Neural Networks for physics

...

Simone Rodini

April 7, 2024

Contents

Outline

How does a computer think?

1. How does a digital computer actually think? A brief window on machine instructions, assembly and programming languages.
2. How can we make a higher-level model of intelligence (or logical reasoning) on a machine? Neural Networks demystified.
3. A specific application: how to use a NN to model an unknown physics phenomena.

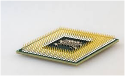
Computer basics

A general idea

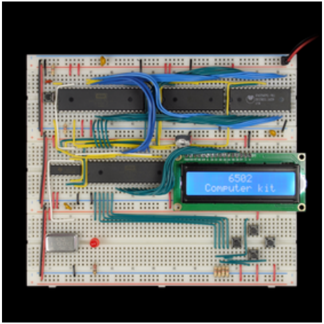
 

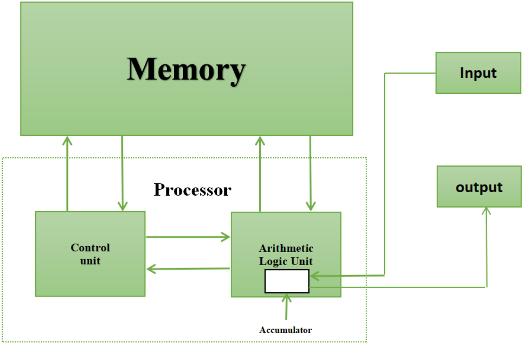
Transistors: the foundation

A general idea

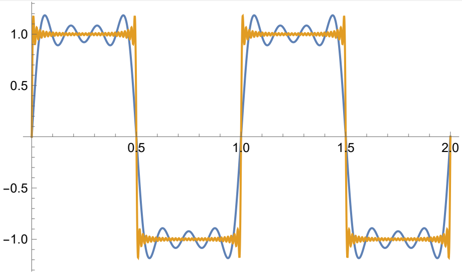
Transistors: the foundation





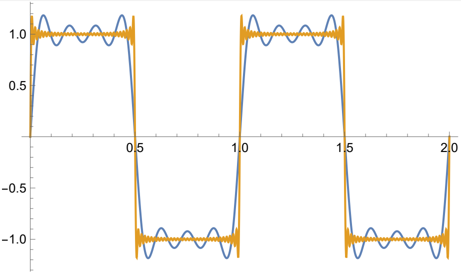
Number representation

All digital computer works storing information as voltage level.
We can store 2 values, conventionally referred to as 0, 1.



Number representation

All digital computer works storing information as voltage level.
We can store 2 values, conventionally referred to as 0, 1.



Why not more? Possible, but it has costs: energy, reliability, scalability etc...
How can we represent numbers using only $0, 1$? Binary!
How to read a base-10 number?

$$67105971\underset{\hookleftarrow}{11}_{10} \equiv 1 \cdot 10^0 + 1 \cdot 10^1 + 1 \cdot 10^2 + 7 \cdot 10^3 + \ldots$$

So, binary = base-2:

$$11011011_2 = 1 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3 + \ldots$$

Number representation

Nowadays hardware can manage 64 bit numbers: up to $2^{63} \lesssim 10^{19}$.
Inconvenient to think in binary $\Rightarrow$ hexadecimal! Use 0 and other 15 symbols to represent numbers: $0, 1, ..., 9$,A,B,C,D,E,F: $\underbrace{1101}_{\text{D}}\underbrace{1011}_{\text{B}} = \text{DB}_{16}$ With actually 64 bits:

$$\underbrace{1101}_{\text{D}}\underbrace{0101}_{\text{5}}\underbrace{0110}_{\text{B}}\underbrace{1111}_{\text{7}}\underbrace{0111}_{\text{C}}\underbrace{0001}_{\text{5}}\underbrace{0110}_{\text{B}}\underbrace{1111}_{\text{7}}\underbrace{0111}_{\text{7}}\underbrace{0111}_{\text{B}}\underbrace{0110}_{\text{7}}\underbrace{1111}_{\text{D}}\underbrace{1100}_{\text{3}}\underbrace{1101}_{\text{A}}\underbrace{0111}_{\text{7}}\underbrace{1011}_{\text{B}}$$
$$= \text{D5B7C5B77B7D3A7B}$$

1 bit: one 1 or 0, 1 byte: eight 1 and 0 like $01000010 \equiv$ two hex number 42.
64 bits $\equiv 8$ bytes.

Addition table:

	0	1
0	0	1
1	1	0*

and multiplication table:

	0	1
0	0	0
1	0	1

*: carrying the 1, like in $9 + 1 = 0$ plus the carried 1.

Beyond positive integers

From any base n to decimal is simple. From base-10 to any base: quotient-remainder algorithm: a tiny bit more annoying to carry out.
Positive integers ✓
Negative integers? Use the most significant bit (the leftmost one) to encode the sign: $0 \Rightarrow$ positive. In formulas

$$b_{N-1}b_{N-2}...b_0 = -b_{N-1}2^{N-1} + \sum_{i=0}^{N-2} b_i 2^i$$

Beyond positive integers

From any base n to decimal is simple. From base-10 to any base: quotient-remainder algorithm: a tiny bit more annoying to carry out.

Positive integers ✓

Negative integers? Use the most significant bit (the leftmost one) to encode the sign: $0 \Rightarrow$ positive. In formulas

$$b_{N-1}b_{N-2}...b_0 = -b_{N-1}2^{N-1} + \sum_{i=0}^{N-2} b_i 2^i$$

For example:

$$1001_2 = -1 \cdot 2^3 + 1 \cdot 2^0 = -8 + 1 = -7$$

Check: $7_{10} = 0111_2$ and

$$\begin{array}{r} 0^1\,1^1\,1^1\,1+ \\ 1\;\;0\;\;0\;\;1= \\ \hline \color{red}{1}0\;\;0\;\;0\;\;0 \end{array}$$

Floating Points

How to represent numbers that are not integers?
In binary we can use inverse powers of 2, like $0.2 = 2 \cdot 10^{-1}$ in binary $0.1_2 = 1 \cdot 2^{-1}$.
On a computer only so many bits: no real numbers!
Could we reserve some bits for the part after the period? Like

$$\underbrace{101...1}_{\text{32-bit}}.\underbrace{110...1}_{\text{32-bit}}$$

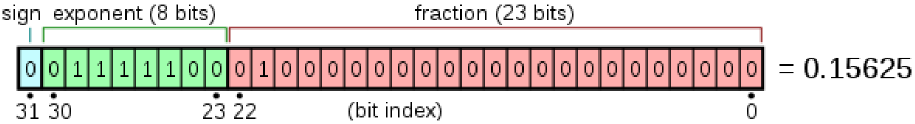
Yes, but impractical: sometimes we want very large numbers, sometimes very small, this has no adaptation capability.

Floating Points

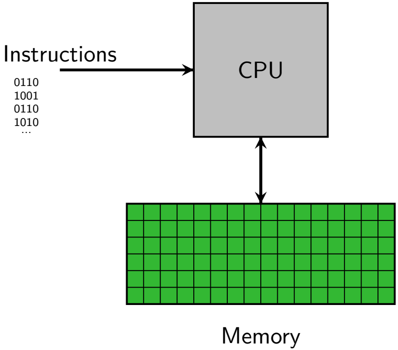
How to represent numbers that are not integers?
In binary we can use inverse powers of 2, like $0.2 = 2 \cdot 10^{-1}$ in binary $0.1_2 = 1 \cdot 2^{-1}$.
On a computer only so many bits: no real numbers!
Could we reserve some bits for the part after the period? Like

$$\underbrace{101...1}_{\text{32-bit}} . \underbrace{110...1}_{\text{32-bit}}$$

Yes, but impractical: sometimes we want very large numbers, sometimes very small, this has no adaptation capability.
Better: scientific notation! For a 32-bit number:



How does we instruct the computer



How does it work (in a very approximate way)?

Level 0: BIOS/UEFI = basic software built-in in the motherboard, manages the very low level hardware

Level 1: OS = manages all the software, it is loaded by the BIOS/UEFI by reading instructions off a particular region of the memory

Level 2: programs = the OS manages your programs, the memory for each of them, where they store information etc.

How does we instruct the computer

On a 64 bit machine each instruction can be up to 15 bytes long(!). The 64 bit refers to the maximum size of each register in the cpu.
Each instruction is literally a sequence of $0, 1$ that triggers some electronics to do 'stuff', like adding two numbers, moving memory, jump to different addresses etc.
It is not manageable to write instructions in binary (or hex) format:

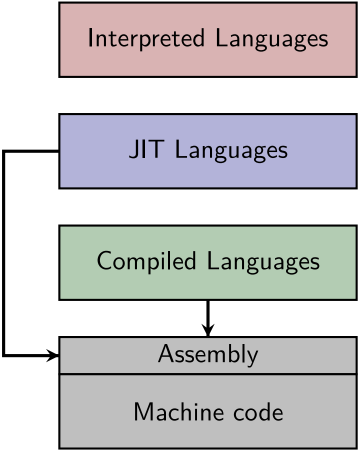
human-readable machine code $\sim$ assembly ASM

```
push rbp
mov  rbp, rsp
mov  DWORD PTR [rbp-4], 1
mov  DWORD PTR [rbp-8], 2
mov  edx, DWORD PTR [rbp-4]
mov  eax, DWORD PTR [rbp-8]
add  eax, edx
mov  DWORD PTR [rbp-12], eax
mov  eax, 0
pop  rbp
ret
```

More readable than a buch of 1 and 0, but still a long way from its C counterpart:

```
int a = 1;
int b = 2;
int c = a + b;
```

Programming languages



Examples

What does it mean to 'compile' a language?

$$\text{Human-readable code} \Longrightarrow \text{Machine code}$$

It is a translation!

Python: interpreted language. What does it mean? It reads the commands, parse them, construct on the fly the logical tree and execute them via previously compiled functions.

Lua: JIT (just-in-time compiled) language. It reads pieces of code, compile them on-the-fly and execute the resulting binary.

C/C++: compiled languages. There is a dedicated piece of software (the compiler) that translates the code to machine instructions. This is done separately from the running of the progrma itself.

Your turn now, let us play a bit!

Neural Networks, the basics