Assignment 1 INT32 versus FP32

Name: Rahul Vigneswaran K

Roll: CS23MTECH02002

Subject: Hardware Architecture for Deep Learning (CS6490)

> Understand the range and precision for INT32 and single-precision FP32 data types. Explain how FP32 can support a larger range as well as better precision than INT32, while both of them can represent 2^32 unique numbers. Illustrate with examples of numbers that can be represented in INT32 but not in FP32 and vice versa. Submit a pdf file.

Part 1: Understand the range and precision for INT32 and single-precision FP32 data types.

Understanding Range and Precision:

Let's take a set of fake data types called **tint** and **mint**.

Range:

- Say if **tint** can represent numbers from 1 to 10. That means the range of **tint** is 10. If **mint** can represent numbers from 1 to 15, that means the range of **mint** is 15.
- We can effectively say that mint has a higher range than **tint**. So, the range of a data type is how large of a number you can cover using that specific data type.
- Say if **tint** can represent numbers between 1 and 10 and **mint** can represent numbers between 11 and 20. Here even though there is no intersection between the numbers that they both can represent and the magnitudes of values of **mint** is higher than **tint**, both **mint** and **tint** have the same range.

Precision:

- Now say **tint** and **mint** both have a range of 20 but **tint** can only represent multiples of 5 and **mint** can only represent multiples of 10.
- That means,
  - **tint** can represent: 5,10,15,20
  - **mint** can represent: 10, 20
- In the above case, even though both **tint** and **mint** have the same range, that is it can cover numbers as high as 20, **mint** can only represent the numbers 10 and 20, that is, **mint** is only precise enough to represent only those two numbers and nothing in-between them while **tint** can represent the number 15 in between 10 and 20.
- In other words, precision is the distance between the successive numbers that the datatype can represent. Therefore the precision of **tint** and **mint** are 5 and 10 respectively.

Understanding INT32 and FP32

First, let's understand each of them individually and then do a comparison.

INT32

- INT32 is a 32-bit signed integer datatype. In this 1 bit is reserved for the sign as this is a signed integer.
- This means we have 31 bits for the integers. 2^31 = 2147483648. The left-most bit would indicate the sign followed by the bits indicating the integer. Therefore INT32 can cover numbers ranging from -2147483648 to 2147483647. There reason why 2147483648 numbers are on the negative side but only 2147483647 on the positive side is because we have also included 0.
- Note that INT32 can represent only integers and there is no way for it to represent fractions. So the minimum possible distance between any of its successive numbers is 1. Therefore the precision is 1.

> **Range: -2147483648 to 2147483647**

> **Precision: 1**

FP32

- FP32 is a 32-bit signed floating point datatype. Therefore we have 32-bits at our disposal. In this 1 bit is reserved for the sign.

- Now we are left with 31 bits. Unlike INT32, this accommodates floating points too. So we have to reserve some bits for the decimal places. This can be done is any way but according to the IEEE 754 format which is universally accepted as FP32, Out of the 32 bits, 8 bits are allotted for the exponent and 23 bits are allotted for the fraction part (also known as mantissa bits).

  Lets first understand the range of the individual parts of the FP32, this would help us build to the actual range and precision.

Components of FP32:

- Sign
  - +ve:0 , -ve:1
- Exponent
  - We have 8 bits. And in that
    - NaN
      - 11111111 indicates NaN errors. But also make sure not all digits of mantissa are 0.
    - Absolute zero
      - When represent numbers in FP32, we go through a step called normalisation which is out of the scope of this assignment. But the thing to note is that when we normalize we always implicitly assume that the fraction part would have the binary digit 1 to the left of the point.
      - But when we set the exponent to 00000000, we set that leading number to be 0. This would allow even lower values compared to the smallest normalized numbers.
    - So we have the remaining 2^8-2 = 254
  - Offset bias:
    - In FP32 we use a offset binary format in order to represent both -ve and +ve exponents without using a dedicated sign bit. For FP32, this offset bias is 127. So whatever number we are able to represent using the 256 available numbers, we have to subtract it with 127 in order to obtain the actual exponent. Using any number above 127 would yield a +ve exponent and anything below 127 would yield a -ve exponent. Except when we set all exponent bits to 0, in that case we keep the exponent as -126 and not 0-127.
    - So to understand the actual range of the exponent we need to find the min and max of (a-127). The range of a is as we saw earlier 1 to 254.
  - So the effective range the exponent is -126 to +127.
- Mantissa
  - We have 23 bits. Since in IEEE 754 format we normalize it, all the bits would be used to represent the numbers after the point and implicitly assume that there is leading 1 before the point. So the maximum fractional part 0.11111111111111111111111 (23 times 1s). But we know that there is implicitly a leading 1. So 1.11111111111111111111111.
  - On its own, the range of mantissa alone is 1 to 1.11111111111111111111111. But as we saw earlier, if we can set all exponent to 0. Then the range would be 0.11111111111111111111111 to 1.11111111111111111111111.
  - Inf:
    - If all digits of exponents are set 1 and all digits of mantissa are set to 0, it means infinity.

Now that we have understood the range of each part of the FP32 properly, lets try to understand the range and precision of FP32 as a whole.

Range of FP32

- Technically the range of FP32 is -inf to +inf as you can set all exponents to 1 and mantissa to 0 and just change the sign bits but that is not in general considered as range. So we will ignore those cases and derive the range.
- +ve range:
  - Min number (leftmost on +ve number line):
    - Sign bit: 0
    - Exponent bits: 00000000
    - Mantissa bits: 00000000000000000000001
    - Final number: + 0.00000000000000000000001 x 2^-126 = + 1.4012984643248170709237295832 9 x 10^−45 (approx)
  - Max number (rightmost on +ve number line):
    - Sign bit: 0
    - Exponent bits: 11111110
    - Mantissa bits: 11111111111111111111111
    - Final number: + 1.11111111111111111111111 x 2^127 = + 3.40282346638528859811704183485 x 10^+38 (approx)
- -ve range:
  - Min number (leftmost on -ve number line):
    - Sign bit: 1
    - Exponent bits: 11111110
    - Mantissa bits: 11111111111111111111111
    - Final number: - 1.11111111111111111111111 x 2^127 = - 3.40282346638528859811704183485 x 10^+38 (approx)
  - Max number (rightmost on -ve number line):
    - Sign bit: 1
    - Exponent bits: 00000000
    - Mantissa bits: 00000000000000000000001

    - Final number: - 0.00000000000000000000001 x 2^-126 = - 1.4012984643248170709237295832 9 x 10^−45 (approx)
- Absolute zero:
  - Sign bit: 0 or 1
  - Exponent bits: 00000000
  - Mantissa bits: 00000000000000000000000
  - Final number: +/- 0.00 x 2^-126 = +/- 0

Range of FP32 is - 3.4 x 10^38 to + 3.4 x 10^38

Precision of FP32

- Unlike in INT32 where the precision is constant throughout the entire range, in FP32 the precision varies alot depending on the the exponent and mantissa part.
- As we saw earlier the precision is the difference between two consecutive number. So lets see the lower bound and upper bound of it on the positive side of the number line.
- Lower bound (High precision):
  - N1: | 0 | 00000000 | 00000000000000000000001 |
    - - 1.40129846432481707092372958329 x 10^−45
  - N2: | 0 | 00000000 | 00000000000000000000000 |
    - - 0
  - Precision:
    - N1-N2: 1.40129846432481707092372958329e−45 - 0 = 1.4 x 10^-45 (approx) (which is very small)
- Upper bound (Low precision):
  - N1: | 0 | 11111110 | 11111111111111111111110 |
    - - 3.40282326356119256160033759537 x 10^+38
  - N2: | 0 | 11111101 | 11111111111111111111111 |
    - - 3.40282346638528859811704183485 x 10^+38
  - Precision:
    - N1-N2: = 2.02 x 10^+31 (approx) (which is alot)

Precision of FP32 ranges from 1.4 x 10^-45 to 2.02 x 10^+31

So the larger the number FP32 tries to represent, the less precise it gets. The smaller the number it tries to represent, the more precise it gets. Also, due to this lack of precision at higher numbers FP32 cant represent the exact number at higher values.

Part 2: Explain how FP32 can support a larger range as well as better precision than INT32, while both of them can represent 2^32 unique numbers.

Range

Both INT32 and FP32 can represent 2^32 unique values because both have 32 bits, but there is no exponent term in INT32. Because of this the largest number it can represent is 2147483647 which is approximately 2.1 x 10^9 and the smallest number is the negative of it. Since FP32 supports exponent term, the largest number it can support is + 3.4 x 10^38 and the smallest is the negative of it. As we can see that the range is much more higher for FP32 when compared to INT32.

Precision

We saw earlier that the precision of FP32 varies alot as we go from 0 to a higher number. While we can say that FP32 has better precision than INT32 at lower numbers, as we go higher toward the maximum INT32 value, the precision gets poorer.

For example lets take the maximum number that INT32 can represent 2,147,483,647,

INT32: The difference between 2,147,483,647 and the next lowest number to it can represent 2,147,483,646 is 1.

FP32: We cant even represent 2,147,483,647 in FP32. So we will take the example of the nearest number FP32 can represent 2,147,483,648. The value prior to it that can be represented by FP32 is 2,147,483,520. The difference between them is 128 which is much higher than 1.

Therefore, FP32 can support a larger range as well as better precision than INT32 "in general" even though they both can represent 2^32 uniques numbers. But there is a trade-off and for higher values, INT32 is more precise than FP32.

Part 3: Illustrate with examples of numbers that can be represented in INT32 but not in FP32 and vice versa.

Due to the mismatch in range and precision that we saw earlier, there is a trade-off in using INT32 and FP32. As we saw in the previous part FP32 is more precise than INT32 at lower numbers but is less precise at higher numbers. Also FP32 has a higher range when compared to INT32.

Example of numbers represented by INT32 but not FP32:

- 2,147,483,647
    - This can be represented exactly by INT32 but the nearest number that FP32 can precisely represent is 2,147,483,648 which is number higher.
- Like we saw in part 2, FP32 looses precision as the number gets higher.

Example of number represented by FP32 but not INT32:

- 2,147,483,647
    - Like we saw in part 1, FP32 has a much higher range than INT32. The maximum value of INT32 is 2,147,483,647. So it cannot represent any value beyond it.
- 1.5
    - Due to lack of ability to represent fractions, INT32 cannot represent any value that is not an integer.

Acknowledgment

I used Omnicalculator tool (https://www.omnicalculator.com/other/floating-point) for converting FP32 notation to base10. I use it only for the final step, not the entire calculation.