EE559 Mini deep-learning framework using Pytorch

Mariam Hakobyan, Nguyet Minh Nguyen, Mazen Fouad A-wali Mahdi
mariam.hakobyan@epfl.ch, minh.nguyen@epfl.ch, fouad.mazen@epfl.ch

I. Introduction

Neural networks, as the name suggests, are machine learning methods that are inspired by information processing and distributed communication nodes in biological systems such as the brain [1]. The current popularity of neural networks is due to several factors that occurred through out the past few years such as, the increased computational power and development of distributed and GPU systems, the availability of labeled data, and the advancement of a number of algorithms that helped tackle and stabilize essential issues in neural networks. There are various applications of neural networks ranging from classification to pattern recognition and prediction which all displayed a boost in performance on benchmark data-sets by the introduction of different models such as Convolutional Neural Network (CNN) [2], Recurrent Neural Network (RNN) [3] et cetera. As any complex system, a neural network can be dissected into basic abstract components and by combining these components the complexity arises. In this project, we will discuss these components and show how we can build a multi-layer perceptron [4] using an open source deep learning platform, Pytorch [5].

II. Abstract components

In this section we will discuss the various building blocks of a neural network in detail. The simplest neural network abstraction could be described as a black box where a certain set of parameters are being updated to satisfy a relationship between an input and an output. Once this black box achieves a desired performance, its' internal state can be saved for future data (or test data) to be passed through it and the outputs are recorded. Decomposing this black box, we can see that it's formed of simpler components such as activation functions, loss functions, optimizers, containers, and lastly layers. The construction of such networks is as follows. The input data is passed from the input layer where it is multiplied by some weights (the parameters of the network), and passed to a neuron in the next layer. For simplicity, let's take a case of a single neuron. The operation of this neuron, is to sum the inputs given to it (the weights * input). This value is then passed through an activation function and a certain output is produced. This process keeps on happening until we reach the output layer. The value of the output layer is then compared with the actual true value and to determine how far we are from the true value a loss function is used. Our goal is to minimize this loss, and update the weights of the network using an optimizer.

A. Activation functions

The use of activation functions in a neural network introduces the non-linearity property to our network, without it our output signal is merely composed of simple linear functions and it wouldn't be able to learn complex mappings of our data [6].

Sigmoid

The sigmoid is a non-linear function that squashes the value of the output in the range [0,1]. The drawback in using the sigmoid function is that it responds poorly to changes in the input when the values are large, hence the gradient is also small which gives rise to the vanishing gradient problem.

$$Sigmoid(x) = \frac{1}{1+e^{-x}} \tag{1}$$

$$\frac{Sigmoid(x)}{dx} = Sigmoid(x) \circ (1 - Sigmoid(x)) \tag{2}$$

Hyperbolic Tangent

The hyperbolic tangent function can be described as a scaled sigmoidal that output values in the range [-1,1]. The difference between the tanh and the sigmoid is that its' derivatives are steeper and strongly negative inputs map to a negative output.

$$Tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3}$$

$$\frac{Tanh(x)}{dx} = 1 - Tanh(x)^2 \tag{4}$$

Rectified Linear Unit

The rectifier linear unit (ReLU) is a piece-wise linear function that will output the same input if positive, otherwise it outputs a zero. The ReLU and it's variants have become the default activation functions for neural networks because they train easier and often lead to better performances [7].

$$ReLU(x) = \begin{cases} 0 & x \leq 0 \\ x & 0 < x \end{cases} \tag{5}$$

$$\frac{d(ReLU(x))}{dx} = \begin{cases} 0 & x \leq 0 \\ 1 & 0 < x \end{cases} \tag{6}$$

Softmax

The softmax function or the normalized exponential function is a function that takes as input real values and outputs a probability. The sum of the output probabilities must add to 1.

$$Smax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}} \tag{7}$$

$$\frac{d(Smax(x_i))}{dx_i} = Smax(x_i) \circ (1 - Smax(x_i)) \tag{8}$$

B. Loss Functions

A loss function can be considered as a performance metric for our network. If the output of the network is far from the target value then the loss is high and vice versa. The goal of the network is to minimize this loss, hence increasing the performance.

Mean Squared Error

The mean squared error measures the average squared difference between the actual output and the estimated one.

$$MSE(y, x) = \sum_{i=1}^{I} (y_i - x_i)^2 \tag{9}$$

$$\frac{dMSE(y_i, x_i)}{d(x_i)} = 2(y_i - x_i) \tag{10}$$

Cross Entropy

Cross entropy measures the performance of a classification model whose output is a probability value between 0 and 1 (usually achieved by the softmax).

$$CE(y, p) = -\sum_{i=0}^{K} \log(p_i)(y_i) \tag{11}$$

$$\frac{CE(y, p)}{d(p_i)} = p_i - y_i \tag{12}$$

C. Optimizers

Optimization algorithms play an important role in neural networks because they help us minimize the error by updating the internal parameters of our model (weights & biases).

Stochastic Gradient Descent

SGD is an optimizer that performs a parameter update for each training sample. It usually performs better because of the frequent updates allowing us to discover and reach better local minima. If a batch is used, then it can be referred to as mini-batch gradient descent.

$$w^{(l)} = w^{(l)} - \gamma(\delta^{(l)}(x^{(l-1)})) \tag{13}$$

D. Layers

A layer is a connection of neurons operating together at a certain depth of the network. There are several types of layers such as the identity, linear, and bi-linear. In this project we will be focusing only on the linear layer.

Linear Layer

The linear layer takes as input and output the number of desired neurons. In this layer we initialize the weights based on Xavier's initialization and the biases to zero. Its' main purpose is to calculate the value of each neuron, which is the weights*input + bias in the forward pass, and compute the gradients in the backward pass. In this layer we also compute the derivatives of the weights and the bias that will later be used in the optimizer.

E. Containers

A container is a module that collects the different layers of the network discussed above and pipes them together to create the structure of the network.

Sequential

A sequential container structures the different layers such as the linear layer and the activation functions in a sequential manner. Each output of a layer is directly connected to the layer after it.

III. Back-propagation

Back-propagation is the main mechanism by which a neural network learns. In other words, it's the transmission of information throughout the network. Two important notions are defined: the forward pass, and the backward pass. In the forward pass the network computes the sums of each input multiplied by the weights as described before and gives an output. The loss function computes how the output is far from the real target, and in the backward pass the network computes what is the effect of a minor change in our learnable parameters that would decrease the loss; hence the use of the derivatives. The optimizer then updates the parameters based on those derivatives.

We can look at the back-propagation as a fancy implementation of the chain rule and goes as follows:

$$\frac{d(L_0)}{d(w_{jk})^{(L)}} = \frac{d(z_j)^{(L)}}{d(w_{jk})^{(L)}} \circ \frac{d(a_j)^{(L)}}{d(z_j)^{(L)}} \circ \frac{d(L_0)}{d(a_j)^{(L)}} \tag{14}$$

$$\frac{d(L_0)}{d(a_k)^{(L-1)}} = \sum_{j=0}^{n_L} \frac{d(z_j)^{(L)}}{d(a_k)^{(L-1)}} \circ \frac{d(a_j)^{(L)}}{d(z_j)^{(L)}} \circ \frac{d(L_0)}{d(a_j)^{(L)}} \tag{15}$$

Where L is the loss, W is a specific weight, a is the activation, and z is the sum before the activation at a given specific layer.

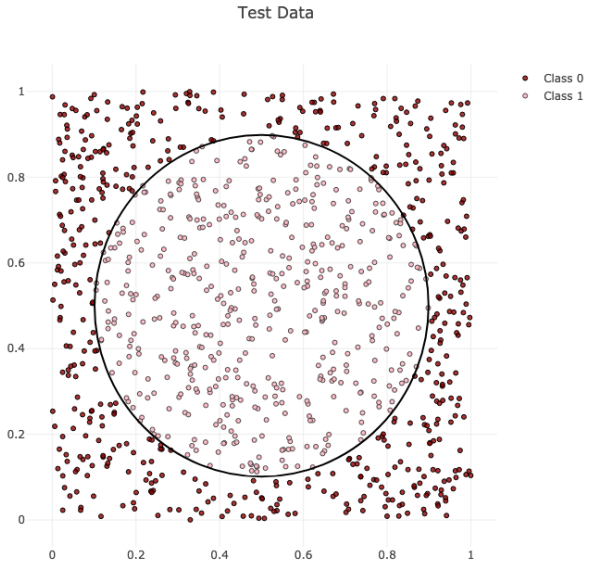

Figure 1. Generated test data.

IV. Dataset

The data-set is a simple one that aims to check if the implementation of the neural network is functioning properly or not. It is composed of 1000 training points and 1000 testing points that are sampled from a uniform distribution [0,1]. Points that are lying outside a disk of radius $1/\sqrt{2 * \pi}$ are labeled 0 and points inside the disk are labeled 1. A visualization of the data can be found in figure 1.

V. The magic

Now that we've discussed all the essential building blocks of a neural network, we can actually build one and see how it performs on the generated data. First, we need to define what is the architecture of our network and what kind of building blocks it's going to use. The network we implemented is composed of an input layer of 2 neurons, 3 hidden layers of 25 neurons each, and an output layer of 2 neurons. Between the layers we use the Relu activation function, and after the output layer we use the Tanh activation function. The network utilizes the mean squared error loss, and uses mini-batch gradient descent to update the parameters. The network is trained over 500 epochs, a batch size of 100, and with shuffling the input data. On the right you can see how the loss and accuracy behave over epochs, and how the predictions of the data look. It is noticed that the errors the network makes is solely around the boundaries of the circle.

VI. Conclusions

In this project, we've learned about the different building blocks of a network and how to construct it.

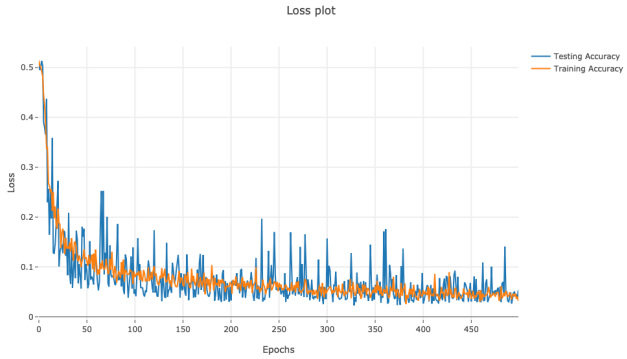

Figure 2. Training and testing loss.

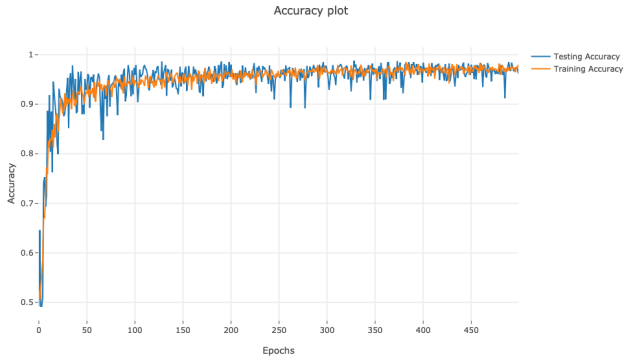

Figure 3. Training and testing accuracy.

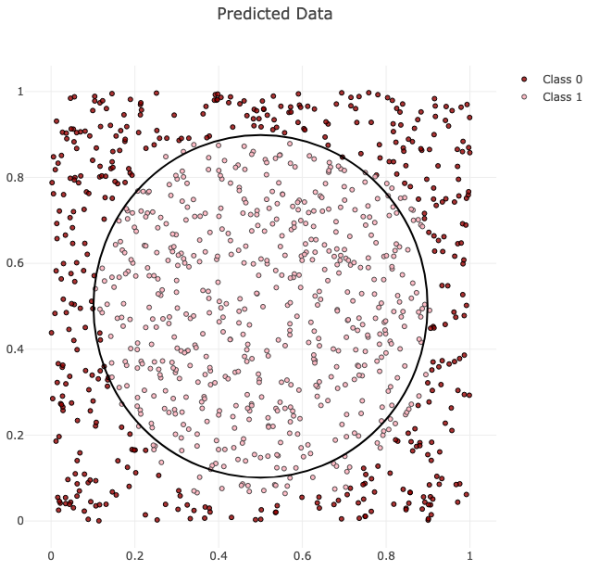

Figure 4. Predicted data.

REFERENCES

[1] https://en.wikipedia.org/wiki/Deeplearning

[2] Krizhevsky, A., Sutskever, I., Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

[3] Mikolov, T., Karafit, M., Burget, L., ernock, J., Khudanpur, S. (2010). Recurrent neural network based language model. In Eleventh annual conference of the international speech communication association.

[4] Haykin, S. (1994). Neural networks: a comprehensive foundation. Prentice Hall PTR.

[5] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., ... Lerer, A. (2017). Automatic differentiation in pytorch.

[6] https://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f

[7] Nair, V., Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In Proceedings of the 27th international conference on machine learning (ICML-10) (pp. 807-814).