Neural Nets - Assignment 1

Alex Zucca[1,*]

[1]*Department of Physics, Simon Fraser University, 8888 University Drive, Burnaby, British Columbia V5A 1S6, Canada*

(Dated: January 25, 2018)

I. INTRODUCTION

This is a short document accompanying the code on Github[1]. We first introduce the mathematical notation necessary to understand how a neural network works. In the following calculations we will refer to the simple neural network shown in Fig. 1.

We introduce the following quantities that help generalizing the concepts for any neural network with 1 hidden layer.

- n_s is the number of samples of our training set.
- n_f is the number of features of out training set. The training matrix is $X \in \mathbb{R}^{n_s \times n_f}$. Also, the number of inputs of the neural network is n_f.
- n_o is the number of outputs of the neural network. The target matrix is $y \in \mathbb{R}^{n_s \times n_o}$.
- n_h is the number of neurons in the hidden layer

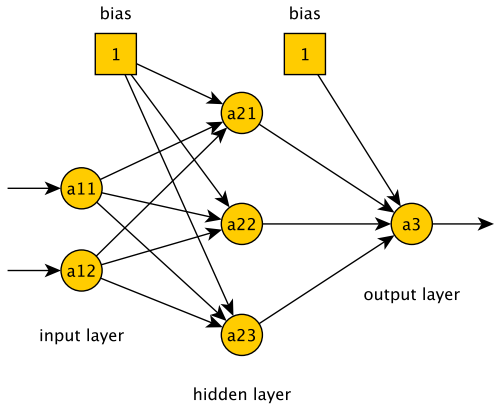

FIG. 1: A simple neural network (Multi-layer Perceptron, MLP) with two input neurons, a hidden layer with 3 neurons and an output neuron. Note the bias units represented with squares.

*Electronic address: azucca@sfu.ca

[1] https://github.com/alexzucca90/Projects/blob/master/MLP_assignment.ipynb

A. Feed-forward

Here we describe how the input is propagated through the neural network in order to produce an output. Let us define $a_1 \in \mathbb{R}^{n_f+1}$ as

$$a^{(1)} = \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_{n_f} \end{pmatrix}. \tag{1}$$

For each neuron i of the hidden layer we have the net input

$$z_i^{(2)} = w_{i,1}^{(1)} a_1^{(1)} + \cdots + w_{i,n_f+1}^{(1)} a_{n_f+1}^{(1)}. \tag{2}$$

that is passed to the activation function φ. In matrix form we have

$$z^{(2)} = W^{(1)} a^{(1)}, \tag{3}$$

where $W^{(1)} \in \mathbb{R}^{n_h \times (n_f+1)}$ is the weights matrix for the input layer. Here, $z_2 \in \mathbb{R}^{n_h}$. We then define the vector $a^{(2)} \in \mathbb{R}^{n_h+1}$ as

$$a^{(2)} = \begin{pmatrix} 1 \\ \varphi(z^{(2)}) \end{pmatrix}, \tag{4}$$

where the activation function φ is applied to each element of $z^{(2)}$. The last step is to compute the activation of the output layer neuron. We have

$$z^{(3)} = W^{(2)} a^{(2)}, \tag{5}$$

where $W^{(2)} \in \mathbb{R}^{n_0 \times (n_h+1)}$ is the weight matrix for the hidden layer. We can then calculate the activation of the output layer as $a^{(3)} = \varphi(z^{(3)})$.

B. Feed-forward for the training set

Here we generalize the procedure described above in case we want to calculate for n_s samples. Let us define the matrix $A_1 \in \mathbb{R}^{(n_f+1) \times n_s}$ as

$$A^{(1)} = \begin{pmatrix} 1 & \dots & 1 \\ x_{11} & \dots & x_{n_s 1} \\ \vdots & \ddots & \vdots \\ x_{1 n_f} & \dots & n_{n_s n_f} \end{pmatrix}. \tag{6}$$

Then the net input for the hidden layer is given by

$$\mathbb{R}^{n_h \times n_s} \ni Z^{(2)} = W^{(1)} A^{(1)}, \tag{7}$$

where $W^{(1)}$ is the same defined in the previous section. Then the activation of the hidden layer is given by

$$\mathbb{R}^{(n_h+1)\times n_s} \ni A^{(2)} = \begin{pmatrix} 1 \dots 1 \\ \varphi(Z^{(2)}) \end{pmatrix}, \tag{8}$$

where we have inserted a row of n_s 1. The next step is to calculate the net input of the output layer, $Z^{(3)}$. We have

$$\mathbb{R}^{n_o \times n_s} \ni Z^{(3)} = W^{(2)} A_2, \tag{9}$$

and the activation of the output layer becomes $A_3 = \varphi(Z)^{(3)}$. The above equations are implemented in the code updated on Github.

C. Back-propagation of the error.

When training the neural net we need to update each element of the weight matrices, $W^{(1)}$ and $W^{(2)}$ according to a rule. Let $E(y, \varphi(Z_3))$ be the objective function that we want to minimize. Using the gradient descent algorithm, at each iteration we update the weights according to the rule

$$w_{ij}^{(L)} \to w_{ij}^{(L)} - \alpha \frac{\partial E}{\partial w_{ij}^{(L)}}, \tag{10}$$

where $L = 1, 2$ and α is the learning rate. The goal of this section is to calculate the second term on the RHS in the equation above. To be more specific and consistent with the `Python` code, let the objective function be the mean squared errors (MSE),

$$E(y, \varphi(Z_3)) = \frac{1}{2} \sum_{i=1}^{n_s} \sum_{j=1}^{n_o} \left(y_{ji} - \varphi(z_{ji})^{(3)} \right)^2. \tag{11}$$

Taking the derivative w.r.t. $w_{mn}^{(L)}$, $L = 1, 2$ yields

$$\frac{\partial E}{\partial w_{mn}^{L}} = - \sum_{i=1}^{n_s} \sum_{j=1}^{n_o} \left(y_{ji} - \varphi(z_{ji}^{(3)}) \right) \frac{\partial \varphi}{\partial z}\bigg|_{z_{ji}^{(3)}} \frac{\partial z_{ji}^{(3)}}{\partial w_{mn}^{(L)}}. \tag{12}$$

Now let us begin with the simplest case, $L = 2$. The last factor of the equation above is

$$\frac{\partial z_{ji}^{(3)}}{\partial w_{mn}^{(2)}} = \delta_{jm} a_{ni}^{(2)}, \tag{13}$$

thus yielding

$$\frac{\partial E}{\partial w_{mn}^{(2)}} = - \sum_{i=1}^{n_s} \left(y_{mi} - \varphi(z_{mi}^{(3)}) \right) \frac{\partial \varphi}{\partial z}\bigg|_{z_{mi}^{(3)}} a_{ni}^{(2)}. \tag{14}$$

The equation above can be cast in matrix form as

$$A_{mn} = \sum_{i=1}^{n_s} B_{mi} C_{in}, \tag{15}$$

where

$$A_{mn} = \frac{\partial E}{\partial w_{mn}^{L}}, \tag{16}$$

$$B_{mi} = - \left(y_{mi} - \varphi(z_{mi}^{(3)}) \right) \frac{\partial \varphi}{\partial z}\bigg|_{z_{mi}^{(3)}} \tag{17}$$

$$C_{in} = a_{ni}^{(2)}, \tag{18}$$

or

$$\nabla_{W^{(2)}} E = - \left[\left(y^T - Z^{(3)} \right) \frac{\partial \varphi}{\partial z}\bigg|_{Z^{(3)}} \right] \cdot A^{(2)\,T}. \tag{19}$$

Eq. (19) can be easily implemented in `Python` with the command `numpy.matmul` or `@`.

Let us now compute the Eq. (12) for $L = 1$. Using the chain rule we find

$$\frac{\partial z_{ji}^{(3)}}{\partial w_{mn}^{(1)}} = w_{j,m+1}^{(2)} a_{ni}^{(1)} \frac{\partial \varphi}{\partial z}\bigg|_{z_{mi}^{(2)}}, \tag{20}$$

so the Eq. (12) becomes

$$\frac{\partial E}{\partial w_{mn}^{(1)}} = - \sum_{i=1}^{n_s} \sum_{j=1}^{n_o} \left(y_{ji} - \varphi(z_{ji}^{(3)}) \right) \frac{\partial \varphi}{\partial z}\bigg|_{z_{ji}^{(3)}} w_{j,m+1}^{(2)} a_{ni}^{(1)} \frac{\partial \varphi}{\partial z}\bigg|_{z_{mi}^{(2)}}, \tag{21}$$

that can be cast in a matrix-tensor form

$$A_{mn} = \sum_{i=1}^{n_s} \sum_{j=1}^{n_o} B_{ji} C_{jimn}, \tag{22}$$

where

$$A_{mn} = \frac{\partial E}{\partial w_{mn}^{L}}, \tag{23}$$

$$B_{ji} = - \left(y_{ji} - \varphi(z_{ji}^{(3)}) \right) \frac{\partial \varphi}{\partial z}\bigg|_{z_{ji}^{(3)}}, \tag{24}$$

$$C_{jimn} = w_{j,m+1}^{(2)} a_{ni}^{(1)} \frac{\partial \varphi}{\partial z}\bigg|_{z_{mi}^{(2)}}. \tag{25}$$

Unfortunately I don't know how to handle rank-four tensor operations in `Python` so I will implement a simple nested `for` loop.

II. THE NEURAL-NET AT WORK

Here we have a look at the results obtained with the following dataset

$$X_{\text{train}} = \begin{pmatrix} -0.5 & -0.5 \\ 0.5 & -0.5 \\ -0.5 & 0.5 \\ 0.5 & 0.5 \end{pmatrix}, \quad y_{\text{train}} = \begin{pmatrix} -0.5 \\ 0.5 \\ 0.5 \\ -0.5 \end{pmatrix}. \tag{26}$$

A. Test 1

This discussion follows the requirements of the assignment. We train 3 different neural networks with 2,4 and 6 hidden neurons respectively for 300 epochs. The learning rate is set as $\alpha = 0.1$ and the momentum parameter is set as $\gamma = 0$. The results are shown in Fig. 2

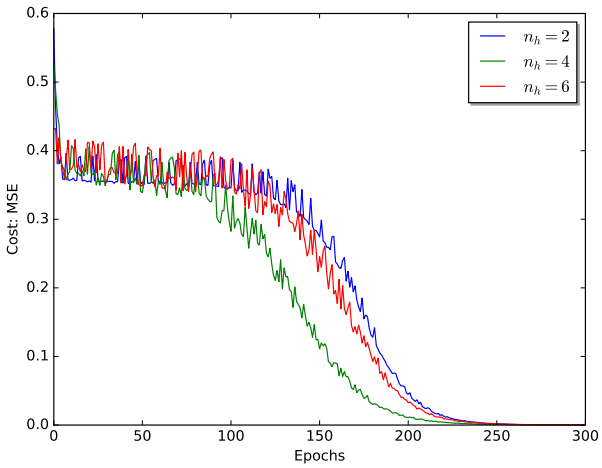

FIG. 2: Results for the test performed in Sect. II A.

B. Test 2

This discussion follows the requirements of the assignment. We train 3 different neural networks with 2,4 and 6 hidden neurons respectively for 200 epochs. The learning rate is set as $\alpha = 0.1$ and the momentum parameter is set as $\gamma = 0.9$. The results are shown in Fig. 3.

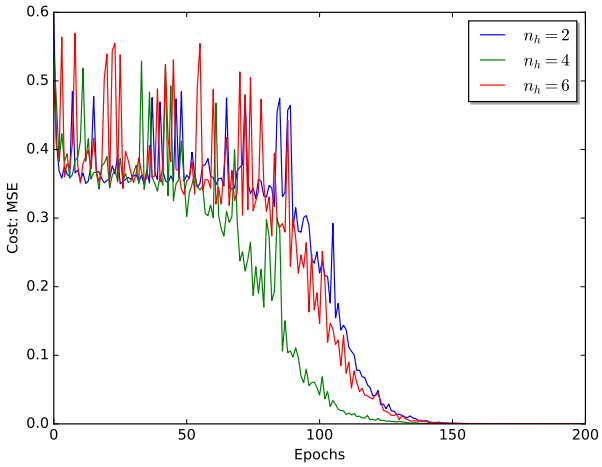

FIG. 3: Results for the test performed in Sect. II B.

III. CONCLUSIONS

Somehow we can see that implementing the momentum stochastic gradient descent helps the convergence speed. Also we see that the fastest convergence is achieved by a neural network with 4 hidden neurons.