量化算子长什么样?

4.1 量化算子

Quantize Function

float value = 1.0; float scale = 0.1;

int qt32 = round_fn(value / scale);

char qt8 = clip(qt32, Q_MIN, Q_MAX)

value - 浮点值；scale - 尺度因子

qt32 - 没有名字；qt8 - 量化值；Q_MIN, Q_MAX - 截断值，

round_fn - 取整函数

4.1.2 取整

Round Function

- Round half to even
- Round half away from zero
- Round half toward zero
- Round half down
- Round half up

4.1.2 取整

Round Function

- assert ppq_numerical_round(1.5, policy=RoundingPolicy.ROUND_HALF_EVEN) == 2
- assert ppq_numerical_round(2.5, policy=RoundingPolicy.ROUND_HALF_EVEN) == 2
- assert ppq_numerical_round(0.5, policy=RoundingPolicy.ROUND_HALF_EVEN) == 0
- assert ppq_numerical_round(-0.5, policy=RoundingPolicy.ROUND_HALF_EVEN) == 0
- assert ppq_numerical_round(1.1, policy=RoundingPolicy.ROUND_HALF_EVEN) == 1
- assert ppq_numerical_round(1.2, policy=RoundingPolicy.ROUND_HALF_EVEN) == 1
- assert ppq_numerical_round(1.3, policy=RoundingPolicy.ROUND_HALF_EVEN) == 1
- assert ppq_numerical_round(-1.1, policy=RoundingPolicy.ROUND_HALF_EVEN) == -1
- assert ppq_numerical_round(-1.2, policy=RoundingPolicy.ROUND_HALF_EVEN) == -1
- assert ppq_numerical_round(-1.3, policy=RoundingPolicy.ROUND_HALF_EVEN) == -1

4.1.2 取整

Round Function

- assert ppq_numerical_round(1.5, policy=RoundingPolicy.ROUND_HALF_UP) == 2
- assert ppq_numerical_round(2.5, policy=RoundingPolicy.ROUND_HALF_UP) == 3
- assert ppq_numerical_round(0.5, policy=RoundingPolicy.ROUND_HALF_UP) == 1
- assert ppq_numerical_round(-0.5, policy=RoundingPolicy.ROUND_HALF_UP) == 0

- assert ppq_numerical_round(1.5, policy=RoundingPolicy.ROUND_HALF_DOWN) == 1
- assert ppq_numerical_round(2.5, policy=RoundingPolicy.ROUND_HALF_DOWN) == 2
- assert ppq_numerical_round(0.5, policy=RoundingPolicy.ROUND_HALF_DOWN) == 0
- assert ppq_numerical_round(-0.5, policy=RoundingPolicy.ROUND_HALF_DOWN) == -1

- assert ppq_numerical_round(1.5, policy=RoundingPolicy.ROUND_HALF_TOWARDS_ZERO) == 1
- assert ppq_numerical_round(2.5, policy=RoundingPolicy.ROUND_HALF_TOWARDS_ZERO) == 2
- assert ppq_numerical_round(-0.5, policy=RoundingPolicy.ROUND_HALF_TOWARDS_ZERO) == 0

4.1.2 取整

Round Function

- assert ppq_numerical_round(1.5, policy=RoundingPolicy.ROUND_HALF_UP) == 2
- assert ppq_numerical_round(2.5, policy=RoundingPolicy.ROUND_HALF_UP) == 3
- assert ppq_numerical_round(0.5, policy=RoundingPolicy.ROUND_HALF_UP) == 1
- assert ppq_numerical_round(-0.5, policy=RoundingPolicy.ROUND_HALF_UP) == 0

- assert ppq_numerical_round(1.5, policy=RoundingPolicy.ROUND_HALF_DOWN) == 1
- assert ppq_numerical_round(2.5, policy=RoundingPolicy.ROUND_HALF_DOWN) == 2
- assert ppq_numerical_round(0.5, policy=RoundingPolicy.ROUND_HALF_DOWN) == 0
- assert ppq_numerical_round(-0.5, policy=RoundingPolicy.ROUND_HALF_DOWN) == -1

- assert ppq_numerical_round(1.5, policy=RoundingPolicy.ROUND_HALF_TOWARDS_ZERO) == 1
- assert ppq_numerical_round(2.5, policy=RoundingPolicy.ROUND_HALF_TOWARDS_ZERO) == 2
- assert ppq_numerical_round(-0.5, policy=RoundingPolicy.ROUND_HALF_TOWARDS_ZERO) == 0

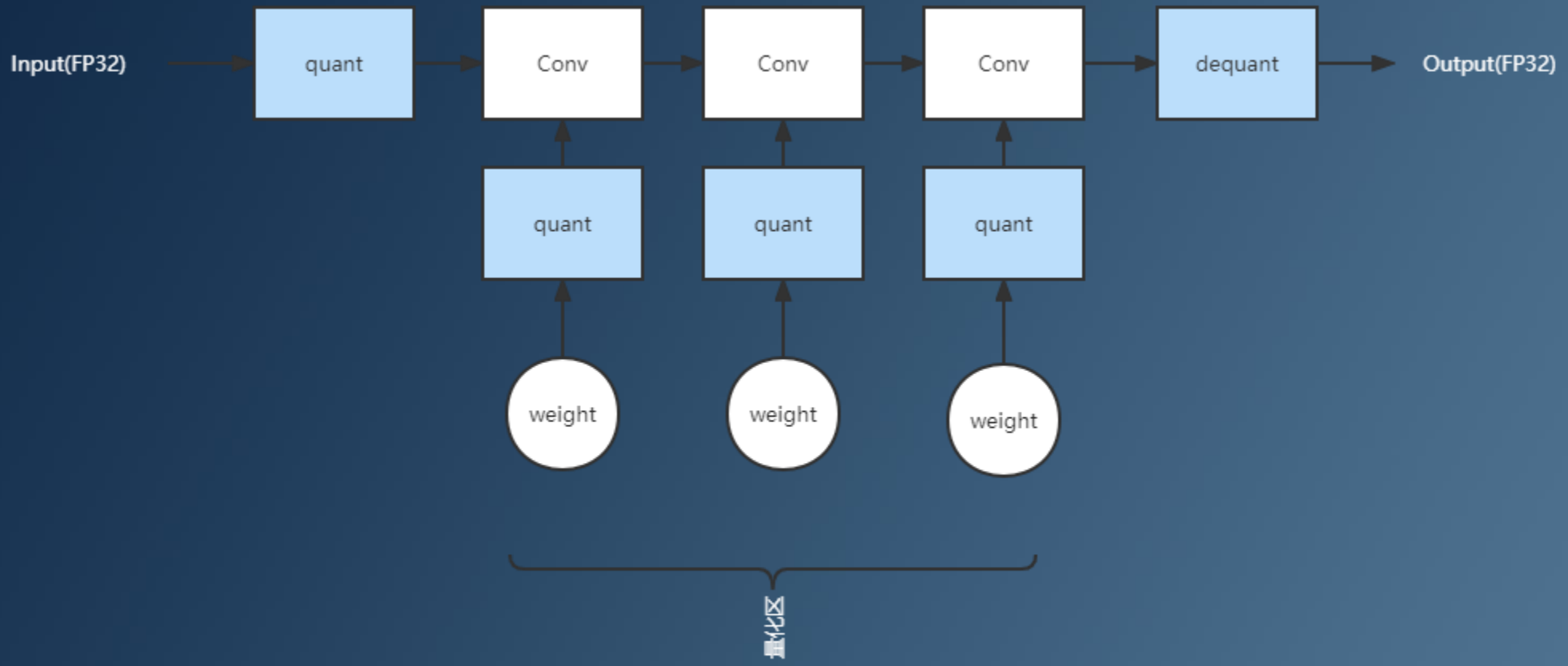
4.1.2 取整

Round Function

```
__device__ void _fix_neuron_v2_device(const Real& src,int& res,
 int val_max,Real val_amp,int method){
 Real res_real_= src*val_amp;
 //method:
 //5: old RNN
 //2: special round for -x.5,
 //3: standard round
 //4: new RNN
...
}
```

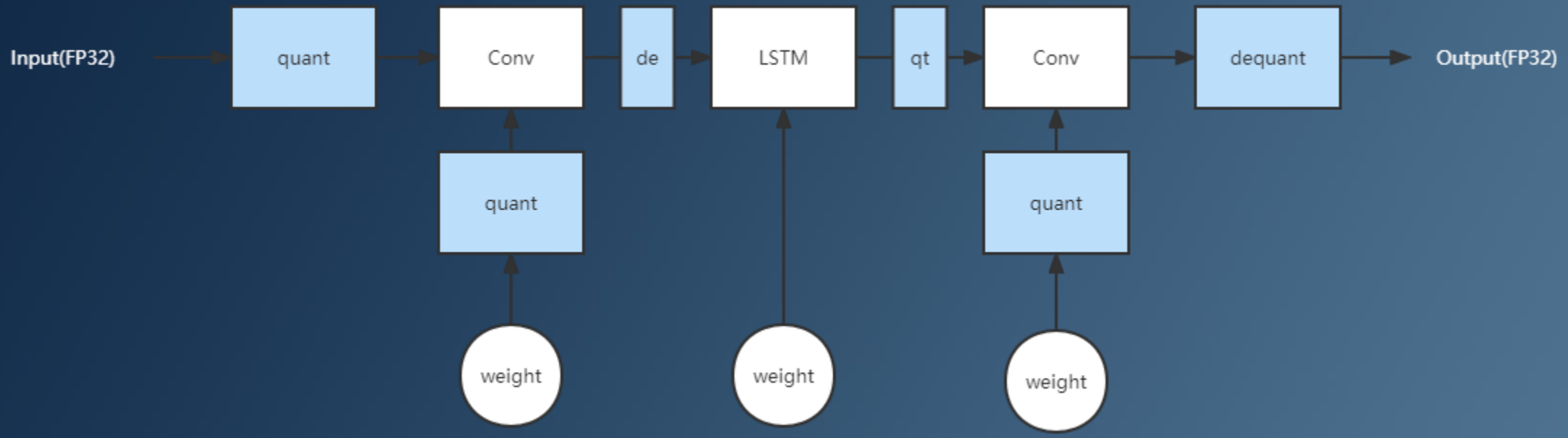
4.1.3 量化子图与全精度子图

Quantized Subgraph



4.1.3 量化子图与全精度子图

Quantized Subgraph



4.1.3 反量化算子
Dequantize Function

```
char value = 1; float scale = 0.1;
float deq = (value * scale);
```

4.1.4 对称与非对称量化

Symmtrical & Asymmtrical Quantization

float value = 1.0; float scale = 0.1;

int qt32 = round_fn(value / scale);

char qt8 = clip(qt32, Q_MIN, Q_MAX)

value - 浮点值；scale - 尺度因子

qt32 - 没有名字；qt8 - 量化值；Q_MIN, Q_MAX - 截断值，

round_fn - 取整函数

4.1.4 对称与非对称量化

Symmtrical & Asymmtrical Quantization

```
float value = 1.0; float scale = 0.1;

int qt32  = round_fn(value / scale + offset);
unsigned char qt8 = clip(qt32, Q_MIN, Q_MAX)
```

value - 浮点值；scale - 尺度因子；offset - 偏移量（零点）

qt32 - 没有名字；qt8 - 量化值；Q_MIN, Q_MAX - 截断值，

round_fn - 取整函数

4.1.4 非对称反量化

Symmtrical & Asymmtrical Quantization

```
char value = 1; float scale = 0.1;
float deq = (value * scale - offset);
```

4.1.4 整数量化

Power-of-2 Quantization

```
float value = 1.0; int shift = 1;
int qt32  = round_fn(value * (2 << shift));
char qt8 = clip(qt32, Q_MIN, Q_MAX)
```

value - 浮点值，scale - 尺度因子，shift - 定点位
qt32 - 没有名字，qt8 - 量化值
Q_MIN, Q_MAX - 截断值，
round_fn - 取整函数

4.1.5 Tensor 量化与通道量化

Per-Tensor & Per-channel Quantization

模式1（PerTensor 量化）：

FP	SCALE	OFFSET	QT8
-0.3	0.1	0	-3
2.1			21
13.2			127
15.7			127

模式2（PerChannel 量化）：

FP	SCALE	OFFSET	QT8
-0.3	0.1	0	-3
2.1			21
13.2	1	0	13
15.7			15

4.1.4 量化模式小结

Quantization Summery

模式1（对称量化）：

```
float value = 1.3; float scale = 0.1;
int qt32 = round(value / scale);
char qt8 = clip(qt32, -128, 127);
```

模式2（非对称量化）：

```
float value = 1.3; float scale = 0.1;
int qt32 = round(value / scale) - offset;
unsigned char qt8 = clip(qt32, 0, 255);
```

模式3（Power of 2）：

```
float value = 1.3; int shift = 4;
int qt32 = round(value << shift);
char qt8 = clip(qt32, -128, 127);
```

模式4（指数量化）：

欢迎自行了解！

4.1.4 PPQ Quantization Policy

QuantizationProperty.ASYMMETRICAL | QuantizationProperty.LINEAR | QuantizationProperty.PER_CHANNEL,
QuantizationProperty.ASYMMETRICAL | QuantizationProperty.LINEAR | QuantizationProperty.PER_TENSOR,
QuantizationProperty.SYMMETRICAL | QuantizationProperty.LINEAR | QuantizationProperty.PER_CHANNEL,
QuantizationProperty.SYMMETRICAL | QuantizationProperty.LINEAR | QuantizationProperty.PER_TENSOR,

QuantizationProperty.ASYMMETRICAL | QuantizationProperty.LINEAR | QuantizationProperty.PER_TENSOR |
QuantizationProperty.POWER_OF_2,

QuantizationProperty.SYMMETRICAL | QuantizationProperty.LINEAR | QuantizationProperty.PER_TENSOR |
QuantizationProperty.POWER_OF_2,

QuantizationProperty.ASYMMETRICAL | QuantizationProperty.LINEAR | QuantizationProperty.PER_CHANNEL |
QuantizationProperty.POWER_OF_2,

QuantizationProperty.SYMMETRICAL | QuantizationProperty.LINEAR | QuantizationProperty.PER_CHANNEL |
QuantizationProperty.POWER_OF_2,