



Denglin Hamming™ V2

dICU Extended Math API

DL-DG/SW-032E-01

2023-4-30

Copyright©苏州登临科技有限公司，2019 - 2025，版权所有。

未经苏州登临科技有限公司事先书面同意，不得以任何形式或方式复制或传播本文件的任何部分。

商标和许可



和其它苏州登临科技有限公司的其它登临科技的图标为苏州登临科技有限公司的商标。本手册中提及的所有其他商标均为其各自所有者的财产。

通知

所购买的产品、服务和特性由苏州登临科技有限公司与客户签订的合同规定。本文件中描述的所有或部分产品、服务和特性可能不在采购范围或使用范围内。除非合同中另有规定，本文件中的所有声明、信息和建议均按“原样”提供，无任何明示或暗示的保证或陈述。

本手册中的信息如有更改，恕不另行通知。本文件在编制过程中已尽一切努力确保内容的准确性，本文件中的所有声明、信息和建议不构成任何明示或暗示的保证。

苏州登临科技有限公司

苏州工业园区扬富路11号南岸新地一期商务楼5号1101室，江苏，中国

<http://www.denglin.ai>

Email : support@denglin.ai

Change History

Version	Change description
01	Initial version.

Table of Contents

Table of Contents

Modules

1. Quantization/dequantization

1.1 Introduction

1.2 Built-in API

hquan

h2quan

h2quan2

hdequan

h2dequan

h2dequan2

3. Sigmoid

3.1 Introduction

3.2 Built-in API

Modules

1. Quantization/dequantization

In neural networks, quantitative models are often used to analyze and process network data. A quantitative model bridges fixed points and floating points, and establishes an effective data mapping relation, so that more benefits are obtained at a lower cost of accuracy loss.

DengLin Minsky™ architecture has designed special quantization and dequantization for AI programs.

1.1 Introduction

Formulas to implement quantization and dequantization:

- Standard floating-point to fixed-point quantization formula:

$$Q = R/S + Z$$

- Standard fixed-point to floating-point dequantization formula:

$$R = (Q - Z) * S$$

In the formulas above, **R** is the real floating-point value, **Q** is the quantized fixed-point value, **Z** is the quantized fixed-point value corresponding to 0 floating-point value, and **S** is the smallest scale that can be represented after fixed-point quantization. **S** and **Z** are quantization parameters; **Q** and **R** can be obtained by formulas.

The quantization and dequantization implemented by the Minsky™ architecture, similar to the FMA instructions, are an abstraction of the standard quantization formulas. Both quantization and dequantization can be calculated with the following formula:

$$result = data * scale + offset$$

In the formula above, **scale** and **offset** need to be converted according to the standard quantization formula:

- For quantization, **scale** = 1/**S** and **offset** = **Z**; **result** is of type uchar; **data**, **scale** and **offset** are of type half.
- For dequantization, **scale** = **S** and **offset** = - **Z** * **S**; **result** is of type half; **data** is of type uchar; **scale** and **offset** are of type half.

To improve the performance of quantization and dequantization, the Minsky™ architecture has optimized the implementation, and implemented instructions for 2 and 4 quantization/dequantization results respectively.

`h2quan` and `h2dequan` can get 2 quantization or dequantization results at a time.

`h2quan2` and `h2dequan2` can get 4 quantization or dequantization results at a time.

1.2 Built-in API

DengLin Hamming™ architecture encapsulated quantization and dequantization instructions to built-in APIs. Users only need to include file `cuda_fp16.h` and call these device functions in the kernel definition file.

This section describes functions that are only supported in device code.

Note: Before using the following functions, you must calculate **scale** and **offset** by using the standard quantization and dequantization formulas. For calculation methods, see [Introduction](#).

hquan

```
__device__ uchar hquan (const __half data, const __half scale, const __half offset)
```

Performs quantization and returns the quantization result.

Parameters

- `data` *const __half*
Read only.
Data to be quantized.
- `scale` *const __half*
Read only.
Scale used to quantize `data`.
- `offset` *const __half*
Read only.
Offset used to quantize `data`.

Returns

uchar

Quantization result.

h2quan

```
__device__ uchar2 h2quan (const __half2 data, const __half2 scale, const __half2 offset)
```

Performs quantization and returns the quantization result.

Parameters

- `data` *const __half2 (float16 x 2)*
Read only.
Data to be quantized.
- `scale` *const __half2 (float16 x 2)*
Read only.
Scale used to quantize `data`.
- `offset` *const __half2 (float16 x 2)*
Read only.
Offset used to quantize `data`.

Returns

uchar2

Quantization result.

h2quan2

```
__device__ uchar4 h2quan2 (const __half2 data0, const __half2 scale0, const __half2 offset0,
const __half2 data1, const __half2 scale1, const __half2 offset1)
```

Performs quantization and returns the quantization result.

Parameters

- `data0` *const __half2 (float16 x 2)*
Read only.
Data to be quantized.
- `scale0` *const __half2 (float16 x 2)*
Read only.
Scale used to quantize `data0`.
- `offset0` *const __half2 (float16 x 2)*
Read only.
Offset used to quantize `data0`.
- `data1` *const __half2 (float16 x 2)*
Read only.
Data to be quantized.
- `scale1` *const __half2 (float16 x 2)*
Read only.
Scale used to quantize `data1`.
- `offset1` *const __half2 (float16 x 2)*
Read only.
Offset used to quantize `data1`.

Returns

uchar4

Quantization result.

hdequan

```
__device__ __half hdequan (const uchar data, const __half scale, const __half offset)
```

Performs dequantization and returns the dequantization result.

Parameters

- `data` *const uchar*

Read only.

Data to be dequantized.

- `scale` *const __half (float16)*

Read only.

Scale used to dequantize `data`.

- `offset` *const __half (float16)*

Read only.

Offset used to dequantize `data`.

Returns

__half

Dequantization result.

h2dequan

```
__device__ __half2 h2dequan (const uchar2 data, const __half2 scale, const __half2 offset)
```

Performs dequantization and returns the dequantization result.

Parameters

- `data` *const uchar2*

Read only.

Data to be dequantized.

- `scale` *const __half2 (float16 x 2)*

Read only.

Scale used to dequantize `data`.

- `offset` *const __half2 (float16 x 2)*

Read only.

Offset used to dequantize `data`.

Returns

__half2 (float16 x 2)

Dequantization result.

h2dequan2

```
__device__ void h2dequan2 (__half2 *ret0, __half2 *ret1, const uchar4 data, const __half2 scale0, const __half2 offset0, const __half2 scale1, const __half2 offset1)
```

Performs dequantization and returns the dequantization result.

Parameters

- `ret0 __half2 (float16 x 2)`

Outputs low half2 of the result to `ret0`.

- `ret1 __half2 (float16 x 2)`

Outputs high half2 of the result to `ret1`.

- `data const uchar4`

Read only.

Data to be dequantized.

- `scale0 const __half2 (float16 x 2)`

Read only.

Scale used to dequantize `data0`.

- `offset0 const __half2 (float16 x 2)`

Read only.

Offset used to dequantize `data0`.

- `scale1 const __half2 (float16 x 2)`

Read only.

Scale used to dequantize `data1`.

- `offset1 const __half2 (float16 x 2)`

Read only.

Offset used to dequantize `data1`.

Returns

void

3. Sigmoid

3.1 Introduction

Function Sigmoid is used to calculate the fast approximate sigmoid of the input argument.

Calculation formula:

$$\text{Sigmoid}(x) = 1/(1 + e^{-x})$$

3.2 Built-in API

```
__device__ float __sigmoidf(float __input)
```

Parameters

`input` *float*

Returns

float

Sigmoid of input.