MANNIX  
FC + ACTIVATION + SW

2020

Authors: Dor shilo & Eliyahu levi

Date: December 30, 2020

**Document Information**

|  |  |
| --- | --- |
| **Document Title:** | **File Information:** |
| MANNIX | **File Name:** Dor\_shilo\_Eliyahu\_levi\_report  **Last time saved:** December 27, 2020  **Saved by:** Dor Shilo, Eliyahu levi |
| **Keywords:** | |
|  | |
| **Abstract:** | |
| This document is the Specification of the FC + ACTIVATION layers and the SW part of the project. | |

Contents

[**Introduction To Neural Network**: 4](#_Toc59964576)

[Neural Networks 4](#_Toc59964577)

[Fully Connected Neural Networks 5](#_Toc59964578)

[Fully Connected layer – Concepts 5](#_Toc59964580)

[**Activation layer:** 7](#_Toc59964581)

[**Implementation - General Description:** 9](#_Toc59964582)

[**Block Diagram:** 9](#_Toc59964583)

[FC: 9](#_Toc59964584)

[Activation: 9](#_Toc59964585)

[**Interfaces:** 10](#_Toc59964586)

[FC: 10](#_Toc59964587)

[Activation: 10](#_Toc59964588)

[**Schedule:** 11](#_Toc59964589)

[**Project Software** 12](#_Toc59964590)

[1.0 Introduction 12](#_Toc59964591)

[2.0 Mannix software manager 12](#_Toc59964592)

[2.1 Loading data to the memory: 12](#_Toc59964593)

[2.2 MANNIX\_convolution\_layer 12](#_Toc59964594)

[2.3 MANNIX\_pull\_layer 13](#_Toc59964595)

[3.0 software timestamp 13](#_Toc59964595)

[3.1 Step One - Pure software 13](#_Toc59964595)

[3.2 Step Two - Managing a Basic Operating System (Software) 13](#_Toc59964595)

[3.3 Third stage - integration of processing unit's within the hardware 13](#_Toc59964595)

[3.4 Step Four - Create a Python Shell for Code / \* Optional \* / 13](#_Toc59964595)

[**References** 16](#_Toc59964596)

# 

# **Introduction To Neural Network**:

## Neural Networks

A Neural Network is built from groups of neurons that are divided into layers.

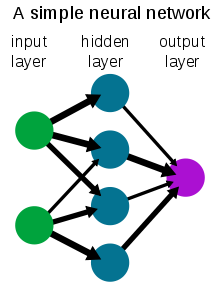
Each and every neuron stores data and is connected to the neurons on the different layers close to his own in a well-defined connection structure.

Neural Networks resemble the human brain neurons – Our own human "AI" – a series of electrical connections that connect one neuron to another in a process called synapses. Those transitions are the key factor of us being able to process information correctly!

Nowadays, Neural Networks idea is being used to solve problems using computers – harnessing computers calculation abilities to process information in the same way human brain's neurons does!

The connection between biological Neural Networks to AI Neural Networks is modeled into weighs. Weight, a number that if positive, reflects an excitatory connection, while negative values mean inhibitory connections. Using those relations, this activity is referred to as linear combination of data and weights granting the computer to process information.

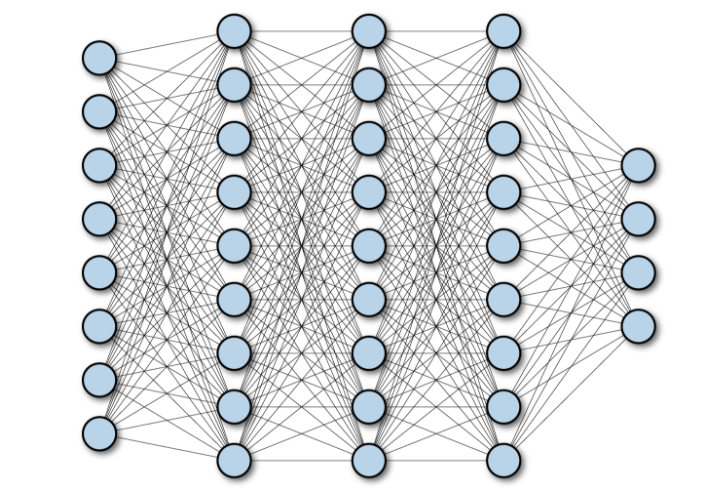
But how does the computer learn?  
One of the most common human behavior is being able to learn from our mistakes and improve.  
This feature is implemented using predictive modeling – The AI start to process the information given to him by "guessing the result" (the computer only controls the weighs values) and by comparing the outcome of his calculation to the correct result provided by the user – the computer learns and adapt in order to be more precise in the next calculation.



## Fully Connected Neural Networks

Neural Networks have different types of architecture that can be used in order to get a different behavior of the network – a faster one, a more precise and any combination of both.  
One of the most basic Neural Network architecture is the Fully Connected Neural Network.

## A Fully Connected Neural Network consists of a series of fully connected layers that connect every neuron in one layer to every neuron in the next layer.



The major advantage of such architecture is that every piece of data is being analyzed – if the network input consists of pictures than every single pixel is passing through the network and Taken into account.  
The liability of such architecture is that it's very big in size (Memory) and usually have a weaker performance.

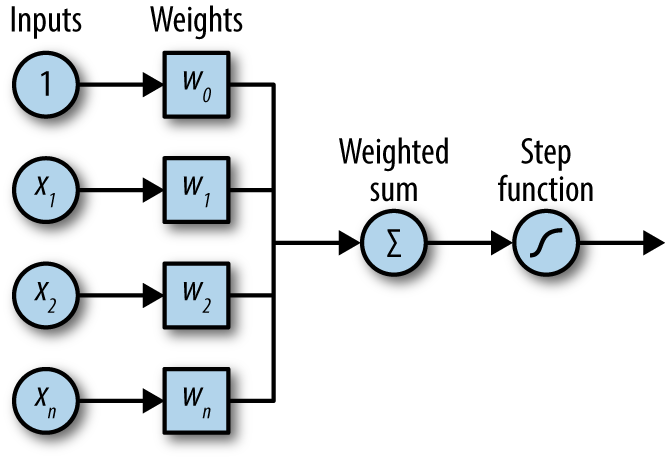
## Fully Connected layer – Concepts

Let's define some basic Concepts:  
In order to make these concepts more understandable we'll follow along with an example –

Let's us try to create a network that deletes every spam email we receive:

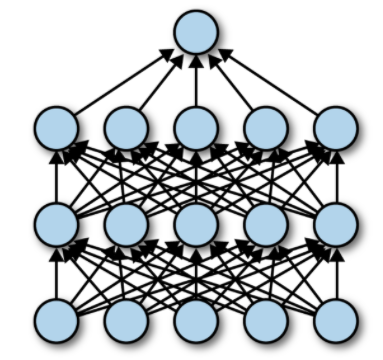
|  |  |  |
| --- | --- | --- |
| Concept | Definition | Example |
| Label | The correct outcome out net is trying to predict. | Is the email we received is spam or not. |
| Feature | an individual measurable property or characteristic of a phenomenon being observed. | * Irregular words * Irregular email address * Time of day |
| Example (concept wise) | A single input:   * Input with label * Input without a label | An email:   * Email I know if spam or not * A new Email The net knows nothing about |
| Hidden layers | Neural network consists of layers – all the layers between the input and output called "Hidden layers" |  |

So now we'll have a look inside a Fully connected layer and understand the way it works:



This is the schematic diagram of a simple Fully Connected layer.  
As we can see – our data (for example every pixel of a by picture - pixels overall ) enter the network.  
The next step is to multiply every pixel in it's corresponding weight value. All those multiplications are being summed to one value that is being normalized using the "Activation function" – we'll discuss on this function shortly.

Another concept called **bias** is added to this scheme – The bias is a single value completing the linear function of the Fully connected layer.

This action happens **in every single neuron** – every neuron in the next layer has it's own weights values corresponding to each neuron in the previous layer.

This suggest that in the output of a layer with M neurons there will be **M different y values**.

Finally We get to the Final layer – where we figure out if the network answer is correct and adjust the weight properly (depends on the amount of different outcomes our system can receive – for example, if the system needs to distinguish between 10 different objects, we will have 10 neurons in the final layer).

# **Activation layer:**

The Activation layer is usually incorporated in the end of a fully connected layer – either as an independent layer in the net or as part of the fully connected layer that came before.

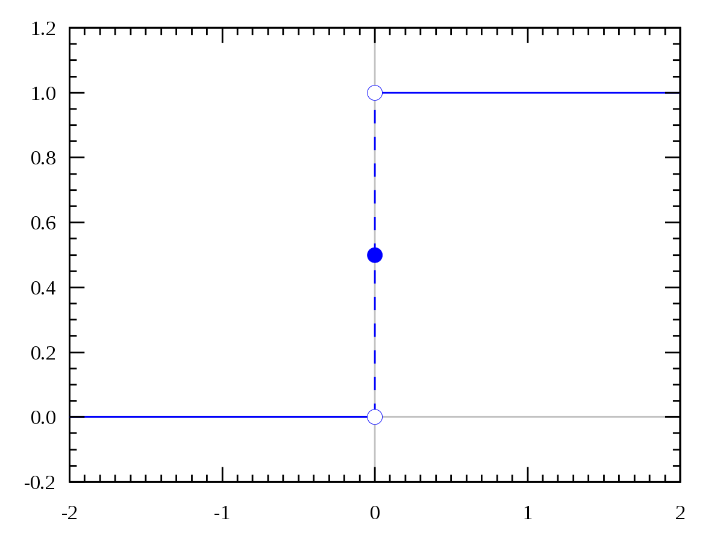
what is this Activation layer and what is it's role?

As we saw in the previous chapter – when data enters a neuron we receive :

Notice the value in the parenthesis can range from to - the neuron does nothing to bound the value (it's mandatory to bound values - especially in hardware!).  
In order to bound this value and normalize all the values, we add **Activation function!**

Here are some examples for different Activation function:

1. **Step function -**This is probably the first function that comes to mind – if Y is below some threshold value, consider it as zero –

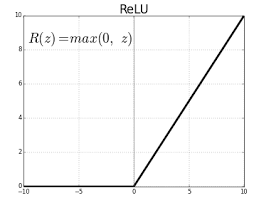


But as you can tell – this ideal function only lives in the dreams of engineers and we must be more realistic.

1. **Relu function –**

Relu function is the "linear" way to solve this problem. Using this function we get a range of values, hence it's not binary activation like the step function –

Where C is the slope.



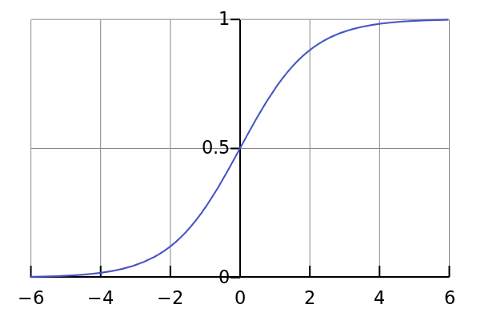
Once again we face the non Continuous problem but in a more gentle way. notice we can still explode to infinity.

On the other hand this function can be a decent option because while eliminating all the negative values, it's still a Hardware buildable function.

1. **Sigmoid function**

Sigmoid function is a well-defined function –

Which gives the plot:



This function does looks smoother and more "step function like" and we get rid of all non-continuous problems – so it's prefect isn't it?!

Well … no! in software it's may be easy – but in MANNIX we deal with bit's so it's harder to define a continues function like sigmoid (it's possible using LUT but it's still on debate in our project).

# **Implementation - General Description:**

The FC block is a part of HW accelerator that makes convolutional Neural Network.  
It is an implementation of the algorithm that was explained in the introduction above.   
It uses a mathematical calculation of dot-product in order to calculate the output matrix.   
FC has an interface to a memory and it's able to ask for data according to the address that has been chosen by the software.

# **Block Diagram:**

## FC:



## Activation:

As of right now – The activation will be a part of the FC module (the multiplication + sum +bias and activation will be part of the ACTIVATION state):



# **Interfaces:**

## FC:

|  |  |  |
| --- | --- | --- |
| Name | I/O | Comment |
| General | | |
| clk | I | clock |
| rst\_n | I | reset negative |
| Memory Interfaces | | |
| mem\_intf\_read\_pic | IF | Read interface from memory |
| mem\_intf\_read\_wgt | IF | Read interface from memory |
| mem\_intf\_write | IF | Write interface to memory |
| Software Interface |  |  |
| fc\_sw\_busy\_ind | O | An output to the software -  1 – FC unit is busy  0 – FC is available (Default) |
| sw\_fc\_if\_vld | I | SW registers can be used/ there was a data change in registers |
| fc\_addr\_x | I | FC Data FIRST address |
| fc\_addr\_z | I | FC return address |
| fc\_xm | I | FC data matrix num of rows |
| fc\_ym | I | FC weight matrix num of rows |
| fc\_yn | I | FC weight matrix num of columns |

## Activation:

As of right now, the activation will be part of the fully connected module.

# **Schedule:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| TASK | Owner | 15-נוב-20 | 22-נוב-20 | 29-נוב-20 | 06-דצמ-20 | 13-דצמ-20 | 20-דצמ-20 | 27-דצמ-20 | 03-ינו-21 | 10-ינו-21 | 17-ינו-21 | 24-ינו-21 | 31-ינו-21 | 07-פבר-21 | 14-פבר-21 | 21-פבר-21 | 28-פבר-21 | 07-מרץ-21 | 14-מרץ-21 | 21-מרץ-21 | 28-מרץ-21 | 04-אפר-21 | 11-אפר-21 | 18-אפר-21 | 25-אפר-21 | 02-מאי-21 | 09-מאי-21 | 16-מאי-21 | 23-מאי-21 | 30-מאי-21 | 06-יונ-21 | 13-יונ-21 | 20-יונ-21 | 27-יונ-21 |
|  |  |  |  |  |  |  |  |  |  |  | מבחנים | מבחנים | מבחנים | מבחנים |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Initial Design Spec writing | Dor |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Initial skeleton implementation | Dor |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Base Version implementation | Dor |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Final Design spec update | Dor |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Final version implementation | Both |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| ASIC feasibility check (synthesis) | Dor |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Productizing and documenting | Both |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| In parallel to all - continues verification | Both |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

# **Project Software**

## 1.0 Introduction

The MANNIX HW accelerator has software tools. The reasons for that are:

1. **Profiling** – writing a model in C or other high-level language to find the bottleneck that we want to accelerate.
2. **Verification** – the system needs to be bit accurate, which mean that the output of the accelerator must be equal (byte resolution) to the output of a software program that simulates the same operation.
3. **Modularity** – We want the project to be modular and flexible to change. The best way to do so in minimal cost is to manage it with software program.

Those three demands will affect the way the program will be written.

## 2.0 Mannix software manager

Following the introduction, we decided to use this model:

#define ADDRX /\*BASE POINTER OF THE ADDRESS WHERE THE IMAGE WILL BE LOADED \*/

#define ADDRY /\*BASE POINTER OF THE ADDRESS WHERE THE WEIGHTS WILL BE STORED\*/

#define ADDR\_CONV\_RES /\*BASE POINTER OF THE ADDRESS WHERE THE CONVOLUTION LAYER RESULT WILL BE STORED\*/

#define ADDR\_ACTIVATION\_RES /\*BASE POINTER OF THE ADDRESS WHERE THE ACTIVATION LAYER RESULT WILL BE STORED\*/

#define ADDR\_PULL\_RES /\*BASE POINTER OF THE ADDRESS WHERE THE MAX PULL LAYER RESULT WILL BE STORED\*/

#define RESULT /\*\*/

define COUNTER /\*SAVE THE NUMBER OF IMAGE \*/

void MANNIX\_NN(image\_ptr\*, weights\*, bias\*) {

    load(image\_ptr, ADDRX); load(weights, ADDRY); load(bias, ADDRY + B);

MANNIX\_convolution\_layer       (ADDRX,  ADDRY,  ADDR\_CONV\_RES);

MANNIX\_non\_linearity\_activation(ADDR\_CONV\_RES, ADDRY, ADDR\_CONV\_RES);

MANNIX\_pull\_layer              (ADDR\_CONV\_RES, ADDRY, ADDR\_PULL\_RES);

MANNIX\_fully\_conneted          (ADDR\_PULL\_RES, ADDRY, RESULT);

}

## 2.1 Loading data to the memory:

**Description:**

Since we are working on a FPGA device which is much slower then ASIC, the first step will be to allocate a memory area to the incoming data.

## 2.2 MANNIX\_convolution\_layer

**Description:**

The convolution layer - The program must send to the processing unit the starting address of data ( ADDRX), the start address of the weights (ADDRY) and the return address to save the output (ADDRZ). It also must send the data length (Xm), it's width (Xn), the window length (Ym) and it's width (Yn).

## 2.3 MANNIX\_pull\_layer

**Description:**

The software must send to the processing unit the start address (ADDRX) and the return address (ADDRZ). It is also must send windows length (Xm), windows width, (Xn),the resulting matrix length (Pm) and the width of the matrix (Pn).

2.4 MANNIX\_non\_linearity\_activation

**Description:**

The unit must send the return address (ADDRZ), this address is also the input address. The software must send the window length (Xm) and window width (Xn). The method of execution will be ReLU.

2.5 MANNIX\_fully\_conneted

**Description:**

The software must send to the processing unit the start address (ADDRX), the weights address (ADDRY), the base address (ADDRB) and the return address (ADDRZ). It is also must send the input vector length (Xn), weight vector length (Ym), weight vector width (Yn) and base length (Bn).

2.6 Variable memory allocation

In the previous paragraph we mentioned many different variables that must be allocated to memory.

As of right now – the allocation is described in the following table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| DOCUMENT | REGISTER | | SOFTWARE | | MANNIX |
| Operation | Address (offset) | | Size | | Name |
|  |  | |  | | CNN LAYER |
| CNN Data window address | 0x0000 | | 31:0 | | **CNN\_ADDRX** |
| CNN weights window address | 0X0004 | | 31:0 | | **CNN\_ADDRY** |
| CNN return address | 0x0008 | | 31:0 | | **CNN\_ADDRZ** |
| CNN input length | 0x000B | | 31:0 | | **CNN\_XM** |
| CNN input width | 0x0010 | | 31:0 | | **CNN\_XN** |
| CNN weights length | 0x0014 | | 31:0 | | **CNN\_YM** |
| CNN weight width | 0x0018 | | 31:0 | | **CNN\_YN** |
|  |  | |  | | PULLING LAYER |
| PULL Data window address | 0x001B | | 31:0 | | **PL\_ADDRX** |
| PULL return address | 0x0020 | | 31:0 | | **PULL\_ADDRZ** |
| PULL input length | 0x0024 | | 31:0 | | **PULL\_XM** |
| PULL input width | 0x0028 | | 31:0 | | **PULL\_XN** |
| PULL result length | 0x002B | | 31:0 | | **PULL\_PM** |
| PULL result width | 0x0030 | | 31:0 | | **PULL\_PN** |
|  |  | |  | | ACTIVATION LAYER |
| ACTIVATIN Data window address | 0x0034 | | 31:0 | | **ACTIV\_ADDRX** |
| ACTIVATION weights window address | 0X0038 | | 31:0 | | **ACTIV\_XM** |
| ACTIVATION return address | 0x003B | | 31:0 | | **ACTIV\_XN** |
|  |  | |  | | FULLY CONNECTED |
| FULLY CONNECTED Data vector address | | 0x0040 | | 31:0 | **FC\_ADDRX** |
| FULLY CONNECTED weights window address | | 0X0044 | | 31:0 | **FC\_ADDRY** |
| FULLY CONNECTED bias vector address | |  | |  | **FC\_ADDRB** |
| FULLY CONNECTED return vector address | | 0x0048 | | 31:0 | **FC\_ADDRZ** |
| FULLY CONNECTED input vector length | | 0x004B | | 31:0 | **FC\_XM** |
| FULLY CONNECTED weights window length | | 0x0050 | | 31:0 | **FC\_YM** |
| FULLY CONNECTED weights window width | | 0x0054 | | 31:0 | **FC\_YN** |
| FULLY CONNECTED bias vector length | | 0x0058 | | 31:0 | **CNN\_BN** |

Note - this is the temporary table! Changes will be made as we move along with the project!

3.0 software timestamp

The software program will be divided to two parts:

1) pure software  
2) writing to the processing unit.

3.1 Step One - Pure software

**Description:**

Building software that simulates the behavior of a MANNIX accelerator. The program will fit the software structure mentioned in the introduction. The behavior of the software will be serial, that is - no unit will start work before it's predecessor has finished. In addition, each function will simulate the behavior of one of the processing unit's in such a way that the input and output of every function will be identical to the processing unit which it replaces.

**Goal:**

* Creating a memory management software shell
* building a model that is easier to test and integrate.
* Modularity.

**Expected date:** 12/02/21

**Notes:**

* The model will be written in C in windows operating system in order to facilitate the transition to the RISC-V code later on.
* The chosen dataset is fashion emnist – this is temporary and progress dependent.

3.2 Step Two - Managing a Basic Operating System (Software)

**Introduction:**

Instead of waiting for it's processing to be completed, new data is sent once it is possible. When we come to do this, we encounter two problems:

1. **Memory allocation management** – for Each image or data that arrives the program must know where it is located.
2. **Address Management for Processing Unit's** - Each processing unit handles information independently of the other unit's. This creates a problem when several images are waiting for the same unit.

The solution to the first problem would be to track an address index. The solution to the second problem would be to manage an address queue for each unit.

**Goal:**

Creating a memory management model in such a way that it can be replaced by a hardware mechanism.

**Notes:**

* This step will also be managed in the software only.
* Adding parallel software components (threads) in order to simulate the hardware mechanism. The intention is to create a situation where several unit's are waiting for the same function.
* Once we have finished processing the image, it's place in memory will be vacated.

**Expected date:** Passover 2021.

* 1. Third stage - integration of processing unit's within the hardware

**introduction:**

in order to use mannix accelerator we need to write to the gpp. We want to create functions that do so and replace the software functions.

**Goals:**

* Implement each hardware module built into the software system separately.
* Convert program code to RISCV code.

**Notes:**

* This phase may be parallel to stage 2 depending on the pace of progress of the construction of the processing unit's.

**Expected date:** two weeks after Passover.

* 1. Step Four - Create a Python Shell for Code / \* Optional \* /

# **References**

FC Intro:

[1] [Adi](https://www.sciencedirect.com/topics/engineering/convolutional-layer) teman course in BIU – "From HW to DL"

[2] <https://www.oreilly.com/library/view/tensorflow-for-deep/9781491980446/ch04.html>

Activation:

[3] https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0