**Report**

Assignment 1, Deep Learning

by Carmel Schablin and Gregory Koushnir

## The task

Use forward deep neural network to classify handwritten digits from MNIST dataset

## The dataset

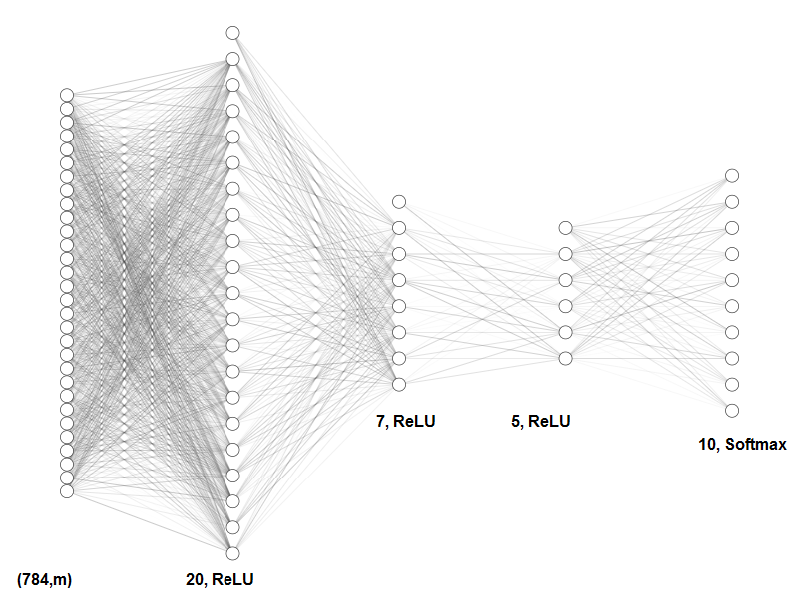
The MNIST dataset is used loaded with the Keras library

keras.datasets.mnist.load\_data()

This dataset contains train and test data including gray-scaled 28x28 images of handwritten digits with their labeled values. The original spread is 60k samples for training and 10k samples for test. For the purpose of our research, the train data is split in a ratio of 80% for a train set and 20% for the validation set. The final split is: 48k train set, 12k validation set, 10k test set.

## The method

The following architecture of the network is used to for the classification task:



The training is done on the flattened image vectors (with length of 784) in batches and afterwards passes through fully connected hidden layers with 20, 7 and 5 units respectively using ReLU activation. The last layer has 10 units, as the number of digit classes and applies Softmax activation.

Before the data is passed to the network, it is transformed in a following way:

* the images are flattened to 1-d array and their values are normalized by dividing by maximal value, so that the values range will be [0,1]
* each label is transformed to a vector with 1 at input-index and zeros at rest, e.g label “3” becomes a vector [0., 0., 0., 1., 0., 0., 0., 0., 0., 0.]

The network is initiated with random weights using normal distribution in a range of [-1,1] and biases are set to 0.

weight\_matrix = np.random.randn(weight\_layer\_shape[0], weight\_layer\_shape[1])\*np.sqrt(2/weight\_layer\_shape[1])

The network is trained using the following parameters:

params, costs = NeuralNet.L\_layer\_model(x\_train.T, y\_train.T, layers\_dims, num\_iterations=1000, batch\_size=32, learning\_rate=0.009)

For each batch, to complete the forward and backward propagation the following steps are taken:

1. the prediction is made with linear\_model\_forward

y\_pred, caches = linear\_model\_forward(x\_sub, params, use\_batchnorm=use\_batch\_norm)

1. the batch loss is computed with compute\_cost

batch\_loss = compute\_cost(y\_pred, y\_sub)

1. gradients are calculated

grads = linear\_model\_backward(y\_pred, y\_sub, caches)

1. networks parameters are being updated

params = update\_parameters(params, grads, learning\_rate)

To check the convergence of the training process, every iteration step starting from the second epoch the accuracy value of the validation set is calculated and compared to the accuracy from the previous 100 steps. It seems to us reasonable to let the network train on the first epoch without interrupting it to make the predictions on the validation set. If there is no improvement at least by , the training process stops.

## The results

The network was trained twice: without and with batchnorm

|  |  |  |
| --- | --- | --- |
|  | w/o batchnorm | With batchnorm |
| Batch size |  |  |
| Number of iterations |  |  |
| Number of epochs |  |  |
| Final accuracy – train set |  |  |
| Final accuracy – validation set |  |  |
| Final accuracy – test set |  |  |

Costs vectors: