# MNIST dataset-NN

ByeongKyu Park (byeonggyu.park)

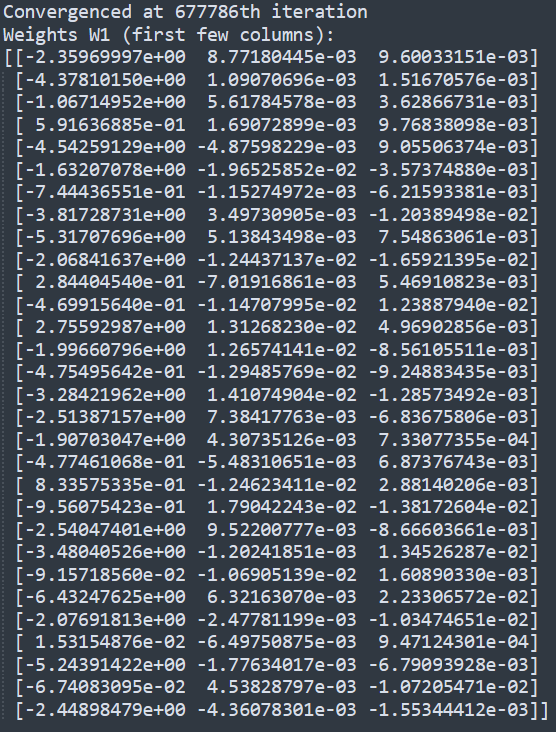
## The programming language used

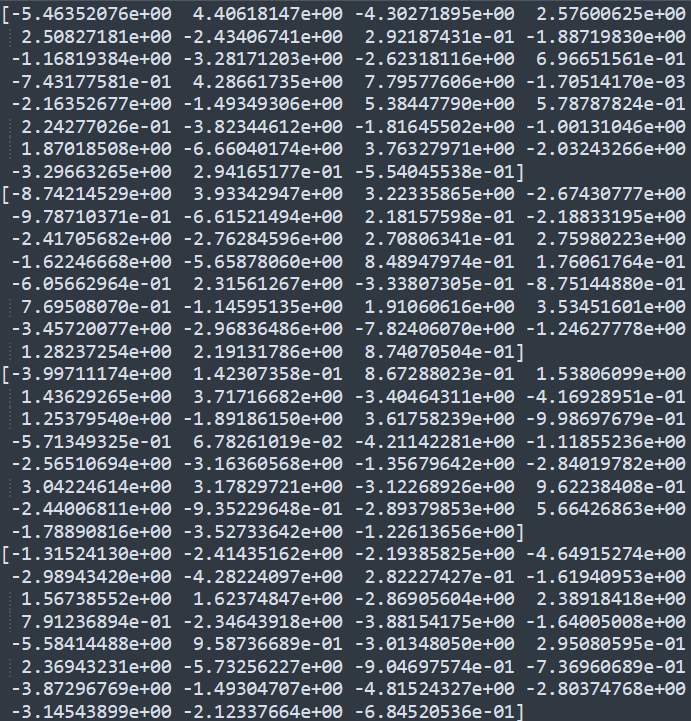
Python

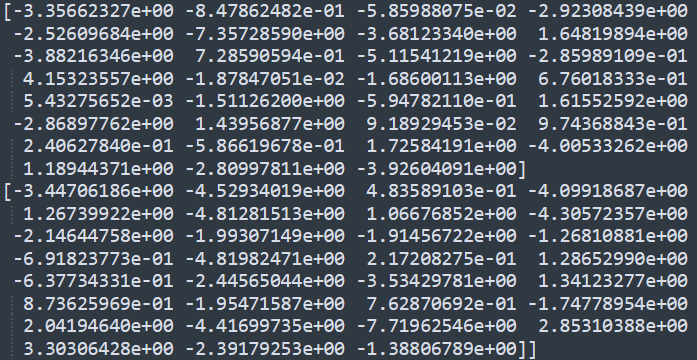
On which values of η lead to a reasonable performance in the gradient descent algorithm.

Approximately 0.1

## Resulting weights W (1) (the first three columns only), W (2) when s2 = 30









(e) Do more units in the hidden layer lead to better accuracy?   
: Increasing the number of units in the hidden layer does enhance the accuracy up to a certain point. Beyond that threshold, however, the accuracy remained relatively constant.

S2=30



S2=60



S2=100



S2=150



S2=200



S2=300



(f) Include any additional information

\* For the project, I employed Stochastic Gradient Descent (fixed ***eta = 0.1***) and set the training to terminate when the gradient's magnitude falls below a threshold of ***1e-12***.

\* The accuracy is calculated as the ratio of correct predictions (with activation ≥ 0.5) to the total number of samples in the test set. Since x\_test.shape[1] includes all test samples regardless of the model's confidence level, this denominator is typically larger than the number of valid (>=0.5) predictions.

\* Used keras.datasets (<https://keras.io/api/datasets/mnist/>)

## (g - extra)

In my project, I adjusted the neural network's hidden layer sizes at line 98 of the code. Initially, the configuration **hidden\_layer\_sizes = [128]** resulted in an accuracy of 97.04%. By modifying this to **hidden\_layer\_sizes = [128, 64]**, the network's accuracy slightly improved to 97.91%. However, further increasing the complexity with **hidden\_layer\_sizes = [128, 64, 32]** significantly extended the training time to over 10 hours. This demonstrates that while tweaking the hidden layers might slightly enhance performance, it also increases computational demands.

***hidden\_layer\_sizes = [128, 64]***

