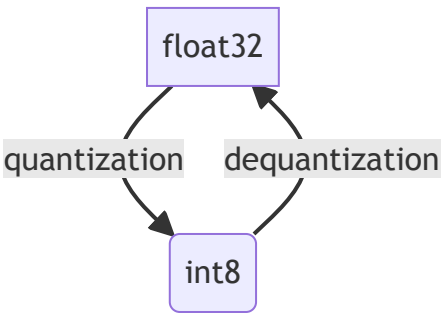
20221119&20_quantization

Quantization

> Quantization for deep learning is the process of approximating a neural network that uses floating-point numbers by a neural network of low bit width numbers.

Today we're going to explore what is meant by quantization in deep learning and why we need to quantize network parameters!! Deep learning uses neural networks to learn and identify complex patterns in data. A neural network is made up of series of interconnected neurons which is considered as layers. Each layer have their own weights and biases and are called as the parameters of neural network. During training these weights will get updated based on the loss function and they'll get stored in memory by neural network. Normally the standard choice is to represent these numbers in 32-bit-floating point values, and it'll provide high accurcy. That's why neural network consumes lots of memory. Usually neural network will contain millions of parameters, and each being stored as float32, memory usage will definitely add up. To deal with this huge memory consumption researchers introduced quantization.

Quantization is the process of reducing precision of weights such that they consume less memory. ie. it is the process of converting data in float32 to smaller precision like int8 and perform operations like convolution, maxpool, etc. and convert the lower precision output back to higer precision float32. The neural network can be quantized after training is finished. However, by far the most effective method for retaining high accuracy is to quantize during training.



There are two basic operations in quantization :

- quantization : converting data to lower precision like int8
- dequantization : converting data back to high precision like float32

Quantization is the starting operation and dequantization is the final operation in the process

---

Precision reduction in quantization

How precision is reduced while quantization, let's make it clear through some examples

Suppose we need to multiply two floating point numbers :

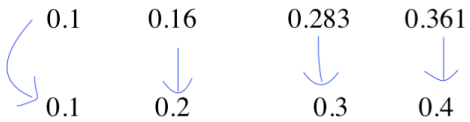
- scale used : 2^15
    - ```
      a = 0.29
      b = 1.04
      s_15 = 2**15
      a*s_15 = 9502.72
      b*s_15 = 34078.72
      ## removed the decimal points
      a_15, b_15 = 9502, 34078
      a*b = 0.3016
      (a_15*b_15)/s_15**2 = 0.3015707768499851
      loss = 0.3016-0.3015707768499851 = 2.9223150014856536e-05
      ```
    - First I assigned values 0.29 & 1.04 to the variables a and b respectively. Then I selected a sacale 2^15 for the process and multiplied each numbers with this scale. Then removed the decimal part of the obtained values and take them as the int8 equivalents of a and b. ie. 9502.72 is taken as 9502 and 34078.72 is taken as 34078. So from now onward our a and b are 9502 & 34078. Multiplying these int8 values & dividing them with square of scale, and multiplication between 0.29 & 1.04 should give us same value. Some loss in precision is acceptable. as we can see the loss from above example is extremely small. So it is a simple example of quantization and the resulting reduction in precision.
- scale used : 2^10
    - ```
      a = 0.29
      b = 1.04
      s = 2**10
      a*s = 296.96
      b*s = 1064.96
      a_10, b_10 = 296, 1064
      a*b = 0.3016
      (a_10*b_10)/s**2 = 0.30035400390625
      loss = 0.3016-0.30035400390625 = 0.0012459960937499792
      ```
- scale used : 2^5
    - ```
      a = 0.29
      b = 1.04
      s = 2**5
      a*s = 9.28
      b*s = 33.28
      a_5 = 9
      b_5 = 33
      ```
      ```

```
(a_5*b_5)/s**2 = 0.2900390625
a*b = 0.3016
0.3016 - 0.2900390625 = 0.01156093749999998
```

- These three examples shows that as we decrease the scaling factor loss is getting increased.

Data types

In deep learning we need to deal with vast amount of data. These data can be stored in different datatypes. We generally deals with continuous data and discretization of them is know as quantization.



Data ranges from `-∞ to +∞`. for example 1/3 is 0.3333... and its decimal points are of infinite in number. So storing such values as it is will eat up a lot of memory. So we'll make them discrete for the sake of memory.

Basics

8 bit = 1 byte

32 bit = 4 byte

Integer Data types

Integer data are of two types : Signed and Unsigned , Signed and Unsigned hyperinteger

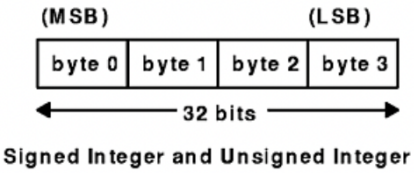
Signed and Unsigned

ref-ibm ref2

Unsigned type of int holds zero and +ve numbers while signed holds -ve ,zero and +ve numbers.

In 32bit integers, Unsigned integer has a range of 0 to 2^{32}-1 and signed has a range of (- $2^{31)}$ to (+ 2^{31})-1

Figure 1. Signed Integer and Unsigned Integer



Signed Integer and Unsigned Integer

In 8bit integers, Unsigned integer has a range of 0 to 2^{8}-1 and signed has a range of (-$2^{8)}$-1 to (+$2^{8)}$

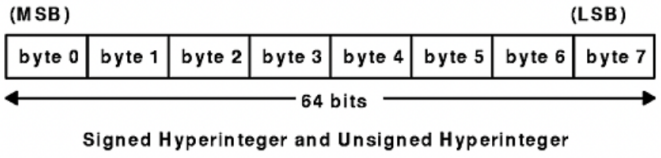
---

integer	storage(byte)	Signed	Unsigned
int32	4 (32 bit)	-2147483648 to 2147483647	0 to 4294967295
int8	1 (8 bit)	-128 to 127	0 to 255

Signed and Unsigned hyperinteger

64 bit (8 byte) numbers are called signed and unsigned hyperinteger

Figure 1. Signed Hyperinteger and Unsigned Hyperinteger



Floting-Point Data types

There are two floating point : Single - Precision Floating Point and Double - Precision Floating Poing

Single - Precision Floating Point

it is also known as float. Length of float is 32 bits or 4 bytes.

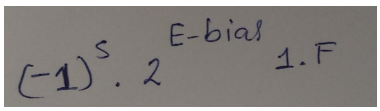
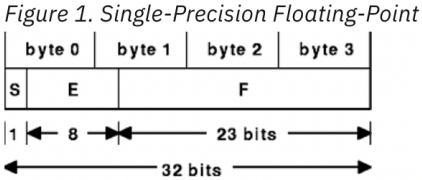
single precision floating point is represented as : 

Figure 1. Single-Precision Floating-Point

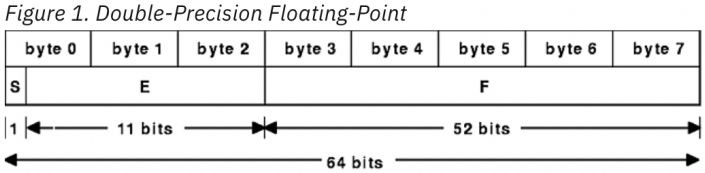


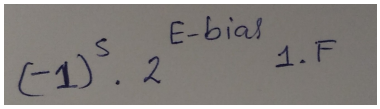
- S : sign of the number
  - This 1 bit field specifies sign
  - 0 for +ve and 1 for -ve
- E : exponent of number in base 2
  - contains 8 bits
  - exponent is biased by 127
- F : fractional part

Double - Precision Floating Poing

it is known as double

Figure 1. Double-Precision Floating-Point



double precision floating point is represented as : 

- S : sign of the number
  - This 1 bit field specifies sign
  - 0 for +ve and 1 for -ve
- E : exponent of number in base 2
  - contains 11 bits
  - exponent is biased by 1023
- F : fractional part

Quantization process

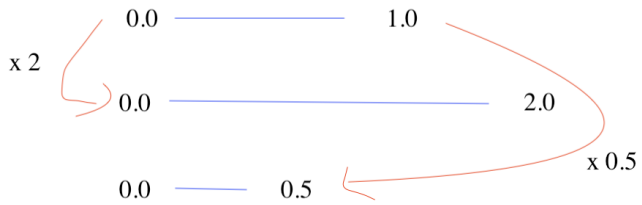
`R = S*(q - zp)`

- R : real value
- S* : scale
  - `scale = (rmax - rmin) / (qmax - qmin)`
  - rmax : maximum of value to be quantized
  - rmin : minimum of value to be quantized
  - suppose we need to map float32 to int8, so qmax will be 128 and qmin will be -127 (128--127 = 255)
- q : quantized value
- zp : zero point
  - It is the quantization point corresponding to zero in real v

After quantization the input which were in float will be mapped to a range of -128 and 127.

scale

multiplication is illustraed belw



During scaling this is happening. We are either shrinking or expanding our data range.

Why we are subtracting zero point from quantized value??

- suppose we need quantize data with rmin = -1 and rmax=1
  - Here our data is symmetric across zero, so subtracting zp is not necessary
- if rmin = -1.5 and rmax = 1
  - In this case our data is asymmetric across zero, so need to subtract zp from quantized value
  - zp is introduced to bring asymmetry to the quantized values

Model has two types of parameters : static and dynamic. static parameters are its weights and dynamic parmeters are its inputs and outputs. during tflite conversion this static parametrs will get automatically quantized. But for dynamic parameters we need to set scale and zero point for quantization. So for this purpose we are using data generators during tflite conversion. During training network will calculate and store scale & zero point for a small part of training data set. data generators are used to create this subsample from whole training data. And with this scale and zero point network will quantize the real time data during inference.