Oct 31st, 1:00 PM - 2:30 PM

Implementation of Artificial Neural Network in Embedded Systems

Roni Kasemi
University for Business and Technology - UBT

Bertan Karahoda
University for Business and Technology, bertan.karahoda@ubt-uni.net

Implementation of Artificial Neural Networks in Embedded Systems

Roni Kasemi[1], Bertan Karahoda[2]

[1]UBT, Faculty of Management of Mechatronics and Robotics, Prishtinë, Kosova
{roni.kasemi,bkarahoda}@ubt-uni.net

Abstract. As we know the artificial intelligence, and specifically artificial neural networks, have improved rapidly in the last decade which leads to the application of these systems in commercial fields like in buying online products, medical applications, financial applications, etc. But we know also that ANN usually are complex systems that need a lot of computing power in order to function properly which limits their application in number of fields, including here embedded systems because of their limited hardware and software properties. In this paper our goal is to implement a fully functional ANN in ATmega328p microcontroller, which will be programmed and trained in microcontroller. Our study case is solar tracker, where we aim to create a functional tracker which works based on ANN and with very limited hardware resources. With this implementation we aim to prove that ANN can function in practical systems without need of high computing power but just with simple low cost embedded system.

Keywords: Artificial Intelligence, artificial neural networks, embedded systems, atmega328p.

1 Introduction

The idea of creating an artificially intelligent machine existed even before the technology was capable of creating such a thing, but early development of AI started in 1950 with an article of Alan Turing published in “Mind” magazine [1]. Since then, various systems and models have been developed like Expert Systems, Genetic Algorithms, Fuzzy Logic, Artificial Neural Networks, etc. But we can easily say that ANN were one of the most advanced systems since they were capable of solving real complicated intelligence problems like winning against world chess and GO champions, detecting various cancers, driving cars autonomously without human intervention, etc.

1.1 Literature review

First developments in neural networks started in 1943 when McCulloch and Pitts proposed a simple nonlinear model of actual neurons, which was followed by Rorenblatt in 1960 with the first proposed model of a layered network of perceptrons. But as you can imagine the first models were limited and were not able to solve even simple XOR tables. But that changed with development of the first multilayer network in 1985 which includes a famous backpropagation algorithm [2].

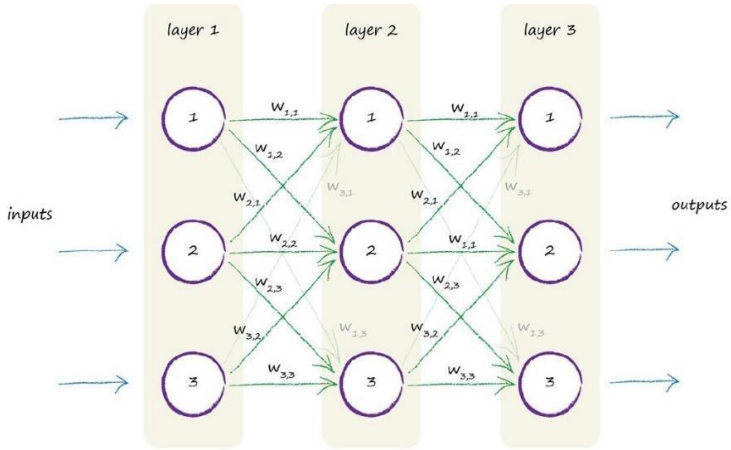

Fig. 1. Multilayer Artificial Neural Network [3]

First step is to calculate the output of the first layer which will be obtained by multiplying each neuron with specific weight and then sum the total as output of the next layer neuron. Then we need to apply to that output an activation function, where we use as example sigmoid function:

$$y = \frac{1}{1+e^{-x}} \tag{1}$$

Where x represents the output we get from the previous calculation. The process is repeated until we get to the final output layer where we get the final result and then compare it with desired or targeted output, where their difference will give us the error [3]. The error value then will be distributed to all the layers back with backpropagation method which can be represented like:

$$\mathbf{Error}_{\text{hidden}} = \mathbf{W}^{T}_{\text{hidden\_output}} * \mathbf{error}_{\text{output}} \tag{2}$$

Then comes maybe the most important part of the whole process which is updating the values of weights and can be presented by the formula below:

$$\Delta W_{jk} = \alpha * E_k * sigmoid(O_k)\left(1 - sigmoid(O_k)\right) * O_j{}^{T} \quad (3)$$

Through this process the ANN can learn from its own errors and get always closer to desired output. The whole process presented above is representing one of the most common ANN structures called Multilayer Perceptron which will be used also in our application of study case. But also, other types of ANN such like Convolutional Neural Networks, Radial Basis Function Neural Networks, Recurrent Neural Networks, etc. use as base model the steps described above, with the difference that each network contains unique elements and algorithms to fit their purpose.

But with increasing the complexity of tasks that are given to solve by ANN we need also to increase the computational power, which brings us to our main problems we try to solve:

Can ANN be implemented in limited devices such as embedded systems?
If so, what is the limit of complexity of applied ANN in an embedded system?
It is possible for an embedded system with trained ANN to control a physical system?

To find answers for these questions first we need to understand the limits of the embedded system that we are going to use. Embedded systems are computational systems that are designed to do specific tasks (e.g. microwaves or televisions) and because of that they usually have specific and limited software and hardware unlike classical computers which are designed for general purpose use. These limitations make it possible for embedded systems to be very compact in size and usually cheaper in price, but also have very limited hardware, especially memory (RAM and ROM), speed of processor, battery lifetime, number of I/O, etc. Another limit is in software which is as a result of hardware, so most of the embedded systems use C programming language [4]. The hardware we are going to use is ATmega328p which uses Harvard architecture and contains separate memories and specific communication lines for program and data which increase the parallel processing. It is an 8-bit microcontroller with three 16-bit registers, 2KB SRAM and 32KB flash [5].

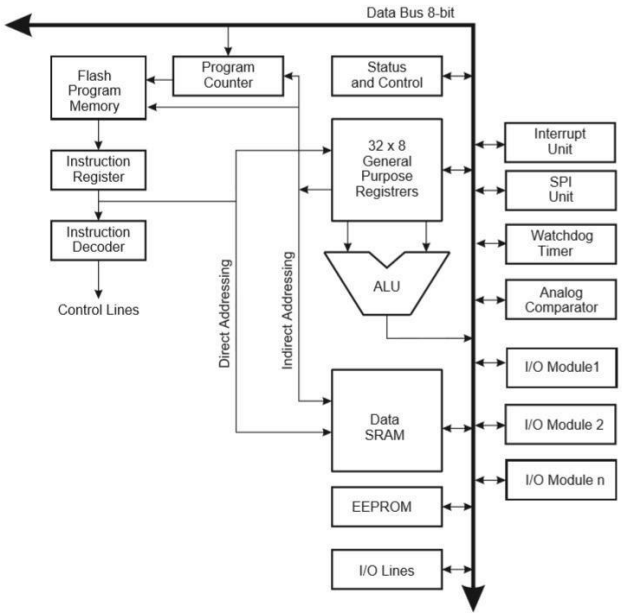

Fig. 2. Architecture of ATmega328p [4]

All of these limited specifications of microcontroller create a real challenge to develop and train a fully functional ANN in it.

1.2 State of the art

During the research for related work**,** we find that most of the publications and projects use computers to train the ANN and then apply trained network in embedded system which is not very similar with our case since we want to train the network in the microcontroller. One of the most relevant work related to our research was a paper from Gary Parker and Mohammad Khan, Connecticut College. They applied dynamic learning using backpropagation method and Arduino Nano as hardware (which have the same microcontroller we are using). Practical approach in this paper was implementation of logic operators AND, OR, XOR and XNOR where LED's were used as output of a system (representing 0 and 1) but what makes it different from our case was that in this project for each neuron an Arduino Nano was used. That gives a lot more processing power since every neuron has its own processing unit and they communicate with each other via I2C protocol and master – slave system [6]. If we compare it with our project the main difference is that in our case**,** we trained a neural network in only one microcontroller which was obviously more challenging due to system limitations.

2 Results

As a practical implementation we chose to build a solar tracker, which is supposed to track the light source not only from East to West but also North and South. So, the system will have 2DOF from two servo motors which will be the outputs of the system, and four photoresistors to detect light source as inputs of the system. In the beginning we are going to tell the system extreme cases for example:

If LDR1 has full light then turn Servo1 for 180 degrees and Servo2 for 10 degrees.

So, we define each extreme case for different combinations and then use ANN to train the system in order to learn every other scenario in between so the solar panel can track the light source smoothly in every direction. First, we need to define Pattern Count which determines different case combinations, Input Nodes for number of input neurons and Output Nodes for neurons to control servo motors. Then we define parameters like: Hidden Nodes, Learning Rate, Momentum, Weights, Success. Each of these parameters need to be changed and adjusted through a trial and error process until we figure out the best values to work with our system. All the other calculations like calculating output of neurons, sigmoid function, calculating error, backpropagation it and update the weights need to be done manually by coding since there is no library with built in functions in C language to do the math part of ANN (like in Python) and also the memory of ATmega328p is very limited for such complex libraries.

After changing and experimenting with parameters we mention above, we find that Learning Rate, Hidden Nodes and Success parameters have the biggest effect on how the system will work. So, Learning Rate determines actually how much the network learns from errors, Hidden Nodes represent the number of neurons in the hidden layer and the Success parameter determines at what percentage of success rate we consider the network successfully trained. At the beginning we start with values: Learning Rate = 0.3, Hidden Nodes = 6, Success = 0.0002 and we get results as below.

Table 1. Results from the first experiments

Input 1	Input2	Input3	Input4	Desired Output1	Desired Output2	Actual Output1	Actual Output2
0	1	1	0	0.2	1	0.20016	0.98587
1	1	0	0	0.8	0.1	0.79986	0.10001
0	0	1	0	0.2	0.5	0.19921	0.50022
1	0	0	0	0.8	0.5	0.80021	0.50009
0	0	0	0	0.5	0.5	0.49982	0.5002
0	0	1	1	0.2	0.1	0.20059	0.09982
1	0	0	1	0.8	1	0.79987	0.98594
1	1	1	1	0.5	0.5	0.50015	0.50086

Time needed to train the network was about 1 minute and it takes 1194 iterations to complete the training. We can see from the table that actual outputs were really close to desired outputs and the solar panel was able to track the source of light. But when movement of light is bit faster, the system almost always starts to shake and loses the control of tracking the light at all.

So, we changed parameters to Learning Rate = 0.1, Success = 0.0001 and Hidden Nodes = 8 and we get the results:

Table 2. Results from the second experiment

Input 1	Input2	Input3	Input4	Desired Output1	Desired Output2	Actual Output1	Actual Output2
0	1	1	0	0.2	1	0.20001	0.98977
1	1	0	0	0.8	0.1	0.79997	0.10004
0	0	1	0	0.2	0.5	0.19985	0.50015
1	0	0	0	0.8	0.5	0.80001	0.50014
0	0	0	0	0.5	0.5	0.50009	0.50011
0	0	1	1	0.2	0.1	0.2001	0.09997
1	0	0	1	0.8	1	0.79999	0.99026
1	1	1	1	0.5	0.5	0.49995	0.50054

We can see that results of actual outputs do not change that much but a big difference was the number of iterations with 5000 and time needed to train the network was

around 4 to 5 minutes. On the practical side we see positive results as the system starts to respond significantly more stable than before. So, we start to search for optimal values and as final values we get Learning Rate = 0.2, Hidden Nodes = 7, Success = 0.0001 and the final results were:

Table 3. Results from the final experiment

Input 1	Input2	Input3	Input4	Desired Output1	Desired Output2	Actual Output1	Actual Output2
0	1	1	0	0.2	1	0.2001	0.99013
1	1	0	0	0.8	0.1	0.79993	0.09996
0	0	1	0	0.2	0.5	0.19961	0.50014
1	0	0	0	0.8	0.5	0.80009	0.50011
0	0	0	0	0.5	0.5	0.50004	0.5001
0	0	1	1	0.2	0.1	0.20033	0.09983
1	0	0	1	0.8	1	0.79995	0.98991
1	1	1	1	0.5	0.5	0.50003	0.50053

Network takes 3200 iterations and about 2 minutes to train and produce almost the same results as before. Also, we can see from the graph below that actual and desired outputs are almost the same.

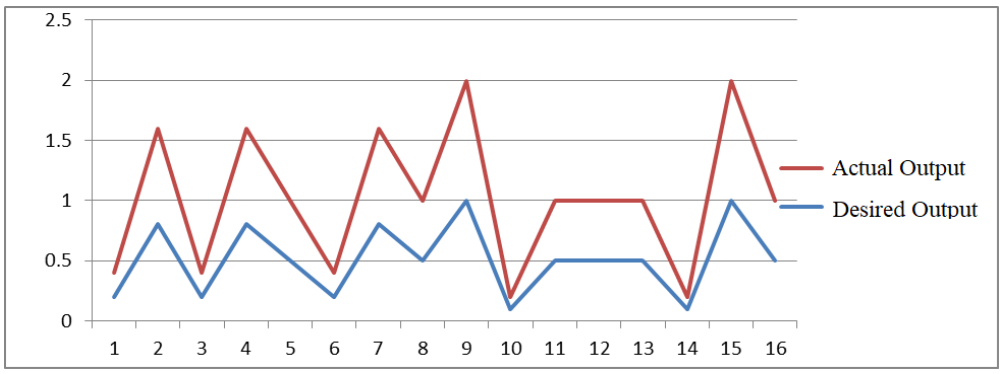

Fig. 3. Graph of the desired and actual outputs

And the final form of applied ANN looks like:

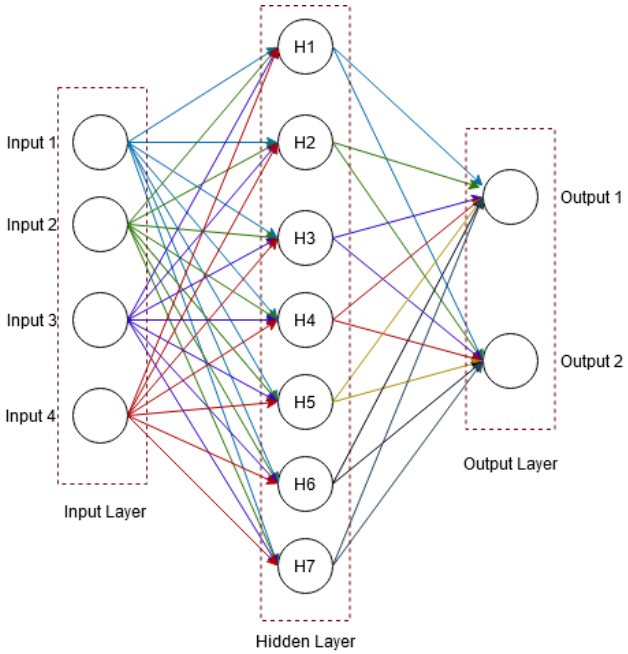

Fig. 4. Final structure of ANN

3 Discussions

We can say that the designed ANN was trained in a very limited embedded system such as ATmega328p microcontroller and was fully functional which was one of the main goals we wanted to achieve in this project. A trained system was also applied on practical project which means it can be used and adapted to real life systems with little changes. When we come to the limits of the microcontroller we use, we can say that the main limitation is the number of hidden neurons in the network. When the number of hidden neurons is above 9 the algorithm often gets stuck in an infinite loop while trying to calculate the actual outputs within the range of Success parameter. Learning Rate and Success parameters are mostly making the training slower but they alone do not cause major problems like the number of hidden neurons. We also do not try to add more than one hidden layer since only adding more neurons causes trouble and adding more layers just would make the network not functional. Also, worth noting is that the photoresistors were used directly without any applied filter or electronics to smooth and stabilize the signal which means that ANN was able by himself to read the raw data from simple photoresistors and produce very good results by tracking the light precisely.

During the research we also find out that the new development board series like ESP and Arduino Pro provide much better and powerful hardware which makes it possible to train more complex networks for solving harder problems. Another thing that makes these boards special is that they have access to the internet and some of their models start to make possible development of much more complex networks in TensorFlow by training the network in cloud and processing the results in real time as IoT systems.

References

1. B. G. Buchanan, "A (Very) Brief History of Artificial Intelligence," AI Magazine, vol. 26, pp. 53-60, 2006.
2. J. Larsen, "Introduction to Artificial Neural Networks," Technical University of Denmark, Lyngby, 1999.
3. T. Rashid, Make Your Own Neural Network, 2016.
4. E. White, Making Embedded Systems: Design Patterns for Great Software, O'Reilly Media, 2011.
5. Atmel, 8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash, Atmel Corporation, 2015.
6. M. K. Gary Parker, "Distributed Neural Network: Dynamic Learning via Backpropagation with Hardware Neurons using Arduino Chips," in International Joint Conference on Neural Networks (IJCNN), 2016.