runlinc AI Project 4:
Machine Learning Demo (E32W Version)

Contents

Introduction

Aim

This project will demonstrate machine learning in predicting the next light that will be clicked by the user.

Background

To make machine predicting like any intelligent human being, this machine needs to be trained or to be imparted knowledge that allows it to think like a human being. The goal of artificial intelligence (AI) as a science is to make machines do things that would require intelligence if they had been done by humans.

There are numerous ways to make a machine think more like a human. The most popular way is the **Neural network**. The neural network is based on the system of human biological neural network (human brain). The brain consists of a densely interconnected set of nerve cells, or basic information-processing units, called **neurons** with the connections, **synapses** between them. A neuron consists of a cell body, **soma**, several fibres called **dendrites**, and a single long fibre called an **axon**.

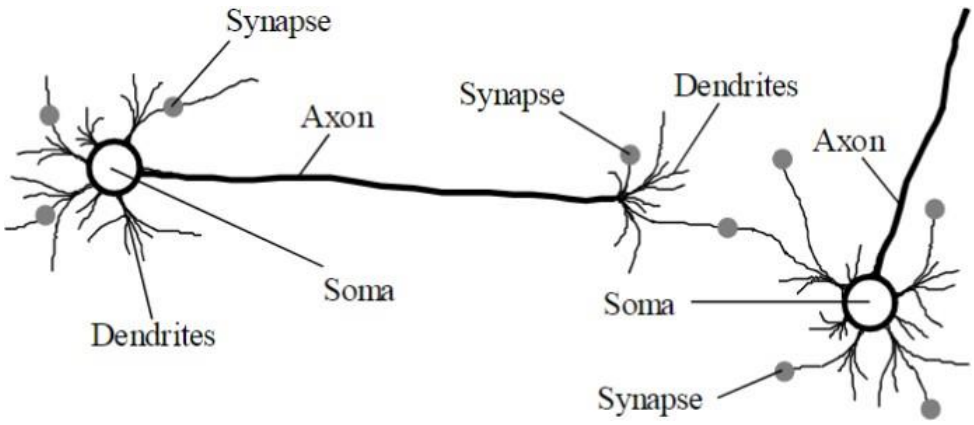

Figure 1: Biological neural network

For the neural network in a machine, they have the same structure but instead of using view biologically, we use blocks, lines, numbers, mathematically viewing on those units.

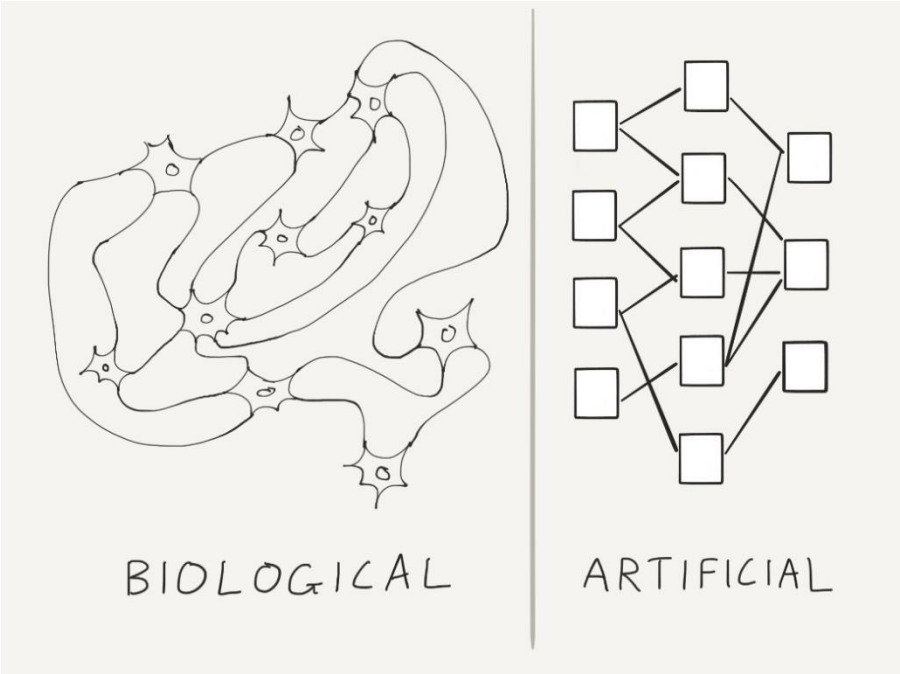

Figure 2: Neural network representation between biological and artificial

Runlinc Background

Runlinc is a web page inside a Wi-Fi chip. The programming is done inside the browsers compare to programming inside a chip. The runlinc web page inside the Wi-Fi chip will command the microchips to do sensing, control, data logging Internet of Things (IoT). It can predict and command.

Part A: Design the Circuit on runlinc

Note: refer to runlinc Wi-Fi setup guide document to connect to runlinc

For our case, our neural network looks like this: (For more information on this diagram, read the appendix)

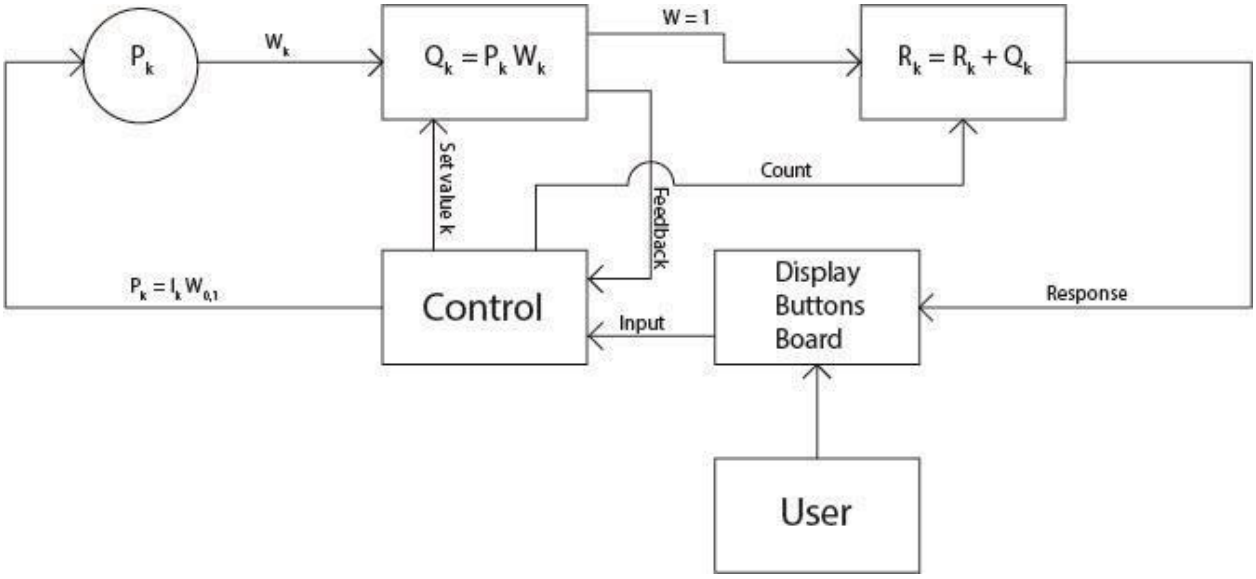

Figure 3: Block diagram of Red-or-Green neural network

Figure 4: On runlinc control page, assign "Light19" in D19, and "Light23" in D23

For D19, it'll be a DIGITAL_OUT with the variable name of "Light19".
For D23, it'll be a DIGITAL_OUT with the variable name of "Light23".

Part B: Build the Circuit

Use the STEMSEL E32 board to connect the hardware. For this project we are using both the left and right I/O ports, with **negative port (-ve)** on the outer side, **positive port (+ve)** on the middle and **signal port (s)** on the inner side (as shown below).

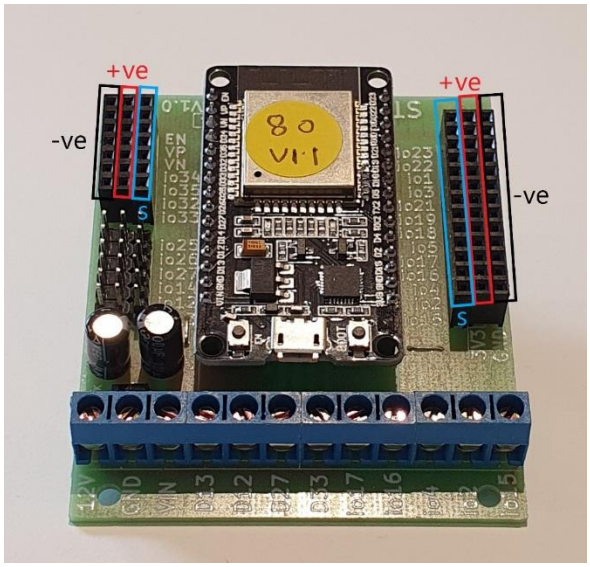

Figure 5: Negative, Positive and Signal port on the E32 board

Wiring Instructions

a) Plug in one LED to io19 on the E32W board.

b) Plug in another LED to io 23 on the E32W board.

c) Make sure all (-ve) pins are on the GND (outer) side of the I/O ports.

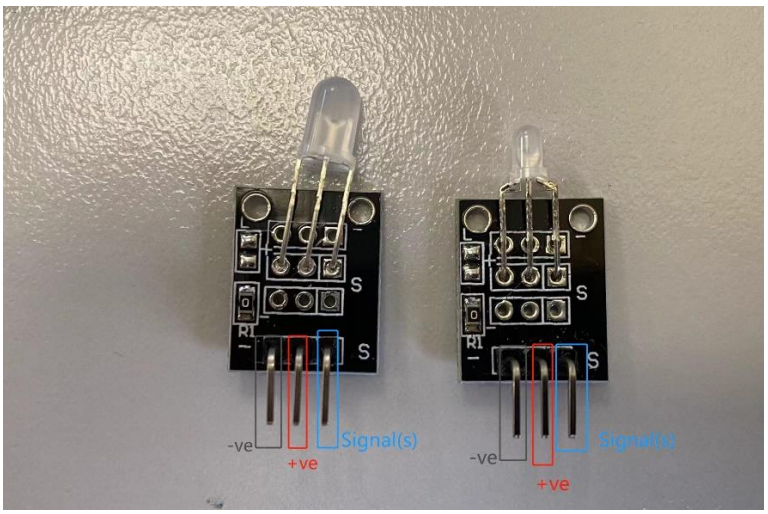

Figure 6: I/O parts with negative, positive and signal pins indicated

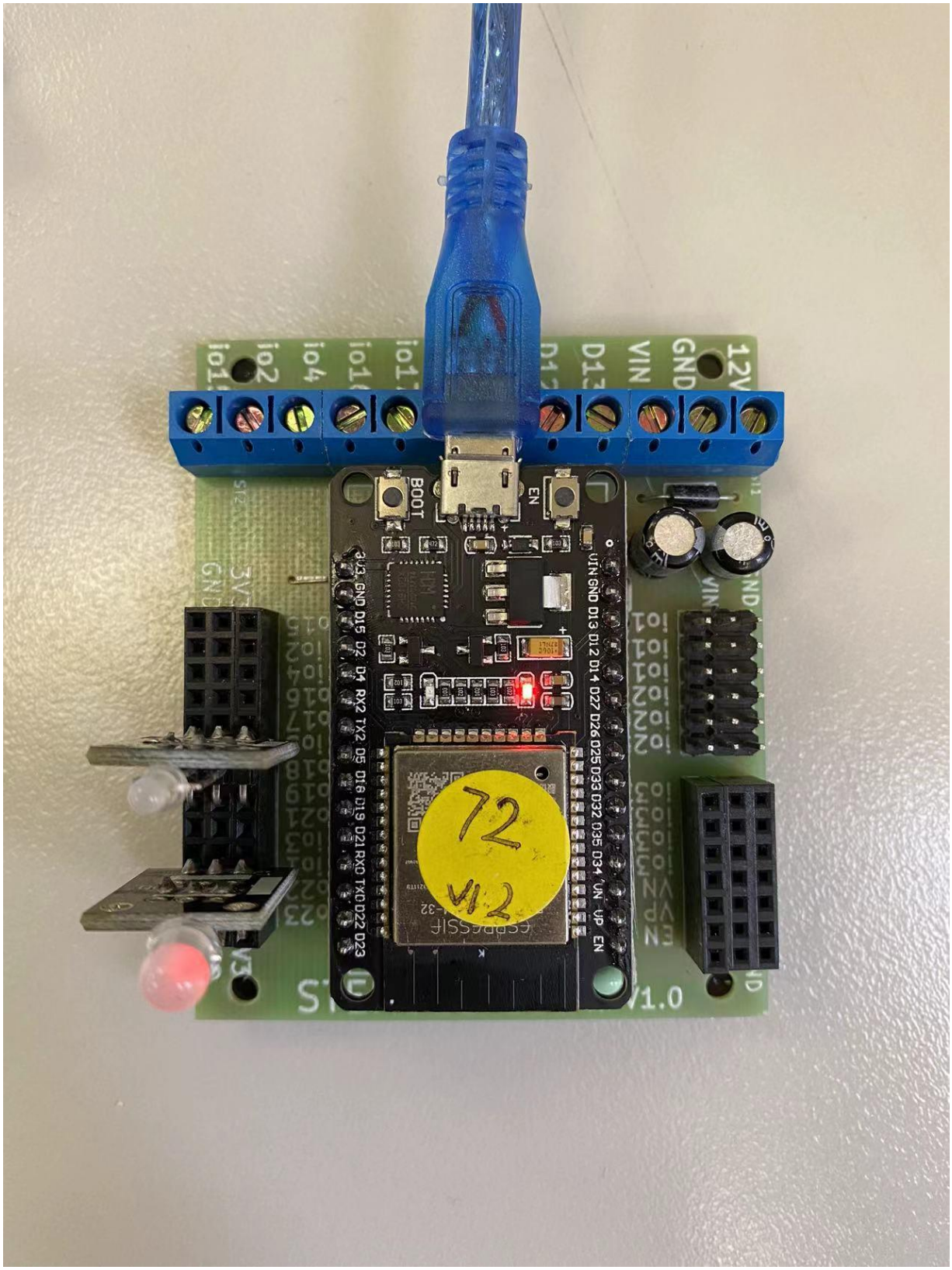

Figure 6: top view of E32W board

Part C: Program the Circuit

HTML:

```
<div>
<h1>Red or Green</h1>
<button style="background-color: rgb(255, 103, 103)"
onclick="play('light19')">Light_19</button>
<button style="background-color: rgb(118, 255, 118)"
onclick="play('light23')">Light_23</button>
<button onclick="play('default')">Reset</button>
</div>
<br>
<laber>Sequence: </laber>
<output id="seq">       </output><br>
<laber>Predict next light will be </laber>
<output id="predict">  </output>
<br><br><br>
```

JavaScript:

```
var n = 8;
var sequence = [];
var count = 0;

function play(light) {
if (sequence.length > 7) { sequence.shift();}
switch (light) {
case 'light19':
sequence.push("1");
calculation();
break;
case 'light23':
sequence.push("0");
calculation();
break;
default:
weight = 0;
sequence = [];
document.getElementById("seq").innerHTML = "      ";
document.getElementById("predict").innerHTML = "";
turnOff(Light19);
turnOff(Light23);
break;
 }
}
```

```
function calculation() {
 var input = [];
 var weightEach = [];
 var weight = 0;
 var SUM = 0; var DEC = 0;
 for (var i = 0; i < sequence.length; i++) { if (sequence[i] == 0) {
  input[i] = 0.1;
 }
 if (sequence[i] == 1) {
  input[i] = 1; count++;
 }
 SUM = SUM + sequence[i] * Math.pow(10, 7 - i);
 DEC = SUM.toString(10);
 weightEach[i] = (input[i] / ((1 / DEC) + n));
 weight += weightEach[i];
 }
 document.getElementById("seq").innerHTML = sequence.join(" ");
 prediction(weight);
}
```

```
function prediction(pred) {
if (sequence.length < 7) { document.getElementById("predict").innerHTML = "       ";
turnOff(Light19);
turnOff(Light23);
} else {
if (pred >= 0.5499999931763907) {
document.getElementById("predict").innerHTML = "light19";
turnOn(Light19);
turnOff(Light23);
}
if (pred <= 0.5374999333107895) {
document.getElementById("predict").innerHTML = "light23";
turnOff(Light19);
turnOn(Light23);
}
if (pred == 0.6624999923295869) {
document.getElementById("predict").innerHTML = "light23";
turnOff(Light19);
turnOn(Light23);
}
if (pred == 0.41249951870961277) {
document.getElementById("predict").innerHTML = "light19";
turnOn(Light19);
turnOff(Light23);
}
}
}
```

Part D: Run the Application

When you finish implementing the code, remember to send the code to the board. Then you can run the machine learning demo at the IP address. It should have the following page:

Figure 7: Webpage

Then you can play around the sequence. You'll find that it'll be slowly able to predict more accurately how you will plan your next sequence. But of course, it won't be highly accurate.

Appendix

The general architecture of an artificial neural network:

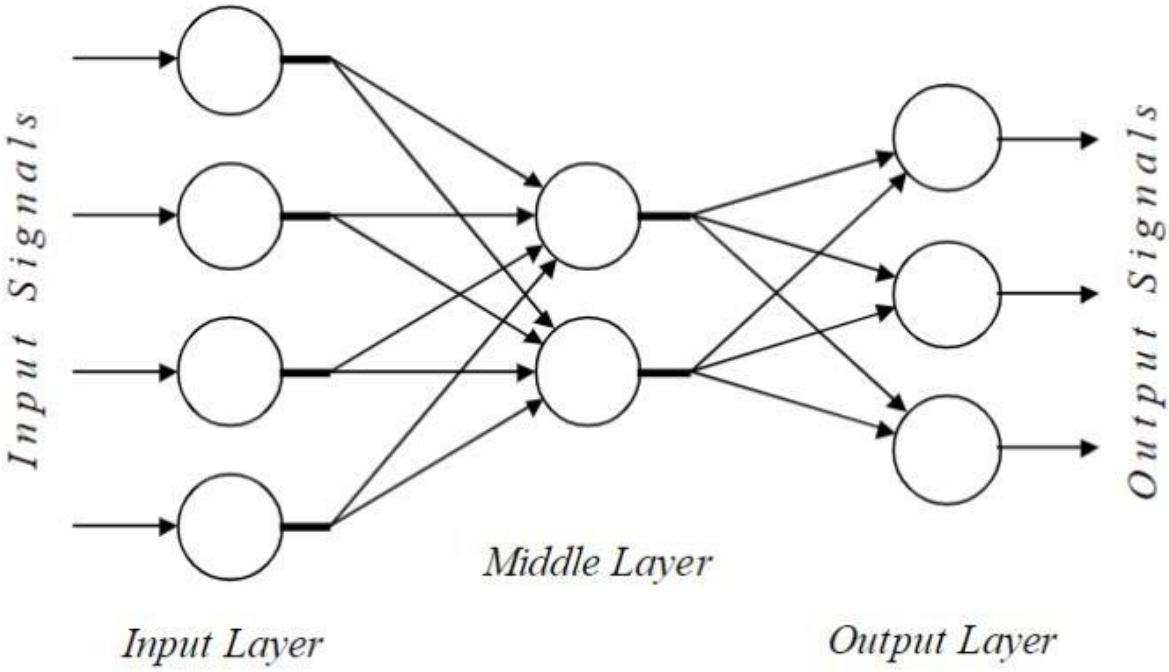

Figure A1: Architecture of an artificial neural network

To put it into an analogy between biological and artificial neural networks, we have:

- Some as Neuron
- Dendrite as Input
- Axon as Output
- Synapse as Weight

And to simplify artificial neural network as a computing element, we have

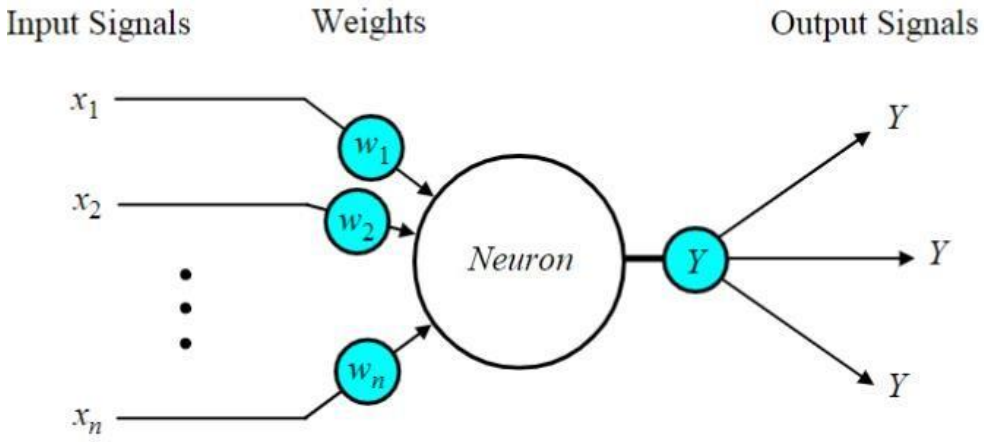

Figure A2: Diagram of a neuron

Before designing our block diagram, figure 3, we need to know our situations:

- 2 inputs with individual weight
- Total of 256 combinations in our sequence, from 00000000 to 11111111 (8 bits)
- Each of the combinations needs to have different weights
- Only record 8 bits

Let's start with 2 inputs with 2 bits. The possible combination of 2 bits, 0 and 1 are:

Decimal	A	B
0	0	0
1	0	1
2	1	0
3	1	1

Table 1: Possible combination of 2 bits

Therefore, in the neural network:

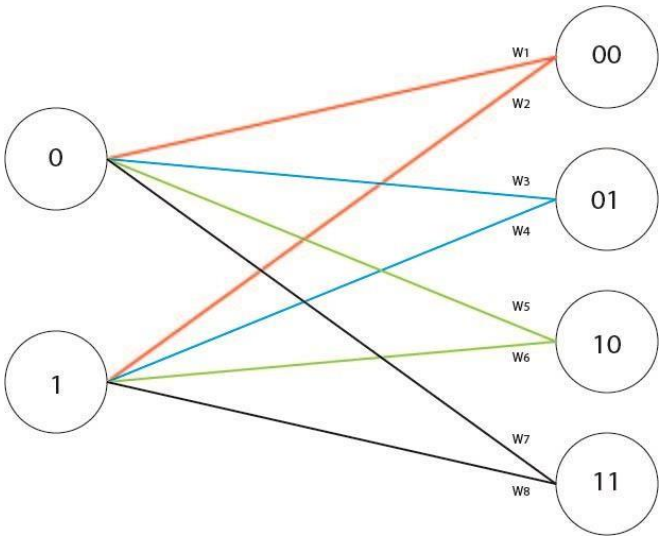

Figure A3: Neural network for 2 bits

Same method with 4 bits and 8 bits:

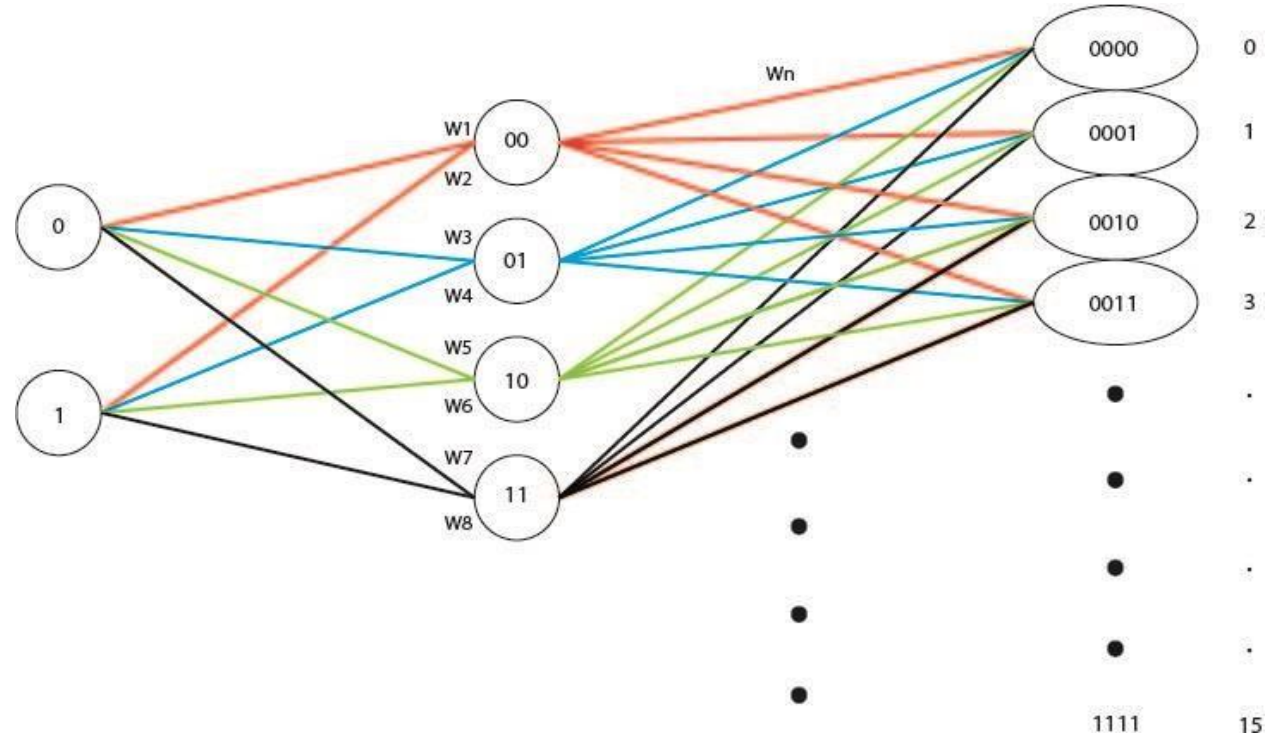

Figure A4: Neural network for 4 bits

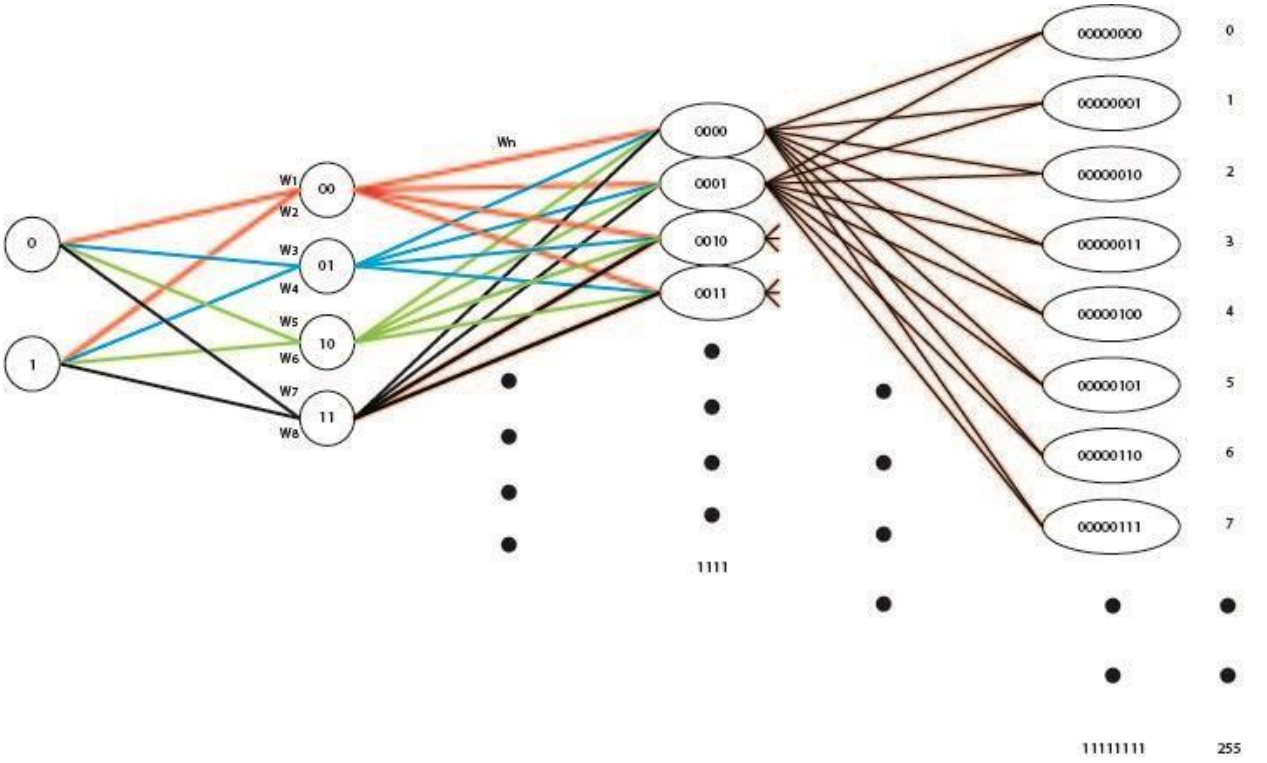

Figure A5: Neural network for 8 bits

Here comes the trick, we could split 8 bits into 4 parts, and it will make things easier. Example:

	AB (first part)	CD (second part)	EF (third part)	FG (forth part)
00000000	00	00	00	00
00000001				01
00000010				10
00000011				11
00000100			01	00
00000101				01
00000110				10
		.		
		.		
		.		
11111111	11	11	11	11

With this split, we could see each part are the combination of 2 bits, and we can group them, and assign a number, k for the order of part to let the machine know which part goes first towards last part. Therefore:

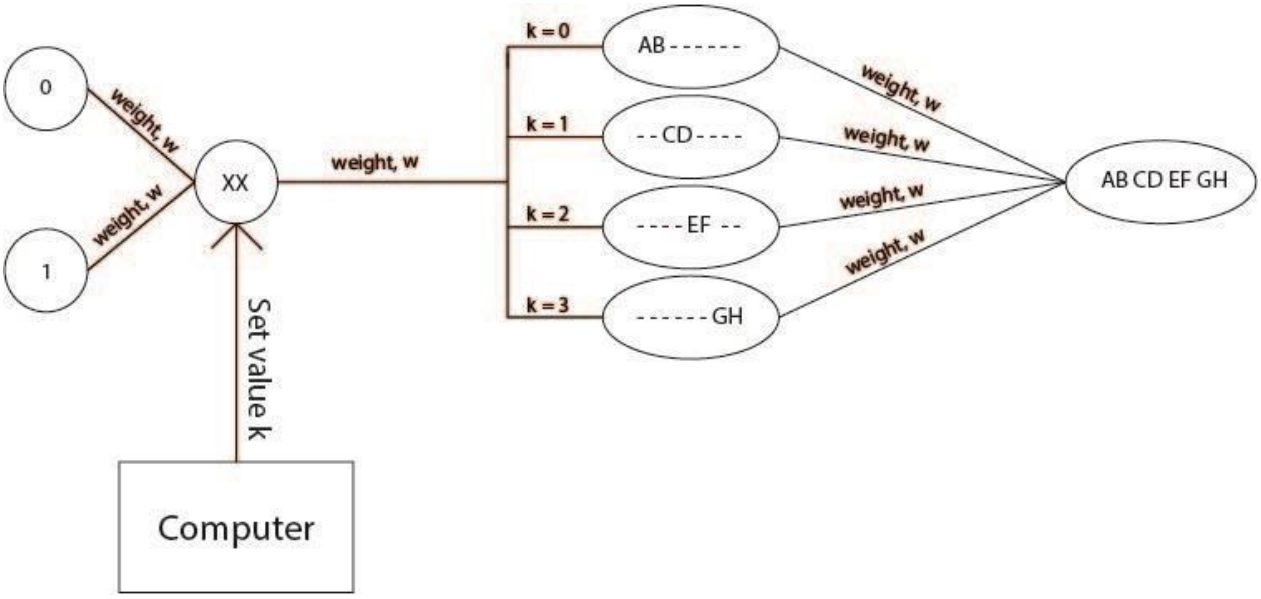

Figure A6: 8 bits into 4 parts block diagram

Furthermore, we can split them from 4 to 2 parts, that means k count from 0 to 7.

	A	B	C	D	E	F	G	H
00000000	0	0	0	0	0	0	0	0
00000001	0	0	0	0	0	0	0	1
00000010	0	0	0	0	0	0	1	0
00000011	0	0	0	0	0	0	1	1
00000100	0	0	0	0	0	1	0	0
00000101	0	0	0	0	0	1	0	1
00000110	0	0	0	0	0	1	1	0
.								
11111111	1	1	1	1	1	1	1	1

Finally, group them into a block, change the computer to control, then add a user:

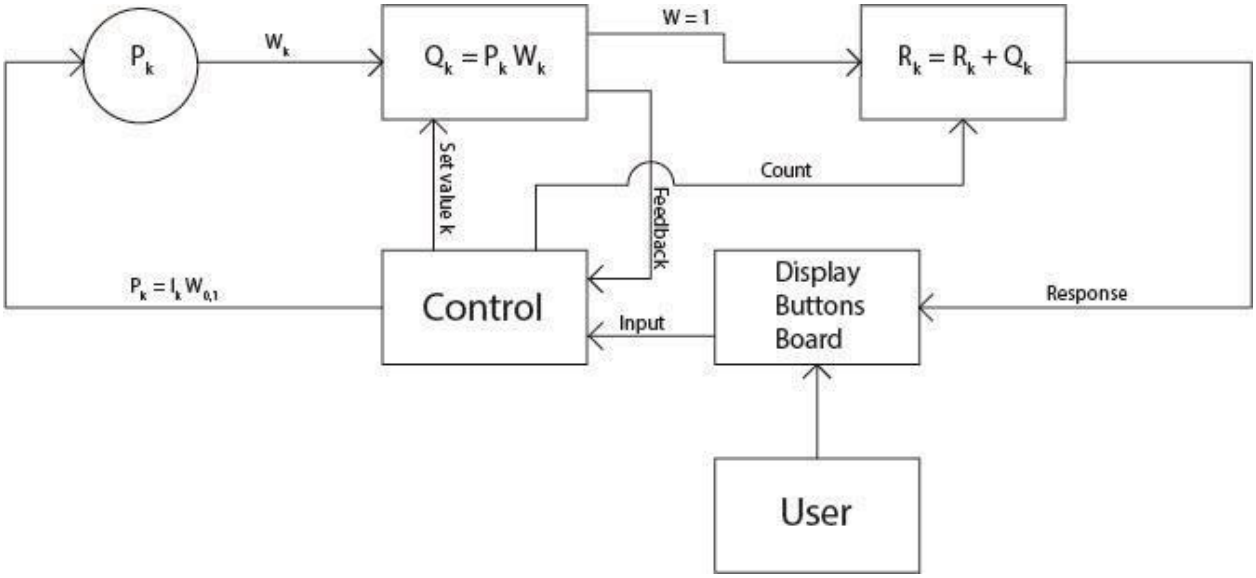

Figure A7: Block diagram of Red-or-Green neural network