

Expressive Power and Universal Approximation Property of Quantized Neural Networks under Fixed-Point Arithmetic

2025 KIAS CAINS Workshop

Park, Yeachan

Sejong University

ychpark@sejong.ac.kr

May 30, 2025

I. Background and motivation

- (i) Quantization of Neural networks
- (ii) Floating-point and fixed-point arithmetic

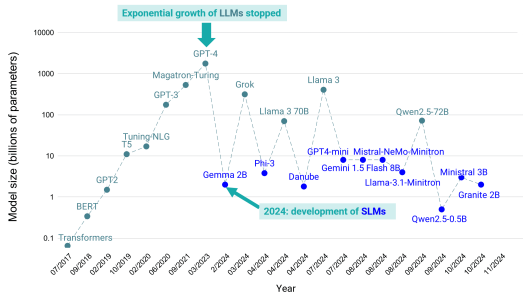
II. Universal approximation theorem

- (i) Universal approximation theorem under real operations
- (ii) Universal approximation theorem under fixed-point arithmetic

Background and motivation

Quantization of Neural networks

Era of large language models



Small language models



	Llama 3	Meta 8 billion parameters
	Phi-3	Microsoft 3.8 billion - 7 billion parameters
	Gemma	Google 2 billion - 7 billion parameters
	Mixtral 8x7B	Mistral AI 7 billion parameters
	OpenELM	Apple 0.27 billion - 3 billion parameters

Quantization of neural networks

Quantization

Floating point

3452.3194



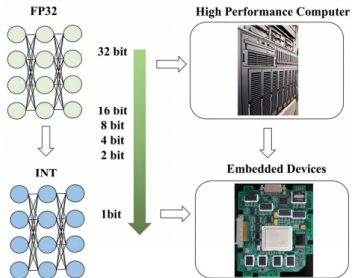
Integer

3452

32 bit

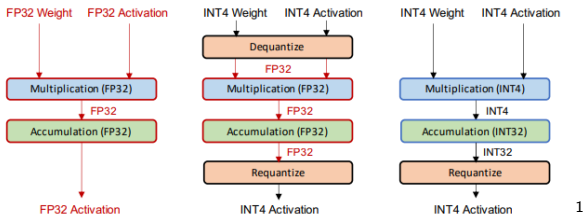


8 bit



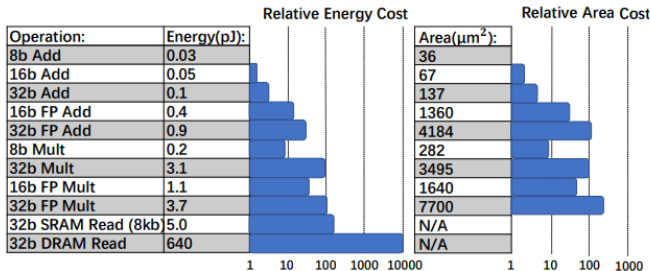
Quantization of neural networks

- Simulated quantization (fake quantization)
 - quantizing weights only
- Integer-only quantization (fixed-point quantization) - quantizing weights and operations (integer arithmetic)



¹Gholami, Amir, et al. "A survey of quantization methods for efficient neural network inference." Low-power computer vision. Chapman and Hall/CRC, 2022. 291-326.

Performance of quantization



2

²Gholami, Amir, et al. "A survey of quantization methods for efficient neural network inference." Low-power computer vision. Chapman and Hall/CRC, 2022. 291-326.

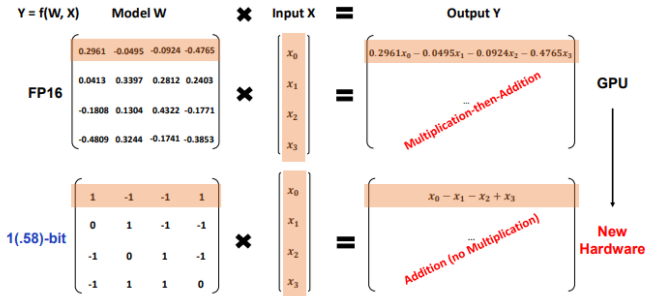
Bitnet : binary LLM

BitNet: Scaling 1-bit Transformers for Large Language Models

Hongyu Wang^{1†} Shuming Ma^{1†} Li Dong¹ Shaohan Huang¹
 Hualjie Wang² Lingxiao Ma³ Fan Yang¹ Ruiping Wang¹ Yi Wu³ Furu Wei^{1*}
[†] Microsoft Research ¹ University of Chinese Academy of Sciences ³ Tsinghua University
<https://aka.ms/GeneralAI>

The Era of 1-bit LLMs: All Large Language Models are in 1.58 Bits

Shuming Ma^{*} Hongyu Wang^{*} Lingxiao Ma³ Lei Wang³ Wenhui Wang³
 Shaohan Huang¹ Li Dong¹ Ruiping Wang¹ Jibong Xue³ Furu Wei^{1*}
<https://aka.ms/GeneralAI>

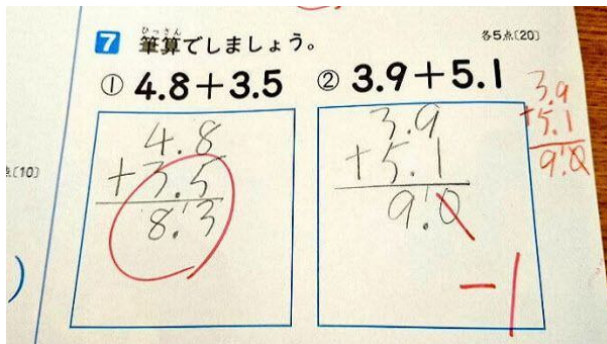


3

³Ma, Shuming, et al. "The era of 1-bit llms: All large language models are in 1.58 bits." arXiv preprint arXiv:2402.17764 1 (2024).

Floating-point and fixed-point arithmetic

Motivation



- What is the right answer? 9 or 9.0?

$0.1 + 0.2 = 0.3$?

```
In [1]: 0.1+0.1 == 0.2
```

```
Out[1]: True
```

```
In [2]: 0.1+0.2 == 0.3
```

```
Out[2]: False
```

```
In [3]: print(0.1+0.1)  
         print(0.1+0.2)
```

```
0.2  
0.30000000000000004
```

- $0.1 + 0.1 = 0.2$
- $0.1 + 0.2 \neq 0.3$
- Why does this happen?

Floating point number

Scientific notation:

$$12.345 = \underbrace{12345}_{\text{significand}} \times \underbrace{10^{-3}}_{\text{base}}^{\text{exponent}}$$

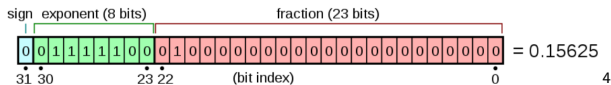
Binary notation:

$$\begin{aligned} & \left(\sum_{n=0}^{p-1} \text{bit}_n \times 2^{-n} \right) \times 2^e \\ &= (1 \times 2^{-0} + 1 \times 2^{-1} + 0 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4} + \dots + 1 \times 2^{-23}) \times 2^1 \\ &\approx 1.5707964 \times 2 \\ &\approx 3.1415928 \end{aligned}$$

- Floating point number : approximately express real numbers using binary notation.
- \neq Fixed point number : express exact real number.

IEEE754 Floating number standard

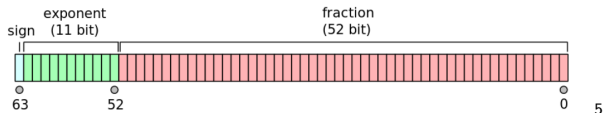
32-bit floating number (single precision):



$$\text{value} = (-1)^{b_{31}} \times 2^{(b_{30} \dots b_{23})_2 - 127} \times (1.b_{22} \dots b_0)_2$$

$$= (-1)^{\text{sign}} \times 2^{E-127} \times \left(1 + \sum_{i=1}^{23} b_{23-i} 2^{-i}\right)$$

64-bit floating number (double precision):



⁴<https://en.wikipedia.org/wiki/Single-precision-floating-point-format>

⁵<https://en.wikipedia.org/wiki/Double-precision-floating-point-format>

Floating point arithmetic rule : Rounding

Rounding Rule:

- Round to nearest, ties to even
- Round to nearest, ties away from zero

Example of rounding to integers using the IEEE 754 rules

Mode	Example value			
	+11.5	+12.5	-11.5	-12.5
to nearest, ties to even	+12.0	+12.0	-12.0	-12.0
to nearest, ties away from zero	+12.0	+13.0	-12.0	-13.0
toward 0	+11.0	+12.0	-11.0	-12.0
toward $+\infty$	+12.0	+13.0	-11.0	-12.0
toward $-\infty$	+11.0	+12.0	-12.0	-13.0

6

⁶<https://en.wikipedia.org/wiki/IEEE754>

Floating point arithmetic rule

- $\lceil \cdot \rceil$: Rounding operation.

Basic rule:

$$x \oplus y = \lceil x + y \rceil, \quad x \otimes y = \lceil x \times y \rceil$$

Multiple operations:

$$x \oplus y \oplus z = (x \oplus y) \oplus z = \lceil \lceil x + y \rceil + z \rceil$$

$$x_1 \oplus \cdots \oplus x_n = (\dots(x_1 \oplus x_2) \oplus \cdots \oplus x_n) = \lceil \dots \lceil \lceil x_1 + x_2 \rceil + \dots \rceil + x_n \rceil.$$

In general, floating point addition/multiplication is not associative:

$$(x \oplus y) \oplus z \neq x \oplus (y \oplus z).$$

```
In [1]: print( (0.1+0.2)+0.3 )
        print( 0.1+(0.2+0.3) )

0.6000000000000001
0.6
```

```
In [2]: print ( (1.0000000000000001 - 1)+0.0000000000000001 )
        print ( 1.0000000000000001 + (- 1+0.0000000000000001) )

1e-17
0.0
```

GNU C Library (libc)

Files

master

Go to file

__sin.c

__sindf.c

__sinl.c

__tan.c

__tandf.c

__tanl.c

acos.c

acosh.c

acoshf.c

acoshl.c

acosl.c

asin.c

asinf.c

asinh.c

ulysse-libc / src / math / exp.c

Code
Blame
135 lines (128 loc) · 4.09 KB

```

81  double exp(double x)
82  {
83      double hi, lo, c, xx;
84      int k, sign;
85      uint32_t hx;
86
87      GET_HIGH_WORD(hx, x);
88      sign = hx > 31;
89      hx ^= 0x7fffffff; /* high word of |x| */
90
91      /* special cases */
92      if (hx >= 0x40062e42) { /* if |x| >= 709.78... */
93          if (isnan(x))
94              return x;
95          if (hx == 0x7ff00000 && sign) /* -inf */
96              return 0;
97          if (x > 709.782712893383973096) {
98              /* overflow if x != -inf */
99              STRICT_ASSIGN(double, x, 0x1p1023 * x);
100             return x;
101         }
102         if (x < -745.13321910194110842) {
103             /* underflow */
104             STRICT_ASSIGN(double, x, 0x1p-1000 * 0x1p-1000);
105             return x;
106         }
107     }

```

Fixed-point arithmetic

- p -bit fixed-point arithmetic : $[0, 2^b - 1]_{\mathbb{Z}} \rightarrow [\alpha, \beta]_{\mathbb{R}}$.

$$x = \alpha + (\beta - \alpha) \frac{k}{(2^b - 1)}, \quad k \in [0, 2^b - 1]_{\mathbb{Z}}.$$

- Quantization:

$$q = \lceil \frac{\text{clip}(x, [\alpha; \beta]) - \alpha}{s} \rceil, \quad s = \frac{2^b - 1}{\beta - \alpha} \quad x \in \mathbb{R}.$$

$$\text{Quant}(x) = \alpha + \frac{q}{s} \in [\alpha, \beta], \quad k \in [0, 2^b - 1]_{\mathbb{Z}}.$$

Fixed-point arithmetic

- Rounding rule : "away from zero".
- accumulation in high-precision.

```

inline void FullyConnected(const uint8* input_data, const Dims<4>& input_dims,
                           int32 input_offset, const uint8* filter_data,
                           const Dims<4>& filter_dims, int32 filter_offset,
                           const int32* bias_data, const Dims<4>& bias_dims,
                           int32 output_offset, int32 output_multiplier,
                           int output_shift, int32 output_activation_min,
                           int32 output_activation_max, uint8* output_data,
                           const Dims<4>& output_dims,
                           gemmlowp::GemmContext* gemm_context) {
    (void)gemm_context; // only used in optimized code.
    TFLITE_DCHECK_LE(output_activation_min, output_activation_max);
    // TODO(benoitjacob): This really should be:
    //   const int batches = ArraySize(output_dims, 1);
    // but the current --variable_batch hack consists in overwriting the 3rd
    // dimension with the runtime batch size, as we don't keep track for each
    // array of which dimension is the batch dimension in it.
    const int batches = ArraySize(output_dims, 1) * ArraySize(output_dims, 2) *
        ArraySize(output_dims, 3);
    const int output_depth = MatchingArraySize(filter_dims, 1, output_dims, 0);
    const int accum_depth = ArraySize(filter_dims, 0);
    TFLITE_DCHECK(IsPackedWithoutStrides(input_dims));
    TFLITE_DCHECK(IsPackedWithoutStrides(filter_dims));

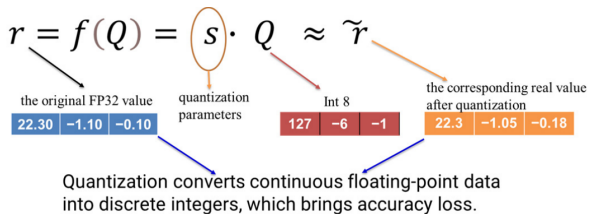
    for (int b = 0; b < batches; ++b) {
        for (int out_c = 0; out_c < output_depth; ++out_c) {
            int32 acc = 0;
            for (int d = 0; d < accum_depth; ++d) {
                int32 input_val = input_data[b * accum_depth + d];
                int32 filter_val = filter_data[out_c * accum_depth + d];
                acc += (filter_val + filter_offset) * (input_val + input_offset);
            }
            if (bias_data) {
                acc += bias_data[Offset(bias_dims, out_c, 0, 0, 0)];
            }
            acc = MultiplyByQuantizedMultiplierSmallerThanOne(acc, output_multiplier,
                                                                output_shift);
            acc += output_offset;
            acc = std::max(acc, output_activation_min);
            acc = std::min(acc, output_activation_max);
            output_data[out_c + output_depth * b] = static_cast<uint8>(acc);
        }
    }
}

```

7

⁷https://github.com/tensorflow/tensorflow/blob/4952f981be07b8bf508f8226f83c10cdafa3f0c4/tensorflow/contrib/lite/kernels/internal/reference/reference_ops.h

Motivation : Rounding error



Motivation : Rounding error

Two-layer network $f : \mathbb{R}^{257} \rightarrow \mathbb{R}$ defined as

$$\begin{aligned} f(\mathbf{x}) = & 2 \left(\text{ReLU} \left(\left\lceil \sum_{i=1}^{129} w_{1,i} x_i \right\rceil \right) \right) + \text{ReLU} \left(\left\lceil \sum_{i=1}^{257} w_{2,i} x_i \right\rceil \right) \\ & + 3 \left(-1 \times \text{ReLU} \left(\left\lceil \sum_{i=1}^{129} w_{3,i} x_i \right\rceil \right) \right) + 2 \left(-1 \times \text{ReLU} \left(\left\lceil \sum_{i=1}^{65} w_{4,i} x_i \right\rceil \right) \right). \end{aligned}$$

$$(w_{1,1}, w_{1,2}, w_{1,3}, \dots, w_{1,129}) = \left(1, -\frac{1}{256}, \dots, -\frac{1}{256} \right),$$

$$(w_{2,1}, w_{2,2}, w_{2,3}, \dots, w_{2,257}) = \left(-1, \frac{1}{256}, \dots, \frac{1}{256} \right),$$

$$(w_{3,1}, w_{3,2}, w_{3,3}, \dots, w_{3,129}) = \left(-1, \frac{1}{128}, \dots, \frac{1}{128} \right),$$

$$(w_{4,1}, w_{4,2}, w_{4,3}, \dots, w_{4,65}) = \left(-1, \frac{1}{128}, \dots, \frac{1}{128} \right).$$

$$f(-\mathbf{1}) = -1, \quad f(\mathbf{1}) = 1,$$

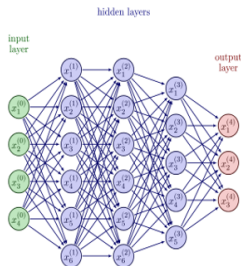
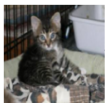
$$\lceil f \rceil(-\mathbf{1}) = 1, \quad \lceil f \rceil(\mathbf{1}) = -1.$$

Universal approximation theorem

Universal approximation theorem under real operations

Expressive power of neural networks

- Neural network represent real-world *unknown* target function



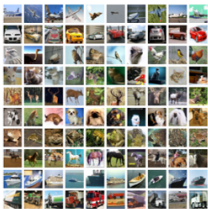
Cat

Dog

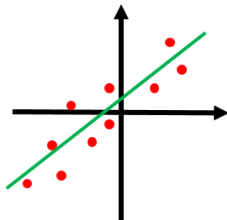
Cat

Expressive power of neural networks

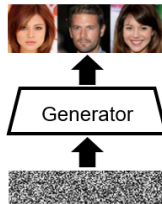
- Real-world unknown target function?
 - No mathematical definition!
 - just *continuous* function.



Classification



Regression



Data generation

Expressive power of neural networks

- Expressive ability:
How much can neural network approximate given target function
- Universal approximation:
Neural network can approximate any continuous functions.

Universal approximation theorem⁸

Theorem (Cybenko (1989), universal approximation for sigmoid activation)

Let $\sigma \in C(\mathbb{R})$ be sigmoid function. Then for any continuous function $f \in C([0, 1]^n, \mathbb{R})$, $\epsilon > 0$ such that there exists a 2-layer network $NN(x)$ such that

$$\|f(x) - NN(x)\| < \epsilon$$

where $NN(x) = W_2\sigma(W_1x + b_1) + b_2$.

⁸Cybenko, George. "Approximation by superpositions of a sigmoidal function." Mathematics of control, signals and systems 2.4 (1989): 303-314.

Universal approximation theorem⁹

Theorem (Leshno (1993), universal approximation for non-polynomial activation)

Let $\sigma \in C(\mathbb{R})$ and assume σ is non-polynomial. Then for any compact set $\mathcal{K} \in \mathbb{N}^d$, continuous function $f \in C(\mathcal{K}, \mathbb{R}^m)$, $\epsilon > 0$ such that there exists a 2-layer network $NN(x)$ such that

$$\|f(x) - NN(x)\| < \epsilon$$

where $NN(x) = W_2\sigma(W_1x + b_1) + b_2$.

⁹Leshno, Moshe, et al. "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function." Neural networks 6.6 (1993): 861-867.

Expressive power of ReLU Network¹⁰

Theorem (Yarotsky (2017), Expressive power of ReLU Network)

Let $\sigma \in C(\mathbb{R})$ be ReLU. Then for continuous function $f \in C([0, 1]^d, \mathbb{R})$, $\epsilon > 0$ such that there exists a ReLU network $NN(x)$ which has $O(\log(\frac{1}{\epsilon}) + 1)$ depth $O(\epsilon^{-d}(\log(\frac{1}{\epsilon}) + 1))$ parameters such that

$$\|f(x) - NN(x)\| < \epsilon.$$

¹⁰Yarotsky, Dmitry. "Error bounds for approximations with deep ReLU networks." Neural Networks 94 (2017): 103-114.

Universal approximation theorem under fixed-point arithmetic

Setup

- $(p + 1)$ -bit symmetric quantization.
- $[-(2^p - 1), 2^p - 1]_{\mathbb{Z}} \rightarrow [\frac{-(2^p+1)}{s}, \frac{2^p-1}{s}]_{\mathbb{R}}$.

$$\mathbb{Q}_{p,s} = \begin{cases} \left\{ \frac{q}{s} : q \in [-2^p + 1, 2^p - 1] \cap \mathbb{Z} \right\} & \text{if } p < \infty, \\ \left\{ \frac{q}{s} : q \in \mathbb{Z} \right\} & \text{if } p = \infty. \end{cases} \quad (1)$$

Condition

For an activation function $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ and $\mathbb{Q}_{p,s}$,

- $\lceil \sigma \rceil(x) = \max \lceil \sigma \rceil(\mathbb{Q}_{p,s})$ for all $x \in \mathbb{Q}_{p,s}$ such that $x \geq z$,
- $\lceil \sigma \rceil(x) < \max \lceil \sigma \rceil(\mathbb{Q}_{p,s})$ for all $x \in \mathbb{Q}_{p,s}$ such that $x < z$.

Lemma

If $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ is monotone and $\lceil \sigma \rceil$ is non-constant on $\mathbb{Q}_{p,s}$, then σ and $\mathbb{Q}_{p,s}$ satisfy the condition.

Furthermore, popular non-monotone activation functions such as GELU, SiLU, and Mish also satisfy the condition for all $\mathbb{Q}_{p,s}$.

First step: gap indicator function

- $q_{\max} = \max \mathbb{Q}_{p,s}.$

$$\phi(x) = \left(-\lceil \sigma \rceil (\lceil x - \alpha + z \rceil) + \lceil \sigma \rceil (q_{\max}) - \lceil \sigma \rceil (\lceil -x + \beta + z \rceil) + \lceil \sigma \rceil (q_{\max}) \right).$$

$$\phi(x) = \begin{cases} 0 & \text{if } x \in [\alpha, \beta], \\ > 0 & \text{if } x \notin [\alpha, \beta] \end{cases}$$

$$\begin{aligned} g(x) &= -\lceil \sigma \rceil \left(\lceil m \times q_{\max} \times \phi(x) + z - \frac{1}{s} \rceil \right) + \lceil \sigma \rceil (q_{\max}) \\ &= \begin{cases} \lceil \sigma \rceil (q_{\max}) - \lceil \sigma \rceil \left(z - \frac{1}{s} \right) & \text{if } x \in [\alpha, \beta], \\ 0 & \text{if } x \notin [\alpha, \beta]. \end{cases} \end{aligned}$$

First step: gap indicator function

For any $q \in \mathbb{Q}_{p,s}$, let m_q be an integer satisfying

$$m_q q_{\max} \times \left(\lceil \sigma \rceil (q_{\max}) - \lceil \sigma \rceil \left(z - \frac{1}{s} \right) \right) > 2q_{\max}.$$

Then we have

$$\begin{aligned} F^q(x) &= \lceil \sigma \rceil (\lceil q + m_q (q_{\max} \times g(x)) \rceil) - \lceil \sigma \rceil (q), \\ &= \begin{cases} \lceil \sigma \rceil (q_{\max}) - \lceil \sigma \rceil (q) & \text{if } x \in [\alpha, \beta], \\ 0 & \text{if } x \notin [\alpha, \beta]. \end{cases} \end{aligned}$$

Second step: universal approximator

$$\mathcal{V}_{\sigma,p,s} = \{ \lceil \sigma \rceil(x) - \lceil \sigma \rceil(y) : x, y \in \mathbb{Q}_{p,s} \}, \quad (2)$$

$$\mathcal{S}_{\sigma,p,s,b}^{\circ} = \left\{ b + \sum_{i=1}^n w_i x_i : n \in \mathbb{N}_0, w_i \in \mathbb{Q}_{p,s}, x_i \in \mathcal{V}_{\sigma,p,s} \ \forall i \in [n] \right\}, \quad (3)$$

$$\mathcal{S}_{\sigma,p,s,b} = \{ \lceil z \rceil : z \in \mathcal{S}_{\sigma,p,s,b}^{\circ} \}. \quad (4)$$

Lemma (2)

If σ is differentiable on $(0, \frac{2}{s})$, $\frac{1}{2} \leq \sigma'(x) < 1$ and $|\sigma(x)| \leq \frac{2^p-1}{s}$ for $x \in (0, \frac{2}{s})$,

ReLU, ELU, SiLU, Mish, and GELU

$$\left| \lceil \sigma \rceil \left(\frac{k+1}{s} \right) - \lceil \sigma \rceil \left(\frac{k}{s} \right) \right| \leq \frac{1}{s}.$$

Third step: universal approximation

Theorem

Let $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ and $p, s \in \mathbb{N}$. Suppose that $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ and $\mathbb{Q}_{p,s}$ satisfy Condition 1. If there exists $b \in \mathbb{Q}_{\infty,s}$ such that

$$\mathcal{S}_{\sigma,p,s,b} = \mathbb{Q}_{p,s}, \quad (5)$$

then σ quantized networks under $\mathbb{Q}_{p,s}$ can universally approximate.

Theorem (universal approximation under fixed-point arithmetic with unbounded exponent)

For any $p \in \mathbb{N}$, $d \in [2^p]$, $f^* \in C([0, 1]^d, \mathbb{R})$, and $\varepsilon > 0$, there exists a σ -quantized network $f(\cdot; \mathbb{Q}_{p,s}) : \mathbb{Q}_{p,s}^d \rightarrow \mathbb{Q}_{p,s}$ of 3 layers such that

$$|f(\mathbf{x}; \mathbb{Q}_{p,s}) - f^*(\mathbf{x})| \leq |f^*(\mathbf{x}) - \lceil f^*(\mathbf{x}) \rceil_{\mathbb{Q}_{p,s}}| + \varepsilon.$$

Thank you