**Report**

Assignment 1, Deep Learning

by Carmel Shablin 305812661

Gregory Koushnir

## Task

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

## Dataset

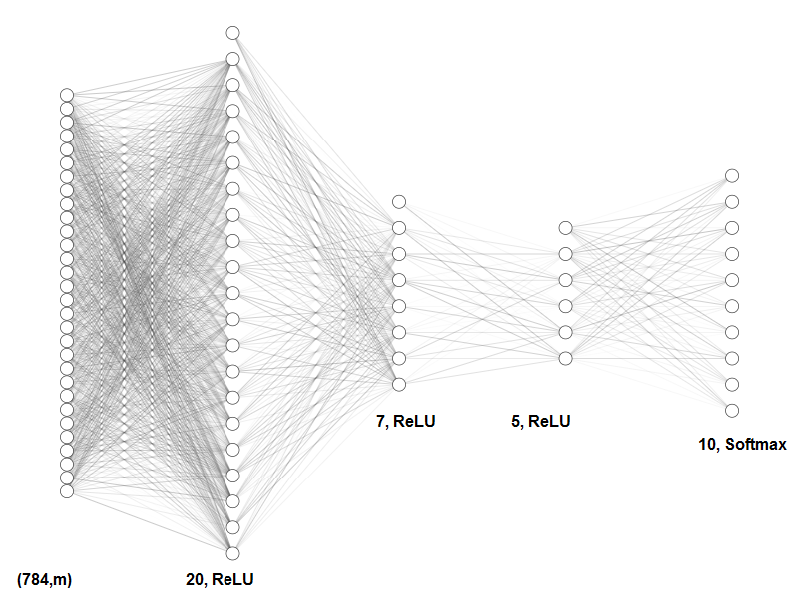
The MNIST dataset is used loaded with the Keras library

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.

## 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 (both normal and uniform distributions were implemented) 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=100000, batch\_size=32, learning\_rate=0.009)

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

1. calculate predictions (forward propagation)

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

by calculating , when is for all the hidden layers, except the last one, which is , we choose log(C) to be the minus maximal value of the input vector, so the exponent will not explode and will remain stable

1. compute batch loss

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

, m – is a number of samples

1. calculate gradients

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

by calculating . ,

1. update networks parameters

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

The stopping criterion- It was noticed that running a predict function on the validation step is very costly, and the stopping criterion of 100 training steps without improvement does not allow the network to converge to the possible values. We decided to change the condition. To check the convergence of the training process, we measure in every **100’th** iteration step starting from the second epoch the accuracy value of the validation set (using the predict function, which calculates the ratio between the correct guesses and the total number of samples) and to compare it to the accuracy from the previous **50** measurements. That means, that we will allow the network to continue the train process on **5000** more batches before the decision that the accuracy can not be improved. Time-wise it is a quicker solution than the one that was advised in the assignment. Also, 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 and the best parameters are returned.

If the training does not converge in a desirable time we use another limiting criterion - the maximal number of iterations.

Some of the implemented functions have a “test\_mode” parameters, which allows to ignore unnecessary/unwanted parts when not in train mode.

As a bonus, drop-out normalization was implemented. During the dropout test, batch normalization was disabled.

* dropout\_forward(A, test\_mode) – is applied before the ReLU activation function. Randomly generated mask is applied on the activation matrix with the *globals.dropout\_keep\_probability* probability to keep each value unmasked. Since we don’t use Dropout in test time, then the expected output of the layer is x (activations) so we make the expectation of layer output to be x instead of p\*x (p\*activations). The outputs are transformed activations and activation cache, which saves the mask.
* dropout\_backwards(dz, dropout\_cache) – is applied after the *\_\_relu\_backward* and is calculated by , where keep\_prob = *globals.dropout\_keep\_probability* and the *da* is masked by the same mask that was saved in *dropout\_cache* by *dropout\_forward*.

## Results

The network was trained twice: without and with batchnorm. The batch normalization was done after ReLU activations, using those formulas:

- activation of i’th sample of the layer, ,

|  |  |  |  |
| --- | --- | --- | --- |
|  | w/o batchnorm | With batchnorm | Dropout (0.1), w/o BN |
| Batch size | 32 | 64 | 32 |
| Number of iterations | 31400 | 79400 | 29700 |
| Number of epochs | 20 | 105 | 19 |
| Learning rate | 0.009 | 0.05 | 0.09 |
| Final accuracy – train set | 0.9525 | 0.9475 | 0.9261 |
| Final accuracy – validation set | 0.9333 | 0.9359 | 0.913 |
| Final accuracy – test set | 0.9363 | 0.938 | 0.9162 |
| Total runtime [sec][[1]](#footnote-1) | 157 | 501 | 152 |

Comparison of the tests:

The final scores are in range of 91.3-95.2%. The best accuracies reached with neither batch normalization nor dropout.

We see that the network does not overfit: train, test and validation set accuracies are very similar.

When looking at costs, multiple curves of costs for each test-case might be seen. This happens because of the cost measurement falls in a different place of epoch.

The convergence when the batchnorm is applied takes considerably more iterations and time (almost thrice), although the learning rate was increased to 0.05. Both regular and dropout runs takes the same amount of time and almost the same amount of iterations.

Costs of the dropout compared to regular are twice as high and much noisier due to the randomness of the architecture, and the accuracies are about 2% lower on every set.

For a full table of cost values please see an EXCEL file in the same directory.

1. Intel Core i5 3317U CPU @ 1.7 GHz, 6 Gb RAM, Intel HD Graphics 4000 [↑](#footnote-ref-1)