INF3490 Mandatory Assignment 2: Multilayer Perceptron

Paul Wieland

Deadline: Tuesday, October 16th, 2018 23:59:00

Contents

1 Introduction

1.1 Task

We will build a Multilayer Perceptron to steer a robotic prosthetic hand. There are 40 inputs of electromyographic signals that we will classify.
There are 8 classification values corresponding to a different hand motion:

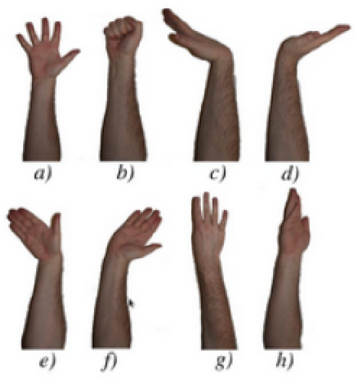

Figure 1: Possible motions [1]

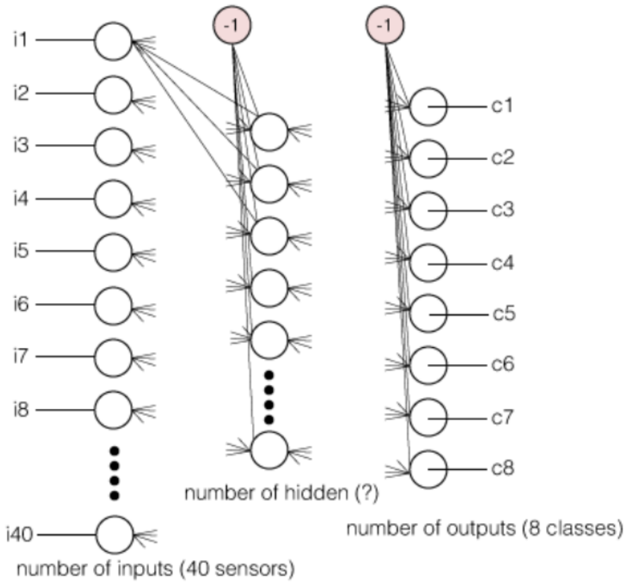

Figure 2: Multilayer Perceptron for our problem [2]

We build a Multilayer Perceptron with 40 entry nodes, that means one node for each input. Then there is one hidden layer with a various number of hidden nodes. For classifying the input, there are 8 output nodes corresponding to the 8 hand motions. We only use one hidden layer to solve this problem.

[1]http://folk.uio.no/kyrrehg/pf/papers/glette-ahs08.pdf
[2]https://www.uio.no/studier/emner/matnat/ifi/INF3490/h18/assignments/assignment-2/assignment_2.pdf

1.2 Training Data

For each input vector:

$$\mathit{input} = [i_1, i_2, i_3, i_4, ..., i_{40}], i_n \in \mathbb{R}, n \in [40] \tag{1}$$

we have a target output vector:

$$\mathit{output} = [c_1, c_2, c_3, c_4, ..., c_8], c_n \in \{0,1\}, n \in [8], \sum_{n=1}^{8} c_n = 1 \tag{2}$$

That means, forwarding the input should result in the given target vector.

2 Implementation

The file *mlp.py* contains the class mlp. There are 5 functions that i will explain in detail.

2.1 Initialization

The function *__init__(self, inputs, targets, nhidden)* has three important parameter that we need to initialize the Multilayer Perceptron.
As the input data is given as a vector, it is a good choice to create two 2D-Array for the two weight layers. As the parameters *inputs* and *targets* have the type *<class 'numpy.ndarray'>*, it is a good idea to work only with numpy arrays.

2.1.1 Dimension of the weight matrix

- weight_matrix_1:
  The input vector in (1) has of course a size of 40 $(len(inputs[n]), n \in [len(inputs) - 1] \cup \{0\})$. But we need to add the *bias_value -1* that can be seen in Figure 2. That means:

$$\mathit{weight\_matrix\_1} \in \mathbb{R}^{41 \times nhidden} \tag{3}$$

- weight_matrix_2:
  There are *nhidden* hidden nodes and 8 $(len(targets[n]), n \in [len(targets) - 1] \cup \{0\})$ exit nodes. So we also need to take into account the *bias_value -1*. That means:

$$\mathit{weight\_matrix\_2} \in \mathbb{R}^{(nhidden+1) \times 8} \tag{4}$$

Both, *weigh_matrix_1* and *weigh_matrix_2*, will be initialized randomly with values in [-1,1].

2.2 Forward

The forward function takes one input vector and runs it on the network. At first, the input vector must be expanded. The reason for this is that we have to take into account the bias value:

$$input \in \mathbb{R}^{1\times 41} \tag{5}$$

Subsequently it is possible to compute the *hidden_values*:

$$hidden\_values = [h_1, h_2, h_3, h_4, ..., h_8] \tag{6}$$

$$h_i = \sum_{n=1}^{41} input[n] \times weight\_matrix\_1[n][i] \tag{7}$$

This operation is done by the function *vec_matr_mult()* (that function can be found in the file *operations.py*) that is doing a vector matrix multiplication with two for-loops.
After that is the activation function applied to all hidden nodes:

$$h_{new,i} = \frac{1}{1+\ exp(-\beta h_i)} \tag{8}$$

The resulting *activation_hidden_values* is also expanded by the bias -1:

$$activation\_hidden\_values \in \mathbb{R}^{1\times(nhidden+1)} \tag{9}$$

So the *output* vector can be calculated easily by another vector matrix multiplication:

$$output = activation\_hidden\_values \cdot weight\_matrix\_2 \tag{10}$$