MicroPython - Neural Network

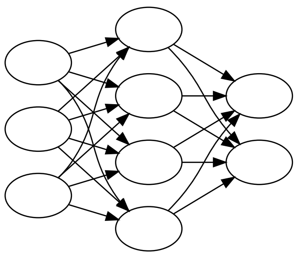
Implement Neural Network Deep Feed Forward on micro-controller using MicroPython

Olivier Lenoir
olivier.len02@gmail.com

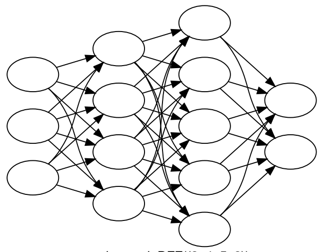
February 14, 2021

Abstract

Implement *Neural Network Deep Feed Forward* on micro-controller using *MicroPython*.
This project is designed in pure *MicroPython*.



neuralnetwork.DFF((3, 4, 2))



neuralnetwork.DFF((3, 4, 5, 2))

1 Requirements

Download *matrix.py*[1] and *neuralnetwork.py*[2] and copy them on your MicroPython board. The same code can be used on your computer with Python.

2 Get Started

I'm going to describe how work *MicroPython - Neural Network* with a very small an simple example. In this classifier we are using a *Sigmoid* activation function as $\sigma(x)$ and his derivative as $\sigma'(x)$. Here is what we want to predict, with i_n as inputs and s_n as the expected classification.

i_1	i_2	i_3	s_1	s_2
0	0	0	1	0
0	0	1	0	1
0	1	1	0	1
0	1	0	1	0
1	1	0	0	1
1	1	1	1	0
1	0	1	0	1
1	0	0	1	0

Table 1: Training set

We use a neural network (3, 4, 2). With 3 values in the input layer, 4 values in the hidden layer and 2 values in the output layer.

[1] matrix.py: `https://gitlab.com/olivier.len02/MicroPython-Matrix/-/blob/master/micropython/matrix.py`

[2] neuralnetwork.py: `https://gitlab.com/olivier.len02/MicroPython-NeuralNetwork/-/blob/master/micropython/neuralnetwork.py`

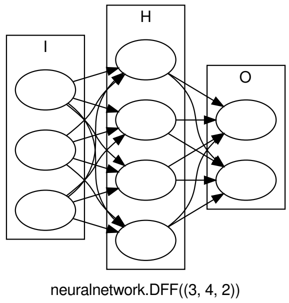


Figure 1: Neural network (3, 4, 2) detail

Input layer is represented by the matrix I and the output by the matrix O. Hidden layer is matrix H. Weights between matrix I and H is matrix W_{ih}. Weights between matrix H and O is matrix W_{ho}.

$$\begin{aligned} \sigma(x) &= \frac{1}{1+e^{-x}} \\ \sigma'(x) &= \sigma(x) \cdot (1-\sigma(x)) \\ \sigma'(x_\sigma) &= x_\sigma \cdot (1-x_\sigma) \end{aligned} \tag{1}$$

$$I = \begin{pmatrix} i_1 & i_2 & i_3 \end{pmatrix} \tag{2}$$

$$W_{ih} = \begin{pmatrix} w^{ih}_{1,1} & w^{ih}_{1,2} & w^{ih}_{1,3} & w^{ih}_{1,4} \\ w^{ih}_{2,1} & w^{ih}_{2,2} & w^{ih}_{2,3} & w^{ih}_{2,4} \\ w^{ih}_{3,1} & w^{ih}_{3,2} & w^{ih}_{3,3} & w^{ih}_{3,4} \end{pmatrix} \tag{3}$$

$$H = \begin{pmatrix} h_1 & h_2 & h_3 & h_4 \end{pmatrix} \tag{4}$$

$$W_{ho} = \begin{pmatrix} w^{ho}_{1,1} & w^{ho}_{1,2} \\ w^{ho}_{2,1} & w^{ho}_{2,2} \\ w^{ho}_{3,1} & w^{ho}_{3,2} \\ w^{ho}_{4,1} & w^{ho}_{4,2} \end{pmatrix} \tag{5}$$

$$O = \begin{pmatrix} o_1 & o_2 \end{pmatrix} \tag{6}$$

$$S = \begin{pmatrix} s_1 & s_2 \end{pmatrix} \tag{7}$$

From those matrices you can propagate the input I to the output O using the following calculations:

$$\begin{aligned} H &= \sigma(I \cdot W_{ih}) \\ O &= \sigma(H \cdot W_{ho}) \end{aligned} \tag{8}$$

If the network is properly trained, the output O should be very close from the expected S matrix. If not, this mind we need to train the artificial neural network with the training data-set and back-propagate the error to adjust weights.

We are now going to use propagated results to back-propagate the error of each layers. E_o and E_h are error of layers O and H. Matrix gW_{ho} and gW_{ih} are the gradient to adjust weights.

$$\begin{aligned} E_o &= (S-O) \times \sigma'(O) \\ gW_{ho} &= H^T \cdot E_o \end{aligned} \tag{9}$$

$$\begin{aligned} E_h &= (E_o \cdot W_{ho}) \times \sigma'(H) \\ gW_{ih} &= I^T \cdot E_h \end{aligned} \tag{10}$$

$$\begin{aligned} W_{ho} &= W_{ho} + gW_{ho} \\ W_{ih} &= W_{ih} + gW_{ih} \end{aligned} \tag{11}$$

Weights are updated. Training continue until we are satisfied with the result.

2.1 Train Neural Network

I recommend training *Neural Network* on a computer. Otherwise you may quickly run into memory error on your MicroPython board, even if you use garbage collect.

```
from matrix import Matrix
import neuralnetwork as nn

# Create neural network with an input layer of 3, an hidden layer of 4
# and an output layer of 2 by default the activation function is sigmoid()
# but a ReLU also exist as relu()
ann = nn.DFF((3, 4, 2))
```

Now let's create a training set with input matrix and output matrix.

```
training_set = [
    [Matrix([[0, 0, 0]]), Matrix([[1, 0]])],
    [Matrix([[0, 0, 1]]), Matrix([[0, 1]])],
    [Matrix([[0, 1, 1]]), Matrix([[0, 1]])],
    [Matrix([[0, 1, 0]]), Matrix([[1, 0]])],
    [Matrix([[1, 1, 0]]), Matrix([[0, 1]])],
    [Matrix([[1, 1, 1]]), Matrix([[1, 0]])],
    [Matrix([[1, 0, 1]]), Matrix([[0, 1]])],
    [Matrix([[1, 0, 0]]), Matrix([[1, 0]])],
    ]
```

We train the network a thousand times with the training set. The learning rate (*lrate*) is set to one by default

```
print('Learning progress')
for i in range(1000):
    for a, s in training_set:
        ann.train(a, s, lrate=1)
    if i % 10 == 0:
        print('.', end='')
```

Check if you are satisfied with the training.

```
def short(a):
    return round(a, 3)


print('=' * 20)
score = True
for a, s in training_set:
    p = ann.predict(a)
    scr = str(p.map(round)) == str(s.map(short))
    print(a.map(short), p.map(short), p.map(round), s.map(short), scr)
    score &= scr
print('Good learning?', score)
```

Print out weights.

```
print('=' * 20)
print('Rounded weights')
for i, w in enumerate(ann.weights):
    print('W{}'.format(i), w.map(round))
```

2.2 Predict

With the trained weights, we can now use our network.

```
from matrix import Matrix
from neuralnetwork import DFF

ann = DFF(
    (3, 4, 2),
    weights=[
        Matrix([[-1, -4, 6, -4], [5, 0, 6, 1], [4, -4, -8, 6]]),
        Matrix([[-10, 10], [5, -5], [7, -7], [6, -6]])
        ]
    )

d = Matrix(((1, 0, 1)))
p = ann.predict(d)
print(p)
```

References

(1) Jean-Claude Heudin, *Comprendre le deep learning, une introduction aux réseaux de neurones*, Science-eBook, Octobre 2016, ISBN 979-10-91245-44-9.

(2) Jean-Claude Heudin, *Intelligence Artificielle, manuel de survie*, Science-eBook, Octobre 2017, ISBN 978-2-37743-000-0.

(3) Damien George, *MicroPython*, George Robotics Limited, https://micropython.org/.

(4) Nicholas H. Tollervey *Programming with MicroPython, embedded programming with MicroPython & Python*, O'Relly, 1st edition, October 2017, ISBN 978-1-491-97273-1.

(5) Olivier Lenoir, *MicroPython - Matrix*, GitLab, January 2021, https://gitlab.com/olivier.len02/MicroPython-Matrix/.

(6) Tutorials Point, *Artificial Neural Network Tutorial*, 2021, https://www.tutorialspoint.com/artificial_neural_network/index.htm

(7) Wikipedia, *Sigmoid function*, wikipedia.org, 2021, https://en.wikipedia.org/wiki/Sigmoid_function