Lab 2: Neural Networks: Implementing a Neural Network from scratch

Iuliia Alekseenko, Zahid Hassan Tushar

I. INTRODUCTION

Artificial neural networks (ANNs) are one of the most powerful and popular tools in Computer Vision due to relatively high accuracy and performance compared to other algorithms in this area. This is why there are many frameworks are provided which make the work with neural networks (NNs) much easier and faster. However, developing the NNs from scratch helps to understand the key concepts behind this idea.

Thus, this laboratory session of Deep Learning in Computer Vision is aimed to get familiar with the implementation of neural networks from scratch in Python with numpy, and understand the principles of training and testing such networks. To do that, the MNIST dataset is used which is commonly used for training various image processing systems. This dataset consists of 70000 grayscale images of handwritten digits (of size 28x28), and can be used both for binary and multiclass classification problems which are solved in this laboratory session.

Despite a wide range of neural networks architectures, each of them consists of the following components:

- An input layer, x
- An arbitrary amount of hidden layers
- An output layer, ŷ
- A set of weights and biases between each layer, w and b
- A choice of activation function for each hidden layer, .

To evaluate the classification accuracy and performance, the three architectures were developed and tested and analysed for the obtained results.

A. Single Neuron

In the first exercise, we build a single neuron with 28 x 28 = 784 inputs - a single sigmoid unit generates the output. This neuron is trained to distinguish between the 0 digit and the others from the MNIST dataset. The graphical representation is shown in Figure 1, where n = 784.

B. A Neural Network with One Hidden Layer

This part of the laboratory session introduces one hidden layer with 64 units. Thus, the input size of 784 is connected to 64 units, and later this hidden layer is connected to the output. The reason behind creating one extra layer is that it potentially leads to better accuracy results. The representation of this idea is represented in Figure 3, where n_I = 784, n_H = 64.

ajulyav@gmail.com, zahidhassantushar@gmail.com

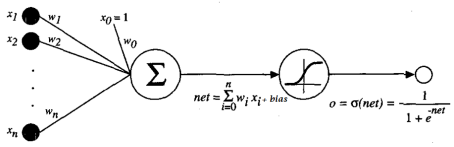

Fig. 1: Graphical representation of a single sigmoid unit

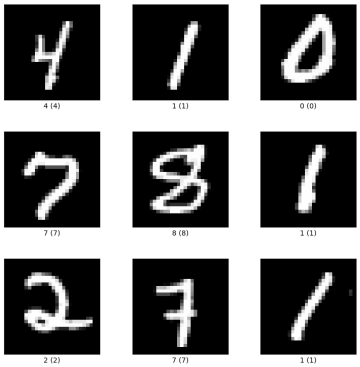

Fig. 2: Example images from the MNIST data set

C. Multiclass Neural Network

In final exercise, the task is changed to a multiclass problem which is recognizing each of 10 different digits from the MNIST dataset individually. This leads to some modifications in the previous architecture, for instance, the output layer should have ten units connected to each of the units in the previous hidden layer as this is a fully-connected network. This architecture is represented in Figure 4.

II. DATA

The MNIST database of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples each of size 28 X 28 pixels with 256 graylevels. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.

This dataset is commonly used when working on small projects to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting [1].

The data set is available in TensorFlow, and some examples from the MNIST corpus are shown in Figure 2.

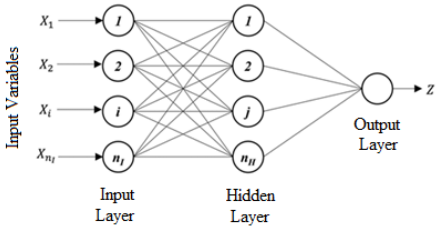

Fig. 3: Graphical representation of a neural network with one hidden layer

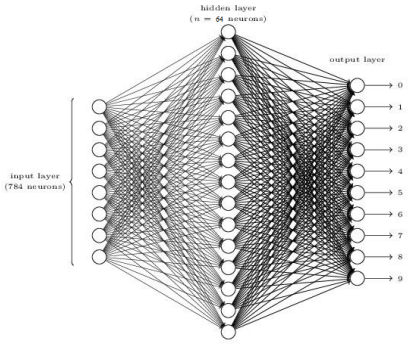

Fig. 4: Graphical representation of a neural network for a multiclass classification problem

III. IMPLEMENTATION

A. Single Neuron

Mainly, coding any neural network consists of two main parts - forward propagation and backward propagation.

In the context of forward propagation, the first step is initializing the weights randomly. We use an uniform distribution in the half-open interval [-0.005,0.005), while the bias is set to 0.

At the stage of forward propagation (making calculation from the input to the output), we multiply the inputs with the corresponding weight values, and then add bias, which gives the linear transformations of given inputs z. Later the sigmoid activation function is applied to z resulting in the post-activation output a. This step is shown in the Equation 1, where σ represents the sigmoid activation, w - weights, b - bias (all accumulated in vectors for better understanding). a^{l-1} is the output given by the previous layer, but in case of a single unit this is the direct input (784 x_train or x_test).

$$a^{(l)} = \sigma\left(\boldsymbol{W}\boldsymbol{a}^{l-1} + \boldsymbol{b}\right) \tag{1}$$

To be more detailed, the sigmoid function curve looks like a S-shape as it is shown in Figure 5. The main reason why this function is used in neural networks is because it exists between 0 and 1. Thus, as the problem to solve is a binary classification, and we should predict the probability as an input (either an input digit is 0 or the other one), the sigmoid is a good choice as probability is in the range of 0 and 1.

However, this function also has some problems when it is applied for hidden layers. Due to the gradient values close to

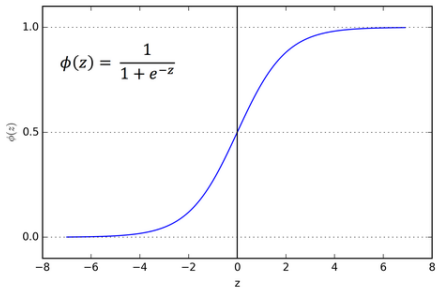

Fig. 5: Sigmoid activation function

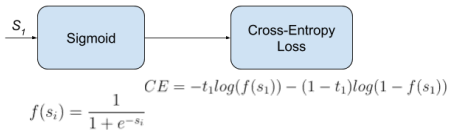

Fig. 6: Loss function

0 (during the backpropagation step) over a large portion of its domain, the training is slower and harder [2].

Then the result after apply ting the activation function is used to calculate the loss function, in particular, the cross-entropy cost as it is shown in Figure 6. It uses the log-likelihood idea to estimate its error. This function is commonly used for training neural networks, but it may stuck in a local minimum which means that the optimal parameters cannot be obtained [4].

The next important phase is a gradient-based learning which is backpropagation.The intuition behind this is that as the loss function shows how far the network's predictions from the groundtruth, backpropagation calculate the gradient of the loss function with respect to each of the weights of the network. The Equation 2 shows the gradient-based backpropogation, where L - is each layer of the network, for a single unit we can consider it as the only, C - cost function (loss).

$$\frac{\partial C}{\partial w^{(L)}} = \frac{\partial C}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial w^{(L)}} \tag{2}$$

Similar set of equations can be applied to the bias as shown

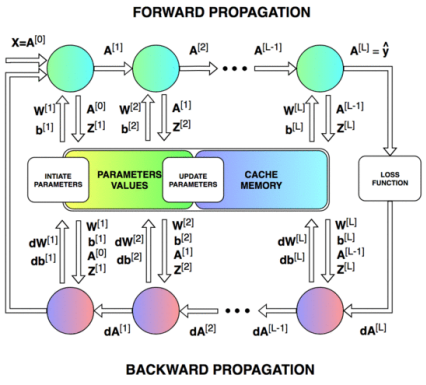

Fig. 7: Forward and back propagation

in Equation 3.

$$\frac{\partial C}{\partial w^{(L)}} = \frac{\partial C}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial b^{(L)}} \tag{3}$$

This enables every weight to be updated individually to gradually reduce the loss function over many training iterations taking into account the learning rate (step size at each iteration when moving toward a minimum of a loss function). This step is shown in Equations 4 and 5.

$$w^{(L)} = w^{(L)} - \text{learning rate} \times \frac{\partial C}{\partial w^{(L)}} \tag{4}$$

$$b^{(L)} = b^{(L)} - \text{learning rate} \times \frac{\partial C}{\partial b^{(L)}} \tag{5}$$

The both forward and back propagation steps in the context of a multi layer neural network are shown in Figure 7. The same principle is used for a single neuron treated as one unit in one layer.

The forward propagation is performed both on the train and test sets to calculate the train and test losses, but only the train set is used to update the parameters (weights and bias) during backpropagation. The number of epochs is 500, the learning rate is 0.25.

B. A Neural Network with one Hidden Layer

Having added the hidden layer with 64 units as shown in Figure 3, we have changed the design of weights and bias. One set of weights and bias is used for the input layer towards the hidden one (64*784), and the other set is applied for the hidden layer towards the output (1*64).

The forward propagation is performed in a similar way like in the previous Subsection, but the sigmoid function is calculated twice - at the input layer, then at the hidden one. The loss function is the same, and it is calculated on the output of the second activation function (at the hidden layer).

The principle of the backward propagation is the same, but now derivatives are calculated at each layer to update all weights and bias values.

C. Multiclass Neural Network

The Figure 4 illustrates that the output layer is changes compared to the single output node in the previous Subsections. The number of output nodes is the same to the number of digits to be classified. Due to this modification, the weights and bias sizes between the hidden layer and output layer are changed to (10*64).

Also, the new loss function is a cross-entropy loss for multiclass problems (Equation 6).

$$\textit{(Multiple class) Cross-entropy} = -\sum_{i}^{C} t_i log(s_i) \tag{6}$$

, where t_i and s_i are the groundtruth and the NN score for each class i in C. Also, the activation function (sigmoid) is applied to the scores before the Loss computation.

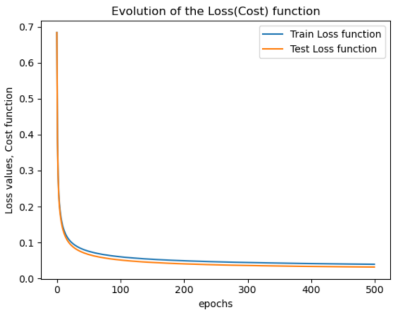

Fig. 8: Loss curves for single neuron model

TABLE I: Accuracy for all models for both Train and Test data

Data	Binary Classification		Multiclass Classification
	Single Neuron	NN with one single neuron	Multiclass NN
Train	98.95%	90.13%	82.54%
Test	99.19%	90.2%	82.32%

Thus, this type of crossentropy is well used to multiclass classification problems, as one example can be considered to belong to a specific category with probability 1, and to other categories with probability 0.

IV. RUNNING THE CODE

First, make sure that the required libraries are installed. The TensorFlow environment should be activated. Open the folder containing the skeleton of the lab code, run the command window in this folder and type the line:

python lab2_skeleton.py to run the single class classification with a single neuron script

python lab2_2_skeleton.py to run the single class classification with a hidden layer script

python lab2_3_skeleton.py to run the multiclass classification with a hidden layer script,

V. RESULTS AND DISCUSSION

A. Single Neuron

As the binary classification problem is considered, all the data instances are transformed into two labels. The digit is considered as the True label while other digits are labeled as False. Since the network was learning only one digit, it was very easy to classify. In Figure 8 the loss curve for training and test data is shown.

The loss was very high at the beginning since the model was learning the parameters (weights and bias). After 100 iterations, the loss is almost constant and falling very slightly. In Table I it can be seen that the accuracy is 98.95% for train data and 99.19% for test data.

B. A Neural Network with one Hidden Layer

When one hidden layer is added, the accuracy drops for both train and test datasets as shown in Table I. The loss curves plotted in Figure 9 shows that before 200 epochs there is a good fit - train and validation losses decrease to a particular point with a minimal gap between them.

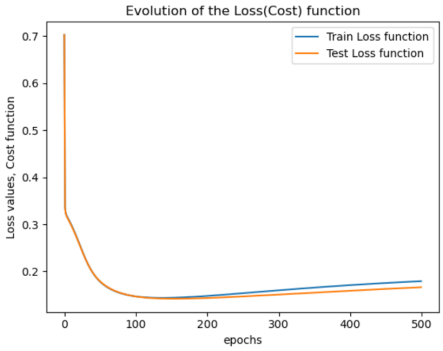

Fig. 9: Loss curves for one hidden layer model

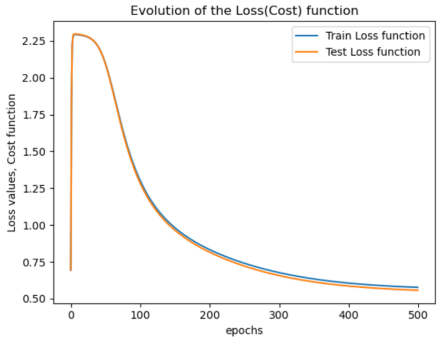

Fig. 10: Loss curves for multiclass NN model

After 200 epochs the loss is growing, and continued training of a good fit will likely lead to an overfit. It could happen because the dataset is relatively small for this model taking into account long training. Although there are many possible solutions to solve this problem, the two simplest are either to use a bigger test dataset or stop training at the point when the loss curve tends to increase.

C. Multiclass Neural Network

In case of a multiclass classification problem, we obtained the accuracy of 82.54 % for train data and 82.35% for test data. Although this performance is lower compared to the previous results, we cannot fairly compare them between each other as this problem is more complex than a binary classification. It's because the network is learning parameters for 10 different classes now unlike the previous cases.

As it is shown in Figure 10, in the beginning of training there is a huge jump in the error. It could happen because of the values of parameters initialized randomly, as they are far from the optimal values. However, later we obtain a good fit when both train and test losses are decreasing. Since the model trained only for 500 iterations, the parameters might have not reached the optimal point yet. Further iterations would certainly lead to better accuracy.

In total, the performance of this simple model for the given multiclass problem is quite high, and can be potentially increased.

VI. CONCLUSIONS

Neural Networks are a powerful tool in the ares of Computer Vision which is able successfully to solve a wide range of tasks. In this laboratory session, we implemented three basic models from scratch - a single neuron, a neural network with one hidden layer for a binary classification problem and for a multiclass classification problem. Even such basic architectures allowed us obtain great accuracy performance. Developing a model from scratch is a good practice to understand the ideas behind training complex networks using popular frameworks. Thus, we implemented and learned the core of any neural model - forward and backward propagation, the idea of activation function, as well as how modify the architecture.

REFERENCES

[1] LeCun, Yann and Cortes, Corinna. "MNIST handwritten digit database." (2010).

[2] Sagar sharma (2017, September 6) Activation Functions in Neural Networks. Retrieved from https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[3] Imad Dabbura (2018, April 1) Coding Neural Network — Forward Propagation and Backpropagtion. Retrieved from https://towardsdatascience.com/coding-neural-network-forward-propagation-and-backpropagtion-ccf8cf369f76

[4] Peltarion, Categorical crossentropy. Retrieved from https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/categorical-crossentropy