \alpha \, db^{[2]}$$
$$W^{[1]} := W^{[1]} - \alpha \, dW^{[1]}$$
$$b^{[1]} := b^{[1]} - \alpha \, db^{[1]}$$

Backward propagation

Forward propagation

Variable	Dimensions	Observations
$A^{[0]} = X$	$784 \times m$	
$Z^{[1]} \sim A^{[1]}$	$10 \times m$	
$W^{[1]}$	10×784	as $W^{[1]}A^{[0]} \sim Z^{[1]}$
$B^{[1]}$	10×1	
$Z^{[2]} \sim A^{[2]}$	$10 \times m$	
$W^{[1]}$	10×10	$W^{[2]}A^{[1]} \sim Z^{[2]}$
$B^{[2]}$	10×1	

Backward propagation

Variable	Dimensions	Observations
$dZ^{[2]}$	$10 \times m$	$(A^{[2]})$
$dW^{[2]}$	10×10	
$dB^{[2]}$	10×1	
$dZ^{[1]}$	$10 \times m$	$A^{[1]}$
$dW^{[1]}$	10×10	
$dB^{[1]}$	10×1	

Methodology (cont.)

$$\begin{bmatrix} 0.01 \\ 0.02 \\ 0.05 \\ 0.02 \\ 0.80 \\ 0.01 \\ 0.01 \\ 0.00 \\ 0.01 \\ 0.07 \end{bmatrix} \xrightarrow[\text{One Hot encoder}]{} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Results

1 layer

3 Epoch

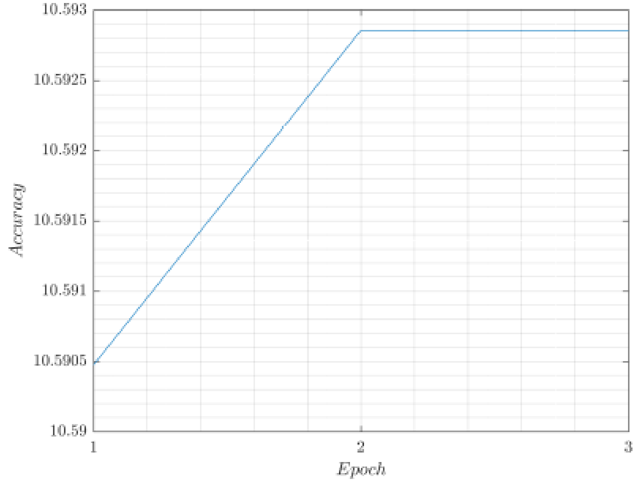


Figure 6. Accuracy for 10 nodes with 3 epoch using RELU. Source: Own.

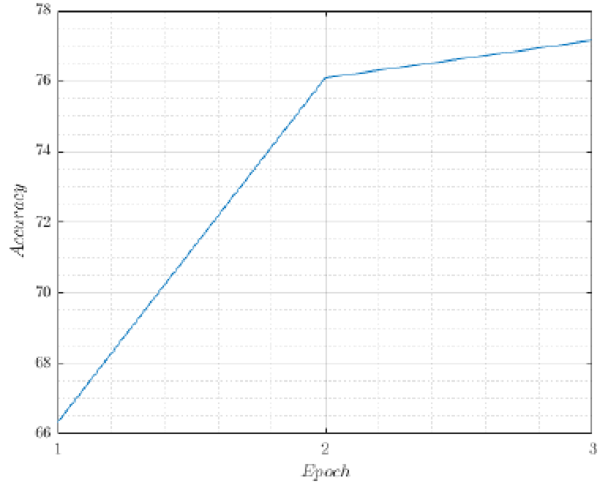


Figure 7. Accuracy for 10 nodes with 3 epoch using Sigmoid. Source: Own.

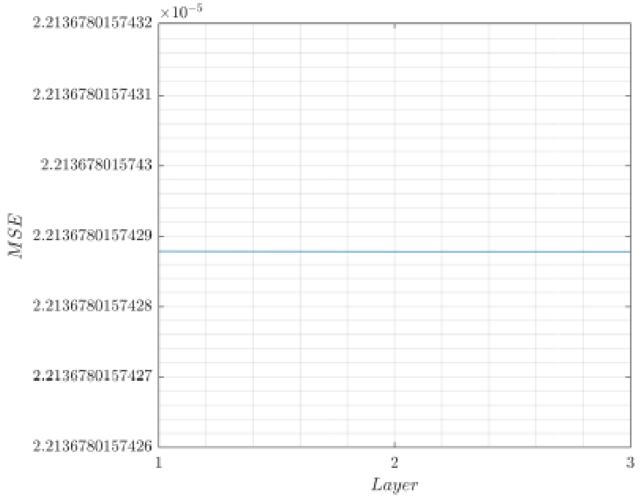


Figure 8. MSE for 10 nodes with 3 epoch using RELU. Source: Own.

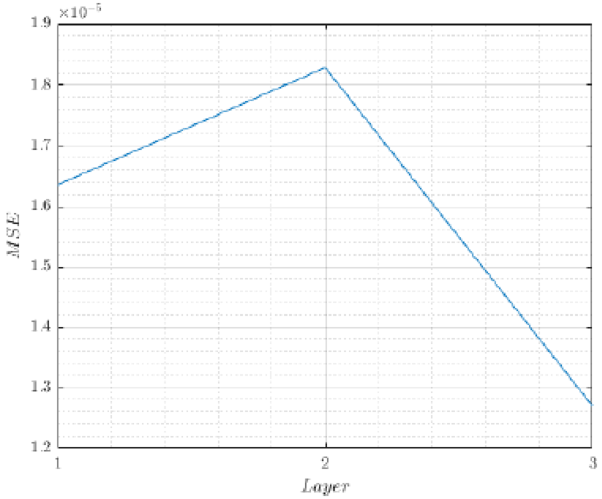


Figure 9. MSE for 10 nodes with 3 epoch using Sigmoid. Source: Own.

Results

1 layer

10 Epoch

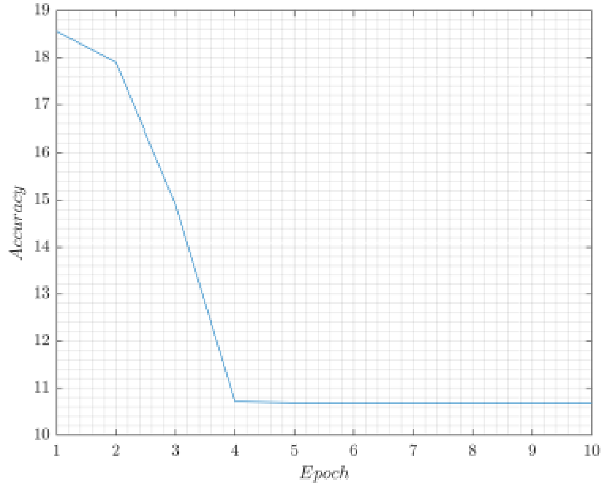


Figure 10. Accuracy for 10 nodes with 10 epoch using RELU. Source: Own.

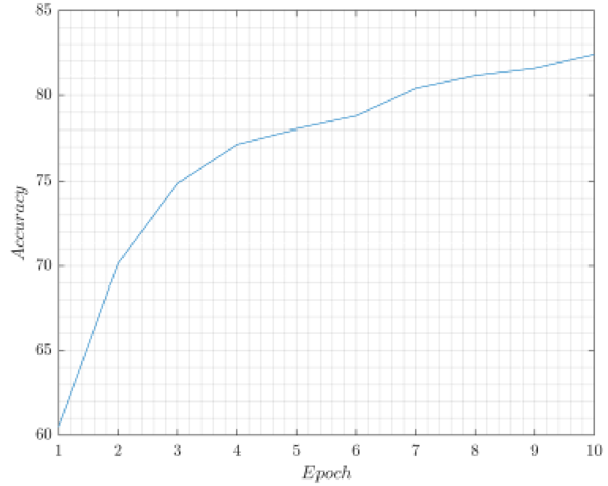


Figure 11. Accuracy for 10 nodes with 3 epoch using Sigmoid. Source: Own.

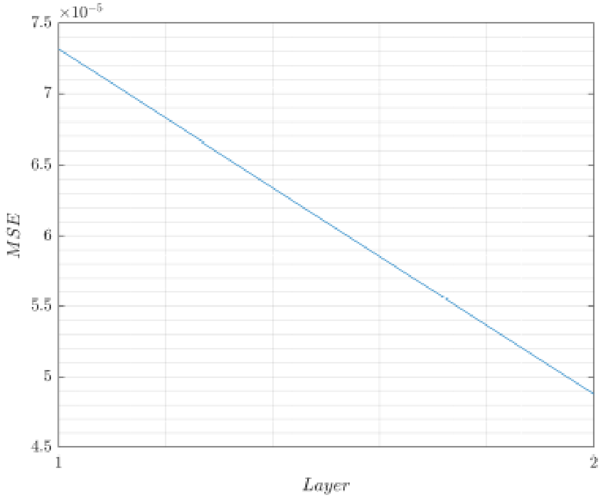


Figure 12. MSE for 10 nodes with 10 epoch using RELU. Source: Own.

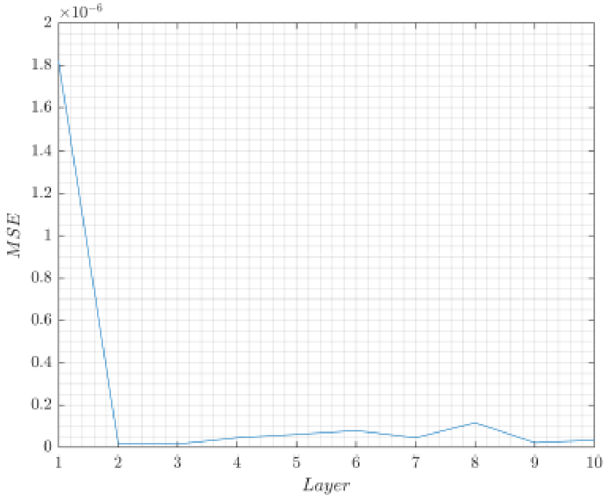


Figure 13. MSE for 10 nodes with 10 epoch using Sigmoid. Source: Own.

Results

1 layer

100 Epoch

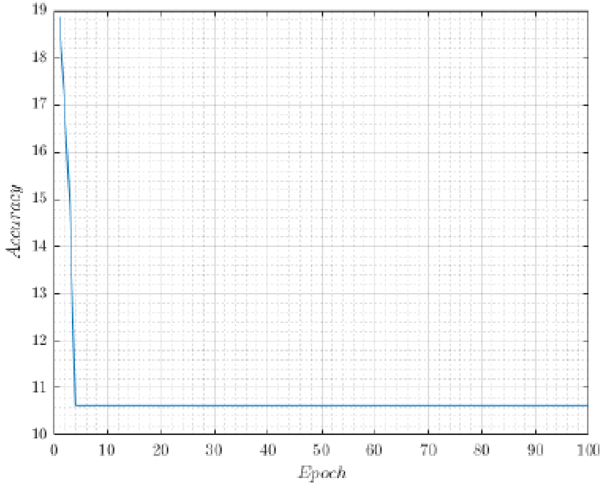


Figure 14. Accuracy for 10 nodes with 100 epoch using RELU. Source: Own.

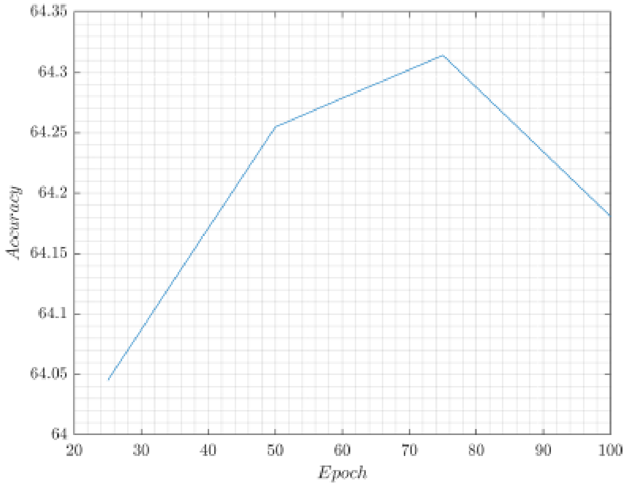


Figure 15. Accuracy for 10 nodes with 100 epoch using Sigmoid. Source: Own.

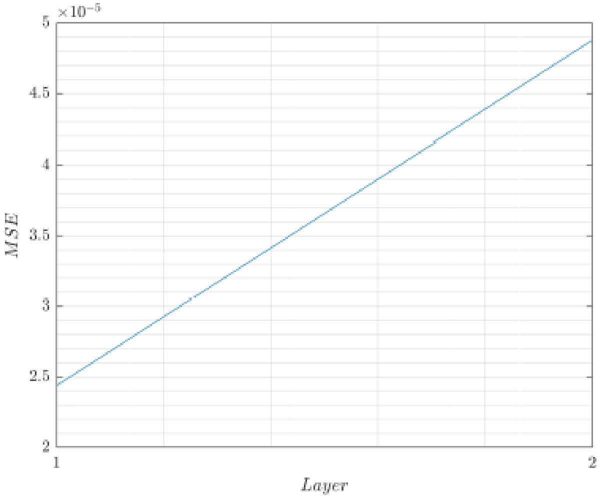


Figure 16. MSE for 10 nodes with 100 epoch using RELU. Source: Own.

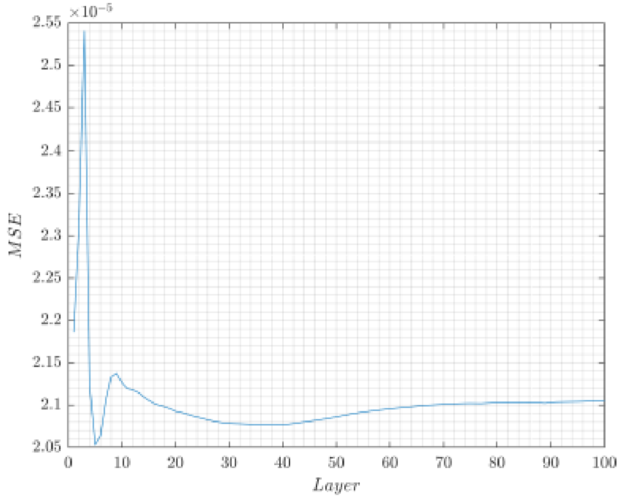


Figure 17. MSE for 10 nodes with 100 epoch using Sigmoid. Source: Own.

Results

2 layers

3 Epoch

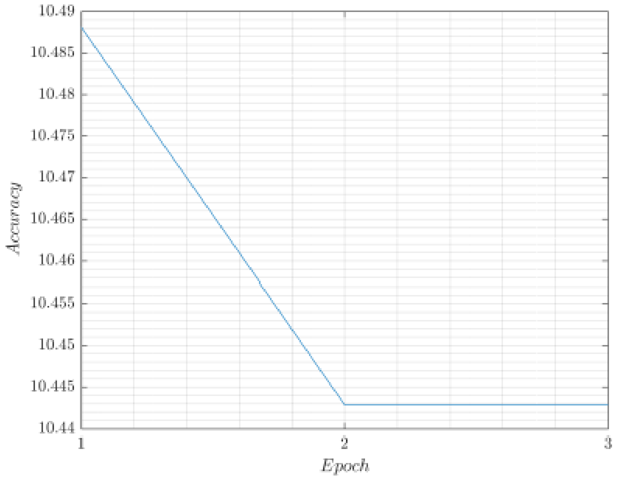


Figure 18. Accuracy for 10 nodes with 3 epoch using RELU and learning rate $\alpha = 0.01$ (2 layers). Source: Own.

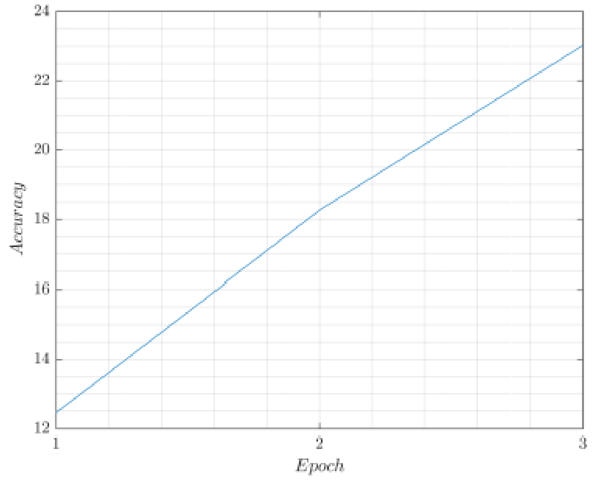


Figure 19. Accuracy for 10 nodes with 3 epoch using Sigmoid and learning rate $\alpha = 0.01$ (2 layers). Source: Own.

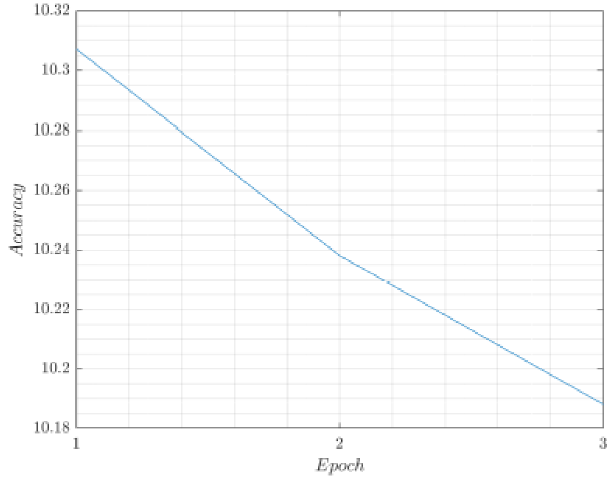


Figure 20. Accuracy for 10 nodes with 3 epoch using ReLU (2 layers) with learning rate $\alpha = 0.1$ (2 layers). Source: Own.

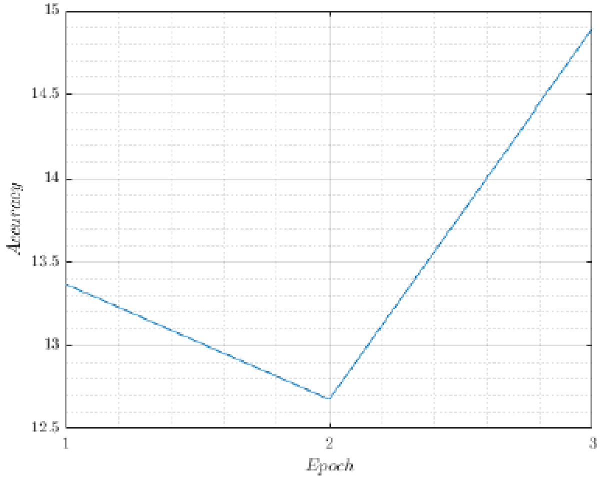


Figure 21. Accuracy for 10 nodes with 3 epoch using Sigmoid (2 layers) with learning rate $\alpha = 0.1$ (2 layers). Source: Own.

Results

2 layers

3 Epoch

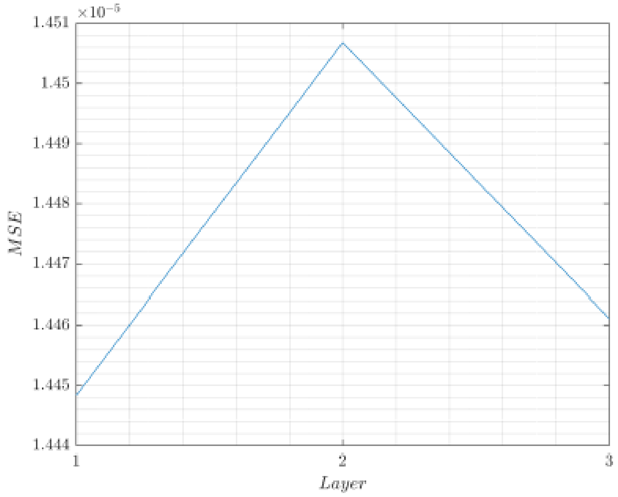


Figure 22. MSE for 10 nodes with 3 epoch using ReLU with learning rate *alpha* = 0.01 (2 layers). Source: Own.

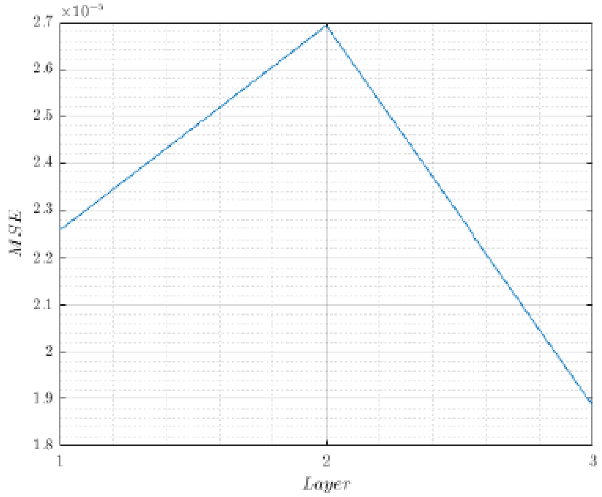


Figure 23. MSE for 10 nodes with 3 epoch using Sigmoid with learning rate *alpha* = 0.01 (2 layers). Source: Own.

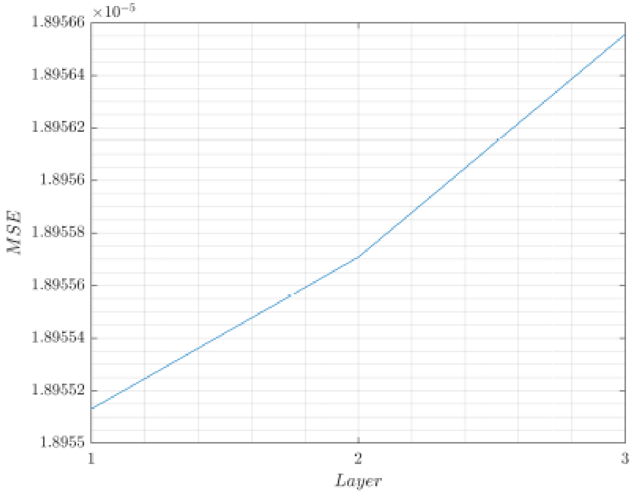


Figure 24. MSE for 10 nodes with 3 epoch using ReLU with learning rate *alpha* = 0.1 (2 layers). Source: Own.

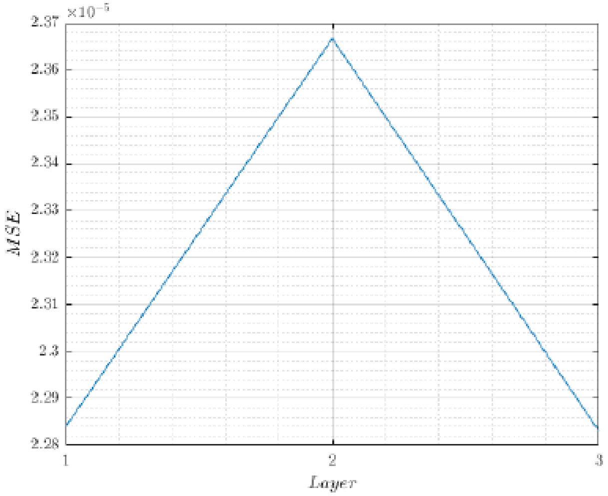


Figure 25. MSE for 10 nodes with 3 epoch using Sigmoid with learning rate $\alpha = 0.1$ (2 layers). Source: Own.

Results

2 layers

10 Epoch

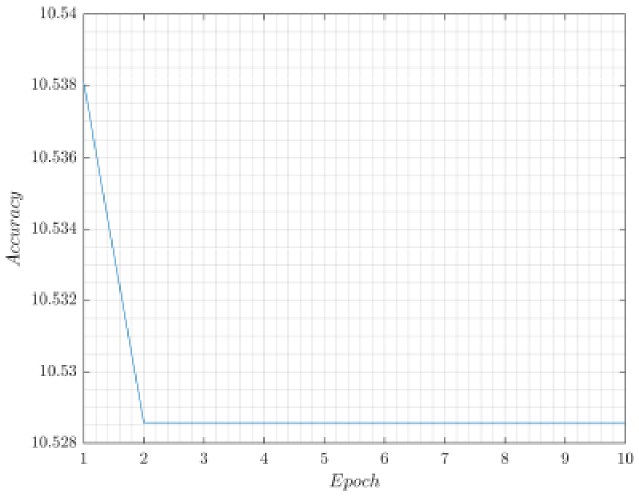


Figure 26. Accuracy for 10 nodes with 10 epoch using RELU (2 layers). Source: Own.

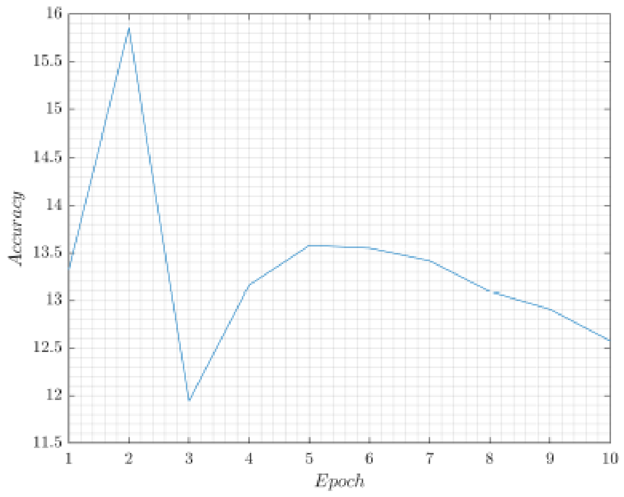


Figure 27. Accuracy for 10 nodes with 10 epoch using Sigmoid (2 layers). Source: Own.

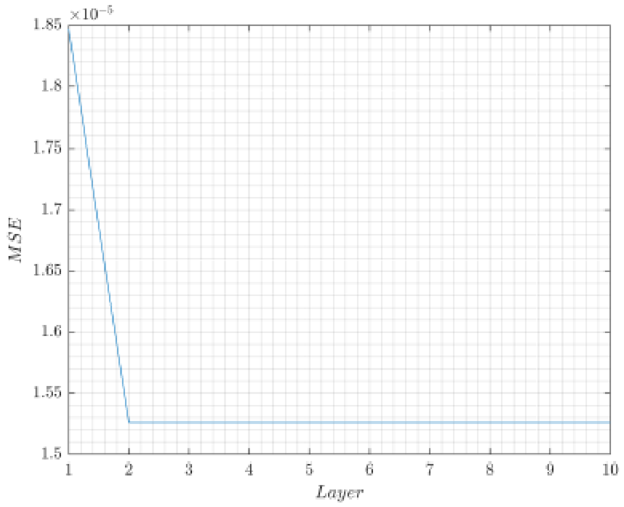


Figure 28. MSE for 10 nodes with 10 epoch using ReLU with learning rate $\alpha = 0.1$ (2 layers). Source: Own.

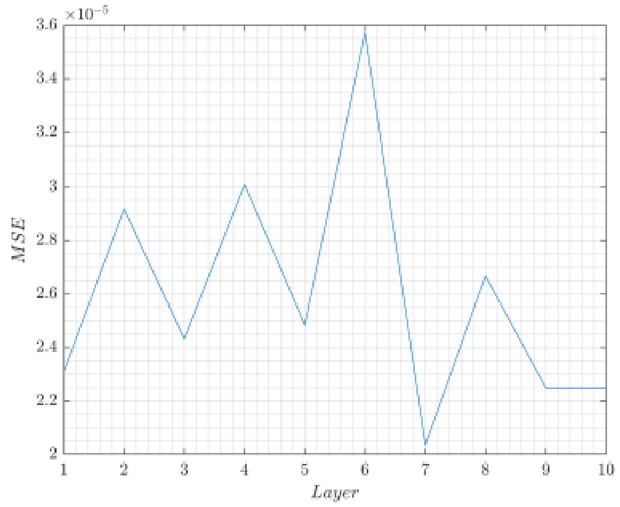


Figure 29. MSE for 10 nodes with 10 epoch using Sigmoid with learning rate $\alpha = 0.1$ (2 layers). Source: Own.

Results

2 layers

100 Epoch

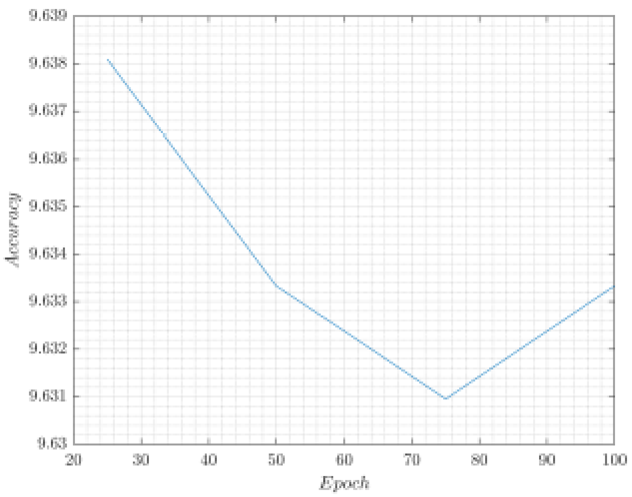


Figure 30. Accuracy for 10 nodes with 100 epoch using RELU (2 layers) $\alpha = 0.1$. Source: Own.

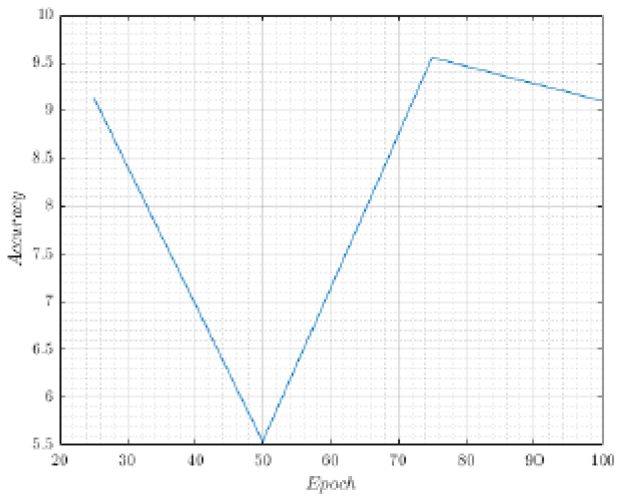


Figure 31. Accuracy for 10 nodes with 100 epoch using Sigmoid (2 layers) $\alpha = 0.1$. Source: Own.

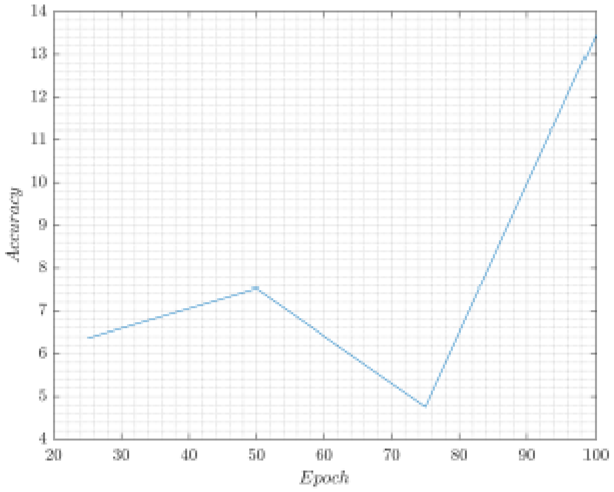


Figure 32. Accuracy for 10 nodes with 100 epoch using Sigmoid with learning rate $\alpha = 0.01$ (2 layers). Source: Own.

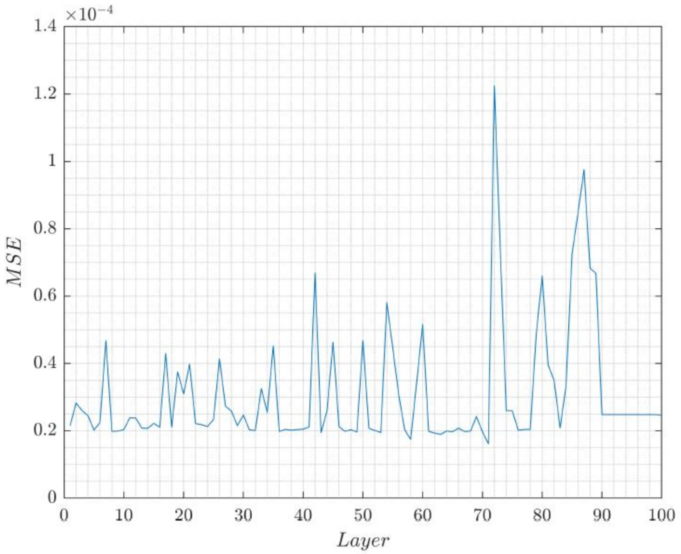
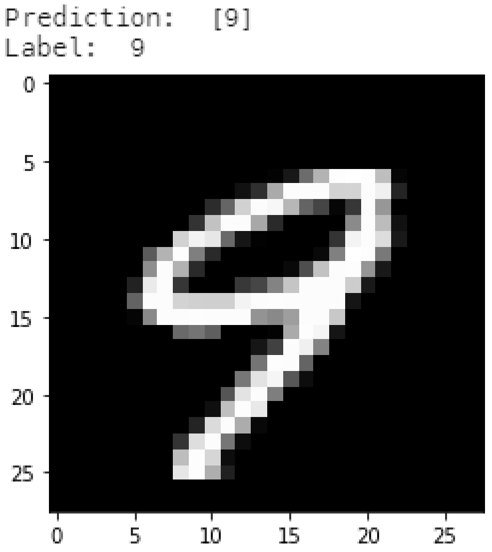
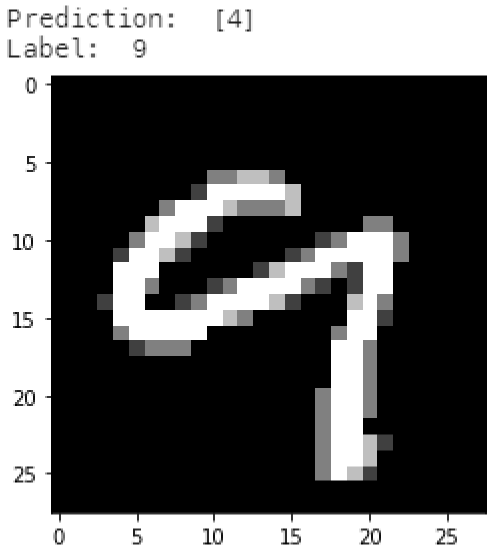
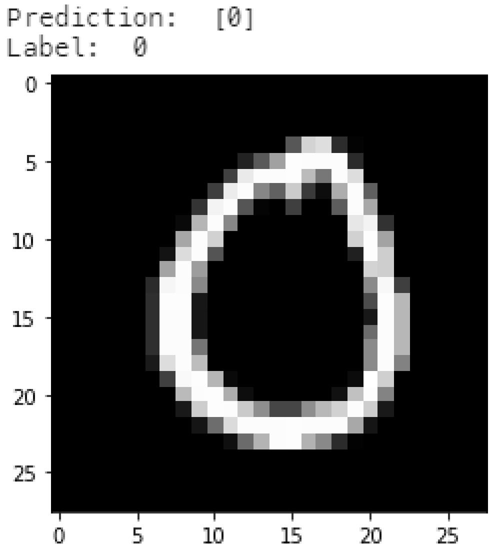


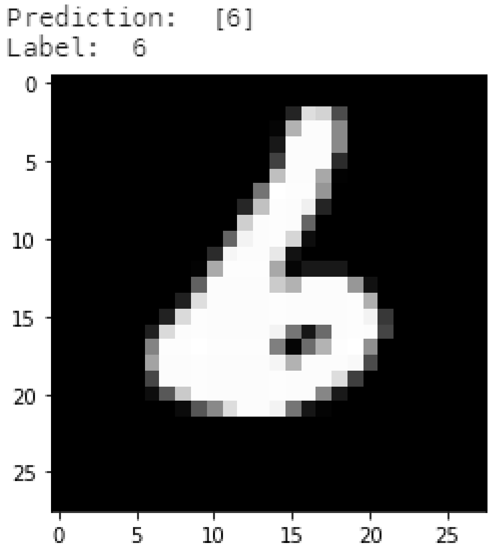
Figure 33. MSE for 10 nodes with 100 epoch using Sigmoid with $\alpha = 0.01$ (2 layers). Source: Own.

Testing the algorithm image







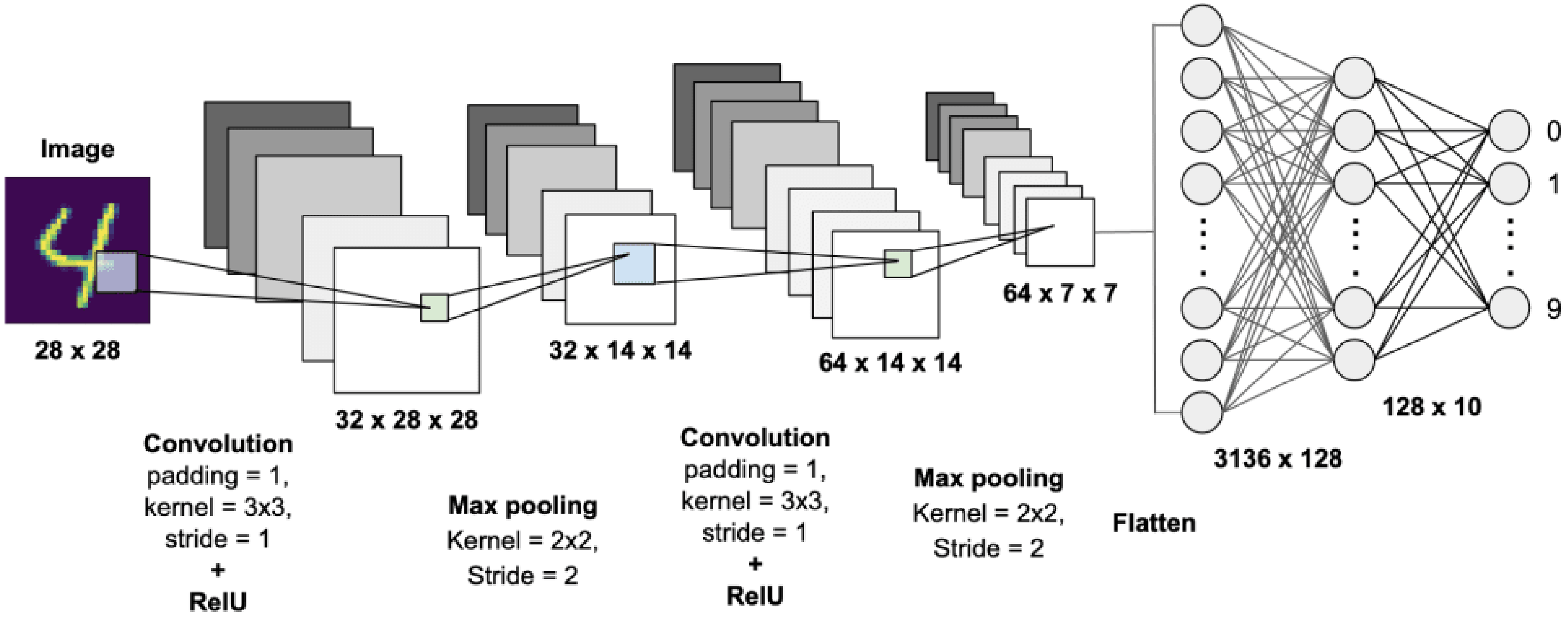


Accuracy achieved: 88.1% with 1000 iterations
(single layer)

[1] Lauterborn W.(2003) Chaotic Systems. https://www.sciencedirect.com/topics/computer-science/chaotic-systems

Further improvements and questions

- How to implement a Convolutional Neural Network?



Convolutional Neural Network [1]

[1] https://dev.to/afrozchakure/cnn-in-a-brief-27gg

Further improvements and questions (cont.)

- Why ReLU activation function does not increase accuracy?

- Potential explanation: when using an input in [0, 255], then when doing the weighted sum for the layer L: $z=a(L-1)w(L)+b(L)$, the value z will often be big too. If z is often big (or even if it's often > 0), let's say around 100, than ReLU(z)=z, and we totally lose the "non-linear" aspect of this activation function! Said in another way: if the input is in [0, 255], then z is often far from 0, and we totally avoid the place where "interesting non-linear things" are going on (around 0 the ReLU function is non linear and looks like __/)... Now when the input is in [0,1], then the weighted sum z can often be close to 0: maybe it sometimes goes below 0 (since the weights are randomly-initialized on [-1, 1], it's possible!), sometimes higher than 0, etc. Then more neuron activation/deactivation is happening... This could be a potential explanation of why it works better with input in [0, 1].

[1] https://datascience.stackexchange.com/questions/18667/relu-vs-sigmoid-in-mnist-example