ICS 35.040
CCS L 71

团 体 标 准

T/AI 115.1—2021

信息技术 神经网络表示与模型压缩 第 1 部分：卷积神经网络

Information technology— Neural network representation and model compression — Part1: Convolutional neural network

2021-10-21 发布 2021-10-21 实施

中 关 村 视 听 产 业 技 术 创 新 联 盟 发 布

目　　次

前　言

本文件按照GB/T 1.1—2020《标准化工作导则　第1部分：标准化文件的结构和起草规则》的规定起草。

T/AI 115在《信息技术　神经网络表示与模型压缩》分为三个部分：

——第1部分：卷积神经网络；

——第2部分：大规模预训练模型;

——第3部分：图神经网络。

本文件为T/AI 115的第1部分。

本文件由新一代人工智能产业技术创新战略联盟AI标准工作组提出。

并文件由中关村视听产业技术创新联盟归口。

本文件起草单位：北京大学、鹏城实验室、深圳市海思半导体有限公司、赛灵思半导体有限公司、杭州海康威视数字技术股份有限公司、北京百度网讯科技有限公司、深圳市腾讯计算机系统有限公司、华为技术有限公司、厦门大学、中国电子技术标准化研究院、中国科学院自动化研究所、浙江大学、中国科学技术大学、上海交通大学、清华大学、中关村视听产业技术创新联盟。

本文件主要起草人：田永鸿、杨帆、纪荣嵘、单弈、陈光耀、燕肇一、郑侠武、浦世亮、谭文明、李哲暘、彭博、钟刚、赵恒锐、段文鸿、胡浩基、李翔、骆阳、王炜、许奕星、李慧霞、林绍辉、王培松、赵依、胡晓光、郑辉煌、蒋佳军、马金成、程健、江帆、朱文武、汪小娟、高文、黄铁军、赵海英、马珊珊。

引　言

神经网络的表示和压缩是人工智能技术体系的重要组成部分，是国民经济各行业应用人工智能的前提。本文件旨在提供神经网络可能涉及的表示和压缩技术方法参考，提升用户对模型的复用效果。使用时，对于模型的表示方法应进行必要的支持，对于压缩技术可根据实际应用场景及技术构成做可选支持，具体的支持方法由后续标准进行补充。对于本标准规定的表示方法不要求平台原生支持，可以通过转换、工具包等形式进行支持。本文件的定义可转化为与特定计算设备、框架匹配的形式和实现。

本文件的发布机构提请注意，声明符合本文件时，可能涉及到7.1、7.2、7.3、7.4、8.1、8.2、8.3、8.4、9.1、9.2、9.3、9.4、9.5、10.1、10.2、10.3、10.4、10.5、11.1、11.2和11.3中如下20项和神经网络处理技术相关的专利的使用。专利名称如下：

PCT/CN2018/101598，一种神经网络计算方法和装置；CN202010144315.9 神经网络模型的训练方法和装置；CN202010143782.X 神经网络模型的量化方法和装置；CN202010144339.4 神经网络模型压缩方法以及装置； CN2016100943049.X 神经网络模型压缩方法以及装置；201810464380.2，一种神经网络模型、数据处理方法及处理装置；PCT/CN2019/085885，一种神经网络模型、数据处理方法及处理装置；201910701012.X, 一种基于结构化剪枝的高效图像分类方法；201810575097.7 神经网络表示标准框架结构；201910722230.1 基于多比特神经网络非线性量化的深度神经网络压缩方法；201810387893.8 一种基于参数范数的神经网络量化算法；201911230340.2 一种模型数据的处理方法、装置及设备；2019104537988 一种应用有界线性整流单元的全整数神经网络系统；201610387878.4 基于张量分解的深度卷积神经网络的加速与压缩方法；201910478617.7 基于神经网络差分的量化方法及系统；PCT/EP2019/053161, Neural Network Quantization Method using Multiple Refined Quantized Kernels for Constrained Hardware Deployment；PCT/EP2018/079063, Data Free Neural Network Quantization for Efficient Inference； US17/017600 Deep Learning Processing Apparatus and Method, Device and Storage Medium；US16/893044 Neural Network Data Processing Apparatus, Method and Electronic Device。

本文件的发布机构对于该专利的真实性、有效性和范围无任何立场。

该专利持有人已向本文件的发布机构承诺，他愿意同任何申请人在合理且无歧视的条款和条件下，就专利授权许可进行谈判。该专利持有人的声明已在本文件的发布机构备案，相关信息可以通过以下联系方式获得：

联 系 人：黄铁军（新一代人工智能产业技术创新战略联盟秘书长）

通讯地址：北京大学理科2号楼2641室

邮政编码：100871

电子邮件：tjhuang@pku.edu.cn

电话：+8610-62756172

传真：+8610-62751638

网址：http://www.aitisa.org.cn

请注意除上述专利外，本文件的某些内容仍可能涉及专利。本文件的发布机构不承担识别这些专利的责任。

信息技术 神经网络表示与模型压缩 第1部分：卷积神经网络

1 范围

本文件规定了适应多种计算要求的高效卷积神经网络模型的基本表示方法与压缩过程。

本文件适用于各种卷积神经网络模型的研制、开发、测试评估过程，以及在端云领域的高效应用。

2 规范性引用文件

下列文件中的内容通过文中的规范性引用而构成本文件必不可少的条款。其中，注日期的引用文件，仅注日期对应的版本适用于本文件；不注日期的引用文件，其最新版本(包括所有的修改单)适用于本文件。

GB/T 5271.34-2006 信息技术 词汇 第34部分：人工智能 神经网络

ISO/IEC DIS 22989 Information technology — Artificial intelligence — Artificial intelligence concepts and terminology

ISO/IEC 2382:2015 Information technology — Vocabulary

3 术语和定义

下列术语和定义适用于本文件。

3.1

编解码表示 codec representation

使用压缩技术减少模型的规模。

注：具体定义参考编解码表示，见第10章。

3.2

层 layer

神经网络中的分级结构。

注：每个网络层包含多个个算子，例如输入层，卷积层，全连接层。

3.3

参考随机向量 reference random vector

整个网络共存的基础符号向量。

3.4

多重INT4量化 multiple int4 quantization

一种量化的方式，将一个张量量化为多个INT4张量的组合的量化方式。

3.5

封装表示 encapsulation representation

支出安全信息和身份验证等接口。

注：具体定义参考模型保护，见第11章。

3.6

分块结构化矩阵 block structured matrix

指可以分为多个块，且每个分块均按照某种规律排列的矩阵。

3.7

分块循环矩阵 block circulant matrix

分块循环矩阵是每个分块为循环矩阵的矩阵。

3.8

共享权重shared weight共享权重值， shared weight value

共享权重张量 shared weight tensor

一个神经网络处理多个不同任务，多个任务共享的权重部分。

3.9

共享偏置 shared bias

共享偏置向量 shared bias vector

用一个神经网络处理多个不同任务，多个任务共享的偏置部分。

3.10

卷积神经网络 convolutional neural network

一种前馈神经网络， 在其至少一层中使用卷积。

注：是人工智能应用中代表的深度学习算法，定义参见ISO/IEC DIS 22989。

3.11

紧凑表示 compact representation

输出通过多种神经网络紧凑表示算法压缩后的序列化格式。

示例：结构化矩阵、多模型间权重共享、稀疏编码的稀疏矩阵、权重量化的量化表、低秩分解的分解矩阵和二值神经网络的位表示。

注：具体定义参考压缩过程、解压过程、数据生成方法，见第7、8、9章。

3.12

基础符号向量 basic symbol vector

包括整个网络共存的基础符号向量，以及每层对应的基础符号向量，其中，每层对应的基础符号向量是以某约定规则从网络的基础符号向量中获得。

3.13

校正输入向量 correct input vector

网络层的输入向量中的元素与扰动向量中对应位置的元素相乘，所得到的向量。

3.14

结构化矩阵 structured matrix

一类特殊的矩阵，可以通过使用较少的数据以及一定的排列规律，构成完整的矩阵。

3.15

卷积核粒度量化 convolution kernel granularity quantization

一种量化的方式，对于卷积层权重，以输出channel切分，逐个3D卷积核进行的量化方式。

3.16

可互操作表示 interoperable representation

对神经网络的语法定义、支持的运算操作定义和权重格式（整型、浮点型和定点型）的定义。

注：具体定义参考神经网络模型的语法和语义，见第6章。

3.17

量化尺度因子 quantitative scale factor

原始张量与量化后的张量的缩放比例。

3.18

模型压缩 model compression

减小神经网络模型规模，提高神经网络模型运行和传输效率的方法。

3.19

模型 model

对应完成单个或多个任务的神经网络。

3.20

偏置 bias, offset

偏置向量 bias vector, offset vector

基于参考值的系统性偏差。

注：具体定义参见ISO/IEC 2382:2015，2124103。

3.21

权重 weight

权重张量 weight tensor

连接权重 connection weight/connection strength/synaptic weight

一个系数， 在它与其他输入值结合前， 乘以人工神经元的输入值。

注：具体定义参见GB/T 5271.34-2006。

3.22

权重聚合维度 weight aggregation dimension

共享权重张量和特有权重向量进行拼接的维度。

3.23

任务 task

用神经网络实现某种功能。

示例：如图像超级分辨率，图像降噪。

3.24

扰动向量，扰动符号向量 disturbance vector

将随机随机向量扩展，使其维数为网络层输入向量的维数，所得到的符号向量。

3.25

神经网络表示 neural network representation

表示神经网络的基本语法、语义、运算、超参数、码流定义方法

3.26

随机向量，随机符号向量 random vector

单个网络层对应的基础符号向量。

3.27

特有权重 unique weight

特有权重值 unique weight value

特有权重张量 unique weight tensor

用一个神经网络处理多个不同任务，对任一任务而言，其不共享的权重部分。

3.28

特有权重列表 unique weight list

用一个神经网络处理多个不同任务，除共享权重值以外的部分，也是多个任务不共享的权重部分的组合。

3.29

特有偏置，特有偏置向量 specific offset vector

用一个神经网络处理多个不同任务，对任一任务而言，其不共享的偏置部分。

3.30

特有偏置向量列表 list of unique offset vectors

用一个神经网络处理多个不同任务，除共享权重值以外的部分，也是多个任务不共享的权重部分的组合。

3.31

循环矩阵 circulant matrix

循环矩阵是结构化矩阵的一种，排列规则为当前行/列为前一行/列的循环移位。

4 缩略语

下列缩略语适用于本文件。

CNN：卷积神经网络（Convolutional Neural Network）

DNN：深度神经网络（Deep Neural Network）

PQ：乘积量化(Product Quantization)

5 约定

5.1 规则

本部分中使用的数学运算符和优先级与C语言使用的类似。但对整型除法和算术移位操作进行了特定定义。除特别说明外，约定编号和计数从0开始。

5.2 算术运算符

算术运算符定义见表1

表 1 算术运算符定义

算术运算符	定义
+	加法运算
−	减法运算（二元运算符）或取反（一元前缀运算符）
×	乘法运算
a^b	幂运算，表示a的b次幂。也可表示上标
/	整除运算，沿向0的取值方向截断。例如，7/4和-7/-4截断至1，-7/4和7/-4截断至-1
÷	除法运算，不做截断或四舍五入
$\frac{a}{b}$	除法运算，不做截断或四舍五入
$\sum_{i=a}^{b} f(i)$	自变量i取由a到b（含b）的所有整数值时，函数$f(i)$的累加和
a % b	模运算，a除以b的余数，其中a与b都是正整数

5.3 逻辑运算符

逻辑运算符定义见表2。

表 2 逻辑运算符定义

逻辑运算符	定义
a && b	a和b之间的与逻辑运算
a || b	a和b之间的或逻辑运算
!	逻辑非运算

5.4 关系运算符

关系运算符定义见表3。

表 3 关系运算符定义

关系运算符	定义
>	大于
>=	大于或等于
<	小于
<=	小于或等于
==	等于
!=	不等于

5.5 位运算符

位运算符定义见表4。

表 4 位运算符定义

位运算符	定义
&	与运算
|	或运算
~	取反运算
a >> b	将a以2的补码整数表示的形式向右移b位。仅当b取正数时定义此运算
a << b	将a以2的补码整数表示的形式向左移b位。仅当b取正数时定义此运算

5.6 赋值

赋值运算定义见表5。

表 5 赋值运算定义

赋值运算	定义
=	赋值运算符
++	递增，x++相当于x = x + 1。当用于数组下标时，在自加运算前先求变量值
--	递减，x--相当于x = x - 1。当用于数组下标时，在自减运算前先求变量值
+=	自加指定值，例如x += 3相当于x = x + 3，x += (-3)相当于x = x + (-3)
-=	自减指定值，例如x -= 3相当于x = x - 3,，x -= (-3)相当于x = x - (-3)

5.7 数学函数

数学函数定义见式（1）～式（10）。

$$Abs(x) = \begin{cases} x & ; \quad x >= 0 \\ -x & ; \quad x < 0 \end{cases} \quad \text{(1)}$$

式中：

x ——自变量x。

$$Ceil(x) = \lceil x \rceil \quad \text{(2)}$$

式中：

x ——自变量x。

$$Clip1(x) = Clip3(0, 2^{BitDepth} - 1, x) \quad (3)$$

式中：

x ——自变量x；

BitDepth ——编码样本精度。

$$Clip3(i, j, x) = \begin{cases} i & ; \quad x < i \\ j & ; \quad x > j \\ x & ; \quad 其他 \end{cases} \quad (4)$$

式中：

x ——自变量x；

i ——下界；

j ——上界。

$$Median(x, y, z) = x + y + z - Min(x, Min(y, z)) - Max(x, Max(y, z)) \quad (5)$$

式中：

x ——自变量x；

y ——自变量y；

z ——自变量z。

$$Min(x, y) = \begin{cases} x & ; \quad x <= y \\ y & ; \quad x > y \end{cases} \quad (6)$$

式中：

x ——自变量x；

y ——自变量y。

$$Max(x, y) = \begin{cases} x & ; \quad x >= y \\ y & ; \quad x < y \end{cases} \quad (7)$$

式中：

x ——自变量x；

y ——自变量y。

$$Sign(x) = \begin{cases} 1 & ; \quad x >= 0 \\ -1 & ; \quad x < 0 \end{cases} \quad (8)$$

式中：

x ——自变量x。

$$Log(x) = log_2 x \quad (9)$$

式中：

x ——自变量x。

$$Ln(x) = log_e x \quad (10)$$

式中：

x ——自变量x；

e ——自然对数的底，其值为2.718281828…。

5.8 结构关系

结构关系定义见表6。

表 6 结构关系符

结构关系符	定义
->	例如：a->b表示a是一个结构，b是a的一个成员变量

5.9 解析过程和解码过程的描述方法

5.9.1 函数

以下函数用于语法描述。假定解码器中存在一个位流指针，这个指针指向位流中要读取的下一个二进制位的位置。函数由函数名及左右圆括号内的参数构成。函数也可没有参数。

5.9.2 保留、禁止和标记

本部分定义的位流语法中，某些语法元素的值被标注为“保留”（reserved）或“禁止”（forbidden）。

“保留”定义了一些特定语法元素值用于将来对本部分的扩展。这些值不应出现在符合本部分的位流中。

“禁止”定义了一些特定语法元素值，这些值不应出现在符合本部分的位流中。

“标记”（marker_bit）指该位的值应为‘1’。

位流中的“保留位”（reserved_bits）表明保留了一些语法单元用于将来对本部分的扩展，解码处理应忽略这些位。“保留位”不应出现从任意字节对齐位置开始的21个以上连续的‘0’。

6 神经网络模型的语法和语义

6.1 数据结构

6.1.1 神经网络结构数据结构定义

本部分针对神经网络层结构的数据结构进行定义（参照Google Protocol Buffer语法），部分词汇内容符合GB/T 5271.34-2006，其模型信息以及计算图的相关网络层结构的编码方式如图1所示。通过多个字段的原信息和字段的值采用顺序相连或嵌套的方式进行压缩存储，字段的元信息中含有对这个字段描述的所有信息。其中每个字段有required，repeated和optional三种格式，required表示该字段是必须的，repeated表示该字段可以重复出现，optional表示该字段可选。

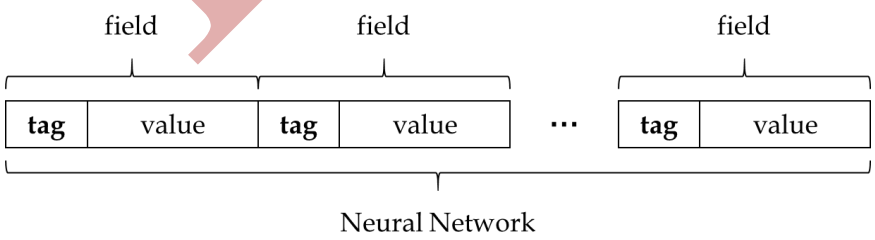


图 1 神经网络结构表示编码结构

针对于不同的数据类型，每组字段和原信息和字段的值采用了不同的编码方式：

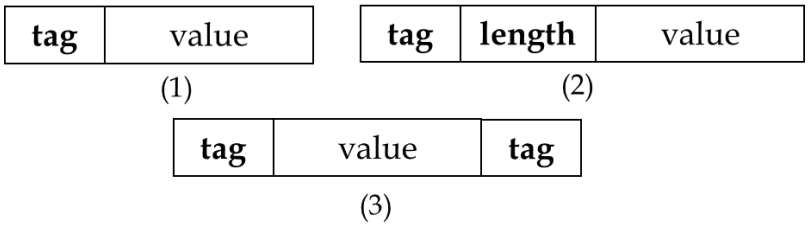


图 2 针对不同类型的编码方式

a) Tag 字段编码：

Tag 字段的编码采用 field_number << 3 | wire_type，其中 field_number 为 Google Protobuffer 中对每个字段定义的标号，wire_type 为该字段的数据类型。最后对上述得到的无符号类型整数做 variant 编码。

b) Int32/int64/uint32/uint64/bool/enum 类型的编码：

该类型采用图 2 中的(1)中的编码方式，Tag 字段采用 Tag 编码方式存放在第一个字节，针对 value 值，对该字段进行 variant 编码存放在第二个字节。

c) String 类型的编码：

该类型采用图 2 中的(2)中的编码方式，第一个字节对 Tag 字段进行编码，第二字节对 string 的长度进行 variant 编码，剩余字节存放 value 值。

d) Float32/Float64 类型的编码：

该类型采用图 2 中的(1)中的编码方式，第一个字节对 tag 字段进行编码，后面字节对 float 的内存直接进行拷贝。

e) 嵌套结构体编码：

该类型采用图 2 中的(1)中的编码方式，第一个字节对 tag 字段进行编码，value 字段不管是单元组还是包装元组，均进行自己的编码序列化。

f) Repeated 类型字段编码：

该类型采用图 2 中的(3)中的编码方式，根据 repeated 个数计算整体长度，对每个 tag 进行 tag 编码，而 value 字段不管是单元组还是包装元组，均进行字节的编码序列化。

6.1.2 神经网络参数数据结构定义

本部分中定义了运算操作的基本内容，每个运算操作包含输入、输出、类型以及属性，并对所有的定义进行了详细的描述。相对于神经网络结构的表示，针对于运算操作，这里将还有网络层参数单独存储，其基本编码方式如图3所示：

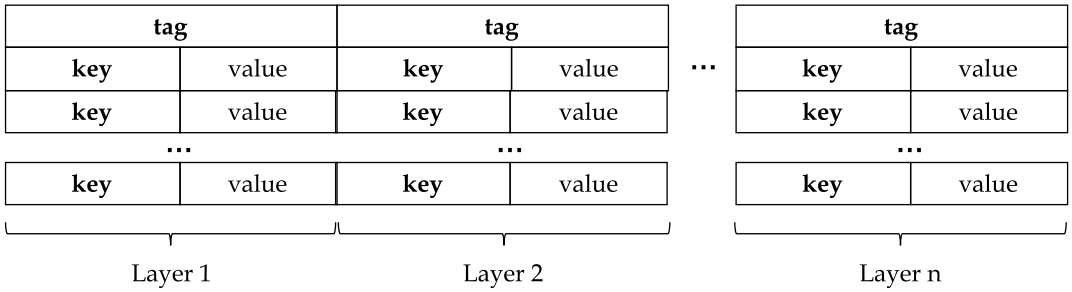


图 3 神经网络参数的编码方式

对于每个运算操作或者神经网络的每一节点，其 tag 字段对应每层自定义名字。每个操作对应多个字段，分别表示每个节点所对应的多组参数。每组参数的 key 对应其关键字或名称，value 字段存储着对应数组。

对于图 3 中 value 中的数据结构表示，对于数组的数据结构编码方式表示如图 4 所示。其包含基本的数据类型、矩阵维度、各维度大小、各维度的跨度以及具体数据。要查找对应数值时，可通过索引、数据类型和各维度的跨度来从 data 中查找。

dtype	float32		数据类型
dim count	2		矩阵维度
dimensions	3	3	各维度的大小
strides	12	4	各维度的跨度
data	*		数据

图 4 数组数据结构编码方式

6.2 语法描述

6.2.1 概述

本部分对神经网络表示的语法进行了定义，采用从粗粒度到细粒度的定义，即从模型结构定义、计算图定义、节点定义等到tensor的定义和基本数据类型的定义，逐层嵌套，构建了整个神经网络表示的基本语法描述。

神经网络表示语法应按照标准规范实现，在由特定计算系统（深度学习平台及相关软硬件）完成，应遵循以下原则：

a) 宜考虑必要层级的定义，包含：
   1) 模型结构定义；
   2) 计算图定义；
   3) 节点定义；
   4) 张量定义；
   5) 基本数据类型定义。
b) 计算系统实现时，应对语法要素按实际需要做出调整，包含但不限于：

1) 关键字（参数）命名；
2) 运算符命名；
3) 数据类型。

6.2.2 模型结构定义

模型结构定义见表7。

表 7 模型结构定义

Model	指定参数	
字段	类型	定义
Version	int64/string	模型表示标准版本
contributors	ContributorsList	模型贡献者信息列表
framework_name	string	模型初始训练框架名称
framework_version	string	模型初始训练框架版本
model_name	string	模型名称
model_version	string	模型版本
doc_url	string	模型描述文档链接
graph	graph(repeated)	模型具体计算图
attribute	map<string, Attribute>	模型属性

6.2.3 贡献者列表定义

贡献者列表定义见表8。

表 8 贡献者列表定义

ContributorList	指定参数	
字段	类型	定义
name	string (repeated)	贡献者姓名
email	string (repeaeted)	贡献者邮箱
institute	string (repeated)	贡献者所在机构

6.2.4 计算图定义

计算图定义见表9。

计算图的id字段表示计算图的唯一序号。计算图可能包含若干子计算图：

a) 神经网络表示中的条件语句、循环语句的内部操作节点和变量节点构成子计算图；

b) 神经网络中的若干操作结点可能会合并为一个新的操作结点，并交由第三方高效计算引擎（如Nvidia TensorRT、Intel Ngraph等）执行，新的操作结点和对应的变量节点可以子计算图的形式表达。

每个计算图的parent_graph_id指向其父计算图，forward_graph_id指向其前向计算图。主计算图无对应的父计算图，其parent_graph_id表示为-1;若计算图无对应的前向计算图，其forward_graph_id表示为-1。

每个计算图包含自己的操作节点和变量节点，操作节点对应着网络中的运算操作，变量节点对应着网络中的变量，包括网络参数和临时变量等。操作节点接收一系列的变量节点作为输入，经过运算操作后输出一系列的变量节点。

表 9 计算图定义

Graph	指定参数	
字段	类型	定义
operator_node	OperatorNode (repeated)	计算图操作节点
variable_type	VariableType(repeated)	计算图变量节点
input_nodes	VariableType (repeated)	计算图输入节点（可选）
output_nodes	VariableType (repeated)	计算图输出节点（可选）
id/name	int64/string	计算图的唯一序号
parent_graph_id	int64	子计算图对应的父计算图的序号（可选）
forward_graph_id	int64	子计算图对应的前向计算图的序号（可选）
sub_graphs/ sub_graphs_ids	Graph (repeated)/ int64 (repeated)	计算图包含的子图列表
attribute	map<string, Attribute>	计算图属性

6.2.5 操作节点定义

操作节点定义见表10。其中，运算操作域定义了操作节点的名称，如abs、relu等；input和output为操作节点的输入变量和输出变量；attribute为操作节点的属性。map是一种映射关系表达，不特定指某种软件中的特定实现。

表 10 操作节点定义

OperatorNode	指定参数	
字段	类型	定义
name	string	节点名称
operator_type	string	节点操作名称
input	map<string, list(VariableType)>	节点输入
output	map<string, list(VariableType)>	节点输出
attribute	map<string, Attribute>	节点属性
doc_string	string	节点描述
definition	string	节点定义
domain	string	作用域（可选）

6.2.6 变量类型定义

变量类型定义见表11。变量类型的name表示变量的名称；data_type表示数据的类型，如稠密张量或稀疏张量等，使用整型数值表示。

表 11 变量类型定义

VariableType	指定参数	
字段	类型	定义
name	string	变量名称
data_type	int64	数据类型
tensor	tensor	变量对应的张量
is_persitable	bool	持久化标记
doc_string	string	变量描述（可选）

6.2.7 属性类型定义

属性类型定义见表12。

表 12 属性类型定义

Attribute	指定参数	
字段	类型	定义
val	Type	属性值
list	list(Type)	属性值序列
name	string	属性名

6.2.8 其他类型定义

其他类型定义见表13。

表 13 其他类型定义

Type	指定参数	
字段	类型	定义
i	int64	整数类型
f	float	单浮点类型
d	double	双浮点类型
s	bytes	位类型
b	bool	布尔类型
vt	vector	数组类型
c	callable	调用类型
v	ValueType	张量数据类型
t	Tensor	张量类型

shape	TensorShape	张量大小类型

6.2.9 张量数据类型定义

张量数据类型定义见表14。

表 14 张量数据类型定义

ValueType	
标号	类型
1	UNDEFINED
2	BINARY
3	INT2
4	INT4
5	INT8

表 14 张量数据类型定义（续）

ValueType	
标号	类型
6	INT16
7	INT32
8	INT64
9	UINT2
10	UINT4
11	UINT8
12	UINT16
13	UINT32
14	UINT64
15	FLOAT16
16	FLOAT32
17	FLOAT64
18	BOOL
19	STRING
20	COMPLEX64
21	COMPLEX128

6.2.10 张量定义

张量定义见表15。

表 15 张量定义

Tensor	指定参数	
字段	类型	定义
name	string	张量名称
version	string	张量版本信息（可选）
shape	TensorShape	张量大小
data_type	ValueType	张量数据类型
content	ValueType(repeated)	张量的数据区域
int32_data	int32_data	张量32位有符号整型数据
uint32_data	uint32_data	张量32位无符号整型数据
int64_data	int64_data	张量64位有符号整型数据
uint64_data	uint64_data	张量64位无符号整型数据
float_data	float_data	张量单精度浮点数数据
double_data	double_data	张量双精度浮点数数据
bool_data	bool_data	张量布尔数据
string_data	string_data	张量字符串数据
format	string	张量数据排布格式
doc_string	string	张量描述（可选）

6.2.11 张量大小定义

张量大小参见表 16。

表 16 张量大小定义

TensorShape	指定参数	
字段	类型	定义
unkown	bool	是否存在大小信息
dim	dimension	维度

6.2.12 维度定义

维度定义见表 17。

表 17 纬度定义

Dimension	指定参数	
字段	类型	定义
size	int64	指定维度
name	string	维度名称

6.3 语义描述

6.3.1 运算操作定义

神经网络模型所含的运算操作，由特定计算系统实现，遵循以下原则：

a) 宜考虑运算涉及的要素，包含：
   1) 可支持的参数；
   2) 可支持的类型。
b) 应按实际需要，做出调整，包含但不限于：
   1) 关键字（参数）命名；
   2) 运算符命名；
   3) 数据类型支持范围。

本章节定义神经网络中所需的运算操作定义，具体定义见表18至表157。

abs运算操作定义见表18。

表 18 abs 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
abs	对输入的数据求绝对值，应用到每一个元素	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64

accuracy 运算操作定义见表 19。

表 19 accuracy 运算操作定义（可选）

运算操作	描述	字段	关键字	定义	数据类型
accuracy	使用输入和标签计算准确率	Input	X	输入张量	float32 float64
			label	数据集标签	int32 int64
		Output	Y	输出张量	float32
		Attributes	k	取每个类别中K个预测值用于计算	int32 int64
			correct	正确预测值的个数	int32 int64
			total	总共的预测值	int32 int64

add 运算操作定义见表 20。

表 20 add 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
add	执行两个输入的加法，在必要条件下将右侧的参数广播以匹配左侧输入的大小	Input	X	输入操作数1	float16, float32, float64
			Y	输入操作数2	float16, float32, float64
		Output	Z	输出张量	float16, float32, float64
		Attributes	axis	计算维度	int
			broadcast	是否允许广播	bool

affine_channel 运算操作定义见表 21。

表 21 affine_channel 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
affine_channel	对输入的每个信道应用单独的仿射变换	Input	X	输入张量	float32 float64
		Output	Y	输出张量	float32 float64
		Attributes	scale	仿射变换的尺度因子	float32 float64
			bias	仿射变换的偏置	float32 float64
			data_format	可选，图像通道格式，默认为“NCHW”	string
			name	可选，网络层输出、权重的前缀标识，默认为None	string
			act	可选，激活函数，默认为None	string

and 运算操作定义见表 22。

表 22 and 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
and	对输入的X和Y执行逻辑运算and操作	Input	X	输入操作数1	bool
			Y	输入操作数2	bool
		Output	Z	输出张量	bool
		Attributes	axis	计算维度	int

			broadcast	是否允许广播	bool

argmax 运算操作定义见表 23。

表 23 argmax 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
argmax	根据提供的维度计算输入张量的最大元素的索引值	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	int64
		Attributes	axis	计算维度	int

argmin 运算操作定义见表 24。

表 24 argmin 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
argmin	根据提供的维度计算输入张量的最小元素的索引值	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	int64
		Attributes	axis	计算维度	int

argsort 运算操作定义见表 25。

表 25 argsort 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
argsort	对输入变量沿给定轴进行升序排列，输出排序好的数据和相应的索引，其维度和输入相同	Input	X	输入张量	float32 float64
		Output	Y	输出张量	float32 float64
			Z	输出索引	int64
		Attributes	axis	对输入张量进行运算对轴	int
			descending	可选，算法排序方向	bool
			name	可选，网络层输出、权重的前缀标识，默认为 None	string

erf 运算操作定义见表 26。

表 26 erf 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
erf	计算输入张量每个元素的误差函数值	Input	X	输入张量	float32 float64
		Output	Y	输入张量	float32 float64
		Attributes	name	可选，网络层输出、权重的前缀标识，默认为None	string

size 运算操作定义见表 27。

表 27 size 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
size	计算输入张量的元素个数	Input	X	输入张量	float32 float64 int32 int64 bool
		Output	Y	输出张量	int64
		Attributes	name	可选，网络层输出、权重的前缀标识，默认为None	string

assign 运算操作定义见表 28。

表 28 assign 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
assign	将输入张量拷贝至输出张量	Input	X	输入张量或numpy array	float16 float32 float64 int32 int64 bool int32, uint8, int16, int8, complex64, uint16, complex128, uint32, uint64
			Value	需要赋给variable的值	float16

					float32 float64 int32 int64 bool int32, uint8, int16, int8, complex64 , uint16, complex12 8, uint32, uint64
		Output	Y	输出张量	float16 float32 float64 int32 int64 bool int32, uint8, int16, int8, complex64 , uint16, complex12 8, uint32, uint64

auc 运算操作定义见表 29。

表 29 auc 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
auc	计算模型评估指标	Input	X	输入张量	float32 float64
			label	训练数据标签	int32 int64
		Output	Y	输出张量	float32 float64
		Attributes	curve	曲线类型，默认为ROC	string
			num_thresholds	将ROC曲线离散化时使用的临界值数，默认为200	int
			topk	预测输出的topk	int

			slide_steps	计算批auc时，不仅用当前步也用先前步	int

average_pool 运算操作定义见表 30。

表 30 average_pool 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
average_pool	根据kernel_shape、strides和pad在输入张量应用平均池化	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64
		Attributes	auto_pad	valid或same，不能和pads一起使用	string
			kernel_shape	核的大小	list of ints
			pads	对于开始和结束，每个维度的填充	list of ints
			strides	每个维度的步长	list of ints
			data_format	输入和输出的数据格式	string

case 运算操作定义见表 31。

表 31 case 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
case	实现if-else语句	Input	Cond	可选，条件子图，结果为bool	bool
		Output	Y	输出张量	float32 float64 int uint bool
		Attributes	pred_fn_pairs	pred为布尔张量，fn为可调用对象callable，返回张量	list of (pred, fn)
			default	可选，可调用对象	callable
			name	可选，网络层输出、权重的前缀标识，默认为None	string

注：条件子图表示一个由若干算子组合而成的计算图结构，该计算图结构可以被 case 算子进行调用，可作为算子输入或属性，由实际框架需求灵活选择。

cast 运算操作定义见表 32。

表 32 cast 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
cast	将输入张量元素转换为由to指向的数据类型	Input	X	输入张量	all
		Output	Y	输出张量	all
		Attributes	to	转换的目标类型	string

ceil 运算操作定义见表 33。

表 33 ceil 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
ceil	向上取整	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64

concat 运算操作定义见表 34。

表 34 concat 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
concat	将输入张量列表进行拼接	Input	X	输入张量列表	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64
		Attributes	axis	合并维度	Int

constant 运算操作定义见表 35。

表 35 constant 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
constant	输出一个恒定值的张量	Output	Output	输出张量	float16, float32, float64
		Attributes	value	输出张量元素的值	tensor

cond 运算操作定义见表 36。

表 36 cond 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
cond	如果 pred 是 True，返回 true_fn() ，否则返回 false_fn()	Input	pred	输入张量	bool
		Output	ture_fn	pred为true时调用	callable
			false_fn	pred为false时调用	callable
		Attributes	name	可选，网络层输出、权重的前缀标识，默认为None	string

conv2d 运算操作定义见表 37。

表 37 conv2d 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
conv2d	构建二维卷积层，根据输入、滤波器参数、步长、填充等参数计算输出特征图	Input	X	输入张量	float16 float32 float64
			W	权重张量	float16 float32 float64
			B	偏置张量（可选）	float16 float32 float64
		Output	Y	输出张量	float16 float32 float64
		Attributes	num_fliters	滤波器的个数	int
			filter_size	滤波器大小	list of ints int
			stride	可选，步长大小	list of ints int
			padding	可选，填充大小	list of ints int
			dilation	可选，膨胀系数大小	list of ints int
			groups	可选，二维卷积层的组数	int
			data_format	卷积x输入的内存排布格式	String

conv3d 运算操作定义见表 38。

表 38 conv3d 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
conv3d	构建三维卷积层，根据输入、滤波器参数、步长、填充等参数计算输出特征图	Input	X	输入张量	float16 float32 float64
			W	权重张量	float16 float32 float64
			B	偏置张量（可选）	float16 float32 float64
		Output	Y	输出张量	float16 float32 float64
		Attributes	num_fliters	滤波器个数	int
			filter_size	滤波器大小	int list of ints
			stride	可选，步长大小	int list of ints
			padding	可选，填充大小	int list of ints
			dilation	可选，膨胀比例大小	int list of ints
			groups	可选，三维卷积层的组数	int
			data_format	卷积x输入的内存排布格式	String

conv2d_transpose 运算操作定义见表 39。

表 39 conv2d_transpose 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
conv2d_transpose	在神经网络中构建一个二维卷积转置层，根据输入、滤波器参数、步长等参数输出特征图	Input	X	输入张量	float16 float32 float64
			W	权重张量	float16 float32 float64
			B	偏置张量（可选）	float16 float32 float64
		Output	Y	输出张量	float16 float32 float64
		Attributes	num_fliters	滤波器的个数	int
			filter_size	滤波器大小	int list of ints
			output_size	可选，输出特征图的大小	int list of ints
			padding	可选，填充大小	int list of ints
			stride	可选，步长大小	int list of ints
			dilation	可选，膨胀比例大小	int list of ints
			groups	可选，二维卷积层的组数	int
			data_format	卷积x输入的内存排布格式	String

conv3d_transpose 运算操作定义见表 40。

表 40 conv3d_transpose 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
conv3d_transpose	在神经网络中构建一个三维卷积转置层，根据输入、滤波器参数、步长等参数输出特征图	Input	X	输入张量	float16 float32 float64
			W	权重张量	float16 float32 float64
			B	偏置张量（可选）	float16 float32 float64
		Output	Y	输出张量	float16 float32 float64
		Attributes	num_fliters	滤波器的个数	int
			filter_size	滤波器大小	int list of ints
			output_size	可选，输出图片的大小	int list of ints
			padding	填充大小	int list of ints
			stride	步长大小	int list of ints
			dilation	可选，膨胀比例大小	int list of ints
			groups	可选，三维卷积层的组数	int
			name	可选，网络层输出、权重的前缀标识，默认为None	string
			data_format	卷积x输入的内存排布格式	String

cos 运算操作定义见表 41。

表 41 cos 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
cos	余弦函数	Input	X	输入张量	float32 float64
		Output	Y	输出张量	float32 float64
		Attributes	name	可选，网络层输出、权重的前缀标识，默认为None	string

cos_sim 运算操作定义见表 42。

表 42 cos_sim 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
cos_sim	余弦相似度算子	Input	X	输入张量	float32
			Y	输入张量	float32
		Output	Z	输出张量	int float

cross_entropy 运算操作定义见表 43。

表 43 cross_entropy 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
cross_entropy	计算输入张量和标签label间的交叉熵	Input	X	输入张量	float32 float64
			label	输入X对应的标签张量值	int64
		Output	Y	输出张量	float32 float64
		Attributes	soft_label	指明label是否为软标签	bool
			ignore_index	指定一个忽略的标签值，此标签值不参与计算	int

crop 运算操作定义见表 44。

表 44 crop 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
crop	根据偏移量和形状，裁剪输入张量	Input	X	输入张量	float32 float64
		Output	Y	输出张量	float32 float64
		Attributes	shape	输出张量的维度	list of ints
			offsets	可选，每个维度上裁剪的偏移量	list of ints
			name	可选，开发人员打印调试信息时使用	string

greater_equal 运算操作定义见表 45。

表 45 greater_equal 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
greater_equal	应用逻辑运算 greater_equal 在输入X和Y	Input	X	输入张量	float32 float64 int32 int64
			Y	输入张量	float32 float64 int32 int64
		Output	Z	输出张量	bool
		Attributes	axis	计算维度	int64
			broadcast	是否允许广播	bool

data 运算操作定义见表 46。

表 46 data 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
data	graph的输入张量，该张量可被计算图中的算子访问。该张量可作为占位符用于数据输入	Output	Y	输出张量	bool float16 float32 float64 int8 int16 int32 uint8
		Attributes	shape	输出张量的维度	list of ints
			dtype	输出张量的类型	string

reciprocal 运算操作定义见表 47。

表 47 reciprocal 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
reciprocal	计算输入张量中每个元素的倒数	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64
		Attributes	name	可选，网络层输出权重的前缀标识，默认为None	string

depth_to_space 运算操作定义见表 48。

表 48 depth_to_space 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
depth_to_space	将输入张量从深度重新排列为空间数据	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64
		Attributes	blocksize	被移动的块大小	int

diag 运算操作定义见表 49。

表 49 diag 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
diag	创建一个方阵，使用输入张量来指定方阵的对角线元素的值	Input	X	输入张量	float32 float64 int32 int64
		Output	Y	输出张量	float32 float64 int32 int64

div 运算操作定义见表 50。

表 50 div 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
div	对输入张量X和Y进行算数运算除法	Input	X	输入操作数1	float16 float32 float64
			Y	输入操作数2	float16 float32 float64
		Output	Z	输出张量	float16, float32, float64
		Attributes	axis	计算维度	int
			broadcast	是否允许广播	bool

dropout 运算操作定义见表 51。

表 51 dropout 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
dropout	随机丢失输入张量部分或简单输出输入张量	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64
		Attributes	is_training	是否训练	bool
			ratio	随机概率（默认为0.5）	float

embedding 运算操作定义见表 52。

表 52 embedding 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
embedding	根据输入信息从嵌入张量中查询对应嵌入信息，函数根据输入的张量维度和数据类型，生成二维嵌入张量	Input	X	输入张量	int64
			W	权重张量	float32 float64
		Output	Y	输出张量	float32 float64
		Attributes	size	嵌入矩阵的维度	list of ints
			is_sparse	是否使用稀疏的更新方式	bool
			is_distributed	是否使用分布式的方式存储嵌入张量	bool
			padding_idx	将输入张量中padding_idx对应的嵌入信息设置为0，默认为None	int long none
			dtype	输出张量的类型	string

floordiv 运算操作定义见表 53。

表 53 floordiv 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
floordiv	逐元素整除算子，并将各个位置的输出元素保存到返回结果中	Input	X	输入张量	float32 float64 int32 int64
			Y	输入张量	float32 float64 int32 int64
		Output	Z	输出张量	float32 float64 int32 int64
		Attributes	axis	可选，Y的维度对应到X维度上时的索引	int32
			name	可选，网络层输出、权重的前缀标识，默认为None	string

elementwise_max 运算操作定义见表 54。

表 54 elementwise_max 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
elementwise_max	逐元素对比输入的两个多维张量，并且把各个位置更大的元素保存到返回结果中	Input	X	输入张量	float32 float64 int32 int64
			Y	输入张量	float32 float64 int32 int64
		Output	Z	输出张量	float32 float64 int32 int64
		Attributes	axis	Y的维度对应到X维度上时的索引	int32
			name	网络层输出、权重的前缀标识，默认为None	string

mod 运算操作定义见表 55。

表 55 mod 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
mod	逐元素取模算子，并将各个位置的输出元素保存到返回结果中	Input	X	输入张量	float32 float64 int32 int64
			Y	输入张量	float32 float64 int32 int64
		Output	Z	输出张量	float32 float64 int32 int64
		Attributes	axis	可选，Y的维度对应到X维度上时的索引	int32
			name	可选，网络层输出、权重的前缀标识，默认为None	string

elementwise_min 运算操作定义见表 56。

表 56 elementwise_min 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
elementwise_min	逐元素对比输入的两个多维张量，并且把各个位置更小的元素保存到返回结果中	Input	X	输入张量	float32 float64 int32 int64
			Y	输入张量	float32 float64 int32 int64
		Output	Z	输出张量	float32 float64 int32 int64
		Attributes	axis	可选，Y的维度对应到X维度上时的索引	int32
			name	可选，网络层输出、权重的前缀标识，默认为None	string

equal 运算操作定义见表 57。

表 57 equal 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
equal	应用逻辑运算equal到输入张量X和Y	Input	X	输入操作数1	float16, float32, float64
			Y	输入操作数2	float16, float32, float64
		Output	Z	输出张量	bool
		Attributes	axis	计算维度	Int
			broadcast	是否允许广播	bool

exp 运算操作定义见表 58。

表 58 exp 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
exp	根据输入张量元素返回其exp指数	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64

expand 运算操作定义见表 59。

表 59 expand 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
expand	根据参数对输入的各维度进行复制，参数为输入的每个维度设置复制次数	Input	X	输入张量	float32 float64 int32 int64 bool
			expand_times	每一维上需要expand的次数，张量或常数List	int32
		Output	Y	输出张量	int32 int64 list of ints
		Attributes	name	可选，网络层输出、权重的前缀标识，默认为None	string

flatten 运算操作定义见表 60。

表 60 flatten 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
flatten	将输入张量平坦为二维矩阵	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64
		Attributes	axis	计算维度	int

floor 运算操作定义见表 61。

表 61 floor 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
floor	向下取整	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64

gather 运算操作定义见表 62。

表 62 gather 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
gather	返回输入张量X索引为indices的内容	Input	X	输入张量	float16, float32, float64
			indices	输入整型张量指数	int32, int64
		Output	Y	输出张量	float16, float32, float64
		Attributes	axis	计算维度	int

gather_nd 运算操作定义见表 63。

表 63 gather_nd 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
gather_nd	gather算子的高维推广，支持多轴同时索引	Input	X	输入张量	int32 int64 float32 float64 bool
			index	索引张量	int32 int64
		Output	Y	输出张量	int32 int64 float32 float64 bool
		Attributes	name	该层的名字，默认为None	string

greater 运算操作定义见表 64。

表 64 greater 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
greater	应用逻辑运算greater在输入X和Y	Input	X	输入操作数1	Tensor
			Y	输入操作数2	Tensor
		Output	Z	输出张量	bool
		Attributes	axis	计算维度	int
			broadcast	是否允许广播	bool

group_norm 运算操作定义见表 65。

表 65 group_norm 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
group_norm	群组归一化	Input	X	输入张量	float32 float64
			W	权重张量	float32 float64
			B	偏置张量	float32 float64
		Output	Y	输出张量	float32 float64
		Attributes	groups	从信道中分离出的群组数量	int32
			epsilon	可选，防止方差除以零	float32
			act	可选，激活函数	string
			data_format	可选，图像通道格式，默认为“NCHW”	string
			name	可选，网络层输出、权重的前缀标识，默认为None	string

hard_max 运算操作定义见表 66。

表 66 hard_max 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
hard_max	返回给定维度最大值的对应大小映射（1表示最大值，0表示其他值）	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64
		Attributes	axis	计算维度	int

less 运算操作定义见表 67。

表 67 less 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
less	应用逻辑运算less到输入张量X和Y	Input	X	输入操作数1	float16, float32, float64
			Y	输入操作数2	float16, float32, float64
		Output	Z	输出张量	bool
		Attributes	axis	计算维度	int
			broadcast	是否允许广播	bool

less_equal 运算操作定义见表 68。

表 68 less_equal 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
less_equal	应用逻辑运算less_equal在输入X和Y	Input	X	输入张量	float32 float64 int32 int64
			Y	输入张量	float32 float64 int32 int64
		Output	Z	输出张量	bool
		Attributes	axis	计算维度	int
			broadcast	是否允许广播	bool

is_empty 运算操作定义见表 69。

表 69 is_empty 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
is_empty	测试输入张量是否为空	Input	X	输入张量	float16 float32 float64 int32 int64
		Output	Y	输出张量	bool
		Attributes	cond	可选，若传入了该参数，则该参数中存储返回给定x的测试结果	int float

isfinite 运算操作定义见表 70。

表 70 isfinite 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
isfinite	测试输入张量是否包含无穷值	Input	X	输入张量	int32 float float64
		Output	Y	输出张量	bool

label_smooth 运算操作定义见表 71。

表 71 label_smooth 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
label_smooth	标签平滑功能	Input	X	输入张量	float32 float64
		Output	Y	输出张量	float32 float64
		Attributes	prior_dist	可选，用于平滑标签的先验分布	int float
			epsilon	可选，混合原始真实分布和固定分布的权重	float
			dtyep	可选，输入张量类型	string
			name	可选，网络层输出、权重的前缀标识，默认为None	string

layer_norm 运算操作定义见表 72。

表 72 layer_norm 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
layer_norm	层归一化	Input	X	输入张量	float32 float64
			W	权重张量	float32 float64
			B	偏置张量	float32 float64
		Output	Y	输出张量	float32 float64
		Attributes	scale	可选，是否在归一化后学习自适应增益	bool
			shift	可选，是否在归一化后学习自适应偏差	bool
			begin_norm_axis	可选，归一化执行维度	int
			epsilon	可选，防止方差除以零	float32
			act	可选，激活函数	string
			name	可选，网络层输出权重的前缀标识，默认为None	string

scatter_nd 运算操作定义见表 73。

表 73 scatter_nd 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
scatter_nd	根据index中的索引值将更新数据添加到一个新的张量中	Input	index	索引	int32 int64
			updates	更新张量	float32 float64
		Output	Y	输出张量	float32 float64
		Attributes	shape	输出张量的形状	List of int
			name	可选，网络层输出权重的前缀标识，默认为Non	string

log 运算操作定义见表 74。

表 74 log 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
log	按照元素计算给定输入张量的自然对数	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64

log_loss 运算操作定义见表 75。

表 75 log_loss 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
log_loss	对输入的预测结果和目标标签进行计算，返回负对数损失值	Input	X	输入张量	float32
			label	输入X对应的标签张量值	float32
		Output	Y	输出张量	float32
		Attributes	epsilon	防止方差除以零	float32
			name	可选，网络层输出权重的前缀标识，默认为None	string

lpnorm 运算操作定义见表 76。

表 76 lpnorm 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
lpnorm	根据给定维度应用Lp归一化（p值限定为1或2）	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64
		Attributes	axis	计算维度	int
			p	标准化的p值（只支持1和2）	int

lstm 运算操作定义见表 77。

表 77 lstm 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
lstm	长短期记忆运算	Input	X	输入张量	float32 float64
			weight_ih	输入的权重张量	float32 float64
			weight_hh	隐状态的权重张量	float32 float64
			bias_ih	输入的偏置张量	float32 float64
			bias_hh	隐状态的偏置张量	float32 float64
		Output	rnn_out	LSTM隐藏输出	float32 float64
			last_h	LSTM最后一步的隐藏状态	float32 float64
			last_c	LSTM最后一步的细胞状态	float32 float64
		Attributes	max_len	LSTM的最大长度	int
			input_size	输入张量的维度	int
			hidden_size	LSTM隐藏状态的维度	int
			num_layers	LSTM的总层数	int
			dropout_prob	可选，忽略比例	float
			is_bidirectional	可选，是否是双向的LSTM	bool
			time_major	可选，是否是时间维在最前	bool
			is_test	可选，是否在测试阶段	bool

mat_mul 运算操作定义见表 78。

表 78 mat_mul 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
mat_mul	矩阵乘法	Input	X	输入张量1	float16, float32, float64
			Y	输入张量2	float16, float32, float64
		Output	Z	输出张量	float16, float32, float64
		Attributes	transpose_x	是否对X转置	bool
			transpose_y	是否对Y转置	bool

max_pool 运算操作定义见表 79。

表 79 max_pool 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
max_pool	根据kernel_shape、pads和strides应用最大池化运算到输入张量X	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64
		Attributes	auto_pad	valid或same，不能和pads一起使用	string
max_pool	根据kernel_shape、pads和strides应用最大池化运算到输入张量X	Attributes	kernel_shape	卷积核大小	list of ints
			pads	对于开始和结束，每个维度的填充	list of ints
			strides	每个维度卷积步长	list of ints
			return_indices	如果等于True，会返回输出最大值的序号	bool
			date_format	channel_first或channel_last	string

max_roi_pool 运算操作定义见表 80。

表 80 max_roi_pool 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
max_roi_pool	根据输入张量X和感兴趣区域，在每个兴趣区域应用最大池化。	Input	X	输入张量	float16, float32, float64
			rois	感兴趣的区域	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64
		Attributes	pooled_shape	ROI pool输出大小	list of ints
			spatial_scale	空间规模大小	float

mul 运算操作定义见表 81。

表 81 mul 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
mul	执行算数运算乘法	Input	X	输入张量1	float16, float32, float64
			Y	输入张量2	float16, float32, float64
		Output	Z	输出张量	float16, float32, float64
		Attributes	broadcast	是否允许广播	bool
			axis	计算维度	int

negative 运算操作定义见表 82。

表 82 negative 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
negative	对输入张量的每个元素进行取反运算	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64

not 运算操作定义见表 83。

表 83 not 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
not	返回每个输入元素的否定	Input	X	输入张量	bool
		Output	Y	输出张量	bool

one_hot 运算操作定义见表 84。

表 84 one_hot 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
one_hot	将输入中的每个id转换为一个one_hot张量	Input	X	输入张量	int32 int64
		Output	Y	输出张量	float32
		Attributes	depth	定义输出张量的长度	int
			allow_out_of_range	输入张量中所包含的id值是否可以大于depth值	bool

or 运算操作定义见表 85。

表 85 or 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
or	执行逻辑运算or到每个输入元素X和Y	Input	X	输入张量1	bool
			Y	输入张量2	bool
		Output	Z	输出张量	bool
		Attributes	axis	计算维度	int
			broadcast	是否允许广播	bool

pad 运算操作定义见表 86。

表 86 pad 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
pad	根据给定的维度和属性对输入张量进行扩充	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64
		Attributes	mode	三种模式：constant(default), reflect, edge	string
			pads	整数的列表表示每个轴的开始和结束的填充元素计数	list of ints
			value	浮点数，表示要填入的值	float

print 运算操作定义见表 87。

表 87 print 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
print	创建一个打印操作，打印正在访问的张量内容	Input	X	输入张量	float16 float32 float64 int32 int64 bool
		Output	Y	输出张量	float16 float32 float64 int32 int64 bool
		Attributes	summarize	打印张量中的元素数目，如果值为-1则打印所有元素	int
			message	打印张量信息前自定义的字符串类型消息	string
			first_n	打印张量的前n个数值	int
			print_tensor_name	可选，指明是否打印张量名称，默认为True	bool
			print_tensor_type	可选，指明是否打印张量类型，默认为True	bool
print	创建一个打印操作，打印正在访问的张量内容	Attributes	print_tensor_shape	可选，指明是否打印张量维度，默认为True	bool
			print_phase	指明打印的阶段，包括 forward ，backward 和 both ，默认为 both	string

pool2d 运算操作定义见表 88。

表 88 pool2d 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
pool2d	在神经网络中构建一个二维池化层，并根据参数配置得到输出	Input	X	输入张量	float32 float64
		Output	Y	输出张量	float32 float64
		Attributes	pool_size	可选，池化核大小	int list of ints
			pool_type	可选，池化类型	string
			pool_stride	可选，池化层的步长	int list of ints
			pool_padding	可选，填充大小	int list of ints
			global_pooling	可选，是否用全局池化	bool
			ceil_mode	可选，是否用ceil函数计算输出高度核宽度	bool
			exclusive	可选，是否在平均池化模型忽略填充值	bool

pool3d 运算操作定义见表 89。

表 89 pool3d 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
pool3d	三维空间池化操作，输入参数的池化配置，根据参数计算输出	Input	X	输入张量	float32 float64
		Output	Y	输出张量	float32 float64
		Attributes	pool_size	池化核的大小	int list of ints
			pool_type	池化类型	string
			pool_stride	池化层的步长	int list of ints
			pool_padding	池化填充	int list of ints string
			global_padding	是否使用全局池化	bool
			ceil_mode	是否使用ceil函数计算输出的深度、高度和宽度	bool
			name	可选，网络层输出权重的前缀标识，默认为None	string
			exclusive	是否在平池化模式忽略填充值	bool
			data_format	输入和输出的数据格式	string

pow 运算操作定义见表 90。

表 90 pow 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
pow	对输入张量进行幂运算	Input	X	输入张量1	float16, float32, float64
			Y	输入张量2	float16, float32, float64
		Output	Z	输出张量	float16, float32, float64
		Attributes	axis	计算维度	int
			broadcast	是否允许广播	bool

random_normal 运算操作定义见表 91。

表 91 random_normal 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
random_normal	产生正态分布的随机张量	Output	Y	输出张量	float16, float32, float64
		Attributes	dtype	输出张量元素的数据类型	string
			mean	正态分布的均值	float
			scale	正态分布的标准差	float
			seed	随机生成器的种子	float
			shape	输出的张量大小	list of ints list of tensor

random_uniform 运算操作定义见表 92。

表 92 random_uniform 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
random_uniform	产生均匀分布的随机张量	Output	Y	输出张量	float16, float32, float64
		Attributes	dtype	输出张量元素的数据类型	string
			high	输出值的上限，默认为1	float
			low	输出值的下限，默认为0	float
			seed	随机生成器的种子	float
			shape	输出的张量大小	list of ints list of tensor

range 运算操作定义见表 93。

表 93 range 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
range	根据step均匀分隔给定数值区间[X, Y)，并返回该分隔结果	Input	start	区间起点（可为张量或常数）	int32 int64 float32 float64
			end	区间终点（可为张量或常数）	int32 int64 float32 float64
			step	步长（可为张量或常数）	int32 int64 float32 float64
		Output	Y	输出张量	int32 int64 float32 float64
		Attributes	dtype	输出张量的数据类型	string

reduce_all 运算操作定义见表 94。

表 94 reduce_all 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
reduce_all	对指定维度上的张量元素进行与逻辑（&）计算，并输出相应的计算结果	Input	X	输入张量	bool
		Output	Y	输出张量	bool
		Attributes	dim	与逻辑运算的维度	list of ints
			keep_dim	是否在输出张量中保留减小的维度	bool

reduce_any 运算操作定义见表 95。

表 95 reduce_any 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
reduce_any	对指定维度上的张量元素进行或逻辑（|）计算，并输出相应的计算结果	Input	X	输入张量	bool
		Output	Y	输出张量	bool
		Attributes	dim	或逻辑运算的维度	list of ints
			keep_dim	是否在输出张量中保留减小的维度	bool

reduce_mean 运算操作定义见表 96。

表 96 reduce_mean 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
reduce_mean	根据给定维度计算输入张量的平均值	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64
		Attributes	axis	计算维度，-1	int
			keepdims	是否保持维度	bool

reduce_max 运算操作定义见表 97。

表 97 reduce_max 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
reduce_max	根据给定维度计算输入张量的最大值	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64
		Attributes	axis	计算维度，-1	int
			keepdims	是否保持维度	bool

reduce_min 运算操作定义见表 98。

表 98 reduce_min 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
reduce_min	根据给定维度计算输入张量的最小值	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64
		Attributes	axis	计算维度，-1	int
			keepdims	是否保持维度	bool

reduce_prod 运算操作定义见表 99。

表 99 reduce_prod 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
reduce_prod	根据给定维度计算输入张量的内积	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64
		Attributes	axis	计算维度，-1	int
			keepdims	是否保持维度	bool

reduce_sum 运算操作定义见表 100。

表 100 reduce_sum 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
reduce_sum	根据给定维度计算输入张量的和	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64
		Attributes	axis	计算维度，-1	int
			keepdims	是否保持维度	bool

reduce_variance 运算操作定义见表 101。

表 101 reduce_variance 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
reduce_variance	根据给定维度计算输入张量的方差	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64
		Attributes	axis	计算维度，-1	int
			keepdims	是否保持维度	bool

relu 运算操作定义见表 102。

表 102 relu 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
relu	应用y = max(0,x)到输入张量X	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64

relu6 运算操作定义见表 103。

表 103 relu6 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
relu6	激活函数	Input	X	输入张量	float32 float64
		Output	Y	输出张量	float32 float64
		Attributes	threshold	阈值，默认值为6.0	float

reshape 运算操作定义见表 104。

表 104 reshape 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
reshape	根据shape调整输入张量X的大小	Input	X	输入张量	float16, float32, int32 int64
			shape	转变的形状	int32 int64
		Output	Y	输出张量	float16, float32, double

reverse 运算操作定义见表 105。

表 105 reverse 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
reverse	输入张量在指定轴axis上进行数据的逆序操作	Input	X	输入张量	int32 int64 float32 float64
		Output	Y	输出张量	int32 int64 float32 float64
		Attributes	axis	逆序运算的轴	int list of ints

rsqrt 运算操作定义见表 106。

表 106 rsqrt 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
rsqrt	激活函数	Input	X	输入张量	float32 float64
		Output	Y	输出张量	float32 float64

round 运算操作定义见表 107。

表 107 round 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
round	四舍五入取整	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	int

shape 运算操作定义见表 108。

表 108 shape 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
shape	返回输入张量的形状大小	Input	X	输入张量	tensor
		Output	Y	输出形状大小	int64

scatter 运算操作定义见表 109。

表 109 scatter 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
scatter	根据index中的索引值将更新数据更新至输入张量	Input	X	输入张量	float32
			index	索引	int32 int64
			updates	更新张量	float32
		Output	Y	输出张量	float32
		Attributes	name	可选，网络层输出权重的前缀标识，默认为None	string
			overwrite	可选，如果index中的索引值有重复且overwrite为True，旧更新值将被新的更新值覆盖；如果为False，新的更新值将同旧的更新值相加。默认值为True	bool

sin 运算操作定义见表 110。

表 110 sin 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
sin	计算输入的正弦值	Attributes	X	输入张量	float16 float32 float64
		Output	Y	输出张量	float16 float32 float64
		Attributes	name	可选，网络层输出权重的前缀标识，默认为None	string

slice 运算操作定义见表 111。

表 111 slice 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
slice	根据给定维度和索引对输入张量进行划分	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64
		Attributes	axes	ends和starts的适用维度	list of ints
			ends	结束索引	list of ints
			starts	开始索引	list of ints

softmax_with_cross_entropy 运算操作定义见表 112。

表 112 softmax_with_cross_entropy 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
softmax_with_cross_entropy	实现softmax交叉熵损失函数	Input	X	输入张量	float32 float64
			label	输入标签张量	float32 float64
		Output	Y	输出张量	float32 float64
softmax_with_cross_entropy	实现softmax交叉熵损失函数	Attributes	soft_label	可选，指明是否将输入标签当作软标签	bool
			ignore_index	可选，指明要无视的目标值	int
			numeric_stable_mode	可选，指明是否使用一个具有更佳数学稳定性的算法	bool

split 运算操作定义见表 113。

表 113 split 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
split	根据split分离输入张量X的部分	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量列表	float16, float32, float64
		Attributes	axis	计算维度	ints
			split	每个输出的大小	list of ints

space_to_depth 运算操作定义见表 114。

表 114 space_to_depth 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
space_to_depth	将输入张量从空间重新排列为深度空间数据	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64
		Attributes	blocksize	被移动的块大小	int32, int64

sqrt 运算操作定义见表 115。

表 115 sqrt 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
sqrt	对输入张量进行开平方运算	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64

square 运算操作定义见表 116。

表 116 square 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
square	对输入张量进行取平方运算	Input	X	输入张量	float16 float32
		Output	Y	输出张量	float16 float32

stack 运算操作定义见表 117。

表 117 stack 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
stack	沿 axis 轴对输入张量进行堆叠操作	Input	X	输入张量列表	float16 float32 int32 int64
		Output	Y	输出张量	float16 float32 int32 int64
		Attributes	axis	对输入张量进行堆叠运算的轴	int

sub 运算操作定义见表 118。

表 118 sub 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
sub	应用算数运算减法到输入张量X	Input	X	输入操作数1	float16, float32, float64
			Y	输入操作数2	float16, float32, float64
		Output	Z	输出张量	float16, float32,f loat64
		Attributes	axis	计算维度	int
			broadcast	是否允许广播	bool

switch_case 运算操作定义见表 119。

表 119 switch_case 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
switch_case	实现C++的switch/case语句	Input	X	输入张量	float16 float32 float64 int32 int64
			branch_index	可选，指定要执行的分支，可调用条件分支函数列表	int32 int64 uint8 callable_list
		Output	Y	输出张量	float16 float32 float64 int32 int64
		Attributes	brach_fns	可选，可调用对象的键（此属性与入参Cond，可选择实现）	list of tuple(int, callable)
			default	可选，可调用对象	callable

topk 运算操作定义见表 120。

表 120 topk 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
topk	返回输入张量前K大的数据	Input	X	输入张量	float16, float32, float64
		Output	values	输出张量的前K大的值	float16, float32, float64
			indices	输出张量的前K打个值的索引	float16, float32, float64
		Attributes	axis	计算维度	int
			k	要查找的前k个的大小	int

squeeze 运算操作定义见表 121。

表 121 squeeze 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型

squeeze	根据给定axes对输入张量进行维度的压缩	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64
		Attributes	axes	计算维度列表	list of ints

transpose 运算操作定义见表 122。

表 122 transpose 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
transpose	矩阵转置	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64
		Attributes	perm	默认情况下转置，否则按给定的值调整	list of ints

unsqueeze 运算操作定义见表 123。

表 123 unsqueeze 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
unsqueeze	根据给定axes对输入张量进行维度的解压缩	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64
		Attributes	axes	计算维度列表	list of ints

unstack 运算操作定义见表 124。

表 124 unstack 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
unstack	将单个维度为D 的张量沿axis 轴拆解为num 个维度为(D-1) 的张量	Input	X	输入张量	float16 float32 float64 int32 int64
		Output	Y	输出张量	float16 float32 float64 int32 int64
		Attributes	axis	可选，对输入张量进行拆解运算所在的轴	int
			num	可选，axis轴的长度	int

xor 运算操作定义见表 125。

表 125 xor 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
xor	进行逻辑运算异或运算	Input	X	输入张量1	bool
			Y	输入张量2	bool
		Output	Z	输出张量	bool
		Attributes	axis	计算维度	int
			broadcast	是否允许广播	bool

scale 运算操作定义见表 126。

表 126 scale 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
scale	进行W * X操作，W是学习的参数	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64
		Attributes	W	系数	tensor
			shape	W的大小	list of ints

shift 运算操作定义见表 127。

表 127 shift 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
shift	进行X+b操作，b是学习的参数	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64
		Attributes	B	偏置	float
			shape	B的大小	list of ints

while_loop 运算操作定义见表 128。

表 128 while_loop 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
while_loop	实现类似while的循环控制功能	Input	X	输入张量列表	list of tensor
		Output	Y	输出张量列表	list of tensor
		Attributes	cond	判断是否执行循环	callable
			body	循环执行结构体	callable
			loop_vars	包含tensor的列表或元组	list of tensor
			is_test	可选，是否在测试阶段执行	bool
			name	可选，网络层输出权重的前缀标识，默认为None	string

zeros 运算操作定义见表 129。

表 129 zeros 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
zeros	创建形状为 shape 、数据类型为 dtype 且值全为0的张量	Output	Shape	需要输出的大小的张量	list of ints
			Y	输出张量	float16 float32 float64 int32
		Attributes	dtype	输出张量的数据类型	string

full_connected 运算操作定义见表 130。

表 130 full_connected 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
full_connected	全连接层	Input	X	输入张量	float16, float32, float64
			W	应用于输出的张量权重	float16, float32, double
		Output	Y	输出张量	float16, float32, float64
		Attributes	axis	batch_size维度，默认为0	int
			units	隐藏单元个数	int

expand_dims 运算操作定义见表 131。

表 131 expand_dims 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
expand_dims	扩展维度	Input	X	输入张量	float16, float32, float64
			ExpandTimes	输入张量，若此输入存在，则忽略expand_times属性	int32 int64
		Output	Y	输出张量	float16, float32, float64
		Attributes	axis	计算维度	int

batch_normalization 运算操作定义见表 132。

表 132 batch_normalization 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
batch_normalization	按照 [18] 所述进行批量标准化	Input	X	输入张量	float16, float32, float64
			W	应用于输出的张量权重	float16, float32, float64
			B	应用于输出的张量偏置	float16, float32, float64
			means	运行时的平均值或预测时的平均值	float16, float32, float64
			var	运行时的方差或预测时的方差	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64
			mean (optional)	操作后运行的平均值	float16, float32, float64
			var (optional)	操作后运行的方差	float16, float32, float64
			saved_mean (optional)	训练时保存的平均值，用于加速梯度计算	float16, float32, float64
			saved_var (optional)	训练时保存的方差，用于加速梯度计算	float16, float32, float64
batch_normalization	按照 [18] 所述进行批量标准化	Attributes	epsilon	用于防止除0的极小值	float
			is_training	判断操作用于训练还是预测	bool
			momentum	计算预测时平均值和方差的因素	float
			spatial	计算整个空间还是各个维度的均值和方差	bool
			data_format	channel_first或channel_last	string

clip 运算操作定义见表 133。

表 133 clip 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
clip	根据MAX和MIN限制输入张量元素的大小。	Input	X	输入张量	float16, float32, float64, int
		Output	Y	输出张量	float16, float32, float64, int
		Attributes	MAX	最大值，大于最大值的将会被最大值取代	float, int
			MIN	最小值，小于最小值的将会被最小值取代	float, int

elu 运算操作定义见表 134。

表 134 elu 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
elu	应用f(x) = alpha * (exp(x) - 1.) for x < 0, f(x) = x for x >= 0到输入张量	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64
		Attributes	alpha	elu的系数，默认为1.0	float

gemm 运算操作定义见表 135。

表 135 gemm 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
gemm	一般矩阵乘法，计算Y = alpha * A * B + beta * C	Input	X	输入张量1	float16, float32, float64
			Y	输入张量2	float16, float32, float64
			Z	输入张量3	float16, float32, float64
		Output	output	输出张量	float16, float32, float64

表 135 gemm 运算操作定义（续）

运算操作	描述	字段	关键字	定义	数据类型
gemm	一般矩阵乘法，计算Y = alpha * A * B + beta * C	Attributes	alpha	X * Y的系数	float
			beta	输入Z的系数	float
			broadcast	输入Z是否允许广播	int
			transX	输入X是否需要转置	int
			transY	输入Y是否需要转置	int

hard_sigmoid 运算操作定义见表 136。

表 136 hard_sigmoid 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
hard_sigmoid	应用y = max（0，min（1，alpha * x + beta））到输入张量	Input	X	输入张量	float16，float32，float64
		Output	Y	输出张量	float16，float32，float64
		Attributes	alpha	默认为2.0	float
			beta	默认为0.5	float

instance_normalization 运算操作定义见表 137。

表 137 instance_normalization 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
instance_normalization	按照 [19] 所述进行批量标准化	Input	X	输入张量	float16，float32，float64
			W	应用于输出的张量权重	float16，float32，float64
			B	应用于输出的张量偏置	float16，float32，float64
		Output	Y	输出张量	float16，float32，float64
		Attributes	epsilon	用于防止除0的极小值	float32

log_softmax 运算操作定义见表 138。

表 138 log_softmax 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
log_softmax	将输入X进行log(X)操作，然后进一步的预处理	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64
		Attributes	axis	计算维度	Int

lrn 运算操作定义见表 139。

表 139 lrn 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
lrn	应用（bias + （alpha / size）* sum（xi ^ 2）^ beta到每个输入元素	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64
		Attributes	alpha	尺度参数	float
			beta	指数	float
			bias	默认为0.1	float
			k	LpNorm 参数	int
			size	要计算的通道数量	int

leaky_relu 运算操作定义见表 140。

表 140 leaky_relu 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
leaky_relu	应用f(x) = alpha * x for x < 0, f(x) = x for x >= 0到每个输入元素	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64
		Attributes	alpha	默认为0.01	float

selu 运算操作定义见表 141。

表 141 selu 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
selu	应用y = gamma * (alpha * e^x - alpha) for x <= 0, y = gamma * x for x > 0到输入每个元素	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64
		Attributes	alpha	默认为1.6732	float
			gamma	默认为1.0507	float

prelu 运算操作定义见表 142。

表 142 prelu 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
prelu	应用f(x) = slope * x for x < 0, f(x) = x for x >= 0到每个输入元素	Input	X	输入张量	float16, float32, float64
			slope	输入斜率张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64

sigmoid 运算操作定义见表 143。

表 143 sigmoid 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
sigmoid	应用y = 1 / (1 + exp(-x))到输入张量X	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64

sigmoid_cross_entropy_with_logits 运算操作定义见表 144。

表 144 sigmoid_cross_entropy_with_logits 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
sigmoid_cross_entropy_with_logits	计算按元素的概率误差	Input	X	输入张量	float32 float64
			label	输入张量对应的标签	float32 float64
		Output	Y	输出张量	float32 float64
		Attributes	ignore_index	被忽略的目标值	int
			name	可选，网络层输出权重的前缀标识，默认为None	string
			normalize	如果为true，则将输出除以除去ignore_index对应目标外的目标数，默认为False	string

softmax 运算操作定义见表 145。

表 145 softmax 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
softmax	应用y = exp(x) / sum(exp(x))到输入张量x	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64
		Attributes	axis	计算维度	int

softplus 运算操作定义见表 146。

表 146 softplus 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
softplus	应用y = ln(exp(x) + 1)到输入张量X	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64

softsign 运算操作定义见表 147。

表 147 softsign 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
softsign	应用(x/(1+|x|))到输入张量X	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64

tanh 运算操作定义见表 148。

表 148 tanh 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
tanh	计算输入张量每个元素的双曲线正切	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64

reduce_log_sum 运算操作定义见表 149。

表 149 reduce_log_sum 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
reduce_log_sum	根据给定维度，计算每个输入元素的LogSum	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64
reduce_log_sum	根据给定维度，计算每个输入元素的LogSum	Attributes	axis	计算维度	int
			keepdims	是否保持维度	bool

reduce_log_sum_exp 运算操作定义见表 150。

表 150 reduce_log_sum_exp 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
reduce_log_sum_exp	根据给定维度，计算每个输入元素的LogSum	Input	X	输入张量	float16, float32, float64
		Output	Y	输出张量	float16, float32, float64
		Attributes	axis	计算维度	int
			keepdims	是否保持维度	bool

resize_bilinear 运算操作定义见表 151。

表 151 resize_bilinear 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
resize_bilinear	对输入的图像数据采用双线性插值方法resize到指定大小	Input	X	输入张量	float32 float64 uint8
			Shape	输入resize后的大小张量	int32 int64
		Output	Y	输出张量	float32 float64 uint8
		Attributes	name	开发人员打印调试信息时使用	string
			align_corners	若为True，则将输入和输出张量的4个角落像素中心对齐，并保留角点像素的值	bool
			align_mode	双线性插值选项	int
			data_format	指定输入的数据格式	string

resize_nearest 运算操作定义见表 152。

表 152 resize_nearest 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
resize_nearest	对输入的图像数据采用最邻近插值方法resize到指定大小	Input	X	输入张量	float32 float64 uint8
			Shape	输入resize后的大小张量	int32 int64
		Output	Y	输出张量	float32 float64 uint8
		Attributes	name	开发人员打印调试信息时使用	string
			align_corners	若为True，则将输入和输出张量的4个角落像素中心对齐，并保留角点像素的值	bool
			data_format	指定输入的数据格式	string

6.3.2 训练操作定义

6.3.2.1 损失函数

本章节定义神经网络训练过程中所需要的运算操作函数定义。损失函数计算函数定义见表 153。

表 153 损失函数计算函数定义（可选）

运算操作	描述	字段	关键字	定义
compute_loss	通过传递一个现有的损失函数名。为每个数据点返回一个标量损失。	Input	y_true	真实标签
			y_pred	预测值
			loss	损失函数
		Output	l	每个样本都有一个标量损失的张量
		Type Constraints	tensor(float16, float32, float64)	限制输入输出的类型为浮点张量

一些损失函数亦可通过较小颗粒度算子组装完成，如mean_squared_error（可用sub、square、cast、mul、reduce_mean组装）。其中的损失函数策略可包含：mean squared error, cross entropy, Kullback-Leibler, margin ranking, negative log likelihood, smooth L1。

训练优化函数可有多种表达，归一化的表达可按表 154 定义。

表 154 训练优化函数定义（可选）

运算操作	描述	字段	关键字	定义
apply_optimize	通过传递被优化的tensor、参数列表和优化器，计算梯度并对参数进行优化。	Input	loss	需要被优化的tensor
			val_list	参数列表
			optimizer	优化器
		Output	grads	梯度
		Type Constraints	tensor(float16, float32, float64)	限制输入输出的类型为浮点张量

其中支持的优化器：

a) SGD(lr=0.01, momentum=0.0, decay=0.0, nesterov=False)
   1) lr：大或等于0的浮点数，学习率
   2) momentum：大或等于0的浮点数，动量参数
   3) decay：大或等于0的浮点数，每次更新后的学习率衰减值
   4) nesterov：布尔值，确定是否使用Nesterov动量
b) RMSprop(lr=0.001, rho=0.9, epsilon=1e-06,momentum=0,centered=False)
   1) lr：大或等于0的浮点数，学习率
   2) rho：大或等于0的浮点数
   3) epsilon：大或等于0的小浮点数，防止除0错误
   4) momentum：大或等于0的浮点数
   5) centered：布尔值，确定是否使用中心化算法
c) Adagrad(lr=0.01, epsilon=1e-06)
   1) lr：大或等于0的浮点数，学习率

2) epsilon：大或等于0的小浮点数，防止除0错误

d) Adadelta(lr=1.0, rho=0.95, epsilon=1e-06)

1) lr：大或等于0的浮点数，学习率

2) rho：大或等于0的浮点数

3) epsilon：大或等于0的小浮点数，防止除0错误

e) Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08)

1) lr：大或等于0的浮点数，学习率

2) beta_1/beta_2：浮点数， 0<beta<1，通常很接近1

3) epsilon：大或等于0的小浮点数，防止除0错误

f) Adamax(lr=0.002, beta_1=0.9, beta_2=0.999, epsilon=1e-08)

1) lr：大或等于0的浮点数，学习率

2) beta_1/beta_2：浮点数， 0<beta<1，通常很接近1

3) epsilon：大或等于0的小浮点数，防止除0错误

g) Nadam(lr=0.002, beta_1=0.9, beta_2=0.999, epsilon=1e-08, schedule_decay=0.004)

a) lr：大或等于0的浮点数，学习率

b) beta_1/beta_2：浮点数， 0<beta<1，通常很接近1

c) epsilon：大或等于0的小浮点数，防止除0错误

6.3.2.2 反向操作定义

本章节定义神经网络表示训练相关的反向操作。反向操作描述神经网络计算图结构的梯度计算行为，是神经网络结构的重要组成部分。神经网络通过梯度更新的方法对参数进行训练，涉及到关键操作为求解损失函数关于参数的梯度。梯度的计算采用反向传播完成，给定输入特征和标准答案，先执行前向操作计算得到损失函数，然后执行反向操作（算子），通过反向传播算法求解模型参数的梯度。本部分通过一些算子来说明前向算子和反向算子的对应关系，大部分反向操作和前向操作存在一一对应关系，反向算子的输入为前向算子的输入、输出（可选）和损失函数值关于输出的导数，反向算子的输出为损失函数值关于前向算子输入变量的导数。其他反向算子可以参考这些示例对前向算子进行变换得到，如果没有特殊说明，默认采用以上规则进行反向算子的定义。

前向运算操作定义见表 155。

表 155 前向运算操作定义

运算操作	描述	字段	关键字	定义
add	执行两个输入的加法，在必要条件下将右侧的参数广播以匹配左侧输入的大小	Input	X	输入操作数 1
			Y	输入操作数2
		Output	Z	输出张量
		Type Constraints	tensor(float16, float32, float64)	限制输入输出的类型为浮点张量
		Attributes	axis (int)	计算维度
			broadcast (bool)	是否允许广播

反向运算操作定义见表 156。

表 156 反向运算操作定义

运算操作	描述	字段	关键字	定义
add_grad	执行两个输入的加法，在必要条件下将右侧的参数广播以匹配左侧输入的大小	Input	X	输入操作数1
			Y	输入操作数2
			dZ	损失函数值关于Z的导数
		Output	dX	损失函数值关于X的导数
			dY	损失函数值关于Y的导数
		Type Constraints	tensor(float16, float32, float64)	限制输入输出的类型为浮点张量
		Attributes	axis (int)	计算维度
			broadcast (bool)	是否允许广播

矩阵乘法前向运算操作定义见表 157。

表 157 矩阵乘法前向运算操作定义

运算操作	描述	字段	关键字	定义
mat_mul	矩阵乘法	Input	X	输入张量1
			Y	输入张量2
		Output	Z	输出张量
		Type Constraints	tensor(float16, float32, float64)	限制输入输出的类型为浮点张量

矩阵乘法反向运算操作定义见表 158。

表 158 矩阵乘法反向运算操作定义

运算操作	描述	字段	关键字	定义
mat_mul_grad	矩阵乘法反向操作	Input	X	输入张量1
			Y	输入张量2
			dZ	损失函数值关于Z的导数
		Output	dX	损失函数值关于X的导数
			dY	损失函数值关于Y的导数

		Type Constraints	tensor(float16, float32, float64)	限制输入输出的类型为浮点张量

全连接前向运算操作定义见表 159。

表 159 全连接前向运算操作定义

运算操作	描述	字段	关键字	定义
fc (fully_connected)	全连接层	Input	X	输入张量
			W	应用于输出的张量权重
			b	偏置
		Output	Y	输出张量
		Type Constraints	tensor(float16, float32, float64)	限制输入输出的类型为浮点张量
		Attributes	axis (int)	batch_size维度，默认为0
			units (int)	隐藏单元个数

全连接反向运算操作定义见表 160。

表 160 全连接反向运算操作定义

运算操作	描述	字段	关键字	定义
fc_grad	全连接层	Input	X	输入张量
			W	应用于输出的张量权重
			b	偏置
			dY	损失函数值关于Y的导数
		Output	dX	损失函数值关于X的导数
			dW	损失函数值关于W的导数
			db	损失函数值关于b的导数
		Type Constraints	tensor(float16, float32, float64)	限制输入输出的类型为浮点张量
		Attributes	axis (int)	batch_size维度，默认为0
			units (int)	隐藏单元个数

7 压缩过程

7.1 多模型

7.1.1 多模型技术定义

多模型压缩技术，其方法通过共享多个模型中的信息，实现总体存储数据量降低的方法。

多模型压缩技术，其中一种方法是在多个模型中，通过共享模型中的部分权重，达到多个模型参数量总和的减少，实现多模型整体压缩的目的。该方法除了可以减少模型的存储空间，在计算过程中，当计算单元在多个模型进行切换时，只需要读取模型中为共享部分的权重，减少需要读入的参数量，达到节省带宽的目的。这种方法可以兼容结构化矩阵、剪枝、量化等其他方法，用于继续压缩本方法压缩后的模型。

多模型残差量化，指量化基于模型在预训练模型的增量上进行。目前，目前，神经网络通常采用预训练模型作为初始化的方式来加速训练，提升网络性能。然而，传统的量化方法没有充分利用预训练模型，直接对原始权重进行量化导致实际应用中往往带有很高的性能损失。多模型差分量化方法，旨在对模型在预训练模型上的增量进行量化，在取得同样量化效果的前提下，尽可能保证了原始的性能。

7.1.2 多模型压缩操作

多模型压缩操作，输入为多个拓扑结构一致的模型，包含权重张量列表Weights_list以及可选的偏置向量列表Bias_list，一个压缩方法选项M_Method。其输出为一个权重张量W_share，一个可选的偏置向量B_share，一个权重张量列表W_specific，一个可选的偏置向量列表B_specific，以及错误指示Error。其中W_share与B_share中，保存多个模型间共享信息。W_specific与B_specific保存多模型间的附加信息。多模型操作定义见表161。

表 161 多模型压缩操作定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
MultimodelCompress	对输入的多个模型进行压缩操作	Input Attributes	Weights_list	输入权重张量列表	List of tensors	NS
			Bias_list	输入偏置向量列表	List of vectors	NS
			M_Method	具体压缩方法选项	value	int
		Output	W_share	输出的共享权重张量	tensor	NS
			B_share	输出的共享偏置向量	vector	NS
			W_specific	输出的特有权重张量列表	List of tensors	NS
			B_specific	输出的特有偏置向量列表	List of vectors	NS
			Error	错误指示	value	int

Q_Method定义见表162。

表 162 Q_Method 定义

Q_Method	
标号	类型
1	UNDEFINED
2	多模型层内权重共享压缩操作
3	多模型残差量化

——如果 Q_Method 的标号为 2，则参考多模型层内权重共享压缩（7.1.2）部分；
——如果 Q_Method 的标号为 3，则参考多模型残差量化（7.1.3）部分。

7.1.3 多模型层内权重共享压缩操作

7.1.3.1 定义

权重共享压缩方法定义针对神经网络处理单元可能执行的多个不同模型，这些模型的一些层拥有相同的神经网络层结构和相同的权重结构。在这些层中，以神经网络层为结构单元，以输出神经元为分组单位，上述神经网络层中一部分输出神经元对应的权重在多个神经网络模型中进行共享，其他的输出神经单元对应的权重则根据执行不同任务进行调整（图 5）。通过以上方法，使得神经网络中对应的网络层在处理信息时，既可以共享一部分多模型通用的用于数据处理的滤波器，也会有一部分针对本任务的优化。

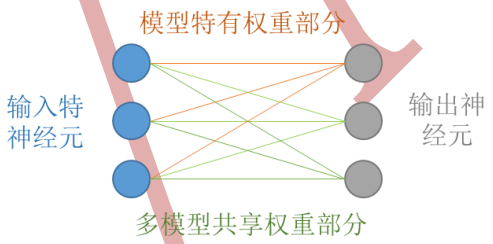


图 5 多模型间层内共享部分权重技术方案

7.1.3.2 权重聚合

权重聚合是指将 M 个维度为 N 且拓扑结构一致的权重张量，通过索引值规定的维度、聚合点位置信息，聚合为一个维度为 N 的共享权重张量和一个由 M 个维度为 N 的特有权重张量构成的列表。其实现流程如下：

a) 检查输入权重张量列表 Weights_list 中的张量数量与输入偏置向量列表 Bias_list 中的向量数量是否相等。如果不相等，错误指示项输出 1，其他输出项输出为 NULL，结束操作。
b) 检查输入权重张量列表 Weights_list 中所有张量是否拓扑结构一致以及输入偏置向量列表 Bias_list 中所有向量是否拓扑结构一致，如有任何不一致，错误指示项输出 2，其他输出项输出为 NULL，结束操作。
c) 如果权重聚合维度 axe 指向输出特征层数量这一维度，将当前状态定义为状态 1，否则定义为状态 2。

d) 在状态 1 下：将输入偏置向量列表 Bias_list 中的所有向量，从聚合点索引位置 slice_indicator，切分为两个向量 C 和 D。将所有输入权重张量列表 Weights_list 中的张量切分出的 B 按顺序组合为新列表 W_specific 并输出，将 A 作为 W_share 输出。将所有输入偏置向量列表 Bias_list 中的向量切分出的 D 按顺序组合为新列表 B_specific 并输出，将 A 作为 B_share 输出。将 Unit 作为 Unit_specific 输出，Error 输出 0。

e) 在状态 2 下：将所有输入权重张量列表 Weights_list 中的张量切分出的 B 按顺序组合为新列表 W_specific 并输出，将 A 作为 W_share 输出。将 Bias_list 作为 B_specific 输出，B_share 输出为空列表。将 Unit 作为 Unit_specific 输出，Error 输出 0。

多模型权重聚合操作的数据定义见表163。

表 163 多模型权重聚合操作数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
多模型权重聚合	将 M 个维度为 N 且拓扑结构一致的权重张量，通过索引值规定的维度、聚合点位置信息，聚合为一个维度为 N 的共享权重张量和一个由 M 个维度为 N 的特有权重张量构成的列表	Input	Weights_list	输入权重张量列表	List of tensors	NS
			Bias_list	输入偏置向量列表	List of vectors	NS
		Attributes	axe	权重聚合维度	value	int
			slice_indicator	聚合点索引	value	int
		Output	W_share	输出的共享权重张量	tensor	NS
			B_share	输出的共享偏置向量	vector	NS
			W_specific	输出的特有权重张量列表	List of tensors	NS
			B_specific	输出的特有偏置向量列表	List of vectors	NS
			Error	错误指示	value	int

多模型权重聚合操作的伪代码见表164。

表 164 多模型权重聚合伪代码

多模型权重聚合	描述符
Multilayer_weight_regroup(Weights_list, Bias_list, axe, slice_indicator, Error) {	
W_specific = []	
B_specific = []	
W_share = NULL	
B_share = NULL	
Error = 0	
if(lens(Weights_list)!= lens(Bias_list)),	
Error=1	

表 164 多模型权重聚合伪代码（续）

多模型权重聚合	描述符
else{	
for (i=0; i< lens(Weights_list)-1; i++){	
if (Weights_list[i].shape != Weights_list[i+1].shape)‖(Bias_list[i].shape != Bias_list[i+1].shape),	
Error=2	
}	
}	
if (Error==1) {	
W_share = slice(Weights_list[0], axe, 0, slice_indicator)	
for (i=0; i< lens(Weights_list); i++){	
W_specific.append(slice(Weights_list[0], axe, slice_indicator,-1))	
}	
if(axe = dimension_of_out_feature_map) {	
B_share = slice(Bias_list[0], axe, 0, slice_indicator)	
for (i=0; i< lens(Bias_list); i++){	
B_specific.append(slice(Bias_list[0], axe, slice_indicator,-1))	
}	
}	
else {	
B_ specific = Bias_list	
}	
}	
return W_share, W_specific, B_share, B_specific, Error	
}	

7.1.4 多模型残差量化

7.1.4.1 多模型残差量化方法定义

首先对于预训练模型，利用其进行模型重训练得到目标模型。在此假设预训练模型权重为W_0，重训练得到的目标模型为$W_i = W_0 + \Delta W$。对上面的残差权重ΔW进行权重共享，即同卷积层中，权重类别数目为2^n(n为量化比特，同时权重共享是针对每个卷积层的，不同卷积层之间不存在权重共享)。通过以上方式，使得神经网络卷积层可以通过权重共享的方式，建立code storage减少模型的存储。

7.1.4.2 权重共享

权重共享，也就是多模型残差量化方法的量化实现部分。权重共享方式参考deep compression的聚类方式。在得到预训练模型重训练获得增量后，对输入的模型差值进行聚类（分层进行）。同时，进行模型的fine-tuning，对于同一卷积层中，权重的梯度被替换为该类所有权重梯度的均值。

算法流程操作叙述：输入部分预训练模型权重张量，DF_model对应为与网络模型的差分权重，n决定量化位数，Weight_list为张量列表。

预训练模型权重为不训练部分（即设置为属性trainable=False），对于输入的差分权重，首先根据给定的量化比特进行K-means聚类（cluster函数）。对应每类只需存储其聚类中心。在重训练阶段（DFQuantization_weight_update，具体操作在9.1.2中介绍），属于同类的权重梯度被更新为该类梯度的均值。重训练结束后，生成对应的共享权重以及相对应的聚类结果，根据聚类结果可以恢复对应每个位置的权重。（残差量化只对对面位置的权重进行量化，对偏置不进行量化，正常存储以及进行训练）。

多模型残差量化的数据定义见表165。

表 165 多模型残差量化数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
多模型残差量化 DFQuantization	对输入的模型进行差分量化	Input	Base_model	输入预训练模型权重张量	tensor	NS
			DF_model	差分权重	tensor	NS
			Weights_list	输入权重张量列表	List of tensors	NS
		Attributes	n	量化位数	value	int
		Output	W_share	输出的共享权重张量	tensor	NS
			W_specific	code storage（权重聚类结果）	List	NS
			Error	错误指示	value	int

权重聚合数据定义见表 166。

表 166 权重聚合数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
权重聚类 Cluster	通过聚类的方式进行权重共享	Input	DF_model	差分权重	tensor	NS
		Attributes	n	量化位数	value	int
		Output	W_share	输出的共享权重张量	tensor	NS
			W_specific	code storage（权重聚类结果）	List	NS
			Error	错误指示	value	int

多模型残差量化的伪代码见表167。

表 167 多模型残差量化伪代码

多模型权重聚合	描述符
DFquantization(Base_model, DF_model, n, Weights_list) {	
W_specific = []	
W_share = NULL	
Error = 0	
if(DF_model == NULL),	
Error=1	
else{	
for (i=0; i< lens(Weights_list)-1; i++){	
W_share[i] W_specific[i], Error _1=Cluster(DF_model[i],n)	K-means聚类
if(Error=1),	
exit()	
}	
}	
if (Error==1)	
exit()	
W_share=DFQuantization_weight_update(W_share,Base_model,gradients,W_specific)	更新梯度
return W_share, W_specific, Error	
}	

7.2 量化

7.2.1 定义

量化压缩技术，其方法是利用给定的原始高精度复杂模型M，将原始模型中参数θ的位宽b由高比特位表示（如32位浮点型），变为低比特位宽表示。在这一过程中，可以把量化看做模型权重从连续空间到离散空间的一种映射关系$q:\theta \rightarrow \theta'$。目标是找到$q$，满足在量化后模型准确性没有明显损失的情况下，将$b$尽可能降低。在量化过程中，可能需要涉及到网络参数的重新训练，内部的参数也将会发生变化。

量化方法具有较强的兼容性，下面对标准中包含的其他压缩方法的联合使用进行描述。

a) 多模型压缩。

采用多模型权重共享以实现多模型的整体压缩，量化也可以在这种方法上进行应用。对于无需重训练的量化方法，可以直接在权重共享后，对权重直接进行量化操作进行压缩。对于需要重训练的量化方法，要按照权重共享时采用的训练方法进行重训练。对于共享的权重和特有的权重，在量化过程中可以依据其重要程度，采用不同的量化位宽，从而提高准确率。

b) 剪枝压缩。

对于剪枝方法，量化压缩可以应用于剪枝后的模型从而实现进一步的优化。在非结构化剪枝操作后，可以获得对应的稀疏掩码表，量化操作对于掩码表中为1的对应权重（即未被剪枝掉的权重）进行操作。而结构化剪枝由于删除了部分结构，因此可以直接对精简后的模型进行量化。

c) 结构化矩阵。

对于结构化矩阵压缩方法，需要先进行量化操作，简化矩阵数据中的内容，随后再按照结构化矩阵的方法进行处理。但是由于查找表量化方法涉及到额外的量化表索引信息，因此对原始权重的操作将转移到量化表索引上来。

在应用场景上，模型量化可以在不同的任务上使用，包括机器视觉任务（图像分类、目标检测、语义分割等），视频处理与分析任务（行为识别、目标跟踪）以及语音处理任务等。大部分的原始模型都是由32位浮点精度进行训练，因此有量化压缩的空间。在使用范围上，量化操作可以用于卷积神经网络中的卷积层、全连接层，循环神经网络中的细胞结构等，使用范围较广。从训练上来说，既可以从头开始直接训练低比特模型，也可以用已经训练好的模型进行压缩。在一些前期的测试中，量化方法在无需重训练的情况下，仍然可以保持原始准确率水平，并有4倍左右的压缩效果。

本章首先介绍两个基础量化操作，之后分别对参数与激活两类数据，提出了不同的量化方案。

7.2.2 基础量化操作

量化操作是对给定的连续类型输入张量，利用不同方法输出离散的量化后张量，作为父类的操作方法的定义与描述见表168和表169。其中，基础量化操作定义见表168。

表 168 基础量化操作定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
Quantization_tensor 基础量化操作	对输入的张量进行稀疏量化操作	Input	X	输入张量	List [tensor]	NS
			granularity	量化粒度	Value	int
			Q_Method	量化方法	Value	int
			kwarg	方法参数	Dict	Dict
		Output	Y	输出张量	list [tensor]	NS

其中的量化粒度granularity为量化方法在输入向量中不同层次结构量化范围的规定，分别包括统一量化、层级量化、通道量化、其他粒度量化。见表169。

表 169 granularity 量化粒度定义

标号	定义
1	统一量化
2	层级量化
3	通道量化
4	其他粒度量化

这些标号中：

——如果 granularity 的标号为 1，则对所有输入张量（假设有 n 个）进行统一量化；

——如果 granularity 的标号为 2，则对所有输入张量（假设有 n 个）按照层粒度进行量化；

——如果 granularity 的标号为 3，则对所有输入张量（假设有 n 个）按照通道粒度进行量化；

——如果 granularity 的标号为 4，则按照用户给定的粒度进行量化。

量化方法Q_Method表示了具体采用的量化方式，对应的标号含义为见表170。

表 170 Q_Method 量化方法定义

标号	作用对象	类型
1	参数、激活	线性量化
2	参数	查找表量化
3		非线性函数映射
4		定点量化
5		INT4量化
6		有界整流单元量化
7	激活	训练截断值
8		定点量化
10		INT4量化
11		有界整流单元量化
12		比例对齐量化

7.2.2.1 线性量化

线性量化首先设置需要量化到的比特位，然后通过对量化层的权重计算对应的尺度变换因子和偏置因子，将原本32比特的全精度权重量化为低比特的数值上去。线性量化的具体数据见表171。

表 171 线性参数量化定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
Linear_quantization 线性量化	将连续表示的高精度网络参数按层映射到低精度的范围	Input	X	粒度索引值对应权重张量	List [tensor]	NS
			X_scale	权重量化尺度因子	List [tensor]	NS
			X_offset	权重量化偏置因子	List [tensor]	NS
			N	量化比特数	value	int
			granularity	量化粒度	Value	int
		Output	Y	线性映射后的权重张量	List [tensor]	NS

线性参数量化的伪代码见表172。

表 172 线性参数量化伪代码

线性参数量化	描述符
def Linear_quantization (X, X_scale, X_offset, granularity,N){	
if(granularity==1): {	
Y = round(add(mul(X_scale, X),X_offset), 0)	
}	
elseif (granularity==2): {	
for(x in X): {	
y = round(add(mul(x_scale, x),x_offset), 0)	
}	
}	
else: {	
for(x in X): {	
for(i in [0: x.channels]): {	
y[i] = round(add(mul(x_scale[i], x[i]),x_offset[i]), 0)	
}	
}	
}	
return Y	
}	

7.2.2.2 查找表量化

查找表量化是指将原始的高精度浮点模型，利用给定的量化中心，转换为索引值表示的张量。量化原则为就近量化，指把原始参数量化到距离最近的量化中心值。量化后的索引张量仅有量化中心个数的索引范围，方便进一步的压缩。本方法主要适用于区间收缩量化生成数据后的压缩，相关的数据生成方法参见第9章内容。对于偏置张量，采取与权重张量相同的方式进行处理。查找表方法的具体数据定义见表173。

表 173 查找表量化定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
Codebook_quantization 查找表量化	将离散表示的张量参数转为依据索引表示的张量	Input	X	粒度索引值对应权重张量	List [tensor]	NS
			X_center	量化中心	List [tensor]	NS
			granularity	量化粒度	Value	int
		Output	Y	量化后的索引张量	List [tensor]	NS

其中，X为输入高精度张量；X_center为量化中心，每一个量化粒度的张量列表对应一组中心点；Y为量化后的索引张量，为整数的数据类型。伪代码见表174。

表 174 查找表量化伪代码

查找表量化	描述符
def Codebook_quantization(X, X_center):{	
X_shape = shape(X)	
if (granularity == 1):{	
X= reshape(X, (-1,1))	
for n in range(0, len(X_center):{	
dis[n] = abs(X –X_center [n])	
}	
Y = argmin(dis, axis=0)	
Y = reshape(Y, X_shape)	
}	
if (granularity == 2):{	
X = reshape(X, (X_shape[0],-1,1))	
for n in range(0,len(X_center):{	
for i in range(0, len(X_center[n]):{	
dis[n][i] = abs(X[n] –X_center[n][i])	
}	
}	
Y[n] = argmin(dis[n], axis=0)	
}	
Y = reshape(Y, X_shape)	
}	
if (granularity == 3){	
X = reshape(X, (X_shape[0], X_shape[1],-1,1))	
for n in range(0,len(X_center):{	
for i in range(0, len(X_center[n]):{	
for j in range(0, len(X_center[n][i]):{	
dis[n][i][j] = abs(X[n][i] –X_center [n][i][j])	
}	
}	
Y[n][i] = argmin(dis[n][i], axis=0)	
}	
Y = reshape(Y, X_shape)	
}	
return Y	
}	

7.2.3 参数量化操作

7.2.3.1 非线性函数映射

在非线性参数量化中，首先对网络权重做一个非线性映射变换，将原本呈现非线性分布的网络权重映射到新的维度空间上，从而呈现一个近似的线性分布，然后对映射后的权重进行线性量化。线性量化首先设置需要量化到的比特位，然后通过对量化层的权重计算对应的尺度变换因子和偏置因子，将原本32 比特的全精度权重量化为低比特的数值上去。该算法不依赖任何数据，通过预训练好的模型权重进行量化操作，因此可以并行处理所有需要量化的卷积层和全连接层。具体算法流程图如图 6 所示：

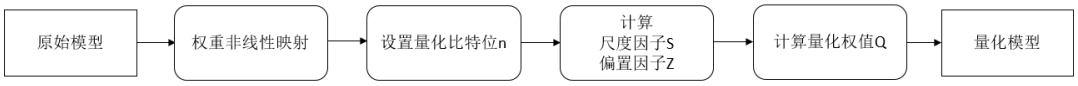


图 6 算法流程图

根据流程图所示，首先对权重进行非线性映射，并且设置需要量化到的比特位后，通过设置比特位的不同，计算出不同的尺度因子和偏置因子，这两个值决定了从全精度数值映射到低比特数值的函数变化。因此，计算出这两个因子之后，就可以求得低比特数值，并替换原本全精度权重，从而得到更小的模型。具体步骤如下。

a) 进行权重映射$\widetilde{\mathrm{W}} = f(\mathrm{W})$：在本章节中，可以使用以下三种映射方法 1)线性映射：$\widetilde{\mathrm{W}} = \mathrm{W}$。2)Log 映射：$\widetilde{\mathrm{W}} = \log_2|\mathrm{W}|$。3) Tanh 映射：$\widetilde{\mathrm{W}} = \tanh(\alpha \mathrm{W})$。

b) 设置量化比特位：根据不同的网络结构，设置不同的比特位 n，其取值范围一般在[1,8]的正整数上。

c) 计算量化层非线性变换后的权重$\widetilde{\mathrm{W}}$对应尺度因子 S 和偏置因子 Z,n 为压缩的比特位数

尺度因子 S 计算公式：

$$S = (\max(\widetilde{\mathrm{W}}) - \min(\widetilde{\mathrm{W}}))/(2^n - 1) \quad (11)$$

式中：

$\widetilde{\mathrm{W}}$——为经过映射之后的权重

S——为尺度因子

n——为压缩的比特位数

偏置因子 Z 计算公式：

$$Z = -\min(\widetilde{\mathrm{W}})/S \quad (12)$$

式中：

Z——为偏置因子

S——为尺度因子

具体操作定义见表175和表176，其中非线性参数量化数据定义见表175。

表 175 非线性参数量化数据定义表

运算操作	描述	字段	关键字	定义	数据类型	数据格式
Weight_non_Linear_quantization 参数非线性量化	将离散表示的高精度网络参数按层映射到低精度的范围	Input	W	粒度索引值对应权重张量	List [tensor]	NS
			granularity	量化粒度	Value	int
			function	非线性函数	Value	int
			N	量化比特数	Value	int
		Output	W_q	线性映射后的权重张量	List [tensor]	NS

具体采用的非线性函数（function）标号定义见表176。

表 176 可选非线性量化函数

标号	类型
1	线性函数
2	Log 函数
3	Tanh 函数
4	其他函数

相关伪代码见表177。

表 177 非线性参数量化伪代码

非线性参数量化	描述符
def Weight_non_Linear_quantization (W, granularity,function,N){	
if(function==1): {	
W = W	
}	
elif (function ==2): {	
W = abs(log(W))	
}	
elif(function ==3): {	
W = tanh(W)	
W_scale = (max(W)-min(W))/(2^n-1)	
W_offset = -1*min(W)/W_scale	
W_q = Linear_quantization (W, W_scale, W_offset, granularity,N)	
}	
return W_q	
}	

7.2.3.2 定点量化

本方法同时量化神经网络的参数（也称为权重，weights）和激活值（activation），量化限定为线性对称量化，每一个张量共享一个缩放系数，同时在定点量化中，缩放系数scale被限定为2的幂次，这样所有的乘法操作就全部可以通过移位来完成，硬件实现上会更加高效，非常适合全整数计算单元进行高效计算，比如FPGA等硬件平台。

定点量化的数据定义见表178。

表 178 定点量化数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
定点量化 fix_quantize_parameter	将高精度网络（FP32）量化成指定比特数的整型网络	Input	W	输入的待量化权重张量	tensor	float32
			B (optional)	网络层偏置张量	tensor	float32
			method	量化方法	value	int
			n	量化比特数	value	int
		Output	W_quantized	量化后的权重张量	tensor	int
			B_quantized(optional)	量化后的偏置张量	tensor	int
			W_p	权重张量的量化位置	value	int
			B_p	偏置张量的量化位置	value	int

具体采用的量化方法标号含义见表179。

表 179 定点量化 method 数据定义

标号	类型
0	NON_OVERFLOW
1	MIN-DIFF

以 8bit 量化为例，定点量化将一组浮点数表示成 8bit 的整数，也就是将浮点数X_f映射成$-128\sim127$之间的一个整数X_q。而缩放系数 scale 是量化间隔，在本操作中，缩放系数被限定为 2 的幂次，也就是 $scale = 2^{-p}$，这样的好处是在可以用移位运算来代替乘法。同时因为有了这样的限制，p 也有了一定的物理含义，它可以理解为小数点的位置，也称为定点位置，因此本量化方法称为定点量化。如图 7 所示，图中蓝色的 8 个数字表示 8bit 有效位。当 p=1 的时候，scale=0.5，相当于定点位置在 1。

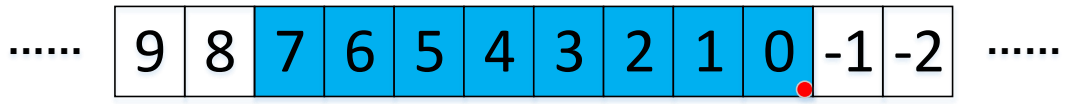


图 7 量化及定点位置

在均匀对称量化的前提下，一旦定点位置 p 确定，也就是 scale 确定之后，所有的量化和反量化操

作都是确定的。总结起来就是公式(13)-(15)，反量化后X_d值的范围是$[-128*2^{-p}, 127*2^{-p}]$，表 185 给出了一些不同 p 值对应的 scale 和所表示的浮点数的范围。

$$scale = 2^{-p} \quad \text{.....(13)}$$

式中：

scale —— 缩放系数；

p —— 定点位置。

$$X_q = round\left(X_f/scale\right) = round\left(2^p * X_f\right) \quad \text{.....(14)}$$

式中：

X_q —— 量化后的整数；

X_f —— 浮点数；

scale —— 缩放系数；

p —— 定点位置。

$$X_d = X_q * scale = 2^{-p} * X_q \quad \text{.....(15)}$$

式中：

X_d —— 反量化后的浮点数；

X_q —— 量化后的整数；

scale —— 缩放系数；

p —— 定点位置。

表 180 不同定点位置 p 对应的 scale 和所表示的浮点数的范围

p	scale	min	max
-2	4	-512	508
-1	2	-256	254
0	1	-128	127
1	0.5	-64	63.5
2	0.25	-32	31.75
3	0.125	-16	15.875
4	0.0625	-8	7.9375

根据前面的分析和总结出来的公式，在均匀对称量化以及缩放系数为 2 的幂次的限制条件下，量化最终需要确定的就是每个张量的缩放系数 scale，等价于小数点位置 p。由于 scale 是离散的数值，可以采用如下两种方法选择出最合适的 scale.

a) 方法 1：Non-overflow

第一种确定定点位置的方法是保证所有待量化的权重都没有溢出，可以表示为式(16)。Non-overflow 的策略保证了所有的数值都没有溢出，但这种策略容易受到野点的影响，一个野点会导致 scale 变大，影响整体的精度。

$$\mathrm{p} = \mathrm{floor}\left(-log_2\left(max(abs(X_{min})/128, abs(X_{\max})/127)\right)\right) \quad \text{.....(16)}$$

式中：

X_{min} —— 权重张量中参数的最小值；

X_{max} —— 权重张量中参数的最大值。

b) 方法 2：min-diff

第二种是保证量化前后，权重的差值尽可能小，见式(17)：

$$p = \text{argmin} \sum_i (X_{di} - X_{fi})^2 \quad \text{.....(17)}$$

式中：

X_{fi} —— 权重张量中第i个浮点数的值；

X_{di} —— 权重张量中第i个浮点数反量化后的值。

这种情况下，部分权重会被饱和截断。一般说来，相对于 Non-overflow，min-diff 方法会鲁棒一些。

表 181 是参数定点量化的伪代码，支持 NON_OVERFLOW 和 MIN_DIFF 两种方法。

表 181 定点量化伪代码

非线性参数量化	描述符
def calc_quantized_value(T, p): {	
sum = 0	
Td = T, Tq = T # dequantized tensor and quantized tensor	
for (idx,value) in enumerate(T): {	
v_q = round(value, pow(2,p))	
v_d = pow(2,-p)*v_q	
Td[idx] = v_d	
Tq[idx] = v_q	
}	
return (Td, Tq)	
}	
def quantize_tensor(T, method, n): {	
nbit_max = pow(2,n-1)-1 # for 8bit, nbit_max=127	
nbit_min = -pow(2,n-1) # for 8bit, nbit_min=-128	
(T_max,T_min) = minmax(T)	
bound = max(abs(T_max/nbit_max), abs(T_min/nbit_min))	
p = floor(-log2(bound))	
(Td,Tq) = calc_quantized_value(T,p)	
if method==0 #"NON_OVERFLOW": {	
return (Tq, p)	
}	
min_diff = L2norm(T, Td)	
best = (Tq, p)	

表 181 定点量化伪代码（续）

非线性参数量化	描述符
for i in range(1,4): {	
(Td,Tq) = calc_quantized_value(T, p+i):	
if L2norm(T, Td) < min_diff: {	
min_diff = L2norm(T, Td)	
best = (Tq, p+i)	
}	
}	
return best	
}	
def fix_quantize_parameter(W, B, method, n): {	
(W_q, W_p) = quantize_tensor(W, method, n)	
(B_q, B_p) = quantize_tensor(W, method, n)	
return (W_q, B_q, W_p, B_p)	
}	

7.2.3.3 INT4 参数量化

INT4量化操作，是指将一个高精度的张量量化为一个INT4张量或多个INT4张量的组合。一般情况下，量化为多个INT4张量时，采用Dual INT4量化。与INT4参数量化对应的激活量化方式为INT4激活量化。参数量化涉及神经网络权重量化和神经网络偏置量化，INT4参数量化中，对于神经网络偏置不作处理，对于神经网络权重，则逐层进行量化，流程如下。

a) 读取当前层输入的待量化权重张量 W 及量化尺度因子列表 sf_w_list，并根据权重量化偏置标识 offset_w 确定量化取值范围。

b) 根据待量化权重张量 W 确定当前网络层的类型，并采用不同粒度的量化方式。

1) 若网络层为卷积层，则进行卷积核粒度量化，即对每个输出 channel 对应的 3D 卷积核张量 W[k]进行量化：对于每一个卷积核张量 W[k]，根据对应的量化尺度因子列表 sf_w_list[k]的长度，确定量化方式为单 INT4 量化或 Multiple INT4 量化（一般为 Dual INT4 量化），并计算量化后的卷积核张量 W_quantized[k]。

2) 若网络层为全连接层，则进行层级粒度量化，即对输入层的权重张量 W 进行量化：根据对应的量化尺度因子列表 sf_w_list，确定量化方式为单 INT4 量化或 Multiple INT4 量化（一般为 Dual INT4 量化），并计算量化后的卷积核张量 W_quantized。

3) 其中，进行 Dual INT4 量化时，量化张量根据最小化重建误差（MMSE）确定。对于待量化张量 T，给定对应的量化尺度因子，令$\tilde{T}_j^1$, $\tilde{T}_j^2$为量化后的张量$\tilde{T}^1$, $\tilde{T}^2$的第 j 个索引上的元素，则$\tilde{T}_j^1$ 和 $\tilde{T}_j^2$的计算方法为式(18)所示：

$$min_{\tilde{T}_j^1,\tilde{T}_j^2}\left(T_j-\alpha_1\,\tilde{T}_j^1-\alpha_2\,\tilde{T}_j^2\right)^2 \Leftrightarrow min_{\tilde{t}\in\mathbb{Z}_q}\left(min_{\tilde{T}_j^2}\left(T_j-\alpha_1\,\tilde{t}-\alpha_2\,\tilde{T}_j^2\right)^2\right)$$

$$\Leftrightarrow min_{\tilde{t}\in\mathbb{Z}_q}\left(T_j-\alpha_1\,\tilde{t}-\alpha_2\left[\frac{T_j-\alpha_1\,\tilde{t}}{\alpha_2}\right]_{\mathbb{Z}_q}\right)^2 \quad\cdots\cdots(18)$$

式中：

$\mathbb{Z}_q$——量化范围。

INT4参数量化的数据定义见表182。

表 182 INT4 参数量化数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
INT4 参数量化 int4_quantize_parameter	将高精度（FP32）表示的神经网络权重量化为 INT4 格式的张量	Input	W	输入的待量化权重张量	tensor	float32
			B (optional)	网络层偏置张量	tensor	NS
			sf_w_list	量化尺度因子列表	list	float32
			offset_w	权重量化偏置标识	value	Boolean
		Attribute	range_min	量化取值范围下界	value	INT4
			range_max	量化取值范围上界	value	INT4
		Output	W_quantized	量化后的权重张量	list[tensor]	INT4
			B_quantized (optional)	量化后的网络层偏置张量	tensor	NS

INT4参数量化的伪代码见表183。

表 183 INT4 参数量化伪代码

INT4参数量化	描述符
def int4_quantize_parameter(W,sf_w_list,offset_w,bit_w=4, B): {	
if offset_w==True: {	
range_max=2**bit_w-1	
range_min=0	
W-=min(W)	
}	

表 183 INT4 参数量化伪代码（续）

INT4参数量化	描述符
else: {	
range_max=2**(bit_w-1)-1	
range_min= -2**(bit_w-1)	
}	
if len(W.shape)>2: {	
W_quantized=[]	
for k in range(len(W)): {	
if len(sf_w_list[k])==1:	
W_quantized[k]=clip(round(W[k]/sf_w_list[k][0]),range_min,range_max)	
if len(sf_w_list[k])==2:	
W_quantized[k]=get_params_double_grid(W[k],sf_w_list[k],range_min,range_max)	
}	
}	
else: {	
if(len(sf_w_list)==1:	
W_quantized=clip(round(W/sf_w_list[0]),range_min,range_max)	
if len(sf_w_list)==2:	
W_quantized=get_params_double_grid(W,sf_w_list,range_min,range_max)	
}	
B_quantized=B	
return W_quantized, B_quantized	
}	

其中，get_params_double_grid()操作的伪代码见表184。

表 184 get_params_double_grid()操作伪代码

get_params_double_grid()操作	描述符
def get_params_double_grid(T,sf,range_min,range_max,bit_w=4): {	
w1=linspace(range_min,range_max,2**bit_w).reshape(1,-1)	
w1=tile(w1,[len(T),1])	
w2=round((T-(w1*sf[0]))/sf[1])	
w2=clip(w2,range_min,range_max)	
t_temp=(T-w1*sf[0]-w2*sf[1])**2	
index=argmin(t_temp,1)	
range_vec=range(w1.shape[0])	
T1=w1[range_vec, index].reshape(T.shape)	

表 184 get_params_double_grid()操作伪代码（续）

get_params_double_grid()操作	描述符
T2=w2[range_vec, index].reshape(T.shape)	
Tq=[T1,T2]	
return Tq	
}	

7.2.3.4 有界整流单元参数量化

本节涉及的参数量化属于线性量化方法，在此作为一个推荐选项，调用基础量化中的线性量化操作和乘法操作。具体定义见表185。

表 185 有界整流单元参数量化定义表

运算操作	描述	字段	关键字	定义	数据类型	数据格式
brelu_quantizer_parameter 有界整流单元参数量化	将连续表示的高精度网络参数按层映射到低精度的范围	Input	X	粒度索引值对应权重张量	List [tensor]	NS
			X_scale	权重量化尺度因子	List [tensor]	NS
			X_offset	权重量化偏置因子	List [tensor]	NS
			N	量化比特数	value	int
			granularity	量化粒度	Value	int
		Output	Y	线性映射后的权重张量	List [tensor]	NS

有界整流单元的参数量化伪代码见表186。

表 186 有界整流单元参数量化伪代码

有界整流单元参数量化	描述符
def brelu_quantizer_parameter (X, X_scale, X_offset, granularity,N):	
Return Linear_quantization(X, X_scale, X_offset, granularity,N)	

7.2.4 激活量化操作

7.2.4.1 训练截断值

量化需要对超出低比特表示空间的异常值进行处理，而基于梯度训练的截断值能够使得量化截断值更具有泛化性、稳定性和自适应性。

训练截断值量化的具体数据定义见表187。

表 187 训练截断值量化定义表

运算操作	描述	字段	关键字	定义	数据类型	数据格式
Trainable_alpha_quantization 训练截断值量化	将量化的截断值按层进行训练	Input	X	待量化的激活值张量	List [tensor]	NS
			Alpha	可训练的截断上界值	List [tensor]	NS
			Granularity	量化粒度	Value	int
			N	量化比特数	Value	int
		Output	X_q	量化后的激活值张量	List [tensor]	NS

其中，X为输入的高精度张量，Alpha是可训练的截断量化参数，X_q是量化后的张量，该算法的伪代码见表188。

表 188 训练截断值量化伪代码

训练截断值量化	描述符
def Trainable_alpha_quantization(X, Alpha, Granularity ,N):{	
X_Scale = 2**(n-1)/Alpha	
X_offset = 0	
X = $\max(\min(X, Alpha), -Alpha)$	
X_q = Linear_quantization (X, X_scale, X_offset, Granularity,N)	
return X_q	
}	

7.2.4.2 定点量化

本小节中针对激活值 feature map 进行统计量化的方法跟 7.2.3.2 小节中针对权重的定点量化方法配套使用。权重和激活值的缩放系数都被限定为 2 的幂次。

与权重不同，激活值取决于神经网络的输入，并不是固定不变的。因此在量化激活值的时候需要一定的校准数据，对每一个 Batch 的输入数据，采用跟权重相同的方法量化激活值，并记录这一个 batch 的定点位置 p. 在运行多个 batch 的校准数据之后，统计每个激活值张量定点位置的分布，选择其中的最大值作为最终的定点位置，完成激活值的量化。

定点量化激活值的数据定义如表 189 所示。

表 189 定点量化激活值数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
fix_quantize_activation 激活值定点量化	将高精度（FP32）表示的神经网络激活量化为指定比特数的整型网络	Input	X	输入的待量化激活张量	tensor	float32
			method	量化方法	value	int
			n	量化比特数	value	int
			calibration_dataset	校准数据集	list[tensor]	float
		Output	X_quantized	量化后的激活张量	tensor	int
			X_p	激活张量的量化位置	value	int

定点量化激活值的伪代码见表 190，这里的伪代码仅展示了计算的过程，没有考虑整体的优化。实际上，我们往往是在需要量化的激活值后面加上量化的 op，把激活值的量化计算放到这个 op 中。这样，对校准数据集进行一轮完整的前向计算可以算出网络中所有激活值的量化结果，而不需要对每一个待量化的激活值，都去进行一次网络的前向计算。网络整体的量化过程请参考定点量化数据生成章节。

表 190 定点量化激活值伪代码

定点量化激活值	描述符
Def fix_quantize_activation(X, method, n, calibration_dataset):	
{	
# p is used to record quantize position for all batches	
p = [0] *len(calibration_dataset)	
for i in range(len(calibration_dataset): {	
batch_data = calibration_dataset[i]	
net.forward(batch_data) # forward with data	
Xi = get_activation(net) #	
_, p[i] = quantize_tensor(Xi, method, n) # position for ith batch	
}	
Xp = count_max(p) # count and select the most frequent position	
(Xd, Xq) = calc_quantized_value(X, Xp)	
return (Xq, Xp)	
}	

7.2.4.3 INT4 激活量化

INT4量化操作，将一个高精度的张量量化为一个INT4张量或多个INT4张量的组合。一般情况下，量化为多个INT4张量时，采用Dual INT4量化。与INT4激活量化对应的参数量化方式为INT4参数量化。逐层对神经网络的激活进行INT4权重量化，流程如下。

a) 读取当前层输入的待量化激活张量 X 及量化尺度因子列表 sf_act_list，并根据激活量化偏置列表 offset_act_list 确定量化取值范围。

b) 根据量化尺度因子列表中的元素，依次进行线性量化，直到循环结束：

1) 若尺度因子列表长度为 1，则直接对激活张量进行量化；

2) 若尺度因子列表长度大于 1，则从第二次开始，每次对激活张量的残余张量进行量化。

INT4激活量化的数据定义见表191。

表 191 INT4 激活量化数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
int4_quantize_activation INT4 激活量化	将高精度表示的神经网络激活量化为 INT4 格式的张量	Input	X	输入的待量化激活张量	tensor	float32
			sf_act_list	量化尺度因子列表	list	float32
			offset_act_list	激活量化偏置列表	list	float32
		Attribute	range_min	量化取值范围下界	value	INT4
			range_max	量化取值范围上界	value	INT4
		Output	X_quantized	量化后的激活张量	list[tensor]	INT4

INT4激活量化的伪代码见表192。

表 192 INT4 激活量化伪代码

INT4激活量化	描述符
def int4_quantize_activation(X,sf_act_list,offset_act_list=None,bit_w=4,X_quantized=[]): {	
if offset_act_list is not None: {	
range_max=2**bit_w-1	
range_min=0	
x_quantized=clip(round((X-offset_act_list[0])/sf_act_list[0]),range_min,range_max)	
out=sf_act_list[0]*x_quantized+offset_act_list[0]	
}	
else: {}	
range_max=2**(bit_w-1)-1	
range_min= -2**(bit_w-1)	
x_quantized=clip(round(X/sf_act_list[0]), range_min, range_max)	

表 192 INT4 激活量化伪代码（续）

INT4激活量化	描述符
out=sf_act_list[0]*x_quantized	
}	
X_quantized.append(x_quantized)	
if len(sf_act_list)>1: {	
X=X-out	
if offset_act_list[1:] is not None:	
X_quantized=int4_quantize_activation(X,sf_act_list[1:],offset_act_list[1:], bit_w=4, X_quantized)	
else:	
X_quantized=int4_quantize_activation(X,sf_act_list[1:],None,bit_w=4, X_quantized)	
}	
return X_quantized	
}	

7.2.4.4 有界整流单元激活量化

有界整流单元激活量化定义见表193。

表 193 有界整流单元激活量化

运算操作	描述	字段	关键字	定义	数据类型	数据格式
BreLU 有界整流单元量化	利用有界整流单元量化特征图	Input	X	输入张量	Tensor	NS
			N_sigma	设置截断比例	value	float
			Scale_X	X 对应的 scale	value	float
			Stage	量化阶段	value	Int
		Attribute	Bound_f	截断值（量化前）	value	float
			n=0	样本数	value	Int
			Bound_i	截断值（量化后）	value	Int
			Mul	乘数	value	Int
			Shift	右移数	value	int
		Output	Y	线性映射后的特征图	List	NS
			Y_scale	Y 对应的 scale	value	float

Stage 参数定义见表 194。

表 194 stage 定义

标号	类型
1	高精度量化阶段
2	训练阶段
3	低精度推理阶段

算法描述：

当 stage 为 1 时，统计 bound_f 和 n。

该方案需要X的缩放因子和偏移因子X_{scale}和X_{offset}。该参数由之前每层的缩放因子和偏移因子计算而来。

给定输入的张量X，首先记录X的所有元素中最大的一定比例的值的最小值作为上界，见表200：

表 195 N_sigma 定义

比例	N-sigma
99.865%	3
99.921%	3.2
99.977%	3.5

利用X的多次训练输入可以得到多个对应的上界b取其均值。把该均值作为该张量的截断值（高精度浮点数）b_f。

b_f的量化：根据该张量对应的低精度整数缩放比例X_{scale}和X_{offset}，求量化后的b_i，并对其进行反量化作为新的b_f：

$$b_i = round(b_f * X_{scale} + X_{offset}) \quad (19)$$

$$b_f = (b_i - X_{offset})/X_{scale} \quad (20)$$

当stage为2时，利用b_f对浮点数张量进行截断：

$$Y_f = \text{clip}(X, \max = b_f) \quad (21)$$

当stage为3时，利用b_i对整数张量进行截断：

$$Y_i = \text{clip}(X, \max = b_i) \quad (22)$$

b默认的初始化参数是b3。如果有验证集，则可以尝试多个b并选取性能最高的一个。

得到b之后就可以调用基础量化中的线性量化（整数域）对Y_i进行量化。 伪代码见表196。

表 196 有界整流单元激活量化伪代码

BReLU	描述符
def BReLU(X,N_sigma=3,Stage,Scale){	
if stage==1:{	
Bound=n_sigma_of(X,N_sigma)	
self.Bound_f+=(self.Bound_f*self.n+Bound)/(n+1)	
Self.n+=1	

表 196 有界整流单元激活量化伪代码（续）

BReLU	描述符
if self.n==50:{	
self.Bound_i=round(self.Bound_f*X_scale/self.n)	
self.mul,self.shift=mul_shift(self.bound_i)	
Y_scale=X_scale*mul/(2**shift)	
self.Bound_f=127/Y_scale	
}	
}	
return X, Y_scale	
elif stage==2:	
return clip(X,self.Bound_f), 1	
else: {	
X = clip(X,self.Bound_i)	
return ((X+2**(shift-1))*mul)>>shift, 1	
}	
def n_sigma_of(x,n_sigma):{	
if n_sigma==3:	
rate=0.99865	
elif n_sigma==3.5:	
rate=0.99977	
else:	
rate=0.99865	
sorted_x,_=torch.sort(x.view(-1))	
index=round(sorted_x.size()[0]*rate)	
bound=sorted_x[index].item()	
return bound	
}	
def mul_shift(bi) {	
for i in range(1,28):	
max_int=(2^(n-1)+0.5)*2**i	
if max_int>bi: {	
mul=round(max_int/bi)	
temp=max_u*mul/2**i	
if(abs(temp-2^(n-1))<0.5): {	
shift=i	

表 196 有界整流单元激活量化伪代码（续）

BReLU	描述符
return mul,shift	
}	
}	
return mul,shift	
}	

7.2.4.5 比例对齐量化

比例对齐量化定义见表197。

表 197 比例对齐量化定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
Scale_syncronization 量化比例对齐	将多个scale不同的输入张量处理为scale相同的张量，计算rescale参数	Input	X	输入张量	List [tensor]	int
			X_scales	X对应的scale	List [tensor]	Float
			Stage	量化阶段	Value	Int
			Method	量化算法	value	Int
			Granularity	量化粒度	value	int
		Attribute	params	量化参数	List [tensor]	NS
		Output	Y	线性映射后的权重张量	List [tensor]	NS

Method仅定义1，即乘后右移量化，Granularity仅定义层级量化。

该方法适用于输入的多个张量下一步将进行相加、相减或连接的操作，并且只适用于有一个以上的张量输入，否则直接跳过。

当stage=1时，计算其量化参数此时的输入X应当提供各自对应的缩放比例，并计算attribute。

用$Y_1, Y_2, \ldots, Y_n$来表示n个输入张量，即X列表中的每个元素，其对应的缩放比例分别为$Y_{scale1}, Y_{scale2}, \ldots, Y_{scalen}$,其上一次卷积对应的卷积核W的缩放比例为$W_{scale1}, W_{scale2}, \ldots, W_{scalen}$。

首先选定输入张量其中的一个作为参考张量Y_0（该参考张量可以由参数指定，默认取第一个），其缩放比例为Y_{scale0}。

对于其他每个张量Y_j进行如下操作（0<j<n）：

计算其scale倍数差距：

$$scale_{diff} = Y_{scale0}/Y_{scalej} \quad \ldots\ldots(23)$$

计算乘数mul和右移数shift：

$$\text{mul}, \text{shift} = \text{mul\_shift\_f}(scale_{diff}) \quad \ldots\ldots(24)$$

其中mul_shift_f方法定义见表198。

针对每个输入张量纪录对应的mul和shift（Y_0对应输出为（1，0））。

注意：当stage为1时需要计算新的Y_{scalej},即$Y_{scalej'}$:

$$Y_{scalej'} = Y_{scale0} * 2^{shift}/mul \quad (25)$$

更新W_{scalej}:

$$W_{scalej} = Y_{scalej'} * W_{scalej}/Y_{scalej} \quad (26)$$

更新Y_{scalej}:

$$Y_{scalej} = Y_{scalej'} \quad (27)$$

当训练时，不做操作。

当低精度推理时，对Y_0以外的张量进行量化操作：

$$Y_{qj} = \left(Y_j * mul + 2^{shift-1}\right) \gg \text{shift} \quad (28)$$

其中>>表示整数右移操作。

其伪代码见表198。

表 198 比例对齐量化伪代码

Scale_syncronization	描述符
Def Scale_syncronization (X,scale,stage,method,granularity) {	
if stage==1: {	
Scale0,pos=scale[0]	
for i in range(len(scale)-1): {	
Mul,shift=mul_shift_f(scale0/scale[i+1])	
Self.params.append([mul,shift])	
}	
return X	
}	
elif stage==2: {	
Return X	
}	
else: {	
For i in range(len(X)-1): {	
Mul,shift=self.params[i+1]	
X[i+1]=(X[i+1]*mul+2^(shift-1))>>shift	
}	
Return X	
}	
}	
def mul_shift_f(scale,precision=0.02): {	
for i in range(0,28): {	
max_int=2**i	

表 198 比例对齐量化伪代码（续）

Scale_syncronization	描述符
if max_int>scale: {	
temp=max_int/ratio	
mul=round(temp)	
if(abs(max_int/mul-scale)<precision*scale): {	
shift=i	
return mul,shift	
}	
}	
}	
return mul,i	
}	

7.3 剪枝

7.3.1 概述

剪枝方法具有较强的兼容性，下面对其他压缩方法的联合使用做一简单分析。

a) 多模型压缩。

采用多模型权重共享以实现多模型的整体压缩，两种剪枝方法在也可以在这种方法上进行应用。首先利用剪枝方法对原始多个模型进行剪枝压缩，然后利用多模型压缩算法进一步压缩。

b) 量化

对于量化方法，可应用于剪枝后的模型从而达到进一步的优化。在非结构化剪枝操作后，可以获得对应的稀疏掩码表，量化操作对于掩码表中为1的对应权重（即未被剪枝掉的权重）进行操作。而结构化剪枝由于删除了部分结构，因此可以直接对精简后的模型进行量化。

c) 结构化矩阵。

对于结构化矩阵方法，需要先进行剪枝操作，剪枝矩阵数据中的内容，随后再按照结构化矩阵进行处理。

7.3.2 剪枝操作

神经网络剪枝操作将预训练神经网络模型的权重按照给定的准则划分重要性先后顺序，根据用户指定的压缩率/加速比例，从网络中删除相应比例重要低的权值的操作。具体定义见表199，200。

表 199 剪枝操作定义

操作	描述	字段	关键字	定义	数据类型	数据格式
Pruning	对输入的权重进行剪枝操作	Input	X	输入权值张量	List of vectors	NS
			granularity	剪枝粒度	Value	Int
			Y	掩码	List of vectors	NAS
		Output	W	输出权值张量	List of vectors	NAS

表 200 剪枝粒度定义

剪枝粒度	
标号	类型
1	UNDEFINED
2	行剪枝/卷积核个数剪枝
3	列剪枝/卷积核形状剪枝

剪枝过程如下。

a) 输入参数：待稀疏化张量列表W;待稀疏化偏置张量列表B;掩码向量列表S;N的维度dim（考虑张量存储格式不一定是NCHW，所以需要用户指定N的维度）；剪枝粒度 granularity。

b) 输出参数：稀疏化的张量列表W_o。

步骤：

用以下步骤遍历张量列表中的所有张量w：

a) 获取 w 的形状。

b) 将w根据N所在的维度进行 (N, C*H*W) w.reshape(w_tensor_shape[dim],-1,1)

c) 根据输入掩码，删除掩码vector中，掩码对应为0 的行。

d) 讲w reshpe为原来的形状，reshape 时，当粒度为行剪枝时，对应的维度进行相应的缩减，当粒度为列剪枝时，直接还原。

操作定义见表201。

表 201 结构化剪枝操作定义

操作	描述	字段	关键字	定义	数据类型	数据格式
剪枝	对网络权重进行结构化剪枝	Input	W	权重张量	List of vectors	NS
			B (optional)	偏置张量	List of vectors	NS
			Y	掩码	List of vectors	NS
			N_dim	N 所在的维度	value	Int
			granularity	粒度	value	Int
		Output	W_o	剪枝之后的权重张量	List of vectors	NS
			B_o	剪枝之后的偏置张量	List of vectors	NS

伪代码见表202：

表 202 剪枝伪代码描述

剪枝	描述符
structured_sparse(W,B,S,N_dim, granularity){	
for(i=0;i<length(W),i++){	
w = W[i];	

表 202 剪枝伪代码描述（续）

剪枝	描述符
b = B[i];	
w_shape = w.tensor_shape;	
w.reshape(w_shape[N_dim],-1);	
b.reshape(w_shape[dim],-1);//reshape 成(N, C)	
If(granularity==2){	
for(j=0;j<length(S[i]);j++){	
If(S[i][j]==0){	
remove(w[j])	
remove(b[j])	
}	
}	
w_shape[dim] = sum(S[i])	
W_o[i] = w.reshape(w_shape)	
}	
If(granularity==3){	
skip_col = S[i] \\\S = Skip_column?	
compress_weights = matrix (shape= (w.row, w.col – len(skip_col))	
compress_bias = vector (shape= (len(b))	
compress_col_idx = 1	
for col_idx = 1 to w.col {	
zero_flag = 1	
for val in w[:, col_idx] {	
if (val != 0){	
zero_flag = 0	
break	
}	
}	
if (zero_flag == 1)	
continue	
else{	
compress_weights [:, compress_col_idx] = w[:, col_idx]	
compress_col_idx = compress_col_idx + 1	
}	
}	
compress_bias[:] = b[:]	
W_o[i] = compress_weights.reshape(w_shape)	

表 202 剪枝伪代码描述（续）

剪枝	描述符
B_o[i] = compress_bias[:]	
}	
}	
}	

7.4 结构化矩阵

7.4.1 结构化矩阵压缩

结构化矩阵压缩操作是指将神经网络中的权重矩阵等数据，压缩为某种格式的结构化矩阵的操作。具体定义见表203，204，205。结构化矩阵压缩操作数据定义见表203。

表 203 结构化矩阵压缩操作数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
CompressStructural Matric	将输入的权重矩阵压缩为给定格式的结构化矩阵的操作	Input	X	输入张量	Tensor	NS
			granularity	结构化矩阵压缩粒度	Value	UINT
			S_Method	结构化矩阵压缩方法	Value	UINT
		Output	Y	输出张量	Tensor	NS

结构化矩阵压缩粒度定义见表 204。

表 204 结构化矩阵压缩粒度定义

granularity	
标号	类型
1	UNDEFINED
2	所有输入张量统一进行结构化矩阵压缩
3	层级粒度结构化矩阵压缩
4	通道级粒度结构化矩阵压缩
5	自定义结构化矩阵压缩粒度

压缩方法S_Method定义见表205。

表 205 压缩方法 S_Method 定义

S_Method	
标号	类型
1	UNDEFINED
2	分块循环矩阵压缩
3	低秩组稀疏分解的结构化矩阵压缩

7.4.2 带符号向量的分块循环矩阵压缩方法

7.4.2.1 定义

循环矩阵的当前行/列为前一行/列的循环移位。当输入矩阵不为方阵时，需要将输入矩阵补齐或截取为方阵进而进行压缩，会导致存储空间浪费或网络性能损失。分块循环矩阵约束输入矩阵由多个子循环矩阵组成，从而实现存储空间的有效利用。进一步的，对使用分块循环矩阵进行训练和推理时，引入符号向量，能够加速网络收敛，提高网络性能。

7.4.2.2 分块循环矩阵压缩操作

当前层使用分块循环矩阵技术压缩时，本操作根据分块循环矩阵和当前层权重矩阵维度信息，对压缩后的权重矩阵进行压缩和存储。

分块循环矩阵压缩的流程如下。

a) 读取当前层分块循环矩阵维度信息；

b) 根据分块循环矩阵尺寸，从上述 2 维权重矩阵中读取子矩阵首行权重数值，存储至压缩后的权重矩阵中；

c) 重复上述操作至遍历整个 2 维权重矩阵，返回压缩后的权重矩阵。

分块循环矩阵压缩操作的具体数据定义见表206。

表 206 分块循环矩阵压缩操作数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
分块循环矩阵压缩 compress_weight_to_w_circ	将当前层权重矩阵使用分块循环矩阵进行压缩	Input	weight	输入的待压缩权重张量	matrix	NS
			circ_size	分块循环矩阵尺寸	value	UINT
		Output	w_circ	压缩后的权重张量	matrix	NS

该操作对应伪码实现见表 207。

表 207 分块循环矩阵压缩操作伪代码

多模型权重聚合	描述符
def compress_weight_to_w_circ(weight, circ_size): {	
height_ori, width_ori = weight.shape()	
for i in range(0, height_ori, circ_size): {	

表 207 分块循环矩阵压缩操作伪代码（续）

多模型权重聚合	描述符
w_circ[i/circ_size][:] = weight[i][:]	
}	
return w_circ	
}	

7.4.2.3 随机向量维数列表及参考随机向量生成操作

为了提高压缩率，整个网络共享全局基础随机符号向量（参考随机向量），每个待压缩层存储当前层的随机符号向量的相关参数（随机向量维数），具体生成流程如下：

a) 根据待压缩层输入特征向量的维度和权重矩阵的分块大小，确定各待压缩层随机向量的维数:
   1) 确定目标区间：以权重矩阵的分块大小和带待压缩层输入特征向量的维度为目标区间端点；
   2) 在目标区间内随机选取一个整数，作为该待压缩层随机向量的维数；
   3) 遍历各待压缩层，得到整个网络的待压缩层随机向量维数列表len_list。
b) 计算参考随机向量的维数：取全部待压缩层随机向量维数的最大值作为参考随机向量的维数；
c) 生成参考随机向量base_sign_vector_network：基于参考随机向量维数随机生成符合二项分布的符号向量。

待压缩层随机向量维数列表len_list生成操作的具体数据定义见表208。

表 208 随机向量维数列表生成操作数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
压缩层随机向量维数生成 generate_len_list	生成网络的待压缩层随机向量维数列表	Input	layer_indicator	使用符号向量和分块循环矩阵技术的层索引	list [int]	UINT
			input_length_list	神经网络输入向量的维数列表	list[int]	UINT
			circ_size_list	分块循环矩阵尺寸列表	list [int]	UINT
		Output	len_list	待压缩层随机向量维数列表	list[int]	UINT

待压缩层随机向量维数列表len_list生成操作的伪代码见表209。

表 209 待压缩层随机向量维数列表生成伪代码

待压缩层随机向量维数列表生成	描述符
def generate_len_list(layer_indicator,input_length_list,circ_size_list): {	
len_list=[]	
for layer in layer_indicator: {}	
input_length=input_length_list[layer]	
circ_size=circ_size_list[layer]	
len_list.append(random.randint(circ_size,input_length))	
}	
return len_list	
}	

参考随机向量base_sign_vector_network生成操作的数据定义见表210。

表 210 参考随机向量生成操作数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
参考随机向量生成 generate_base_sign_vector_network	生成整个网络的参考随机向量	Input	len_list	待压缩层随机向量维数列表	list[int]	UINT
		Output	base_sign_vector_network	网络的参考随机向量	vector	UINT

参考随机向量base_sign_vector_network生成操作的伪代码见表211。

表 211 参考随机向量生成操作伪代码

参考随机向量生成	描述符
def generate_base_sign_vector_network(len_list,n=1,p=0.5): {	
len_vector=max(len_list)	
base_sign_vector_network=random_binomial(len_vector,n,p)	
base_sign_vector_network[sign_vector<0.5]=-1	
return base_sign_vector_network	
}	

7.4.3 低秩组稀疏分解的结构化矩阵压缩方法

7.4.3.1 定义

网络卷积层W为一个4维的张量，该四个维度分别为输出的通道数、输入的通道数、卷积核的高和宽。将卷积核的高和宽进行合并，从而重组为一个三维的张量W'。通过进行高维的SVD分解，将W'转化为三个小尺寸的张量：$w1$、$w2$和$w3$，对W'进行近似。其中$w1$、$w2$和$w3$的大小可以通过控制W输入通道的秩和输出通道的秩，即$w1$输出的通道数量和$w2$输出的通道数量进行压缩率的调整。

为了提高高维SVD算法的准确性，可以对近似的每个子张量进行分组迭代计算，不断的进行更新，提高对原张量的近似。

对于网络模型中的1×1卷积层和全连接层，实际上可以同样的看作权重张量和输入数据的矩阵乘积。以全连接层S为例，直接进行SVD分解，得到两个小尺寸的张量：$s1$和$s2$。通过控制$s1$的输出大小（core_size）控制压缩率。即，将输入数据和S进行计算分解为输入数据和$s1$、$s2$连续进行计算。

7.4.3.2 低秩组稀疏分解的结构化矩阵中卷积层分解压缩操作

卷积层分解压缩的流程如下：

a) 通过计算压缩率得到对应于输入通道的秩（R1）、对应输出通道的秩（R2）和分组数目（groups）（具体见 9.4.2.1）；

b) 读取当前层权值参数，并将初始化每个分组的低秩张量为 0；

c) 对于 i（i=1,...,groups）个分组，计算出除该分组以外其他分组和原始张量的残差，并对残差进行高维 SVD 分解，获得该分组所对应的低秩张量；

d) 重复步骤 c) 直至所有子张量不在发生变化，或达到最高迭代上限，得到压缩后的 groups 个子张量，并组合成 3 个压缩后的权重张量。

低秩组稀疏分解的结构化矩阵中卷积层分解压缩操作的具体数据定义见表212。

表 212 卷积层分解压缩操作数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
低秩组稀疏分解的结构化矩阵中卷积层分解压缩	将当前卷积层层权重矩阵使用低秩组稀疏分解进行压缩	Input	weight	输入的待压缩权重张量	matrix	NS
			R1	对应于输入通道的秩	value	UINT
			R2	对应于输出通道的秩	value	UINT
			groups	分组数目	value	UINT
		Output	w1	压缩后的权重张量1	matrix	NS
			w2	压缩后的权重张量2	matrix	NS
			w3	压缩后的权重张量3	matrix	NS

该操作对应伪码实现见表 213。

表 213 卷积层低秩组稀疏分解压缩操作伪代码

卷积层低秩组稀疏分解压缩	描述符
def BTD_process(weight, R1, R2, groups): {	
C_out, C_in, H, W = shape(weight)	
W = reshape(weight, C_out, C_in, -1)	
r1 = R1/groups	
r2 = R2/groups	
d_error = 1.0	
iter = 0	
last_error = 100.0	
W1 = [zeros(r1, C_in)) for _ in range(groups)]	
W2 = [zeros(r2, r1, h, w) for _ in range(groups)]	
W3 = [zeros(C_out, r2) for _ in range(groups]	
max_iter = 1000	
while (iter < max_iter and d_error > 1e-5): {	
for i in range(groups): {	
sum_ = zeros(shape(weight))	
for j in range(groups): {	
sum_ = sum_ + W2[j]×_1W1[j] ×_2W3[j]	$\times_n$:张量的 n-mode 乘.scikit-tensor==0.1
}	
res = W – sum_ + W2[i] ×_1W1[i]×_2W3[i])	$\times_n$:张量的 n-mode 乘.scikit-tensor==0.1
W1[i], W2[i], W3[i] = HOSVD(res, r1, r2)	
}	
d_error = norm(res)/norm(W)	
}	
w1 = concat(W1, axis=0)	
w2 = concat(W2, axis=1)	
w3 = concat(W3, axis=1)	
return w1, w2, w3	
}	

其中高维 SVD 分解的伪代码见表 214。

表 214 高维 SVD 分解伪代码

高维SVD分解	描述符
def HOSVD(weight, r1, r2): {	
U = []	
for order_idx in range(3): {	

表 214 高维 SVD 分解伪代码（续）

高维SVD分解	描述符
U[order_idx], _, _ = svd(unfold (weight, order_idx))	svd: numpy==1.18.1; unfold: scikit-tenor==0.1
}	
Core = weight ×_1transpose(U[0])×_2transpose(U[1])	$\times_n$:张量的n-mode乘,scikit-tensor==0.1
×_3transpose(U[2])	
return U[0], Core×_3U[2], U[1]	
}	

7.4.3.3 低秩组稀疏分解的结构化矩阵中全连接层和 1x1 卷积层压缩操作

全连接层和1x1卷积层分解压缩的流程如下：

a) 通过计算压缩率确定中间层输出的大小（core_size）（具体见 9.4.2.1）；读取当前层权值参数；

b) 对参数进行 SVD 分解，返回近似张量 w1 和 w2。

低秩组稀疏分解的结构化矩阵中全连接层和 1x1 卷积层压缩操作的具体数据定义见表 215。

表 215 低秩组稀疏分解的结构化矩阵中全连接层分解压缩数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
低秩组稀疏分解的结构化矩阵中全连接层分解压缩	将当前层权重矩阵使用低秩组稀疏分解进行压缩	Input	weight	输入的待压缩权重张量	matrix	NS
			core_size	对应于中间层的输出大小	value	UINT
		Output	w1	压缩后的权重张量1	matrix	NS
			w2	压缩后的权重张量2	matrix	NS

该操作对应伪码实现见表 216。

表 216 低秩组稀疏中全连接层分解压缩伪代码

低秩组稀疏中全连接层分解压缩	描述符
def SVD_process(weight, core_size): {	
u, sigma, v = svd(weight)	svd: numpy==1.18.1
w2 = mat_mul(u[:, :core_size], sigma[:core_size])	
w1 = v[:core_size, :]	
return w1, w2	
}	

8 解压过程（解码表示）

8.1 多模型

8.1.1 多模型解压操作

多模型解压过程，即将压缩的多模型数据解码为非压缩状态的，符合神经网络表示规范的数据格式。多模型解压操作，输入为一个共有权重张量W_share，一个可选的共有偏置向量B_share，一个特有权重张量列表W_specific，一个可选的特有偏置向量列表B_specific，一个压缩方法选项M_Method。其输出为多个拓扑结构一致的模型，包含权重张量列表Weights_list以及可选的偏置向量列表Bias_list，以及错误指示Error。其中W_share与B_share中，保存多个模型间共享信息。W_specific与B_specific保存多模型间的特有附件信息。具体定义见表217，218。多模型解压操作数据定义见表217。

表 217 多模型解压操作数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
DecompressMultimodel	对输入的多个模型进行解压操作	Input	W_share	共享权重张量	tensor	NS
			B_share	共享偏置向量	vector	NS
			W_specific	特有权重张量列表	List of tensors	NS
			B_specific	特有偏置向量列表	List of vectors	NS
			M_Method	具体解压方法选项	value	int
		Output	Weights_list	权重张量列表	List of tensors	NS
			Bias_list	偏置向量列表	List of vectors	NS
			Error	错误指示	value	int

解压Q_Method定义见表218。

表 218 解压 Q_Method 定义

Q_Method	
标号	类型
1	UNDEFINED
2	多模型层内权重共享解压
3	多模型残差量化解压

——如果 Q_Method 的标号为 2，则参考多模型层内权重共享解压（8.1.2.1）部分；
——如果 Q_Method 的标号为 3，则参考多模型残差量化解压（8.1.3）部分。

8.1.2 多模型层内权重共享压缩的解压操作

8.1.2.1 多模型内权重共享压缩的解压

多模型内权重共享压缩的解压包括三种操作：

a) 将压缩的多模型解压为多模型的非压缩数据：对于 N 个任务中的每个任务，逐个获取共享权重值以及对应的特定权重值。

b) 指定需解压的特有模型，并单独返回该模型的非压缩数据：只需要获取共享权重值以及与需解压的特有模型对应的特定权重值即可，而无需获取与其他特有模型对应的特定权重值。

c) 从一种特有模型，切换到另一种指定的特有模型：无需重新获取共享权重值，只需获取与指定任务对应的特有权重值即可。

8.1.2.2 解压输出多模型

解压输出多模型是指将采用多模型层内权重共享压缩方法生成的一个维度为 N 的共享权重张量和一个由 M 个维度为 N 的特有权重张量构成的列表，通过索引值规定的维度，解压为一个包含 M 个维度为 N 且拓扑结构一致的权重张量的列表。

其实现流程如下。

a) 如果权重聚合维度 axe 指向输出特征层数量这一维度，检查输入的特有权重张量列表 W_specific 中的张量数量与待解压权重张量 B_specific 的数量相等，如果不相等，错误指示项输出 1，其他输出项输出为 NULL，结束操作。

b) 检查输入的特有权重张量列表 W_specific 中所有张量是否拓扑结构一致以及输出的特有偏置向量列表 B_specific 中所有向量是否拓扑结构一致，如有任何不一致，错误指示项输出 2，其他输出项输出为 NULL，结束操作。

c) 获取共享权重和每一个非压缩的神经网络模型的特有权重，共享偏置和每一个非压缩的神经网络模型的特有偏置。

d) 将共享权重张量 W_share 与特有权重张量列表 W_specific 中的每一项特有权重张量，在权重聚合维度 axe 这一维度进行合并，并添加至输出权重张量列表 Weights_list。将共享偏置向量 B_share 与特有偏置向量列表 B_specific 中的每一项特有偏置向量，在权重聚合维度 axe 这一维度进行合并，并添加至输出偏置向量列表 Bias_list。将权重张量 W_share 在权重聚合维度 axe 这一维度的数量作为聚合点索引 slice_indicator 输出。

解压输出多模型操作的数据定义见表219。

表 219 解压输出多模型数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
解压输出多模型	将通过多模型层内权重共享压缩技术压缩的数据，通过索引值规定的维度、模型数量，解压为 M 个维度为 N 且拓扑结构一致的权重张量	Input	W_share	输入的共享权重张量	tensor	NS
			B_share	输入的共享偏置向量	vector	NS
			W_specific	输入的特有权重张量列表	List of tensors	NS
			B_specific	输入的特有偏置向量列表	List of vectors	NS
		Attributes	axe	权重聚合维度	value	int
		Output	Weights_list	输出权重张量列表	List of tensors	NS
			Bias_list	输出偏置向量列表	List of vectors	NS
			slice_indicator	聚合点索引	value	int

解压输出多模型操作的伪代码见表220。

表 220 解压输出多模型伪代码

解压输出多模型	描述符
Decompress_Multilayer_weight(W_share, B_share, W_specific, B_specific) {	
Weights_list = []	
Bias_list = []	
slice_indicator = 0	
Error = 0	
if (axe = dimension_of_out_feature_map){	
if(lens(W_specific)!= lens(B_specific)), Error=1	
}	
if(Error ==0) {	
for (i=0; i< lens(W_specific)-1; i++){	
if (W_specific [i].shape != W_specific [i+1].shape), Error=2	
}	
}	
if(Error ==0) {	

表 220 解压输出多模型伪代码(续)

解压输出多模型	描述符
for (i=0; i< lens(W_specific); i++) {	
Weights_list.append(Concate(W_share, W_specific[i], axe))	
Bias_list.append(Concate(B_share, B_specific[i], axe))	
}	
slice_indicator = W_share.shape(axe)	
}	
return Weights_list, Bias_list, slice_indicator, Error	
}	

8.1.2.3 解压输出指定模型

解压输出指定模型是指将采用多模型层内权重共享压缩方法生成的一个维度为N的共享权重张量和一个由M个维度为N的特有权重张量构成的列表，根据指定的模型序号，输出该模型编号对应的未压缩的神经网络模型。

其实现流程如下。

a) 通过任务索引，确定模型序号 Unit_indicator。

b) 获取共享权重和对应特有权重值：从输入的特有权重张量列表 W_specific 中取出对应的特有模型权重张量 W，并从输入的特有偏置向量列表 B_specific 中取出对应的特有模型偏置向量 B。

c) 将输入的共享权重张量 W_share 与 W 在权重聚合维度 axe 这一维度进行合并，并输出至 Weights。

d) 将输入的共享偏置向量 B_share 与 B 在权重聚合维度 axe 这一维度进行合并，并输出至 Bias。

解压输出指定模型操作的数据定义见表221。

表 221 解压输出指定模型数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
解压输出指定模型	将通过多模型层内权重共享压缩技术压缩的数据，通过索引值规定的维度、待提取模型编号，解压指定模型	Input	W_share	输入的共享权重张量	tensor	NS
			B_share	输入的共享偏置向量	vector	NS
			W_specific	输入的特有权重张量列表	List of tensors	NS
			B_specific	输入的特有偏置向量列表	List of vectors	NS
		Attributes	axe	权重聚合维度	value	int
			Unit_indicator	模型序号	value	int
		Output	Weights	输出权重张量	tensors	NS
			Bias	输出偏置向量	vectors	NS

解压输出指定模型操作的伪代码见表222。

表 222 解压输出指定模型伪代码

解压输出指定模型	描述符
Get_indicated_layer_weight (W_share, B_share, W_specific, B_specific, Unit_indicator, axe) {	
Weights = Concate(W_share, W_specific[Unit_indicator], axe)	
Bias = Concate(B_share, B_specific[Unit_indicator], axe)	
return Weights, Bias	
}	

8.1.2.4 切换输出指定模型

切换输出指定模型指从采用多模型层内权重共享压缩方法生成的神经网络模型中解压输出的指定模型一切换输出解压的指定模型二。采用多模型层内权重共享压缩方法生成的神经网络模型包括 M（为正整数）个网络层，该M个网络层中的第i（1≤i≤M，i为整数）个网络层具有共享权重值和N组特有权重值。共享权重值用于执行N个任务中的每个任务，即在第i个网络层中执行N个任务的任一任务时均使用该共享权重值。N组特有权重值中的每组特有权重值用于执行N个任务中的一个任务，且每组特有权重值与N个任务中的一个任务一一对应。

由于共享权重值在 N 个未压缩的神经网络模型中共有，因此，在切换输出指定模型的场景中，第 i 个网络层无需重新获取共享权重值，只需获取与指定输出模型对应的特有权重值并替换原来的特有权重值为该特有权重值即可。

其实现流程如下。

a) 获取对应序号模型的特有权重：从输入的特有权重张量列表 W_specific 中取出对应的特有模型权重张量 W_specific_idx，并从输入的特有偏置向量列表 B_specific 中取出对应的特有模型偏置向量 B_specific_idx。

b) 替换对应序号模型的特有权重：将输入权重张量Weights的特有权重部分替换为W_specific_idx，将输入偏置张量 Bias 的特有偏置部分替换为 B_specific_idx.

获取对应序号模型的特有权重的数据定义见表223。

表 223 提取对应序号模型特有权重数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
提取对应序号模型的特有权重	将通过多模型层内权重共享压缩技术压缩的数据，提取指定模型的特有权重和特有偏置	Input	W_specific	输入的特有权重张量列表	List of tensors	NS
			B_specific	输入的特有偏置向量列表	List of vectors	NS
		Attributes	Unit_indicator	模型序号	value	int
		Output	W_specific_idx	输出特有权重张量	tensors	NS
			B_specific_idx	输出特有偏置向量	vectors	NS

伪代码见表224。

表 224 提取对应序号模型特有权重伪代码

提取对应序号模型的特有权重	描述符
Get_indicated_layer_weight_specific_idx (W_specific, B_specific, Unit_indicator) {	
W_specific_idx = W_specific[Unit_indicator]	
B_specific_idx = B_specific[Unit_indicator]	
return W_specific_idx, B_specific_idx	
}	

替换对应序号模型的特有权重的数据定义见表225。

表 225 替换对应序号模型特有权重数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
替换对应序号模型的特有权重	用提取到的对应序号的模型的特有权重替换原始非压缩的模型的特有权重部分	Input	Weights	输入权重张量	tensors	NS
			Bias	输入偏置向量	vectors	NS
			W_specific_idx	输入特有权重张量	tensors	NS
			B_specific_idx	输入特有偏置向量	vectors	NS
		Output	Weights	输出权重张量	tensors	NS
			Bias	输出偏置向量	vectors	NS

伪代码见表226。

表 226 替换对应序号模型特有权重伪代码

替换对应序号模型的特有权重	描述符
Replace_indicated_layer_weight_specific_idx (Weights, Bias, W_specific_idx, B_specific_idx) {	
Weights[len(W_specific_idx):-1]=W_specific_idx	
Bias[len(B_specific_idx):-1]=B_specific_idx	
return Weights, Bias	
}	

8.1.3 多模型残差量化的解压操作

8.1.3.1 解压操作定义

多模型残差量化的解压操作为以下部分：首先，取得预训练模型的权重，然后取得特定位置增量模型的差分权重，将两者相加赋予目标网络模型。

8.1.3.2 解压输出目标模型

解压输出目标模型是指基于已经存在的预训练模型权重，增量的获得目标模型权重。流程如下：

首先，解压已有的预训练模型权重，然后，解压对应的目标模型差分权重，将两者结合得到最终的目标模型权重。

解压输出目标模型的数据定义见表227。

表 227 解压输出目标模型数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
解压输出指定模型	通过目标模型权重与差分模型权重结合，恢复目标模型权重	Input	W_share	输入的共享权重张量	tensor	NS
			W_specific	权重聚类结果	List	NS
			Base_model	预训练模型权重	Tensor	NS
			Weights_list	权重张量列表	List of tensors	NS
		Output	Weights_target	目标模型权重张量	Tensor	NS
			Bias	输出偏置向量	vectors	NS

伪代码见表 228。

表 228 解压输出目标模型（基于预训练模型）伪代码

解压输出目标模型（基于预训练模型）	描述符
Decompress_Multilayer_weight(W_share, B_share, W_specific, B_specific) {	
Weights_list = [] If (len(W_specific)==len(Weights_list))	
Error=0	
else	
Error=1	
if(Error ==0) {	
for (i=0; i< lens(W_specific)-1; i++){	
Weight_target[i]=Base_model[i]+W_share[W_specific[i]]	
}	
}	
return Weights_target, Error	
}	

8.2 反量化

8.2.1 定义

反量化指的是将量化后为整数类型的压缩数据，恢复到原始精度数据类型的过程。反量化主要为了方便运算，解决不同设备和运行阶段的兼容问题。大部分的反量化操作来源于同名的量化操作，相关内容可以参照上一章。本章首先介绍两个基础反量化操作，之后分别对参数与激活两类数据，提出了不同的反量化方案。

8.2.2 基础反量化操作

反量化操作是对给定的离散类型输入张量，利用不同方法输出高精度的反量化张量，作为父类的操作方法的定义与描述见表229，230。其中，基础反量化操作定义见表229。

表 229 基础反量化操作定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
Dequantization_tensor 基础反量化操作	对输入的张量进行反量化操作	Input	X	输入张量	List [tensor]	NS
			granularity	反量化粒度	Value	int
			D_Method	反量化方法	Value	int
			kwarg	方法参数	Dict	Dict
		Output	Y	输出张量	List [tensor]	NS

其中的量化粒度granularity为反量化方法在输入向量中不同层次结构反量化范围的规定，分别包括统一反量化、层级反量化、通道反量化、其他粒度反量化，见表230。

表 230 granularity 反量化粒度定义表

标号	定义
1	统一反量化
2	层级反量化
3	通道反量化
4	其他粒度反量化

这些标号中：

——如果 granularity 的标号为 1，则对所有输入张量（假设有 n 个）进行统一反量化；

——如果 granularity 的标号为 2，则对所有输入张量（假设有 n 个）按照层粒度进行反量化；

——如果 granularity 的标号为 3，则对所有输入张量（假设有 n 个）按照通道粒度进行反量化；

——如果 granularity 的标号为 4，则按照用户给定的粒度进行反量化。

反量化方法D_Method表示了具体采用的反量化方式，对应的标号含义见表231。

表 231 D_Method 反量化方法定义表

标号	作用对象	类型
1	参数、激活	线性反量化
2	参数	查找表反量化
3		非线性函数映射反量化
4		定点反量化
5		INT4反量化
6	激活	训练截断值反量化
7		定点反量化
8		INT4反量化

8.2.2.1 线性反量化

线性反量化根据输入的比特位，然后通过对量化之后的数据，利用尺度变换因子和偏置因子，将原本8比特的低比特还原为全精度张量。

线性反量化的具体数据定义见表232。

表 232 线性反量化数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
Linear_dequantization 线性反量化	将离散表示的高精度网络参数按层映射到低精度的范围	Input	X	粒度索引值对应权重张量	List [tensor]	NS
			X_scale	权重量化尺度因子	List [tensor]	NS
			X_offset	权重量化偏置因子	List [tensor]	NS
			Granularity	量化粒度	Value	int
			N	量化比特数	Value	int
		Output	Y	线性映射后的权重张量	List [tensor]	NS

线性反量化伪代码见表233。

表 233 线性反量化伪代码

线性反量化	描述符
def Linear_dequantization (X, X_scale, X_offset, granularity,N):{	
if(granularity==1): {	
Y = (X – X_offset)/X_scale	
}	
elif (granularity==2): {	
for (x in X): {	
y = (x – x_offset)/x_scale	
}	
}	
else: {	
for(x in X): {	
for(i in [0: x.channels]): {	
y[i] = (x[i] – x_offset[i])/x_scale[i]	
}	
}	
}	
return Y	
}	

8.2.2.2 查找表反量化

查找表的反量化操作是指依据给定的查找表索引与查找表值，恢复出原始离散高精度参数的过程。本方法主要适用于区间收缩反量化，相关的数据生成方法参见第9章内容。查找表反量化将输入的索引张量矩阵，按照其中元素的索引值，找到所对应高精度参数值，并赋值到输出张量的相同位置。

相应的操作符定义见表234。

表 234 查找表反量化定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
Codebook_ dequantization 查找表反量化	将离散表示的网络参数还原为原始精度。	Input	X_i	使用索引值表示的张量	List [tensor]	NS
			X_center	权重查找表值	List [tensor]	NS
			granularity	量化粒度	Value	int
		Output	Y	层索引值对应张量	List [tensor]	NS

查找表反量化的伪代码见表235。

表 235 查找表反量化伪代码

查找表反量化	描述符
def Codebook_dequantization(X_i, X_center, granularity):{	
Y = []	
if (granularity == 1) or (granularity == 2):{	
for i in range(0, len(X_center)):{	
Y[i] = X_center[X_i[i]]	
}	
}	
if (granularity == 3):{	
for i in range(0, len(X_center)):{	
for j in range(0, len(W[i])):{	
Y[i][j] = X_center[i][j][X_i[i][j]]	
}	
}	
}	
return Y	
}	

8.2.3 参数反量化操作

8.2.3.1 非线性函数映射还原

该部分与之前第七章节中的线性参数量化部分对应，当读取模型进行计算的时候，我们通过公式(29)，恢复为全精度权重进行计算，其中S位尺度因子，Z为偏置因子，Q为量化之后的权重值，f为对应的非线性函数：

$$W' = f^{-1}[S(Q - Z)] \quad (29)$$

式中：

Z——为偏置因子

S——为尺度因子

Q——为量化之后的权重值

f——为对应的非线性函数

W'——为还原之后的权重值

其中，f根据编号为不同函数的反函数，具体操作见表236。

表 236 非线性反量化数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
Weight non Linear_dequantization 参数非线性量化	将离散表示的低精度网络参数进行还原	Input	W_q	量化之后的权重张量	list[tensor]	NS
			W_scale	权重量化尺度因子	list[tensor]	NS
			W_offset	权重量化偏置因子	list[tensor]	NS
			Granularity	量化粒度	value	int
			Function	非线性函数	value	Int
			N	量化比特数	value	int
		Output	W	还原之后的权重张量	list[tensor]	NS

其中部分可选的非线性函数（Function）定义见表237。

表 237 可选非线性量化函数

标号	类型
1	线性函数
2	指数函数
3	Tanh 函数
4	其他函数

整个非线性量化算法的伪代码见表238。

表 238 非线性反量化伪代码

非线性反量化	描述符
def Weight_non_Linear_dequantization (W_q, W_scale, W_offset, granularity,function,N){	
if(function==1): {	
W_q=W_q	
}	
elif (function ==2): {	
W_q = e^(W_q)	
}	
elif(function ==3): {	
W_q = atanh(W_q)	
W = Linear_dequantization (W, W_scale, W_offset, granularity,N)	
}	
return W	
}	

8.2.3.2 定点反量化

如 7.2.3.2 节所述，定点量化将一组浮点数X_f表示成 8bit 的整数X_q以及一个定点位置 p（对应缩放系数scale $= 2^{-p}$），反量化则是利用X_q和 p，反量化出浮点数X_d，也就是：

$$X_d = X_q * scale = 2^{-p} * X_q \quad \text{.....(30)}$$

反量化后X_d的范围 $[-128 * 2^{-p}, 127 * 2^{-p}]$，参数定点反量化的数据定义见表 239。

表 239 查找表反量化伪代码

运算操作	描述	字段	关键字	定义	数据类型	数据格式
fix_dequantize_parameter 参数定点反量化	将低精度表示的神经网络参数反量化为高精度（如FP32）格式的张量	Input	W_quantized	量化后的权重张量	tensor	int
			B_quantized	量化后的偏置张量	tensor	int
			W_p	权重张量的量化位置	value	int
			B_p	偏置张量的量化位置	value	int
		Output	W_dequantized	反量化后的权重张量	tensor	float
			B_dequantized	反量化后的偏置张量	tensor	float

参数定点反量化的伪代码见表 240。

表 240 定点反量化伪代码

定点反量化	描述符
def dequantize(X_q, p): {}	
return X_q*pow(2,-p)	
}	
def fix_dequantize_parameter(W_quantized, B_quantized, W_p, B_p): {	
return (dequantize(W_quantized,W_p), dequantize(B_quantized,B_p)	
}	

8.2.3.3 INT4 参数反量化

INT4反量化操作，将一个INT4张量或多个INT4张量组合反量化为高精度（FP32）的张量。一般情况下，量化为多个INT4张量时，采用Dual INT4量化，因此多INT4反量化时也一般为Dual INT4反量化。与INT4参数反量化对应的激活反量化方式为INT4激活反量化。参数反量化涉及神经网络权重反量化和神经网络偏置反量化，INT4参数反量化中，对于神经网络偏置不作处理，对于神经网络权重，则逐层进行反量化，流程如下。

a) 读取当前层输入的量化后的 INT4 权重张量 W_quantized 及量化尺度因子列表 sf_w_list 和权重

量化偏置列表 offset_w_list。

b) 根据量化权重张量 W_quantized 确定当前网络层的类型，并采用不同粒度的反量化方

1) 若网络层为卷积层，则对每个输出 channel 对应的 3D 卷积核张量 W_quantized[k]进行反量化，计算得到卷积核重建张量 W_recon[k]。

2) 若网络层为全连接层，则对量化权重张量 W_quantized 进行层级反量化，计算得到权重重建张量 W_recon。

INT4参数反量化的数据定义见表241。

表 241 INT4 参数反量化数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
int4_recon_quantized_parameter INT4 参数反量化	将 INT4 格式的权重张量反量化为高精度格式的张量	Input	W_quantized	量化后 INT4 权重张量	list[tensor]	INT4
			B_quantized (optional)	量化后的网络层偏置张量	list[tensor]	NS
			sf_w_list	量化尺度因子列表	list	float32
			offset_w_list	权重量化偏置列表	list	float32
		Output	W_recon	反量化后的重建权重张量	list[tensor]	float32
			B_recon (optional)	反量化重建后的网络层偏置张量	list[tensor]	NS

INT4权重反量化的伪代码见表242。

表 242 INT4 参数反量化伪代码

INT4参数反量化	描述符
def int4_recon_quantized_parameter(W_quantized,sf_w_list,offset_w_list, B_quantized): {	
if len(W_quantized.shape)>2: {	
for k in range(len(W_quantized)): {	
if len(sf_w_list[k])==1:	
W_recon[k]=W_quantized[k]*sf_w_list[k][0]+offset_w_list[k]	
elif len(sf_w_list[k])==2:	
W_recon[k]=W_quantized[k][0]*sf_w_list[k][0]+/	
W_quantized[k][1]*sf_w_list[k][1]+offset_w_list[k]	
}	
}	
else: {	

表 242 INT4 参数反量化伪代码（续）

INT4参数反量化	描述符
if len(sf_w_list)==1:	
W_recon=W_quantized*sf_w_list[0]+offset_w_list}	
elif len(sf_w_list)==2:	
W_recon=W_quantized[0]*sf_w_list[0]+W_quantized[1]*sf_w_list[1]+offset_w_list	
}	
B_recon=B_quantized	
return W_recon, B_recon	
}	

8.2.4 激活反量化操作

8.2.4.1 训练截断值

在反量化操作中，是将量化值 X 进行线性映射，将原本在低比特空间中的值，映射到新的浮点空间中。与先前的量化部分对应，当读取模型进行计算的时候，通过量化反函数$X = X_Scale(X_q - X_offset)$，恢复为全精度权重进行计算。训练截断值反量化操作的具体数据定义见表 243。

表 243 训练截断值数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
训练截断值	将量化的截断值按层进行训练	Input	X_q	量化后激活值张量	List [tensor]	NS
			Alpha	可训练的截断上界值	List [tensor]	NS
			Granularity	量化粒度	Value	int
			N	量化比特数	Value	int
		Output	X	反量化后的激活值输入输出	List [tensor]	NS

训练截断值反量化的伪代码见表244。

表 244 训练截断值反量化伪代码

训练截断值反量化	描述符
def Trainable_Alpha_DeQuantization(X_q, Alpha, Granularity ,N):{	
X_Scale = (2**(n-1)-1)/Alpha	
X_offset = 0	
X= Linear_dequantization (X_q, X_scale, X_offset, Granularity,N)	
return X	
}	

8.2.4.2 定点反量化

激活值定点反量化的数据定义见表 245。

表 245 激活值定点反量化数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
fix_dequantize_activation 激活值定点反量化	将低精度表示的神经网络激活值反量化为高精度（如FP32）格式的张量	Input	X_quantized	量化后的激活张量	tensor	float32
			X_p	激活张量的量化位置	value	int
		Output	X_dequantized	反量化后的激活张量	tensor	float

激活值定点反量化的伪代码见表 246。

表 246 激活值定点反量化伪代码

激活值定点反量化	描述符
def dequantize(X_q, p):	
{	
return X_q*pow(2,-p)	
}	
def fix_dequantize_activation(X_quantized, X_p):	
{	
return dequantize(X_quantized,X_p)	
}	

8.2.4.3 INT4 激活反量化

INT4反量化操作，将一个INT4张量或多个INT4张量组合反量化为高精度（FP32）的张量。一般情况下，量化为多个INT4张量时，采用Dual INT4量化，因此多INT4反量化时也一般为Dual INT4反量化。与INT4激活反量化对应的参数量化方式为INT4参数反量化。逐层对神经网络的激活进行反量化，流程如下。

a) 读取当前层输入的量化后的激活张量 X_quantized 及量化尺度因子列表 sf_act_list 和量化偏置列表 offset_act_list。

b) 对输入层的量化激活张量进行层级线性反量化。

INT4激活反量化的数据定义见表247。

表 247 INT4 激活反量化数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
int4_recon_quantized_activation INT4 激活反量化	将 INT4 格式的激活张量反量化为高精度格式的张量	Input	X_quantized	量化后的 INT4 激活张量	list[tensor]	INT4
			sf_act_list	量化尺度因子列表	list	float32
			offset_act_list	激活量化偏置列表	list	float32
		Output	X_recon	反量化后的重建权重张量	tensor	float32

INT4激活反量化的伪代码见表248。

表 248 INT4 激活反量化伪代码

INT4激活反量化	描述符
def int4_recon_quantized_activation(X_quantized,sf_act_list,offset_act_list): {	
X_recon=zeros(X_quantized[0].shape)	
for(i=0;i<len(sf_act_list);i++):	
X_recon+=sf_act_list[i]*X_quantized[i]+offset_act_list[i]	
return X_recon	
}	

8.3 反稀疏化/反剪枝操作

8.3.1 定义

神经网络反稀疏化操作是指将神经网络的权重，由剪枝之后的网络权重重新还原回原来的尺寸，对于空缺的网络权重部分进行补零操作。剪枝操作定义见表249。

表 249 剪枝操作定义

操作	描述	字段	关键字	定义	数据类型	数据格式
反稀疏化	对网络权重进行结构化还原	Input	W	稀疏权重张量	list[tensor]	NS
			B (optional)	稀疏偏置张量	list[tensor]	NS
			S	掩码	list[vector<int>]	NS
			granularity	粒度	Value	Int
		Output	W_u	剪枝之后的权重张量	list[tensor]	NS

剪枝粒度定义见表250。

表 250 剪枝粒度定义

剪枝粒度	
标号	类型
1	UNDEFINED
2	行剪枝/卷积核个数剪枝
3	列剪枝/卷积核形状剪枝

8.3.2 反稀疏化

反稀疏过程如下：

输入参数：待还原稀疏张量列表W;待还原稀疏偏置张量列表B;掩码向量列表S;N所在的维度N_dim（考虑张量存储格式不一定是NCHW，所以需要用户指定需要裁剪的维度）。

输出参数：还原后的张量列表W_o。

步骤：

用以下步骤遍历张量列表中的所有张量w：

a) 获取 w 的形状 w_tensor_shape；

b) 将w进行 reshape(w,w_tensor_shape[dim],-1,1)；

c) 初始化一个新的全部为零，对应维度与掩码长度一致的权重张量w_u；

d) 根据输入掩码，在掩码vector中，将对应为1的行/列赋值为w对应的行，其余值补零;

操作定义见表251。

表 251 反稀疏化定义

操作	描述	字段	关键字	定义	数据类型	数据格式
反稀疏化	对网络权重进行结构化还原	Input	W	稀疏权重张量	list[tensor]	NS
			B (optional)	稀疏偏置张量	list[tensor]	NS
			S	掩码	ist[vector<int>]	NS
			N_dim	N所在的维度	value	int
			granularity	粒度	value	int
		Output	W_u	反稀疏化之后的权重张量	list[tensor]	NS
			B_o	反稀疏化之后的偏置张量	list[tensor]	NS

伪代码见表252。

表 252 结构化稀疏数据生成

结构化稀疏数据生成	描述符
Un_structured_sparse(W,B,S,N_dim, Skip_Column, granularity){	
for(i=0;i<length(W),i++){	

表 252 结构化稀疏数据生成（续）

结构化稀疏数据生成	描述符
w = W[i];	
b = B[i];	
w_shape = w.tensor_shape;	
w.reshape(w_shape[N_dim],-1);	
If(granularity==2){	
New_tensor_shape_w = w_shape;	
New_tensor_shape_w[dim] = length(S[i]);	
New_tensor_shape_b = b_shape;	
New_tensor_shape_b[dim] = length(S[i]);	
W_o[i] = Tensor(New_tensor_shape_w);	
B_o[i] = Tensor(New_tensor_shape_b)	
k=0;	
for(j=0;j<length(S[i]);j++){	
If(S[i][j]==1){	
W_o[i][j] = w[k]; B_o[i][j]=b[k]k++;	
}	
}	
}	
If(granularity==3){	
skip_col = Skip_Column[i]	
org_weights = matrix (shape= (w.row, w.col + len(skip_col))	
org_bias = vector (shape= (len(b))	
new_col_idx = 1	
for col_idx = 1 to org_weights.col{	
if (col_idx in skip_col)	
org_weights[:,new_col_idx] = 0	
else{	
org_weights[:, col_idx] = w[:, new_col_idx]	
new_col_idx = new_col_idx + 1	
}	
}	
org_bias[:] = b[:]	
W_o[i] = org_weights	
B_o[i] = org_bias	
}	
}	
}	

8.4 结构化矩阵

8.4.1 结构化矩阵解压

结构化矩阵技术通过设计按照某种规律排列的矩阵，从而使用较小的参数表示或回复原始矩阵，结构化矩阵解压缩操作是指将神经网络中压缩为某种格式的结构化矩阵，解压为权重矩阵的操作。具体定义见表253，254，255。结构化矩阵解压操作数据定义见表253。

表 253 结构化矩阵解压操作数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
DecompressStructuralMatric	将压缩为给定格式的结构化矩阵解压为权重矩阵的操作	Input	X	输入张量	Tensor	NS
			granularity	结构化矩阵解压粒度	Value	UINT
			D_Method	结构化矩阵解压方法	Value	UINT
		Output	Y	输出张量	Tensor	NS

结构化矩阵解压操作粒度定义见表254。

表 254 结构化矩阵解压操作粒度定义

granularity	
标号	类型
1	UNDEFINED
2	所有输入张量统一进行结构化矩阵解压
3	层级粒度结构化矩阵解压
4	通道级粒度结构化矩阵解压
5	自定义结构化矩阵解压粒度

结构化矩阵D_Method定义见表255。

表 255 结构化矩阵 D_Method 定义

D_Method	
标号	类型
1	UNDEFINED
2	分块循环矩阵解压
3	低秩组稀疏分解的结构化矩阵解压

8.4.2 带符号向量的分块循环矩阵的解压缩方法

8.4.2.1 分块循环矩阵解压操作

对于使用符号向量和分块循环矩阵技术的层，输入的权重参数为分块结构化的矩阵对应的压缩矩阵。在神经网络的训练与推理中，计算该网络层的输出之前，需要将输入的权重参数解压为压缩矩阵对应的分块结构化的矩阵。与7.4.1.1的压缩过程相对应的，分块循环矩阵解压的流程如下：

a) 从压缩后的分块循环矩阵中读取子分块循环矩阵首行元素向量；
b) 根据首行元素向量，循环移位展开得到压缩前的对应子分块循环矩阵，存储至解压后的权重张量矩阵对应位置；
c) 循环上述操作至解压所有子分块循环矩阵，返回解压后的完整权重张量矩阵。

分块循环矩阵解压操作的具体数据定义见表256。

表 256 分块循环矩阵解压操作数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
分块循环矩阵解压 decompress_w_circ_to_weight	将使用分块循环矩阵压缩的当前层权重矩阵进行解压	Input	w_circ	已压缩的权重张量	matrix	NS
			circ_size	分块循环矩阵尺寸	value	UINT
		Output	weight	解压后的权重张量	matrix	NS

分块循环矩阵解压操作的伪代码实现见表257。

表 257 分块循环矩阵解压操作伪代码

分块循环矩阵解压	描述符
def decompress_w_circ_to_weight(w_circ, circ_size): {	
height_circ, width_circ = w_circ.shape()	
height_ori = height_circ*circ_size	
for i in range(height_circ): {	
for j in range(0, width_circ, circ_size): {	
vector_circ = w_circ[i][j:j+circ_size-1]	
for k in range(circ_size): {	
weight[i*circ_size+k][j:j+circ_size-1] = circ_shift(vector_circ, k)	
}	
}	
}	
return weight	
}	

循环移位操作(右移)circ_shift的具体数据定义见表258。

表 258 循环移位向量生成操作数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
循环移位向量生成	将输入向量进行循环移位	vector	输入的向量	vector	UINT	vector
		shift_	指定的移动位数	value	UINT	shift_
		shift_vector	右移shift位后的向量	vector	UINT	shift_vector

循环移位操作(右移)circ_shift的伪代码实现见表259。

表 259 循环移位向量生成伪代码

循环移位向量生成(右移)	描述符
def circ_shift(vector, shift): {	
shift_vector = zeros(len(vector))	
for i in range(len(vector)): {	
if (i+shift)>len(vector)-1: {	
shift_vector[mod(i+shift, len(vector))] = vector[i]	
}	
else: {	
shift_vector[i+shift] = vector[i]	
}	
}	
return shift_vector	
}	

8.4.2.2 扰动向量的生成操作

对于使用符号向量和分块循环矩阵技术的层，在训练与推理中，计算该网络层的输出之前，需要将输入向量转换为校正输入向量。校正输入向量中的每个元素为原始输入向量中的元素和扰动向量对应位置的元素的乘积。扰动向量由对应层的随机向量扩展而来，其实现流程如下：

a) 由神经网络的参考随机向量 base_sign_vector_network 及压缩层随机向量维数列表 len_list，生成各层随机向量：从参考随机向量中截取长度为 len_list[i]的向量作为第i个待压缩层的随机向量；

b) 生成扰动向量：基于随机向量，采用循环移位或者向量元素复制的方式生成与压缩层的输入向量的维数相等的向量作为该层的扰动向量。

扰动向量生成操作的数据定义见表260。

表 260 扰动向量生成操作数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
扰动向量生成 generate_sign_vector_list	生成压缩层对应的扰动向量	Input	len_list	输入的压缩层随机向量长度列表	list[int]	UINT
			base_sign_vector_network	神经网络的参考随机向量	vector	UINT
			extend_list	扰动向量生成方式	list[int]	UINT
			len_target	压缩层扰动向量维数列表	list[int]	UINT
		Output	sign_vector_list	压缩层扰动符号向量	list[vector]	UINT

扰动符号向量生成操作对应的伪代码实现见表261。

表 261 符号向量生成伪代码

符号向量生成	描述符
def generate_sign_vector_list(len_list, base_sign_vector_network, extend_list, len_target): {	
sign_vector_list = []	
for i in range(len(len_list)): {	
base_sign_vector = base_sign_vector_network[:len_list[i]]	
sign_vector_list.append(extend_base_to_sign(base_sign_vector, len_target[i], extend_list[i]))	
}	
return sign_vector_list	
}	

其中，从随机向量生成扰动向量的操作extend_base_to_sign的具体数据定义见表267，268。其中，扩展符号向量操作数据定义见表262。

表 262 扩展符号向量操作数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
扰动符号向量的生成 extend_base_to_sign	从基础随机符号向量生成扰动符号向量	Input	base_sign_vector	输入的基础随机符号向量	vector	Int
			len_extend	扰动符号向量长度	value	Int
			method	扰动符号向量生成方式	value	Int
		Output	extend_sign_vector	待压缩层扰动符号向量	vector	Int

振动符号向量生成 method 定义见表 263。

表 263 扰动符号向量生成方式 method 定义

method	
标号	类型
1	循环移位生成
2	复制生成

从随机向量生成扰动向量的操作extend_base_to_sign的伪代码见表264。

表 264 扩展符号向量伪代码

扩展符号向量	描述符
def extend_base_to_sign(bae_sign_vector, len_extend, method): {	
extend_sign_vector = zeros(len_extend)	
if method == 1: {	
for i in range(0, len_extend, len(base_sign_vector): {	
extend_sign_vector[i:i+len(base_sign_vector)-1] = circ_shift(\	
base_sign_vector, i/len(base_sign_vector))	
}	
}	
elif method ==2: {	
for i in range(0, len_extend, len(base_sign_vector): {	
extend_sign_vector[i:i+len(base_sign_vector)-1] = base_sign_vector	
}	
}	
else: {	
return error	
}	
return extend_sign_vector	
}	

8.4.2.3 使用符号向量和分块循环矩阵技术的网络层的计算操作

对于使用符号向量和分块循环矩阵技术的层，在训练与推理中，计算该网络层的输出时，输入向量为校正输入向量，校正输入向量中的每个元素为原始输入向量中的元素和扰动向量对应位置的元素的乘积；输入权重为解压后的分块循环化的矩阵。在计算时，输入为扰动向量extend_sign_vector，原始输入向量layer_in，解压后的权重矩阵weight，偏置B(optional)，输出为layer_out。 其实现流程如下：

a) 将原始输入向量中的元素和扰动向量对应位置的元素进行乘积，得到校正输入向量；

b) 将校正输入向量、解压后的权重矩阵、偏置输入至该网络层，计算相应的输出。

网络层计算操作的数据定义见表265。

表 265 网络层计算操作数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
网络层输出向量生成 cal_sign_circ	生成使用符号向量和分块循环矩阵技术的网络层的输出向量	Input	layer_in	网络层原始输入向量	vector	NS
			sign_vector	网络层的扰动向量	vector	UINT
			weight	解压后的权重张量	matrix	UINT
			B(optional)	偏置向量	vector	NS
		Output	layer_out	网络层输出向量	vector	NS

网络层计算操作伪代码见表 266。

表 266 网络层计算操作伪代码

网络层计算	描述符
def cal_sign_circ(layer_in, weight, sign_vector, B): {	
sign_layer_in = mul(layer_in, sign_vector)	
layer_out = mul(sign_layer_in, weight)	
if B: {	
layer_out = layer_out+ B	
}	
return layer_out	
}	

8.4.3 低秩组稀疏分解的结构化矩阵解压方法

8.4.3.1 概述

对于网络中的3 × 3卷积所对应张量的压缩，其为高维SVD得到的分解结果。因此对张量进行解压，只需要对子张量进行张量的n-mode乘积即可对原矩阵进行近似还原，得到解压的结果。

对于网络中的1 × 1卷积所和全连接层的参数张量的压缩，其为SVD分解得到的结果，对这些张量进行解压，只需要对两个张量进行矩阵乘法即可。

8.4.3.2 低秩组稀疏分解的结构化矩阵中卷积层分解解压操作

卷积层解压缩的流程如下：

a) 将每个子张量还原为原来的子张量分组；

b) 进行分别对每组的子张量进行 n-mode 乘法，计算出每个分组的对原张量的近似值，并相加，得到解压的结果。

低秩组稀疏分解的结构化矩阵中卷积层解压操作的具体数据定义见表 267。

表 267 低秩组稀疏分解的结构化矩阵中卷积层解压操作数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
低秩组稀疏分解的结构化矩阵中卷积层解压	对低秩组稀疏所得到的压缩矩阵进行解压	Input	w1	压缩后的权重张量1	matrix	NS
			w2	压缩后的权重张量2	matrix	NS
			w3	压缩后的权重张量3	matrix	NS
			Weight_shape	原张量的维度信息	vector	UINT
			R1	对应于输入channel的秩	value	UINT
			R2	对应于输出channel的秩	value	UINT
			groups	分组数目	value	UINT
		Output	approx	解压后的权重张量	matrix	NS

该操作对应伪码实现见表 268。

表 268 低秩组稀疏中卷积层解压缩伪代码

低秩组稀疏中卷积层解压缩	描述符
def BTD_decompression(w1, w2, w3, Weight_shape, R1, R2, groups): {	
r1 = R1/groups	
r2 = R2/groups	
w1 = [w1[:,idx*r1:idx*r1+r1] for idx in range(groups)]	
w2 = [w2[idx*r1:idx*r1+r1,:] for idx in range(groups)]	
w3 = [w3[:,idx*r2:idx*r2+r2] for idx in range(groups)]	
approx = zeros(Weight_shape)	
for i in range(groups): {	
approx += w2[i]×_1w1[i]×_2w3[i]	$\times_n$:张量的 n-mode 乘.scikit-tensor==0.1
}	
return approx	
}	

8.4.3.3 秩组稀疏分解的结构化矩阵中全连接层和 1x1 卷积层解压操作

全连接层和1x1卷积层解压的流程为，对两个子张量进行矩阵乘法，得到原张量的近似张量。秩组稀疏分解的结构化矩阵中全连接层和1x1卷积层解压操作的具体数据定义见表269。

表 269 秩组稀疏分解的结构化矩阵中全连接层和 1x1 卷积层解压操作数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
秩组稀疏分解的结构化矩阵中全连接层和1x1卷积层解压	对低秩组稀释所得到的压缩矩阵进行解压	Input	w1	输入的待压缩权重张量	matrix	NS
			w2	对应于中间层的输出大小	matrix	NS
		Output	approx	解压后的权重张量	matrix	NS

该操作对应伪码实现见表 270。

表 270 低秩组稀疏中全连接层解压缩伪代码

低秩组稀疏中全连接层解压缩	描述符
def SVD_decompression(w1, w2) {	
approx = mat_mul(w2, w1)	
return approx	
}	

9 数据生成方法

9.1 定义

本文件中规定的神经网络压缩技术，部分技术需要输入的网络模型权重符合一定的结构、数据、超参数等要求。为了满足这些要求，需要通过在训练过程中添加一系列的限制和特殊的训练方法，以获得可采用压缩方法的神经网络模型数据。在本章节将对于前两章节描述的神经网络模型压缩方法所限定的数据的生成方法进行描述。

本章节包括训练数据生成方法以及权重数据生成方法。其中9.2章节主要描述训练数据生成方法，9.3至9.6章节分别描述不同的神经网络压缩方法涉及到的权重数据生成方法。

神经网络训练、推理过程所用数据的处理，应遵循以下原则：

a) 在使用前，应按特定计算系统的实现及用户要求完成组织；

b) 数据处理方法所涉及的要素，应按实际需要，做出调整，包含但不限于：

1) 方法命名；

2) 参数（关键字）的选择及命名；

3) 参数类型及格式。

9.2 训练数据生成方法

9.2.1 需要真实数据的训练数据生成方法

9.2.1.1 概述

该部分主要描述了当已经存在真实训练数据时，如何对真实数据进行扩充，或使用其他方法得到更多的真实数据。

现有的方法包括传统的数据扩充方法，通过旋转、平移、缩放等操作对原始图像进行扩充；以及通过使用真实数据训练好的生成对抗网络（GAN），将噪声输入转化为图片输出。

9.2.1.2 数据扩充方法

本节介绍了如何使用旋转、平移、缩放等基本操作对已有的真实数据进行扩充。

具体的，当已有真实输入图片X时，可对X进行旋转角度α，X中的像素在x轴(y轴)上整体移动n个位置，X中的像素整体沿x轴（y轴）进行中心翻转，图片X整体进行缩放等操作，得到新的图片X’。

数据扩充方法的操作定义见表271。

表 271 数据扩充方法操作定义

运算操作	描述	字段	关键字	定义	类型	格式
数据扩充	对已有的真实数据进行扩充	Input	X	输入真实数据	tensor	NS
			α	旋转角度	int	UINT
			n	平移距离	int	UINT
			axis	平移方向（x轴或y轴）	bool	BOOL
			n_row	缩放后的宽	int	UINT
			n_col	缩放后的高	int	UINT
		Output	X’	输出扩充数据	tensor	NS

数据扩充方法的伪代码见表272。

表 272 数据扩充方法伪代码

数据扩充方法	描述符
def data_augmentation(X, α, n, axis , n_row, n_col, type): {	
if type == ‘rotate’:	
X’ = rotate(X, α)	
elif type ==’translation’:	
X’ = translate(X, n, axis=axis)	
elif type == ‘scale’:	
X’ = scale(X, n_row, n_col)	
elif type == ‘flip’:	
X’ = flip(X, axis=axis)	
return X’	
}	
def rotate(X, α): {	
X’ = np.zero((X.row, X.col))	
for i in range(X.row): {	
for j in range(X.col): {	
i’ = round(i*cosα+j*sinα)	

表 272 数据扩充方法伪代码（续）

数据扩充方法	描述符
j’ = round(-i*sinα+j*cosα)	
X’[i’,j’] = X[i,j]	
}	
}	
return X’	
}	
def translate(X, n, axis): {	
X’ = np.zero((X.row, X.col))	
if axis==’x’:	
for j in range(X.col):	
if j+n>=0 and j+n<X.col:	
X’[:, j+n] = X[:, j]	
if axis==’y’:	
for i in range(X.row):	
if i+n>=0 and i+n<X.row:	
X’[i+n, :] = X[i, :]	
return X’	
}	
def scale(X, n_row, n_col): {	
X’ = np.zeros((round(X.row*n_row), round(X.col*n_col)))	
for i in range(X’.row):	
for j in range(X’.col):	
X’[i, j] = X[round(i/n_row), round(j/n_col)]	
return X’	
}	
def flip(X, axis): {	
X’ = np.zero((X.row, X.col))	
if axis==’x’:	
for j in range(X.col):	
X’[:, j] = X[:, X.col-1-j]	
if axis==’y’:	
for i in range(X.row):	
X’[i, :] = X[X.row-1-i, :]	
return X’	
}	

9.2.1.3 生成对抗网络生成数据

本节介绍了如何利用真实训练数据训练生成对抗网络，之后使用训练好的生成对抗网络生成更多的数据。

具体的，给定输入噪声Z，生成模型G输出生成图片G(Z)。判别模型D输入两种类型的图片，一种是真实图片X，另一种是生成模型生成的图片G(Z)，判别模型需要判断出输入的图片是否来自于真实图片。G和D相互对抗，共同训练。训练完成后，输入噪声Z给生成模型G，即可得到生成的图片G(Z)。

生成对抗网络的操作定义见表273。

表 273 生成对抗网络操作定义

运算操作	描述	字段	关键字	定义	类型	格式
生成对抗网络训练及数据生成	利用真实数据训练生成对抗网络，并利用训练好的生成对抗网络生成数据	Input	X	输入真实数据	tensor	NS
			Z	输入噪声	tensor	NS
			generator_graph	生成器结构	graph	
			discriminator_graph	判别器结构	graph	
			optimizer	优化器	optimizer	
			W_g	生成器权重张量	list[tensor]	NS
			W_d	判别器权重张量	list[tensor]	NS
		Output	image	生成的图片	tensor	NS

生成对抗网络的伪代码见表274。

表 274 生成对抗网络伪代码

生成对抗网络	描述符
Generator_train(X, Z, generator_graph, discriminator_graph, optimizer, W_g, W_d, batch_size, epoch, total_iteration) {	
for epoch in range(epochs): {	
for iter in range(total_iteration): {	
batch_noise = generate_noise_batch(batch_size, Z)	
gen_imgs = generator_graph(batch_noise, W_g)	
real_imgs = generate_real_batch(batch_size, X)	
outputs = discriminator _graph(concat(gen_imgs, real_imgs))	
loss = gan_loss(outputs)	
loss.backward()	
optimizer.step()	
}	
}	
return W_g	

表 274 生成对抗网络伪代码（续）

生成对抗网络	描述符
}	
Generate_image(noise, W_g, generator_graph): {	
return generator_graph(noise, W_g)	
}	

9.2.2 无需真实数据的训练数据生成方法

9.2.2.1 概述

本节主要描述了当无法获取任何真实训练数据时，如何生成与真实训练数据类似的数据来帮助压缩原始模型。本节描述的训练数据生成方法，可以有效的兼容现有的压缩算法（例如：量化等），并将压缩后的网络应用到视觉通用任务中（例如：分类、分割等）。

9.2.2.2 利用生成网络生成训练数据

本节介绍如何训练生成网络来得到与真实训练数据相似的生成数据。当生成网络训练完毕后，输入噪声通过生成网络，即可得到生成的训练数据。生成网络的训练步骤如下：

步骤 S1：初始化生成网络，读取原始的深度神经网络；

步骤 S2：随机生成一组高斯分布的向量，并将其输入生成网络中，得到一组生成样本；

步骤 S3：将生成网络生成的样本输入原始网络中，得到神经网络对该组样本的特征和分类结果；

步骤 S4：利用本发明所提出的损失函数，对原始网络的输出计算损失；

步骤 S5：根据学习率等参数，更新生成网络中的参数；

步骤 S6：重复步骤 S2-S5，直至该生成网络模型收敛。

具体的，随机初始化一组向量$\{z_1, z_2, \cdots, z_n\}$，与一个生成网络$G$，将随机向量输入生成网络中，得到生成的样本$\{x_1, x_2, \cdots, x_n\}$，其中$x_i = G(z_i), i = 1,2, \cdots, n$。之后，将生成样本输入给定的待压缩深度神经网络$N_T$中，得到相对应的输出$\{y_1, y_2, \cdots, y_n\}$与特征$\{f_1, f_2, \cdots, f_n\}$。

之后，设计三个不同的损失函数，即交叉熵损失函数、特征激活损失函数和信息熵损失函数，并线性叠加得到最终的损失函数。

若生成样本在给定的神经网络的输出为$\{y_1, y_2, \cdots, y_n\}$，它们相对应的预测类别为$\{t_1, t_2, \cdots, t_n\}$（即取最大值对应的类别），则交叉熵损失函数$L_c$定义为：

$$\mathscr{L}_c = \frac{1}{n}\sum_i \mathscr{H}_c(y_i, t_i) \quad (31)$$

式中：

H_c——标准的交叉熵。具体定义为$H_c(y, t) = -[y \log t + (1 - y)\log(1 - t)]$。

定义信息熵损失函数L_{in}为：

$$\mathscr{L}_{in} = \mathscr{H}_{in}(\sum_i y_i / n) \quad (32)$$

式中：

H_{in}——标准的信息熵，具体定义为$H_{in} = \sum_{i=1}^{N} x_i \log x_i$。

之后，引入解码网络D，构建一个和生成网络的完全相反的推理网络，输入为生成网络的输出，输出为与生成网络输入维度一致的向量。通过计算解码网络的输出与生成网络输入的随机向量的差值L_{re}，来进行推理更新生成网络和推理网络：

$$L_{re} = \sum_i L_{MSE}(z_i, D(x_i)) \quad (33)$$

式中：

L_{MSE}——标准的均方差损失，具体定义为$L_{MSE}(a,b) = \sum_j (a_j - b_j)^2$。

最后，将上述三个损失函数线性加和，就得到关于生成网络的最终损失函数L：

$$L = L_c + \alpha L_{in} + \beta L_{re} \quad (34)$$

式中：

α——关于信息熵损失函数的超参数；

β——关于交叉熵损失函数的超参数。

通过深度深度神经网络的反向传播算法，使用上述损失函数L对生成网络进行迭代优化训练，直至收敛，可以得到一个训练好的生成网络。随机输入一组高斯分布的向量，生成网络将会输出与真实样本相似的训练样本来辅助神经网络的压缩训练。

训练数据生成方法的操作定义见表 275。

表 275 训练数据生成方法操作定义

运算操作	描述	字段	关键字	定义	类型	格式
训练数据生成	训练生成网络用来生成训练数据	Input	generator_graph	生成器结构	graph	
			model_graph	待压缩网络结构	graph	
			decoder_graph	解码器结构	graph	
			optimizer	优化器	optimizer	
			W_g	生成器权重张量	list[tensor]	NS
			W_m	待压缩网络权重张量	list[tensor]	NS
			W_d	解码器权重张量	list[tensor]	NS
			weight_c	交叉熵损失超参数	float	FP32
			weight _in	信息熵损失超参数	float	FP32
			weight _re	重构损失超参数	float	FP32
			latent_dim	噪声输入的长度	int	UINT
		Output	image	生成的训练数据	tensor	NS

训练数据生成方法的伪代码如见表 276。

表 276 训练数据生成方法伪代码

训练数据生成方法	描述符
def generator_train(noise, generator_graph, model_graph, decoder_graph, optimizer, W_g, W_m, W_d, weight _c, weight _in, weight _re, batch_size, epoch, total_iteration): {	
for epoch in range(epochs): {	
for iter in range(total_iteration): {	
batch_noise = generate_random_vectors(batch_size, latent_dim)	随机生成噪声
gen_imgs = generator_graph(batch_noise, W_g)	噪声输入生成模型，得到生成数据
outputs = model_graph(gen_imgs, W_m)	生成数据输入待压缩网络得到输出
pred = outputs.max(1)	输出最大的概率作为预测类别
recon = decoder_graph(gen_imgs, W_d)	生成数据输入解码器，得到重构噪声
loss_c = cross_entropy_loss(outputs, pred)	交叉熵损失
loss_re = mse_loss(recon, batch_noise)	重构损失
loss_in = information_entropy_loss(outputs)	信息熵损失
loss = weight_c*loss_c + weight_in*loss_in + weight_re*loss_re	
loss.backward()	反向传播
optimizer.step()	更新W_g, W_m, W_d
}	
}	
return W_g	训练好的生成器参数
}	
def generate_image(noise, W_g, generator_graph): {	
return generator_graph(noise, W_g)	
}	

9.3 多模型

9.3.1 多模型层内权重共享数据生成方法

9.3.1.1 多模型权重更新数据定义

使用多模型层内权重共享压缩技术，需要通过在训练过程中对多模型中的模型权重添加限制，并获得支持此技术的权重结构，这里进行多模型权重更新的算子描述。具体定义见表277。

表 277 多模型权重更新数据定义

运算操作	描述	字段	关键字	定义	类型	数据类型
多模型权重更新	训练过程中，更新多模型权重的方法和操作	Input	x	训练数据输入	tensor	NS
			y	训练数据真值	tensor	NS
			Graph	网络图	Function	
			Optimizer	优化器	function	
			W_share	共享权重张量	tensor	NS
			B_share	共享偏置向量	vector	NS
			W_specific	特有权重张量列表	List of tensors	NS
			B_specific	特有偏置向量列表	List of vectors	NS
			model_indicator	训练数据对应的模型索引	value	Int
			loss_factor	误差归一化系数列表	vector	float
多模型权重更新	训练过程中，更新多模型权重的方法和操作	Attributes	axe	权重聚合维度	int	Int
		Output	W_share_updated	更新后的共享权重张量	tensor	NS
			B_share_updated	更新后的共享偏置向量	vector	NS
			W_specific_updated	更新后的特有权重张量列表	List of tensors	NS
			B_specific_updated	更新后的特有偏置向量列表	List of vectors	NS

对于采用本技术压缩的神经网络，可以采用 8.1.2.3 章节中的方法解压输出指定模型获得对应模型的权重，其结构与传统神经网络相同，可以兼容传统神经网络的正向推理操作。而对于多模型层内权重共享技术的整体数据训练方法，具体有两种实现方式，下文进行介绍。

9.3.1.2 多模型层内权重共享数据生成方法一

模型层内权重共享压缩的待压缩数据生成方法一，层内先提取特有权重，在进行数据处理。训练的具体步骤如下。

a) 确定共享权重的多个模型中所需权重的神经网络层位置，多个模型对应的该神经网络层其拓扑结构相同。

b) 获取训练对象：从训练数据输入x，训练数据真值y中获取训练样本batch，包含多个任务。

c) 在一次训练迭代中，对训练样本逐个进行正向计算。对任意一个网络层：

1) 获取共有权重和特有权重：确定该神经网络层中共享权重 W_share 与独立权重间的区分位置索引。

2) 提取针对该训练样本的特有权重，与共享权重合并，偏置进行相同处理。

d) 计算（统计）误差loss: 根据网络图Graph确定训练样本的正向计算结果，结合训练数据真值计算loss，并且根据训练样本对应的模型索引，用loss_factor加权调整loss。

e) 调整特有权重值以及加权调整共享权重值：反向传递加权调整后的loss，进行梯度计算，并调整训练样本对应的模型的权重（共有权重和对应模型的特有权重）。

f) 重复步骤b)～e)，完成一个epoch。

g) 重复步骤b)～f)，训练网络更新参数至网络收敛或达到训练次数上限。

伪代码见表278。

表 278 多模型数据生成方法一伪代码

多模型数据生成方法一	描述符
Multilayer_weight_update_train(x, y, Graph, Optimizer, W_share, B_share, W_specific, B_specific, model_indicator, axe,batch_size,epoch) {	x, y为训练集，非单个样本；model_indicator为对应列表
for epoch in range(epochs): {	
for ite in range(int(len(trainX)/batch_size)): {	
batch_x=x[ite:ite+batch_size]	
batch_y=y[ite:ite+batch_Size]	
batch_model_indicator=modek_indicator[ite:ite+batch_size]	
for i in range(len(batch_x)): {	
idx=batch_model_indicator[i]	
W = Concate(W_share, W_specific[idx], axe)	
B = Concate(B_share, B_specific[idx], axe)	
Result = Graph(batch_x[i],W,B)	
Loss = compute_loss(Result, batch_y[i]) * loss_factor[idx]	
W_gradient, B_gradient = compute_gradient(W, B, Loss, Optimizer)	
W_gradient_share = slice(W_gradient[0], axe, 0, W_share.shape(axe))	
W_gradient_specific = slice(W_gradient[0], axe, W_share.shape(axe), -1)	
B _gradient_ share = slice(B_gradient[0], axe, 0, B_share.shape(axe))	
B _gradient_specific = slice(B_gradient[0], axe, B_share.shape(axe), -1)	
W_share = W_share + W_gradient_share	
B_share=B_share + B_gradient_share	
W_specific[idx] += W_gradient_specific	
B_specific[idx] += B_gradient_specific	
}	
}	
}	
W_share_updated=W_share	
W_specific_updated=W_specific	
B_share_updated=B_share	
B_specific_updated=B_specific	
return W_share_updated, W_specific_updated, B_share_updated, B_specific_updated	
}	

9.3.1.3 多模型层内权重共享数据生成方法二

多模型层内权重共享压缩的待压缩数据生成方法二，层内不提取特有权重，对输入数据进行处理后，提取共享输出数据和对应的特有输出数据至下一层，如图 8所示，训练的具体步骤如下。

a) 确定共享权重的多个模型中所需权重的神经网络层位置，多个模型对应的该神经网络层其拓扑结构相同。
b) 获取训练对象： 从训练数据输入x，训练数据真值y中获取训练样本batch，包含多个任务。
c) 在一次训练迭代中，对训练样本逐个进行正向计算。对任意一个网络层：
   1) 获取共有权重和特有权重：通过确定该神经网络层中共享权重W_share与独立权重间的区分位置索引
   2) 构建一个神经网络层Ln，合并多个模型的神经网络层权重（共享权重+特有权重）于Ln。
   3) 通过过滤器选择共享输出数据和特有输出数据：在某次训练迭代中，针对训练样本设定过滤器的输出，仅训练样本对应的模型的过渡层神经元数据可以通过，输入至下一层中。
d) 计算（统计）误差loss: 根据网络图Graph确定训练样本的正向计算结果，结合训练数据真值计算loss，并且根据训练样本对应的模型索引，用loss_factor加权调整loss。
e) 调整特有权重值以及加权调整共享权重值： 反向传递加权调整后的loss，进行梯度计算，并调整训练样本对应的模型的权重（共有权重和对应模型的特有权重）。
f) 重复步骤b) ~ e)，完成一个epoch。
g) 重复步骤b) ~ f)，训练网络更新参数至网络收敛或达到训练次数上限。

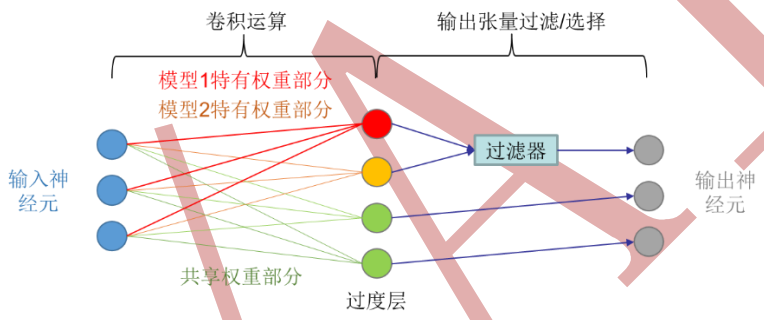


图 8 多模型间层内部分权重共享技术训练方法

伪代码见表279。

表 279 多模型数据生成方法二伪代码

多模型权重共享数据生成方法二	描述符
Multilayer_weight_update_train2(x, y, Graph, Optimizer, W_share, B_share, W_specific, B_specific, model_indicator, axe,batch_size,epoch,perc) {	x, y为训练集，非单个样本；model_indicator为对应列表。
for epoch in range(epochs): {	perc为共享权重比例
for ite in range(int(len(trainX)/batch_size)): {	
batch_x=x[ite:ite+batch_size]	
batch_y=y[ite:ite+batch_Size]	
batch_model_indicator=modek_indicator[ite:ite+batch_size]	
for i in range(len(batch_x)): {	
idx=batch_model_indicator[i]	
layer_in=batch_x[i]	
for layer in Graph.layers: {	

表 279 多模型数据生成方法二伪代码（续）

多模型权重共享数据生成方法二	描述符
layer_W= Concate(layer.W_share, layer.W_specific, axe)	
layer_B= Concate(layer.B_share, layer.B_specific, axe)	
layer_out=layer.operate(layer_in,layer_W,layer_B)	
#filtering	
share_end=perc*layer_out.shape(axe)	
len_specific=(1-perc)*layer_out.shape(axe)/len(set(model_indicator))	
layer_out_share=slice(layer_out,axe,0,share_end)	
layer_out_specific=slice(layer_out,axe,share_end+idx*len_specific, share_end+(idx+1)*len_specific)	
layer_out=Concate(layer_out_share,layer_out_specific)	
layer_in=layer_out	
}	
Result = layer_out	
Loss = compute_loss(Result, batch_y[i]) * loss_factor[idx]	
W_gradient, B_gradient = compute_gradient(W, B, Loss, Optimizer)	
W_gradient_share = slice(W_gradient[0], axe, 0, W_share.shape(axe))	
W_gradient_specific = slice(W_gradient[0], axe, W_share.shape(axe), -1)	
B _gradient_share = slice(B_gradient[0], axe, 0, B_share.shape(axe))	
B _gradient_specific = slice(B_gradient[0], axe, B_share.shape(axe), -1)	
W_share = W_share + W_gradient_share	
B_share=B_share + B_gradient_share	
W_specific[idx] += W_gradient_specific	
B_specific[idx] += B_gradient_specific	
}	
}	
}	
W_share_updated=W_share	
W_specific_updated=W_specific	
B_share_updated=B_share	
B_specific_updated=B_specific	
return W_share_updated, W_specific_uodated, B_share_updated, B_specific_updated	
}	

9.3.2 多模型残差量化训练方法

采用多模型残差量化的同时，需要采取对应的训练方法来提升量化后的性能。训练方法是，固定预训练模型权重，通过同训练量化后的共享权重，进一步提升网络的性能。梯度更新的方式为，属于同类的梯度被替换为聚类结果中同类的均值之和。具体定义见表280。

表 280 多模型残差量化数据生成数据定义

运算操作	描述	字段	关键字	定义	类型
DFQuantization_weight_update	多模型残差量化通过重训练提升网络性能	Input Attributes	W_share_original	初始的共享权重	tensor
			Base model	预训练模型权重	tensor
			W_gradients	权重对应的梯度	Tensor
			W_specific	权重聚类结果(code storge)	List of tensors
		Output	W_share_output	输出的共享权重张量	tensor

伪代码见表 281。

表 281 多模型残差量化训练方法伪代码

多模型残差量化训练方法	描述符
DFQuantization_weight_update(W_share, Base_model,gradients,W_specific): {	
Epoch=100	
W_share_output = W_share_original	
for(j=0;j<Epoch;j++){	
gradients =Gradients_compute(W_share_output, Base_model)	反向传播权重梯度计算
gradients=average(W_specific)	根据聚类结果求同类梯度均值
Lr = 1e-4	
for(i=0;i<len(W_specific);i++)	
W_share_output[i]=W_share[i]-Lr*gradients[i]	根据均值化的梯度进行梯度下降
}	
Return W_share_output	
}	

9.4 量化

9.4.1 权重训练方法

9.4.1.1 INT4 参数量化数据生成

使用INT4量化配置神经网络时，对神经网络模型进行压缩的过程包括权重量化、量化尺度因子修正和激活量化，神经网络的输出根据量化后的权重张量和激活张量计算得到。INT4参数量化数据生成对应的激活量化数据生成方式为INT4激活量化数据生成。

INT4参数量化中，对于神经网络的偏置不做处理，对于神经网络权重，则逐层进行量化。INT4参数量化数据生成的流程如下。

a) 对于神经网络的权重逐层进行 INT4 量化，权重量化尺度因子的计算方法为：
   1) 对于输入的神经网络模型的每一层权重张量进行单 INT4 量化，得到量化后的权重张量和对应的权重量化尺度因子。其中量化时，对于卷积层采用卷积核粒度量化，对于全连接层采用层级粒度量化。
   2) 根据重建误差和预定义的误差阈值，定位需要进行 Multiple INT4 量化的网络层。
   3) 对于所定位的层进行 Multiple INT4 量化，通过最小化原始张量和 Multiple INT4 量化重建张量的距离确定量化尺度因子。令W为网络层待量化张量，$\alpha_i, \widetilde{W_l}, i = 1 \dots t$ 为 Multiple 量化对应的尺度因子和量化张量，则$\alpha_i, \widetilde{W_l}$的计算方法为：

$$\min_{\alpha_1, \widetilde{W_1} \in I_{k_1}} \left\{ \min_{\alpha_2, \widetilde{W_2} \in I_{k_2}} \left\{ \dots \min_{\alpha_n, \widetilde{W_n} \in I_{k_n}} d\left(W, \sum_{i=1}^{n} \alpha_i \widetilde{W_l}\right) \right\} \right\} \quad \cdots\cdots(35)$$

   式中：

   d ——距离度量；

   I_{k_i}——量化允许范围。

   一般情况下，采用 Dual INT4 量化，尺度因子通过 dual 线性搜索含二次项的重建误差实现。

b) 调整权重量化尺度因子，量化因子修正系数的计算方法为：选取一个校准集，根据量化后的神经网络模型的输出和原始非量化模型的输出，以使两个输出的距离最小为目标，优化量化尺度因子。令γ_l为第l层的量化因子修正参数，则修正后的量化张量为$\widehat{W_l} = \gamma_l\left(\alpha_l \widetilde{W_l}\right)$。$\gamma_l$的计算方法为：

$$\gamma_l = \min_{\gamma}\left(\sum_l \beta_l d\left(f(X, W_l), f(X, \widehat{W}_l(\gamma))\right)\right) \quad \cdots\cdots(36)$$

式中：

$0 \le \beta_l \le 1$——第l层的加权。

INT4参数量化数据生成的数据定义见表282。

表 282 INT4 参数量化数据生成的数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
int4_quantization_parameter INT4 参数量化数据生成	对神经网络逐层进行无训练的权重量化	Input	W	待量化的权重张量	list[tensor]	float32
			B (optional)	网络层偏置张量	list[tensor]	NS
			Graph	网络图	function	/
			mse_thresh	量化重构误差阈值	value	float32
			cali_sf	尺度因子优化校准集	tensor	float32
		Attributions	offset_w	权重量化偏置标识	value	Boolean
			bit_w	量化位数	value	int
			n_points	线性搜索样本点数	value	int
		Output	W_quantized	量化后的权重张量	list[tensor]	signed int4
			B_quantized (optional)	量化后的网络层偏置张量	list[tensor]	NS
			sf_w_list	权重量化尺度因子列表	list	float32
			dual_list	网络层进行 Dual INT4 量化标识列表	list	Boolean

INT4参数量化数据生成的伪代码见表283。

表 283 INT4 参数量化数据生成伪代码

INT4参数量化数据生成	描述符
def int4_quantization_parameter(W,Graph, mse_thresh,offset_w=False, cali_sf, B): {	
W_quantized,sf_w_list,dual_list=quantize_weights(W,mse_thresh,offset_w=False)	
sf_w_list=optimize_sf(W,W_quantized,sf_w_list,Graph, cali_sf)	
B_quantized=B	
return W_quantized, sf_w_list, B_quantized, dual_list	
}	

INT4参数量化数据生成的第一部分是权重量化，quantize_weights()操作的伪代码见表284。

表 284 quantize_weights()操作伪代码

quantize_weights()操作	描述符
def quantize_weights(W,mse_thresh,offset_w=False,bit_w=4): {	
dual_list=[]	
sf_w_list=[]	
W_quantized=[]	
if offset:	
W-=min(W)	
for i,w in enumerate(W): {	
if len(w.shape)>2:	Conv layer
for k in range(len(w):	
w_quantized[k],scaling_factor[k]=quantize(w[k],bit_w=4,offset_w,\ n_points=500)	
else:	FC layer
w_quantized,scaling_factor=quantize(w,bit_w=4,offset_w,n_points=500)	
w_recon=int4_recon_quantized_parameter(w_quantized,scaling_factor,\ offset_w_list=None)	
mse=mean((w-w_recon)**2)	
if mse>mse_thresh: {	Dual INT4
dual_list.append(True)	
if len(w.shape)>2:	Conv layer
for k in range(len(w):	
w_quantized[k],scaling_factor[k]=\	
dual_quantize(w[k],bit_w=4,offset_w,n_points=30)	
else:	
w_quantized,scaling_factor=dual_quantize(w,bit_w=4,\ offset_w,n_points=30)}	FC layer
}	
else: {	
dual_list.append(False)	
}	
W_quantized.append(w_quantized)	
sf_w_list.append(scaling_factor)	
}	
return W_quantized,sf_w_list,dual_list	
}	

其中，int4_recon_quantized_parameter() 操作见INT4参数反量化章节，quantize()操作的伪代码见表285。

表 285 quantize()操作伪代码

quantize()操作	描述符
def quantize(w,bit_w=4,offset_w,n_points=500): {	
if offset: {	
max_range_w = max(w) - min(w)	
range_max = 2. ** bit_w - 1.	
range_min = 0.	
start_alpha, end_alpha = 1e-16, 1.1 * max_range_w / range_max	
}	
else: {	
max_range_w = max(abs(w))	
range_max = 2. ** (bit_w - 1) - 1	
range_min = -2. ** (bit_w - 1)	
start_alpha, end_alpha = -1.1 * max_range_w / range_max, \ 1.1 * max_range_w / range_max	
}	
alpha_vec = linspace(start_alpha, end_alpha, n_points)	
res_vec=[[] for i in range(len(alpha_vec)]	
for i,alpha in enumerate(alpha_vec): {	
wq=clip(round(w/alpha),range_min,range_max)	
res_vec[i]=max(abs(w-wq*alpha))	
}	
scaling_factor=alpha_vec[argmin(res_vec)]	
w_quantized=clip(round(w/scaling_factor),range_min,range_max)	
return w_quantized,scaling_factor	
}	

dual_quantize()操作的伪代码见表286。

表 286 dual_quantize()操作伪代码

dual_quantize()操作	描述符
def dual_quantize(w,bit_w, offset_w, n_points): {	
if offset: {	
max_range_w=max(w)-min(w)	
range_max=2.**bit_w-1	
range_min=0	
w-=min(w)	

表 286 dual_quantize()操作伪代码（续）

dual_quantize()操作	描述符
start_alpha, end_alpha = 1e-16, 1.1 * max_range_w / range_max	
}	
else: {	
max_range_w = max(abs(w))	
range_max = 2. ** (bit_w - 1) - 1	
range_min = -2. ** (bit_w - 1)	
start_alpha, end_alpha = -1.1 * max_range_w / range_max, \ 1.1 * max_range_w / range_max	
}	
alpha_vec = linspace(start_alpha, end_alpha, n_points)	
factors=[]	
for r0 in alpha_vec:	
for r1 in alpha_vec:	
factors.append([r0,r1])	
res_vec=[[]for i in range(len(factors))]	
for i,fct in enumerate(factors): {	
wq=get_params_double_grid(w,fct,range_min,range_max,bit_w=4)	
res_vec[i]=mean((wq[0]*fct[0]+wq[1]*fct[1]-w)**2)	
}	
scaling_factor=factors[argmin(res_vec)]	
w_quantized=get_params_double_grid(w,fct,range_min,range_max,bit_w=4)	
return w_quantized,scaling_factor	
}	

其中，get_params_double_grid()操作见INT4参数量化章节。

INT4量化数据生成的第二部分是优化尺度因子，optimize_sf()操作的伪代码见表287。

表 287 optimize_sf()操作伪代码

optimize_sf()操作	描述符
def optimize_sf(W,W_quantized,sf_w_list,Graph,cali_sf,epochs=25): {	
gamma=[[] for i in range(len(sf_w_list))]	
W_recon=[[] for i in range_len(sf_w_list)]	
for l in range(len(sf_w_list)):	
W_recon[l]=int4_recon_quantized_parameter(W_quantized[l],sf_w_list[l],\ offset_w_list=None)	
W.trainable=False	
W_recon.trainable=False	

表 287 optimize_sf()操作伪代码（续）

optimize_sf()操作	描述符
gamma.trainable=True	
optimizer=RMSprop(gamma*W_recon)	
for epoch in epochs: {	
for batch in cali_sf: {	
out_quantized_gamma=Graph(batch,gamma*W_recon)	
out_orig=Graph(batch,W)	
loss=mean((out_quantized_gamma-out_orig)**2)	
loss.backward()	
optimizer.step()	
}	
}	
for l in range(len(sf_w_list)):	
sf_w_list[l]=gamma[l]*sf_w_list[l]	
return sf_w_list	
}	

其中， int4_recon_quantized_parameter () 操作见第INT4参数反量化章节。

9.4.1.2 定点量化数据生成

本小节定点量化的数据生成，限制条件与 7.2.3.2 参数定点量化，7.2.4.2 激活值定点量化，8.2.3.2 定点反量化一致。本小节介绍网络整体的定点量化流程以及定点训练的方法。定点量化数据生成的数据定义见表 288。

表 288 定点量化数据生成的数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
fix_quantize 定点量化整个网络	定点量化整个网络中的参数和激活值	Input	graph	网络图	function	/
			method	量化方法（0: non-overflow, 1: min-diff）	value	int
			n	量化比特数	value	int
			calibration_dataset	校准数据集	list[tensor]	float
		Output	graph_quantized	量化后的网络图	function	/

定点量化数据生成的伪代码见表 289。

表 289 定点量化数据生成伪代码

quantize_op操作	描述符
class quantize_op(): {	
def __init__(self, method, n): {	
self.is_calibration = False	
self.method = method	
self.n = n	
self.position = 0	
self.position_distribution = {}	
}	
def forward(self, X): {	
if self.is_calibration: {	
(Xd, Xp) = quantize_tensor(X, self.method, self.n)	
self.position_distribution[Xp] += 1 # update distribution	
# update position according to current distribution	
self.position = count_max(self.position_distribution)	
return Xd	
}	
else: {	
Xd, Xq = calc_quantized_value(X, self.position)	
return Xd	
}	
}	
}	
def fix_quantize(graph, method, n, calibration_dataset): {	
graph_quantize = insert_quantize_op_after_activation(graph)	
for (W,B) in graph_quantize.all_weights(): {	
fix_quantize_parameter(W, B, method, n)	
}	
graph_quantize.setmode(“calibration”)	
for i in range(len(calibration_dataset)): {	
batch_data = calibration_dataset[i]	
graph_quantize.forward(batch_data)	
}	
return graph_quantize	
}	

9.4.1.3 定点训练

量化能带来资源消耗的减少，运算速度的增加，当然也会存在一些问题和挑战，最主要的问题就是精度损失。对于大部分卷积神经网络，比如 Inception 系列和 Resnet 系列，8bit 量化能够比较好的保持网络的精度，精度损失可以控制在 2%甚至 1%以内，但对于 MobileNets 这类含有 Depth-wise 卷积的网络，量化的精度损失会更大一些，有的在量化后甚至都没有什么精度。而对于更低比特数的量化，比如 4bit 和 2bit，精度的损失就更为明显。在这种情况下，定点训练，也叫量化训练（quantize finetuning/quantize-aware training）就是对抗精度损失的重要手段。

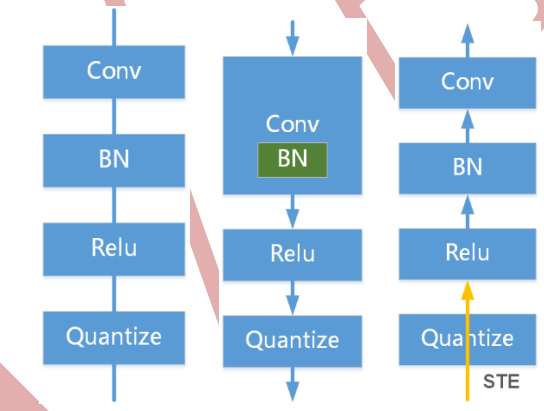
定点训练过程中的数据生成方法，这里只介绍训练相关的处理，包括量化操作梯度的计算以及对 Batchnorm 的处理。

a) 量化操作梯度的计算：

量化操作本身是一个不可导的操作，因此在反向的时候不能直接计算梯度，可以采用 STE 的方法来处理量化操作的梯度，简单说来就是直接让梯度传递到量化操作的上一层。图 9 a)展示了一个卷积神经网络的常见结构，在 Conv, BN, Relu 之后进行量化操作，图 9 c)展示了反向传播时对量化层（FixNeuron）的处理方法。

b) 对 Batchnorm 的处理

Batchnorm（BN）是神经网络训练中经常采用的一项技术，它能加速网络收敛，提高网络的性能。对于量化来说，独立的 BN 层经过量化之后会引入比较大的精度损失，因此在量化校准的过程中，BN 层往往被合并到卷积层之后再整体进行量化。对于量化训练，我们也采用类似的策略，在神经网络前向计算的时候，将 BN 合并到卷积层之后进行量化，如图 9 b)，而在反向传播的时候，分别对 BN 层和 Conv 层进行更新，如图 9 c)。前向合并量化，反向独立更新，这样的量化训练策略即克服了 BN 层单独量化容易带来精度损失的问题，又保留了 BN 层在训练过程中的优点。实验结果也表明，这是是一种行之有效的量化训练方法。



a）量化网络示例　　b) 定点训练前向计算　　c) 定点训练反向计算

图 9 定点训练

定点训练的数据定义见表 290。

表 290 定点训练的数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
fix_quantize_training 定点训练整个网络	定点训练整个网络中的参数和激活值	Input	graph	网络图	function	/
			method	量化方法（0: non-overflow, 1: min-diff）	value	int
			n	量化比特数	value	int
			training_dataset	校准数据集	list[tensor]	float
		Output	graph_quantized	量化后的网络图	function	/

定点训练的伪代码见表 291。

表 291 定点训练的伪代码

定点训练操作	描述符
class Quantize_op(): {	
def __init__(self, method, n): {	
self.is_calibration = False	
self.is_training = False	
self.method = method	
self.n = n	
self.position = 0	
self.position_distribution = {}	
}	
def forward(self, X): {	
if self.is_calibration or self.is_training: {	
(Xd, Xp) = quantize_tensor(X, self.method, self.n)	
self.position_distribution[Xp] += 1 # update distribution	
self.position = count_max(self.position_distribution)	
return Xd	
}	
else: {	
Xd, Xq = calc_quantized_value(X, self.position)	
return Xd	
}	
}	

表 291 定点训练的伪代码（续）

定点训练操作	描述符
def backward(self,delta): {	
# backward, STE	
return delta	
}	
}	
def quantize_training_forward(graph, batch_data): {	
graph_fold = fold_batchnorm(graph)	
return graph_fold.forward()	
}	
def fix_quantize_training(graph, method, n, training_dataset): {	
Graph_quantize = insert_quantize_op(graph)	
graph_quantize.setmode(“training”)	
for i in range(len(training_dataset)): {	
batch_data = calibration_dataset[i]	
output = quantize_training_forward(graph_quantize,batch_data)	
loss = criterion(output, label)	
loss.backward()	
optimizer.update()	
}	
return graph_quantize	
}	

9.4.1.4 区间收缩量化数据生成

区间收缩量化，指的是在单个区间中的权重，在训练过程中向量化中心点逐渐靠近的过程，主要用于低比特量化。区间收缩参数量化数据生成的操作定义见表292。

表 292 区间收缩量化数据生成定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
Shrink_quantization 区间收缩量化数据生成方法	将离散表示的高精度网络参数，通过区间收缩量化方法，生成量化后参数。	Input	W	待量化的权重张量	List [tensor]	float32
			B (optional)	网络层偏置张量	List [tensor]	float32
			Graph	网络图	function	/
			num_centers	量化中心数	Value	int
			method	中心获取方式	Value	int
			optimizer	优化器	class	/
			X	训练数据	List [tensor]	float32
			Y	训练标签	List [tensor]	int
			get_center	用户方法	function	/
		Output	W_quantized	量化后的权重张量	List [tensor]	float32
			B_quantized (optional)	量化后的网络层偏置张量	List [tensor]	float32

对于每一个需要量化的层的权重，根据其粒度，基于区间收缩的量化流程如下。

a) 聚类：

1) 如果 method 等于 0，则采用 K-Means 方法聚类，参考 K-Means 数据生成过程获得大小为 num_centers 的查找表值 W_v，同时返回查找表索引值 W_i；如果 method 等于 1，则采用用户自定义的方法生成大小为num_centers的查找表值W_v,同时返回查找表索引值W_i。

2) 按照 W_v 的绝对值大小进行排序，作为不同区间的量化顺序。

3) 将权重分为收缩区间S,重训练区间 R和冻结区间F。

b) 收缩：

收缩是区间量化的主要过程。对于收缩区间内的权重，采用 L1 损失使其在训练靠近量化中心并且随着训练进行，将区间内的权重逐渐赋值为权重中心。这一过程 L1 损失计算的流程图如下所示：

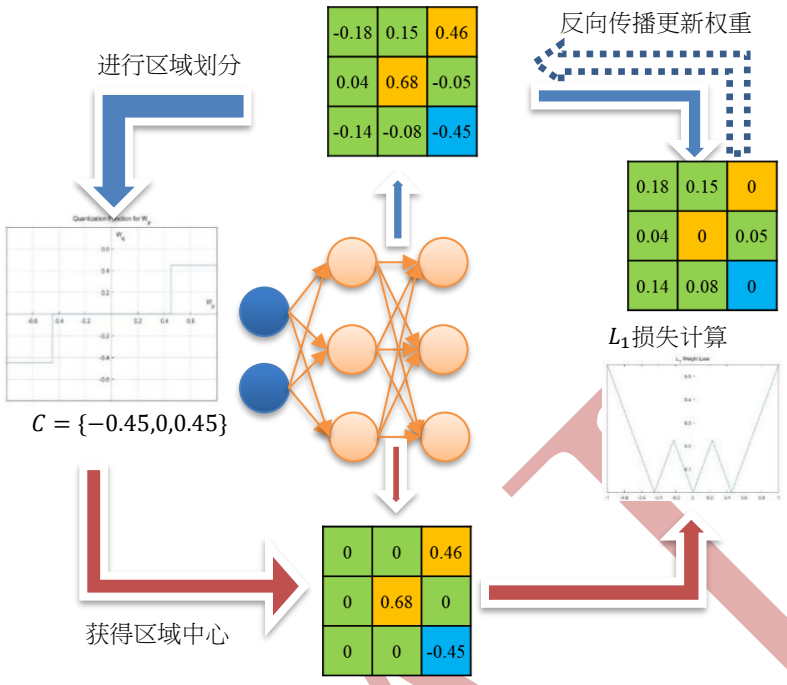


图 10 L1 损失计算示意图

图中，绿色为收缩区间中的权重，蓝色为冻结区间的权重，黄色为重训练区间的权重。对于每层收缩区间中的权重，采用 L1 损失，如公式(37)：

$$L_1 = \sum|w - c| \quad if\ w \in S \tag{37}$$

式中：

w——神经网络模型权重；

c——当前阶段收缩区域的量化中心；

S——收缩区域范围。

在训练的过程中，按照区间内权重离量化中心的绝对距离，随着训练的进行，由小至大将权重赋值为量化中心值，从而达到量化的效果，如公式(38)-(41)。

$$e_{R0} = s|c - e_0| \tag{38}$$

$$e_{R1} = s|c - e_1| \tag{39}$$

$$Q = [c - e_{R0}, c + e_{R1}] \tag{40}$$

$$w_s(p,q) = \begin{cases} c & ,if\ w \in Q \\ w(p,q) & ,if\ w \notin Q \end{cases} \tag{41}$$

式中：

e_0、e_1——收缩区间S的端点（满足$e_0 < e_1$），即相邻两区间量化中心的中点；

e_{R0}、e_{R1}——量化过程中，量化中心到两个收缩区域端点的实时距离；

s——比例系数（满足$0 \le s \le 1$）；

c——当前阶段收缩区域的量化中心；

Q——收缩量化区域范围；

$w(p,q)$——原始网络模型，位于(p,q)位置的权重值；

$w_s(p,q)$——收缩量化操作后，位于(p,q)位置的权重值。

对于已量化的权重，将其在反向传播时的梯度置零，这就意味着其权重值不会随着训练的进行

而改变，从而达到冻结的效果。

c) 重训练

重训练是补偿量化带来损失的重要手段。对于重训练区间中的权重，采用正常的训练方式，如交叉熵损失如公式(42)：

$$L_R = L_{CE}\left(f\left(x^{(i)};\theta\right),y^{(i)}\right) \quad \text{.....(42)}$$

式中：

$x^{(i)}$——第 i 个批次的数据；

$y^{(i)}$——第 i 个批次的标签；

θ——神经网络模型权重；

f——神经网络前向推理过程；

L_{CE}——交叉熵损失函数；

L_R——重训练损失。

为了防止重训练的权重在反向传播后的值位于已冻结的区间，对其进行范围限制，使其不能超过重训练区间范围。

d) 总损失与训练

总损失是类别的交叉熵损失和 L1 权重损失。

从神经网络训练集中选取t个样本$\{x^{(1)},x^{(2)},\ldots,x^{(t)}\}$，用于网络的学习优化，其中$x^{(i)}$所对应目标为$y^{(i)}$；

计算神经网络的分类误差损失L_R，计算总量化损失L_S，以一定比例相加得到总损失L，见式（43）：

$$L = L_R + L_S = L_{CE}\left(f\left(x^{(i)};\theta\right),y^{(i)}\right) + \alpha\cdot\sum L_{1_l} \quad \text{.....(43)}$$

式中：

$x^{(i)}$——第 i 个批次的数据；

$y^{(i)}$——第 i 个批次的标签；

θ——神经网络模型权重；

f——神经网络前向推理过程；

L_{CE}——交叉熵损失函数；

L_S——量化损失；

L_R——重训练损失，见式(42)；

α——量化损失系数；

L_{1_l}——第l层的 L1 损失，见式(37)；

L——总损失。

使用式(44)链式法则计算梯度：

$$g = \frac{1}{t}\nabla_\theta \sum_i^t L_i \quad \text{.....(44)}$$

式中：

t——一个批次的数据数量；

L_i——第 i 个批次数据的总损失；

∇_θ——对权重求偏导数；

g——权重梯度。

依次更新权重参数：

$$\theta = \theta - \varepsilon g \quad (45)$$

式中：

θ——神经网络模型权重；

ε——优化器的学习率。

伪代码见表 293。

表 293 基于区间收缩的量化伪代码

基于区间收缩的量化	描述符
def shrink_weight(weight, num_center, method, alpha, d_s, \	
X, Y, optimizer, graph,get_center=None) :{	
S, R, F = zeros(shape(weight)), ones(shape(weight)), zeros(shape(weight))	
for i in range(0, num_center):{	
if (method == 0):{	
C_i = k_means(weight, num_center) # k-means聚类函数	
c_arg = argsort(abs(C_i))	
}	
elif (method == 1):{	
C_i = get_center(weight, num_center) #用户自定义函数	
c_arg = argsort(abs(C_i))	
}	
e_0, e_1 = get_endpoints(C_i, i) # 将相邻量化中心的中点作为区间端点	
s = 0	
c = C_i[c_arg == i]	
S = weight[weight>e_0 and weight<e_1]	
R = R - S	
while (s <= 1):{	
for (x,y) in (X, Y):{	
L1 = sum(abs(S – c))	
e_R0 = s * abs(c-e_0)	
e_R1 = s * abs(c-e_1)	
M_f = [S > c – e_R0 and S < c – e_R1]	
weight[M_f] = c	
y_p = graph(x)	
l_c = compute_loss(y_p, y, loss)	
l_all = l_c + alpha * L1	
w_g = compute_gradient(l_all, [weight[R], S-weight[M_f]], optimizer)	
weight = weight + w_g	

表 293 基于区间收缩的量化伪代码（续）

基于区间收缩的量化	描述符
s = s + d_s	
F = F + R	
}	
}	
}	
}	

9.4.2 激活训练方法

9.4.2.1 INT4 激活量化数据训练

使用INT4量化配置神经网络时，对神经网络模型进行压缩的过程包括权重量化、量化尺度因子修正和激活量化，神经网络的输出根据量化后的权重张量和激活张量计算得到。INT4激活量化数据生成对应的参数量化数据生成方式为INT4参数量化数据生成。

对于神经网络的激活逐层进行 INT4 量化,激活量化尺度因子的计算方法为：

a) 获取校准集，通过最小化校准集上的原始激活张量和重建激活张量的误差，进行 INT4 量化，确定对应的激活量化尺度因子。令校准集尺寸为M，A_i^l为第 i 个样本对应的第l层的激活张量，则第一个尺度因子的计算方法见式（46）：

$$\beta_{l,1} = argmin_{\beta}\left\{\sum_{i=1}^{M}\left\|A_i^l - \beta[A_i^l/\beta]\right\|^2\right\} \quad (46)$$

式中：

$[\cdot]$——$[\cdot]_{\mathbb{Z}_q}$；

$\mathbb{Z}_q$——量化范围。

b) 获取权重量化时定位的进行 Multiple INT4 量化的网络层，进行 Multiple INT4 量化，一般为 Dual INT4 量化。第二个尺度因子根据量化残差部分获得，计算方法见式（47）：

$$\beta_{l,2} = argmin_{\beta}\left\{\sum_{i=1}^{M}\left\|A_i^l - \beta_{l,1}[A_i^l/\beta_{l,1}] - \beta\left[\frac{A_i^l-\beta_{l,1}\left[\frac{A_i^l}{\beta_{l,1}}\right]}{\beta}\right]\right\|^2\right\} \quad (47)$$

式中：

$[\cdot]$——$[\cdot]_{\mathbb{Z}_q}$；

$\mathbb{Z}_q$——量化范围。

INT4激活量化数据生成的数据定义见表294。

表 294 INT4 激活量化数据生成数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
int4_quantization_activation INT4 量化数据生成	对神经网络逐层进行无训练的激活量化	Input	X	待量化的激活张量	tensor	float32
			Graph	网络图	function	/
			dual_list	网络层进行 Dual INT4 量化标识列表	list	Boolean
			cali_act	激活量化校准集	tensor	float32
			offset_act	激活量化偏置标识	value	Boolean
		Attribute	bit_w	量化位数	value	int
			n_points	线性搜索样本点数	value	int
		Output	X_quantized	量化后的激活张量	list[tensor]	signed/unsigned int4
			sf_act_list	激活量化尺度因子列表	list	float32

INT4激活量化数据生成操作的伪代码见表295。

表 295 INT4 激活量化数据生成操作伪代码

INT4激活量化数据生成操作	描述符
def int4_quantization_activation(X,offset_act,dual_list,cali_act, bit_w=4,n_points): {	
sf_act_list=[]	
for l in range(cali_act.layers): {	
x=cali_act[l]	
x_quantized,sf=cal_sf(x,offset_act,dual_flag=False, bit_w=4,n_points)	
x_recon=int4_recon_quantized_activation(x_quantized,sf,offset_act_list=None)	
if dual_list[i]==True: {	
residual_x=x-x_recon	
residual_quantized,residual_sf=cal_sf(residual_x,offset_act,\ dual_flag=True, bit_w=4,n_points)	
x_quantized.append(residual_quantized)	
sf.append(residual_sf)	
}	

表 295 INT4 激活量化数据生成操作伪代码（续）

INT4激活量化数据生成操作	描述符
sf_act_list.append(sf)	
}	
X_quantized=[]	
for i,x in enumerate(X): {	
x_quantized=int4_quantize_activation(x,sf_act_list[i],offset_act_list\	
=None,bit_w=4,x_quantized=[])	
X_quantized.append(x_quantized)	
}	
return X_quantized,sf_act_list	
}	

其中，int4_recon_quantized_activation()操作INT4激活反量化章节，int4_quantize_activation()操作见INT4激活量化章节，cal_sf()操作的伪代码见表296。

表 296 cal_sf()操作伪代码

cal_sf()操作	描述符
def cal_sf(x, offset_act, dual_flag, bit_w=4, n_points): {	
if offest_act: {	
max_range_x=max(x)-min(x)	
range_min=0	
range_max=2**bit_w-1	
beta_vec = linspace(0.1 *max_range_x/range_max, max_range_x/range_max, n_points)	
}	
else: {	
max_range_x=max(abs(x))	
range_max = 2. ** (bit_w - 1) - 1	
range_min = -2. ** (bit_w - 1)	
if dual_flag:	
beta_vec = linspace(-max_range_x/range_max, max_range_x/range_max,\ 2 * n_points)	
else:	
beta_vec = linspace(-max_range_x/range_max, max_range_x/range_max,\ 2 * n_points)	
}	
res_vec=[[]for i in range(beta_vec)]	
for i, beta in enumerate(beta_vec):	

表 296 cal_sf()操作伪代码（续）

cal_sf()操作	描述符
res_vec[i]=(clip(round(x/beta),range_min,range_max)-x)**2	
scaling_factor=beta_vec[argmin(res_vec)]	
x_quantized=clip(round(x/scaling_factor),range_min,range_max)\	
return x_quantized, scaling_factor	
}	

9.4.2.2 训练截断值量化训练数据生成

将截断值进行参数化，首先需要对参数进行初始化，为了能够使参数收敛的更快，需要基于预训练好的模型和给定的训练数据，按照粒度需求，对需要量化层的最大值进行统计，经过一段时间的训练，可以得到每个粒度的截断值的训练值，该值可直接用于推理过程中，如前文所述。该算法依赖于预训练模型和训练数据，可以并行处理所有需要量化的卷积层和全连接层。具体算法流程图如下：

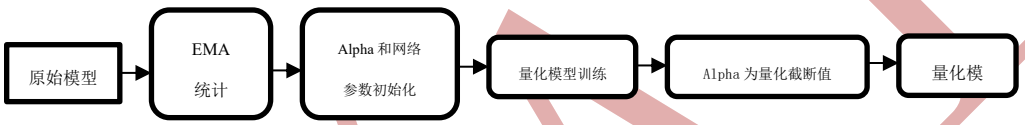


图 11 算法流程图

主要流程如下：

a) 给定一个预训练好的模型，通过 EMA 的统计方式，根据粒度需求，对网络中需要量化的层的值的范围进行统计，得到训练截断参数 Alpha 的初始值；具体的统计方法见式（48）：

$$\alpha_t^l = \beta \cdot \alpha_{t-1}^l + (1-\beta) \cdot mean(max(X_1^l), max(X_2^l), \dots, max\,(X_m^l)) \quad \text{.....(48)}$$

式中，

t——mini-batch 迭代次数

l——指的是粒度所指示的量化 Tensor

X ——是卷积层输出

α ——为训练截断参数

β ——为平滑系数

b) 利用步骤 a）中的 Alpha 初始值，给 Alpha 赋值；

c) 给定训练 Epoch 数、量化 bit 数、Alpha 和预训练模型，进行训练；

d) 重复步骤 c），直到网络的 Loss 不再变化（收敛），执行步骤 e）

e) 获得 Alpha，可作为截断值，并根据公式 $X\_Scale = \frac{2^{n-1}-1}{Alpha}$ ，生成量化 Scale 值。

训练截断值量化的具体数据定义见表297。

表 297 训练截断值量化数据生成数据定义表

运算操作	描述	字段	关键字	定义	数据类型	数据格式
Alpha_quantization 训练截断值量化	将静态统计量化截断值转换为可训练的量化截断值	Input	X	粒度索引值对应待量化的张量	List [tensor]	NS
			M	预训练模型	List [tensor]	NS
			Gamma	指数移动平滑统计常量	Value	Float
			Epoch	训练 epoch 数	Value	Int
			Granularity	量化粒度	Value	int
			N	量化比特数	Value	int
		Output	X_Scale	量化尺度因子	List [tensor]	NS

训练截断值量化数据生成伪代码见表298。

表 298 训练截断值量化数据生成伪代码

训练截断值量化数据生成	描述符
def Alpha_quantization (M, X, Epoch, Gamma, Alpha, Granularity, N) :{	
for epoch in range(epochs):	
if (epoch==0):	
Alpha = EMA(X,Gamma)	EMA=公式（1）
else:	
M.forward(Alpha,X, Granularity ,N)	M前向过程中，插入基于Alpha的量化与反量化操作
X_Scale =(2**(n-1) -1)/Alpha	
return X_Scale	
}	

9.5 剪枝

该部分主要描述剪枝当中掩码生成，即，给定神经网络结构权重，如何输出对应新的神经网络结构权重以及掩码张量。剪枝数据生成见表 299。

表 299 剪枝数据生成

操作	描述	字段	关键字	定义	数据类型	数据格式
掩码生成	对于给定的神经网络结构权重输出对应的掩码	Input	X	输入权值张量	list[tensor]	NS
			P_Method	剪枝方法	value	INT
			granularity	粒度	Value	Int
		Output	Y	输出掩码张量	list[tensor]	NS

剪枝粒度定义见表300。

表 300 剪枝粒度定义

剪枝粒度	
标号	类型
1	UNDEFINED
2	行剪枝/卷积核个数剪枝
3	列剪枝/卷积核形状剪枝
4	自定义剪枝粒度

P_Method 定义见表 301。

表 301 P_Method 定义

P_Method	
标号	类型
1	UNDEFINED
2	ALUM剪枝
3	增量正则化剪枝

9.5.1 稀疏数据生成方法

结构化稀疏过程如下：

输入变量为：X 四维张量，表示输入网络的训练集合；Y 二维张量，要求 X.tensorshape[0] == Y.tensorshape[0]；graph 网络模型定义图；W 权重张量list;lambda对应每一层稀疏化超参数；eplson 对应的稀疏化要求 epslon;

对于输入数据X，Y 对应的网络结构和权重W，我们希望使W实现尽可能高的行稀疏化程度，并且在对应数据集的损失最小。特别的，我们引入两个中间变量F，K，F与K的尺寸保持一致，我们希望对以下损失函数，对于W，F，K求解再输入数据为X,Y的情况下的最优值：

$$\mathcal{L}_{W,F,K}(X,Y)=\mathcal{L}_{X,Y}(W)+\sum_{l=1}^{L}\lambda||F^{l}||_{2,1}+\sum_{l=1}^{L}trace\left(K^{l}(W^{l}-F^{l})\right)+\sum_{l=1}^{L}\frac{1}{2}||W^{l}-F^{l}||_{F}^{2} \quad \ldots(49)$$

式中：

$\mathcal{L}_{X,Y}(W)$ ——为对应网络在数据X Y 上的softmax损失对于每一层

W^{l}——对第l层的网络权重

F^{l},K^{l}——对第l层的中间变量

X,Y——输入数据

我们分别优化对应的W^{l},F^{l},K^{l},

a) 分别初始化K=0,F=W

b) 在输入数据X 和Y 中，采样出一个batch x,y

c) 对于网络权重W,删除与W无关的项之后，可以固定F，K，引入中间矩阵$T^l = F^l - \frac{1}{2}K^l$直接利用$\mathcal{L}_{x,y}(W^l) + \frac{1}{2}||W^l - T^l||_F^2$,输入compute_loss计算损失函数，并且利用compute_grade求得W^l 的梯度，进行梯度更新：

d) 对于矩阵F,固定W以及K，引入中间矩阵$U^l = W^l + \frac{1}{2}K^l$,我们直接利用式（50），对F的每一行进行更新:

e) $F_i^{l*} = U_i^l\left(\max\left\{||U_i^l||_2 - \lambda, 0\right\}\right)/||U_i^l||_2$ .....(50)

式中，

i——为F矩阵的行索引即F_i代表对应F矩阵第i行。

F^l——对第l层的中间变量

F^l——对第l层的中间变量

U_i^l——对第l层的计算中间矩阵

f) 计算出新的F以及W之后，K的更新方式为：$K^{l\{k+1\}} = K^{l\{k\}} + W^l - F^l$

g) 当$\left\|\mathbf{W}^{l\{k+1\}} - \mathbf{F}^{l\{k+1\}}\right\|_F \leq \varepsilon$，或者 $\left\|\mathbf{F}^{l\{k+1\}} - \mathbf{F}^{l\{k\}}\right\|_F \leq \varepsilon$，l=l+1,返回步骤 a)，否则返回 b）。

h) 当所有层都遍历一次之后，跳出循环，算法结束。

操作定义见表302。

表 302 结构化剪枝数据生成

运算操作	描述	字段	关键字	定义	类型	数据格式
结构稀疏数据生成	给定网络结构训练集合生成结构稀疏的数据以及掩码	Input	X	输入数据	tensor	NS
			Y	输入标签	vector<int>	NS
			graph	网络结构	graph	
			optimizer	优化器	optimizer	
			W	权重张量	list[tensor]	NS
			lambda	层超参数	vector<int>	NS
			epslon	稀疏程度	value	FLOAT
		Output	W	权重张量	list[tensor]	NS
			F	掩码	list[vector]	NS

伪代码定义见表303。

表 303 行稀疏数据生成

行稀疏数据生成	描述符
Row_sparse(X,Y,graph,W,optimizer,lambda,epslon){	
List[tensor] K=0;	
List[tensor] F = W;	
for (i=0;i<W.shape[0];i++){	
For (x,y in X,Y){	
T = F[i] – 0.5*K[i];	
y_pred = graph(W,x);	
Loss = compute_loss(y,y_pred,loss=cross_entropy); # no cross_entropy?	
Loss = Loss + 0.5* compute_loss(W[i][:],T[i][:],loss=mean_squared_error);	
W_g = compute_gradient(Loss, W[i], optimizer) ;# SGD中已经定义了lr 下一步是否需要？	
W[i] = W[i] + W_g;	
U = W[i]+0.5K[i];	
F_old[i] = F[i]	
}	
For (j=0;j<W[i].shape[0];j++){	
F[i][j] = U[j]*(max(sqrt(sum(U[j][:]**2)-lambda,0)))/ sqrt(sum(U[j][:]**2));	
}	
K[i] = K[i]+W[i]-F[i];	
If (sqrt(sum((K[i]-F[i])**2))<epslon or sqrt(sum((F[i]-F_old)**2))){	
Break;	
}	
}	
For (x,y in X,Y){# finetune	
y_pred = graph(W.*F,x);	
Loss = compute_loss(y,y_pred,loss=cross_entropy);	
W_g = compute_gradient(Loss, W[i], optimizer);	
W = W + W_g.*F;	
}	
}	

9.5.2 增量正则化剪枝

定义：

W_2d：模型所有参数层的权值矩阵，以(N, C*H*W)的二维形式存储或使用；

B：模型所有参数层的偏置矩阵；

L_sparsity：当前层的目前稀疏率；

Sparsity：模型所有参数层所要达到的目标稀疏率；

列参数组：权值矩阵的行数为N，列数为C*H*W，按照列数可划分为C*H*W个列参数组；

Target_reg:各列参数组的正则阈值，当某个权重列的正则化因子达到正则阈值时，相应的权重将被删除；

Reg_func：正则增量函数，用于为不同的权重列分配对应的正则增量，其中，重要性小的权重列将被分配更大的正则增量，重要性大的权重列将被分配更小的正则增量；

Reg_val：经过Reg_func后每个列参数组的新的正则化值；

Offer_mask：权值删除操作，用于将权值张量中待删除位置权重置零；

W_sparse：模型所有参数层的稀疏权值张量；

B_sparse：模型所有参数层的稀疏偏置张量。

定义增量正则化方法所需的具体操作定义见表304、表305和表306。其中，增量正规化数据见表304。

表 304 增量正则化数据生成

运算操作	描述	字段	关键字	定义	类型	数据格式
ID_Reg	增量正则化稀疏，依次分析网络各层参数，生成层内参数重要性顺序，输出剪枝掩码	Input	W_2d	权重矩阵	list[tensor]	NS
			B	偏置向量	list[vector<float>]	NS
			Sparsity	各层目标稀疏率	list[float]	NS
			Target_reg	正则阈值	value	FLOAT
			Reg_increase	正则化增量	list[vector<float>]	NS
			Iter	正则增量调整间隔	value	INT
			X	输入数据	tensor	NS
			Y	输入标签	vector<int>	NS
			dtype	数据类型	ValueType	
			granularity	剪枝粒度	value	INT
		Output	W_2d_mask	掩码矩阵	list[tensor]	NS
			B_mask	掩码向量	list[vector<float>]	NS

正则增量函数定义见表305。

表 305 正则增量函数定义

运算操作	描述	字段	关键字	定义	类型	数据格式
Reg_func	正则增量函数	Input	W_2d	权重矩阵	tensor	NS
		Output	Reg_increase	正则化增量	vector<float>	NS

参数置零操作见表306。

表 306 参数置零操作

运算操作	描述	字段	关键字	定义	类型	数据格式
Offer_mask	参数置零操作，给定Mask，将参数中对应位置的值置零	Input	A	参数矩阵	tensor	NS
			A_mask	掩码矩阵	tensor	NS
			B	参数向量	vector<float>	NS
			B_mask	掩码向量	vector<float>	NS
		Output	A_sparse	稀疏参数矩阵	tensor	NS
			B_sparse	稀疏参数向量	vector<float>	NS

增量正则化剪枝只支持granularity为3的模式，对所有输入权值张量（假设有n个）按照逐层的顺序进行列剪枝，增量正则化剪枝操作ID_Reg的过程如下：

a) 对每一层的权值张量进行 reshape(N,C*H*W)的展开操作，获取二维权值矩阵，其中矩阵的行数为 N，列数为 C*H*W，按照列的数量划分 C*H*W 个列参数组；
b) 使用增量正则化稀疏操作，配置相关超参数 Sparsity、Target_Reg、Reg_func，从模型的每一层获取参数 W_2d 和 B，并调用操作 ID_Reg 在矩阵的列维度上进行剪枝操作，在输出端获得掩码 W_2d_mask、B_mask，以及新的正则化值 Reg_val；
c) 对每一层的权值矩阵进行权值删除 Offer_mask 操作，得到稀疏权值 W_sparse、B_sparse，最后通过 reshape(N,C,H,W)的变换操作还原为训练框架所需的张量形式；
d) 使用数据集对模型的参数进行训练更新。

增量正则化剪枝的算法流程如下：

第1步：准备预训练好的model，网络结构，训练配置文件，数据集等，并设置目标稀疏率与剪枝超参数Sparsity、Target_Reg、Reg_func；

第2步：训练过程初始化，模型中权重的参数初始化为预训练model中的值；

第3步：逐层检测该层的当前稀疏率，判断当前稀疏率是否达到给定的目标稀疏率，若均达到，则跳到第7步，若未达到，则执行下一步；

第4.1步：使用梯度下降法迭代训练网络M个Iter；

第4.2步：对于网络的各个层，使用L1-norm对各层权重的列进行排序，对前M个iter的排序进行平均，并对平均后的排名再次排序得到该层每个列参数组的重要性排名；

第4.3步：用第4.2步得到的列参数组重要性排名作为正则增量函数的输入，将函数的输出值作为各个列参数组的正则化增量，并使更新到各个列参数组的正则化值上得到Reg_val；

第4.4步：逐列判断更新后的该列参数组的正则化值是否大于正则阈值，若大于则将掩码矩阵中对应的列置零，若小于则不处理；

第4.5步：使用权值删除操作Offer_mask，将掩码矩阵与权重矩阵相乘；

第5步：将新的正则化值加入loss函数；

第6步：反向传播中，使用compute_gradient操作逐层计算梯度，将掩码矩阵与梯度相乘屏蔽剪枝的列参数组的梯度，并对权重进行更新，然后回到第4步；

第7步：结束剪枝，保存model，并对得到的model进行重训练；

第8步：输出重训练中准确率最高的模型。

剪枝流程伪代码见表307。

表 307 增量正则化剪枝伪代码

增量正则化剪枝	描述符
[1] ID_Reg(W_2d, B, Sparsity, Reg_func, Iter, Target_reg, X, Y, dtype, granularity): {	
Initialize W_2d_mask, B_mask with ones for each layer	
while (1) {	
for layer = 1 to NumLayer {	
Update L_sparsity	
if (L_sparsity < Sparsity[layer])	
continue	
else {	
if (all L_sparse >= Sparsity) {	
save model	
break while	
}	
else	
continue	
}	
Get W_2d[layer], W_2d_mask[layer], B[layer], B_mask[layer]	
W_sparse, B_sparse = Offer_mask(W_2d[layer], W_2d_mask[layer], B[layer], B_mask[layer])	
forward(W_sparse, B_sparse)	
}	
if (iter % Iter == 0) {	
for layer = 1 to NumLayer{	
Get W_2d[layer], W_2d_mask[layer] from layer	
for col = 1 to W_2d.cols {	
Reg_sort = sort(abs(W_2d[:, col], iter)	
Reg_increase = Reg_func(Reg_sort)	
Use Reg_increase to update regularizer and get Reg_val	
if (Reg_val >= Target_reg)	
Update W_2d_mask[layer]	
loss += Reg_val	
}	
}	
grad = compute_gradient(loss, val_list, optimizer, graph)	
for layer = NumLayer to 1{	
Calculate W_2d_grad[layer], B_grad[layer] with grad	

表 307 增量正则化剪枝伪代码（续）

增量正则化剪枝	描述符
W_grad_sparse, B_grad_sparse = Offer_mask(W_2d_grad[layer], W_2d_mask[layer], B_grad[layer], B_mask[layer])	
W_2d[layer] = W_2d[layer] – learning_rate* W_grad_sparse	
B[layer] = B[layer] – learning_rate * B_grad_sparse	
}	
}	
iter = iter + 1	
}	
W_2d_mask_list = list[], B_mask_list = list[]	
for layer = 1 to NumLayer{	
Get W_2d_mask[layer], B_mask[layer]	
W_2d_mask_list.append(W_2d_mask[layer])	
B_mask_list.append(B_mask[layer])	
}	
return W_2d_mask_list, B_mask_list	
}	
[2] Retrain(X, Y, graph) {	
for(x,y in X,Y) {	
W_sparse, B_sparse = Offer_mask(W_2d, W_2d_mask, B, B_mask)	
y_pred = graph(W_sparse, B_sparse , x)	
loss = compute_loss(y,y_pred,loss=cross_entropy)	
grad = compute_gradient(loss, W_sparse, B_sparse , optimizer)	
Calculate W_2d_grad, B_grad with grad	
W_grad_sparse, B_grad_sparse=Offer_mask(W_2d_grad, W_2d_mask, B_grad, B_mask)	
W_2d[layer] = W_2d[layer] – learning_rate* W_grad_sparse	
B[layer] = B[layer] – learning_rate * B_grad_sparse	
}	
}	
[3] Output retrained model	

9.6 结构化矩阵

9.6.1 结构化矩阵数据生成

结构化矩阵数据生成方法，基于训练数据x和y，通过网络图Graph和优化去Optimizer，对使用结构化矩阵进行压缩的网络层权重张量W_ori进行更新，获得更新后的结构化权重张量W_structural。具体定义见表308。

表 308 结构化矩阵权重更新数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
结构化矩阵权重更新	训练过程中，使用结构化矩阵技术压缩权重的方法和操作	Input	x	训练数据输入	data	NS
			y	训练数据真值	label	NS
			Graph	网络图	Function	/
			Optimizer	优化器	function	/
			W_ori	原权重张量	tensor	NS
			B_ori	原偏置向量	vector	NS
		Output	W_structural	更新后的权重张量	tensor	NS

9.6.2 带符号向量的分块循环矩阵的待压缩数据生成方法

使用符号向量和分块循环矩阵技术对模型进行压缩时，需要通过在训练过程中约束模型中的对应层权重结构，并获得对应的权重数据。对于使用符号向量和分块循环矩阵技术的整体数据训练方法，其训练步骤如下：

a) 确定使用符号向量和分块循环矩阵技术的层位置；

b) 对给定网络输入，逐层依次正向并计算最终loss，特别的：

1) 若当前层不使用符号向量和分块循环矩阵技术，正常正向推理；

2) 若当前层使用符号向量和分块循环矩阵技术，调用node_forward_sign_circ()进行正向推理：

- 节点输入向量x与扰动符号向量sign_vector相乘得到扰动后的输入向量sign_x；
- 解压当前层压缩后的分块循环矩阵权重w_circ；
- 扰动后的输入向量和解压后的权重矩阵相乘，返回该输出；

c) 根据正向推理计算得到的loss，计算反向梯度更新，特别的：

1) 若当前层不使用符号向量和分块循环矩阵技术，正常反向梯度更新；

2) 若当前层使用符号向量和分块循环矩阵技术，调用node_backward_sign_circ_for_w_circ()进行反向梯度更新：

- 根据当前node对应loss计算当前展开后的权重矩阵对应的梯度矩阵；
- 累加分块循环矩阵中相同元素对应的梯度并计算平均值；
- 根据上述梯度平均值，计算分块矩阵权重值并更新；

d) 重复步骤b~c，训练网络，更新参数至网络收敛或达到指定条件。

使用符号向量和分块循环矩阵技术的权重更新的数据定义见表309。

表 309 使用符号向量和分块循环矩阵技术的权重更新数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
使用符号向量和分块循环矩阵技术的权重更新	使用符号向量和分块循环矩阵技术时，训练过程中的权重更新的方法和操作	Input	x	训练数据输入	data	NS
			y	训练数据真值	label	NS
			Graph	网络图	Function	/
			Optimizer	优化器	function	/
			W_ori	原权重张量	tensor	NS
			B_ori	原偏置向量	vector	NS
		Attributes	layer_indicator	使用符号向量和分块循环矩阵技术的层索引	list	Int
			sign_vector	符号向量列表	list [vector]	NS
			circ_size	分块循环矩阵尺寸列表	list	Int
		Output	W_structural	更新后的权重张量	tensor	NS

使用符号向量和分块循环矩阵技术的权重更新的伪代码见表310。

表 310 使用符号向量和分块循环矩阵技术的权重更新的伪代码

使用符号向量和分块循环矩阵技术的网络权重更新	描述符
def Block_circulant_weight_update(x, y, Graph, Optimizer, W_ori, layer_indicator): {	
{	
layer_in = x	
for layer in range(1,NumLayer): {	
if layer_indicator[layer] == False: {	
layer_out = forward(layer_in, W_ori[layer], B_ori[layer])	
}	
else: {	
layer_out = node_forward_sign_circ(layer_in, W_ori[layer], \ sign_vector[layer], circ_size[layer], B_ori[layer])	
}	
layer_in = layer_out	
}	
Result = layer_out	
Loss = compute_loss(Result, y)	

表 310 使用符号向量和分块循环矩阵技术的权重更新的伪代码（续）

使用符号向量和分块循环矩阵技术的网络权重更新	描述符
for layer in range(NumLayer ,1,-1): {	
if layer_indicator[layer] == False: {	
W_ori[layer], B_ori[layer] = backward(Loss[layer], W_ori[layer], B_ori[layer])	
}	
else: {	
W_structural[layer] = node_backward_sign_circ_for_w_circ(Loss[layer], \ W_ori[layer], circ_size[layer], Optimizer)	
}	
}	
return W_structural, W_ori, B_ori	
}	

对应的正向操作node_forward_sign_circ()见表311。

表 311 node 正向推理数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
node正向推理 node_forward_sign_circ	训练过程中，使用符号向量和分块循环矩阵技术的node正向推理	Input	x	当前node输入	vector	NS
			w_circ	压缩后的node权重	matrix	NS
			sign_vector	当前node对应的扰动符号向量	vector	NS
			circ_size	当前node分块循环矩阵尺寸	value	Int
			B(optional)	偏置向量	vector	NS
		Output	output	当前node的输出	vector	NS

对应的反向操作node_backward_sign_circ_for_w_circ()见表312。

表 312 node 反向梯度更新数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
node反向梯度更新 node_backward_sign_circ_for_w_circ	训练过程中，使用符号向量和分块循环矩阵技术的node的反向梯度更新	Input	loss	当前node对应loss	list[tensor]	NS
			w_circ	当前node对应压缩后的分块循环矩阵	matrix	NS

表 312 node 反向梯度更新数据定义（续）

运算操作	描述	字段	关键字	定义	数据类型	数据格式
node反向梯度更新 node_backward_sign_circ_for_w_circ	训练过程中，使用符号向量和分块循环矩阵技术的node的反向梯度更新	Input	circ_size	当前node对应分块循环矩阵尺寸	value	Int
			Optimizer	优化器	function	
		Output	w_circ	更新后压缩权重矩阵	matrix	NS

对使用此技术的当前层，正向流程伪代码见表313。

表 313 node 正向流程伪代码

使用符号向量和分块循环矩阵技术的当前node层正向流程	描述符
def node_forward_sign_circ(x, w_circ, sign_vector, circ_size, B): {	
sign_x = mul(x, sign_vector)	
output = mul(sign_x, decompress_W_circ_to_weight(w_circ, circ_size))	
if B: {	
output = output + B	
}	
return output	
}	

对于使用该技术的当前层，反向流程伪代码见表314。

表 314 node 反向流程伪代码

使用符号向量和分块循环矩阵技术的当前 node 反向流程	描述符
def node_backward_sign_circ_for_w_circ(loss, w_circ, circ_size, Optimizer): {	
gradient_weight = compute_gradient(loss, Optimizer)	
height_ori, width_ori = gradient_weight.shape()	
for i in range(0, height_ori, circ_size)：	
for j in range(0, width_ori, circ_size): {	
for k in range(circ_size): {	
gradient_sum = 0	
for l in range(circ_size): {	
if(k+l > circ_size):	
shift = k+l-circ_size	
else:	
shift = k+l	

表 314 node 反向流程伪代码（续）

使用符号向量和分块循环矩阵技术的当前 node 反向流程	描述符
gradient_sum += gradient_weight[i+l][j+shift]	
}	
w_circ[i/circ_size][j] = w_circ[i/circ_size][j] – gradient_sum/circ_size	
}	
}	
return w_circ	
}	

9.6.3 低秩组稀疏分解的结构化矩阵的权重数据生成方法

9.6.3.1 概述

使用低秩组稀疏分解的结构化矩阵技术对模型进行压缩时，首先需要计算得到超参数，通过张量分解得到一个初始化的权重，之后需要进行finetune对权重信息进行更新，获得更精准的权重信息。

9.6.3.2 超参数 R1、R2、groups 和 core_size 的确定方法

对于一个4维的卷积层，其参数量为$C_{out} \times C_{in} \times K \times K$，其中$C_{out}$为输出数据的维度大小，$C_{in}$为输入数据的维度大小，$K$为卷积核的大小。进行低秩组稀疏分解后，该卷积层被分为三个小尺寸的卷积层，第一个卷积核的大小为$R1 \times C_{in} \times 1 \times 1$，第二个卷积层的大小为$R2 \times r1 \times K \times K$（$r1 = R1/groups$，对第二个卷积层的输入通道设置了groups个分组），第三个卷积核的大小为$C_{out} \times R2 \times 1 \times 1$。

通过当前层的输入通道数量、输出通道数量和压缩率$ratio$的大小，根据输入通道和输出通道维度，$groups$一般选择1~4，$R1$和$R2$的大小一般正比于C_{in}和C_{out}，并且满足公式52，从而确定超参数$group$、$R1$和$R2$:

$$C_{out} \times C_{in} \times K \times K \times ratio = R1 \times C_{in} \times 1 \times 1 + R2 \times r1 \times K \times K + C_{out} \times R2 \times 1 \times 1 \ldots (51)$$

式中：

C_{in}——输入通道数；

C_{out}——输出通道数；

K——卷积核高、宽；

$ratio$——压缩率；

$R1$——对应于输入通道的秩；

$R2$——对应于输出通道的秩。

同样的，对与网络中的全连接层和1×1卷积层，设其大小为$W \times H$，则通过压缩后得到两个近似张量，第一个张量的大小为$W \times core_size$，第二个张量的大小为$core_size \times H$，设置压缩率$ratio$，通过求解公式52可以得到超参数$core_size$:

$$W \times H \times ratio = W \times core\_size + core\_size \times H \ldots\ldots\ldots\ldots\ldots\ldots (52)$$

式中：

W——输入通道（对于1×1卷积层）或神经元（对于全连接层）个数；

H——输出通道（对于1×1卷积层）或神经元（对于全连接层）个数；

$ratio$——压缩率；

$core\_size$——矩阵分解秩。

9.6.3.3 低秩组稀疏分解的结构化矩阵的权重数据生成流程

其训练过程如下：

a) 通过人工选择和计算，确定超参数；
b) 准备预训练好的模型参数、训练所需的参数、数据集等；
c) 对网络中的卷积层权重参数和全连接层的参数进行提取，并对提取的这些参数进行分解压缩；
d) 将压缩分解后的模型参数和预训练好的其他层的参数对压缩后的网络模型进行赋值；
e) 根据epoch调整学习率，进行正向推理并得到loss，计算梯度并进行反向传播，更新网络的权重参数；
f) 重复 e) 训练网络直到设置的训练轮数epoch。

低秩组稀疏分解的结构化矩阵的权重数据生成方法中的数据数据定义见表315。

表 315 低秩组稀疏分解的结构化矩阵技术的权重更新数据定义

运算操作	描述	字段	关键字	定义	数据类型	数据格式
低秩组稀疏分解的结构化矩阵技术的权重更新	低秩组稀疏分解的结构化矩阵技术时，训练过程中的权重更新的方法和操作	Input	x	训练数据输入	Data	NS
			y	训练数据真值	Label	NS
			Graph	压缩后的网络图	Function	/
			Optimizer	优化器	Function	/
			epoch	训练的轮数	value	Int
			Pretrained_weight	预训练模型权重	Tensor	NS
			R1	每个卷积层的输入channel的秩	List	Int
			R2	每个卷积层的输出channel的秩	List	Int
			groups	每个卷积层的分组数	List	Int
			core_size	全连接层中间输出的数量	List	Int
		Output	BTD_weight	更新后的权重张量	Tensor	NS

该操作对应伪码实现见 316。

表 316 低秩组稀疏分解的结构化矩阵的权重数据生成流程伪代码

低秩组稀疏分解的结构化矩阵的权重数据生成流程	描述符
def BTD_update_weight(Pretrained_weight, Graph, R1, R2, groups, core_size, batch_size, epoch): {	
svd_idx = 0	
hosvd_idx = 0	
Pretrained_layers = Pretrained_weight.keys	
BTD_weight = {}	
for layer_idx in range(len(Pretrained_layer)): {	
key = Pretrained_layers[layer_idx]	
weight = Pretrained_weight[key].weight	
if (Pretrained_weight[key].type == "FC")): {	
w1, w2 = SVD_process(weight, core_size[svd_idx])	
svd_idx = svd_idx+1	
BTD_weight[key+"w1"].weight = w1	
BTD_weight[key+"w2"].weight = w2	
}	
elif (Pretrained_weight[key].type == "Conv") and shape(weight)[3] == 1: {	
w1, w2 = SVD_process(weight, core_size[svd_idx])	
svd_idx = svd_idx + 1	
BTD_weight[key+"w1"].weight = w1	
BTD_weight[key+"w2"].weight = w2	
}	
elif (Pretrained_weight[key].type == "Conv"): {	
w1, w2, w3 = BTD_process(weight, R1[hosvd_idx], R2[hosvd_idx], groups[hosvd_idx])	
hosvd_idx = hosvd_idx + 1	
BTD_weight[key+"w1"].weight = w1	
BTD_weight[key+"w2"].weight = w2	
BTD_weight[key+"w3"].weight = w3	
}	
else: {	
BTD_weight[key].weight = weight	
}	
}	
for epoch_idx in range(epoch): {	
local_lr = adjust_learning_rate(optimizer, epoch_idx)	调整学习率
for ite in range(len(x)/batch_size): {	
batch_x = x[ite*batch_size:ite*batch_size+batch_size]	
batch_y = y[ite*batch_size:ite*batch_size+batch_size]	
out = Graph(batch_x, BTD_weight)	
loss = compute_loss(out, batch_y)	

表 316 低秩组稀疏分解的结构化矩阵的权重数据生成流程伪代码（续）

低秩组稀疏分解的结构化矩阵的权重数据生成流程	描述符
gradient = compute_gradient(BTD_weight, loss, optimizer)	
update_weight(BTD_weight, gradient)	
}	
}	
reture BTD_weight	
}	

10 编解码表示

10.1 神经网络模型权重压缩位流的语法和语义

编解码表示方法的使用，应遵循以下原则：

a) 计算系统应按实际应用要求，选用编解码方法；

b) 模型权重压缩位流操作的实现，应依据应用需要，做出调整，包含但不限于：

1) 操作函数的命名及参数；

2) 流头及层次的组成；

3) 位流解析、解码涉及的数据结构及其初始值。

四维权重张量（[R][S][C][K]）可以重整为三维（[RS][C][K]）张量。这里，[RS]是核的形状，[C]是输入特征集的数目，[K]是输出特征集的数目。三维张量可以拆分为三维编码树(CTU3D)。CTU3D更进一步递归拆分为三维编码单元(CU3D)。CTU3D最大尺寸为RSx64x64，最小CU3D尺寸为RSx8x8。

八叉树是一种树型数据结构，其中每个内部节点正好有八个子节点。三维八叉树通过沿z、y、x轴递归地将三维张量分为八个子树。三维编码树有四种：三维同一树(3D-Unitree)，三维八叉树(3D-Octree)，三维标签树(3D-Tagtree)和三维同一及标签树(3D-UniTagtree)。图12是三维编码树有三个深度的一个例子。

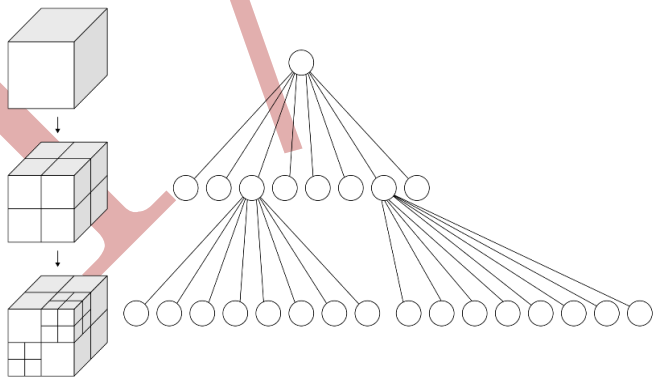


图 12 三维编码树有三个深度的一个例子

三维同一树:非最大深度的节点值0表示其所有子节点都有相同的绝对值，否则，其节点值为1。最大深度节点的值0表示与该节点共享同一父节点的所有节点都具有相同的码本索引或系数绝对值，否则其节点值为1。

三维八叉树: 最大深度的节点值0表示其所有子节点的值都为0，否则，其节点值为1。最大深度节点的值0表示码本索引或系数值为0，否则，其节点值为1。

三维标签树:最大深度节点的值为码本索引值或系数值。非最大深度节点的值为其子节点的绝对值的最小值。

三维同一及标签树:在这种组合方法中，独立地对2D[CK]矩阵的序列进行编码，使用2D四叉树结构为每个2D矩阵构造同一树和标签树。对于任意一个节点，如果其对应的同一值是0，则使用同一树的编码方式进行编码，否则使用标签树的编码方式进行编码。

以下函数用于语法，语义和解码描述。

size(): 是容器的成员函数，回容器中实际元素的个数。

resize(num): 是容器的成员函数，新设置该容器的大小。

begin() : 是容器的成员函数，返回指向容器第一个元素的迭代器。

end() : 是容器的成员函数，返回指向容器最后一个元素的迭代器。

erase(pos): 是容器的成员函数，除pos位置的数据。

insert(pos，elem): 是容器的成员函数，pos位置插入一个elem拷贝。

push_back(elem): 是容器的成员函数，pos位置插入一个elem拷贝。

create_tensor3d(dst, min_cu3d_height, min_cu3d_width, height, width)

初始化函数定义见表317。

表 317 **create_tensor3d** 函数的定义

函数定义
create_tensor3d(dst, min_cu3d_height, min_cu3d_width, height, width){
depth=1
depth_h = ceil(height / min_cu3d_height)
depth_w = ceil(width / min_cu3d_width)
vdh.push_back(depth_h)
vdw.push_back(depth_w)
while (depth_h != 1 || depth_w != 1) {
depth_h = (depth_h + 1) >> 1
depth_w = (depth_w + 1) >> 1
vdh.push_back(depth_h)
vdw.push_back(depth_w)
++depth
}
dst.resize(depth)
for (d = depth - 1; d >= 0; --d) {
dst[d].resize(vdh[depth - 1 - d])
for (dh = 0; dh < vdh[depth - 1 - d]; ++dh)
dst[d][dh].resize(vdw[depth - 1 - d])
}
}

create_tensor6d(dst, min_cu3d_height, min_cu3d_width, height, width, zdepth)
初始化函数定义见表318。

表 318 **create_tensor6d** 函数的定义

函数定义
create_tensor6d(dst, min_cu3d_height, min_cu3d_width, height, width, zdepth){
depth=1
depth_h = ceil(height / min_cu3d_height)
depth_w = ceil(width / min_cu3d_width)
vdh.push_back(depth_h)
vdw.push_back(depth_w)
while (depth_h != 1 || depth_w != 1) {
depth_h = (depth_h + 1) >> 1
depth_w = (depth_w + 1) >> 1
vdh.push_back(depth_h)
vdw.push_back(depth_w)
++depth
}
dst.resize(depth)
for (d = depth - 1; d >= 0; --d) {
dst[d].resize(vdh[depth - 1 - d])
for (dh = 0; dh < vdh[depth - 1 - d]; ++dh) {
dst[d][dh].resize(vdw[depth - 1 - d])
for (dw = 0; dw < vdw[depth - 1 - d]; ++dw) {
dst[d][dh][dw].resize(zdepth)
for (zz = 0; zz < zdepth; ++zz) {
minh = min_cu3d_height << (depth - 1 - d)
minw = min_cu3d_width << (depth - 1 - d)
mh = min(minh, height - dh * minh)
mw = min(minw, width - dw * minw)
dst[d][dh][dw][zz].resize(mh)
for (hh = 0; hh < mh; ++hh) {
dst[d][dh][dw][zz][hh].resize(mw)
}
}
}
}
}
}

create_tree3d(dst, min_cu3d_height, min_cu3d_width, height, width, zdepth, bmix)

初始化函数定义见表319。

表 319 create_tree3d 函数的定义

函数定义
create_tree3d(dst, min_cu3d_height, min_cu3d_width, height, width, zdepth, bmix){
depth=1
depth_h = ceil(height / min_cu3d_height)
depth_w = ceil(width / min_cu3d_width)
vdh.push_back(depth_h)
vdw.push_back(depth_w)
while (depth_h != 1 || depth_w != 1) {
depth_h = (depth_h + 1) >> 1
depth_w = (depth_w + 1) >> 1
vdh.push_back(depth_h)
vdw.push_back(depth_w)
++depth
}
dst.resize(depth)
for (d = depth - 1; d >= 0; --d) {
dst[d].resize(vdh[depth - 1 - d])
for (dh = 0; dh < vdh[depth - 1 - d]; ++dh) {
dst[d][dh].resize(vdw[depth - 1 - d])
for (dw = 0; dw < vdw[depth - 1 - d]; ++dw) {
minh = min_cu3d_height << (depth - 1 - d)
minw = min_cu3d_width << (depth - 1 - d)
mh = min(minh, height - dh * minh)
mw = min(minw, width - dw * minw)
layer_h = mh, layer_w = mw, layer_cd = (bmix) ? 1 : zdepth, L = 1
tensor = dst[d][dh][dw]
vh.push_back(layer_h)
vw.push_back(layer_w)
vcd.push_back((bmix) ? zdepth : layer_cd)
while (layer_h != 1 || layer_w != 1 || layer_cd != 1) {
layer_h = (layer_h + 1) >> 1
layer_w = (layer_w + 1) >> 1
layer_cd = (layer_cd + 1) >> 1
vh.push_back(layer_h)
vw.push_back(layer_w)
vcd.push_back((bmix) ? zdepth : layer_cd)

表 319 create_tree3d 函数的定义（续）

函数定义
++L
}
tensor.resize(L)
for (layer = L - 1; layer >= 0; --layer) {
tensor[layer].resize(vcd[L - 1 - layer])
for (zz = 0; zz < vcd[L - 1 - layer]; ++zz) {
tensor[layer][zz].resize(vh[L - 1 - layer])
for (hh = 0; hh < vh[L - 1 - layer]; ++hh) {
tensor[layer][zz][hh].resize(vw[L - 1 - layer])
}
}
}
}
}
}
}

get_bitdepth(w**)**

计算权重的bitdepth。函数定义见表320。

表 320 get_bitdepth 函数的定义

函数定义
get_bitdepth(w){
n=1
while (w>>1)
++n
return n
}

dequantw(q, bitDepth, stepSize**)**

反量化函数定义见表321。

表 321 dequantw 函数的定义

函数定义
dequantw(q, bitDepth, stepSize){
clipMinimum = -(1 << bitDepth)
clipMaximum = (1 << bitDepth) – 1
sign = (q < 0) ? -1 : 1
dQ = abs(q) * stepSize
clipDq = sign * dQ
return clipDq
}

10.2 权重压缩位流的语法描述

10.2.1 权重压缩位流定义

权重压缩位流由位流头和一个或多个层组成。位流头作用于所有层。每一层由一个层头和一个或多个子层组成。层头作用于每一子层。权重数据包含在子层中。它可以用一维数组表示,也可以用三维编码树表示。权重压缩位流定义见表322。

表 322 权重压缩位流定义

权重压缩位流定义	描述符
nnc() {	
nnc_header()	
for(layerIdx=0; layerIdx< total_trainable_layer;++ layerIdx){	
layer_header()	
sublayerIdx=0	
do{	
weightDim = sublayer_dim[sublayerIdx]	
sublayerScanOrder = sublayer_scan_order[sublayerIdx]	
includeBias = (sublayerIdx+1<total_sublayer&&WeightDim!=1 &&sublayer_dim[sublayerIdx+1]==1)	
if(weightDim == 1){	
SlayerBitDepth = sublayer_bitdepth[sublayerIdx]	
SlayerStepSize = SubLayerStepSize[sublayerIdx]	
array1d(sublayerIdx)	
end_of_last_layer_flag	ae(v)
}	
else {	
if(includeBias){	

表 322 权重压缩位流定义（续）

权重压缩位流定义	描述符
SlayerBitDepth = sublayer_bitdepth[sublayerIdx+1]	
SlayerStepSize = SubLayerStepSize[sublayerIdx+1]	
array1d(sublayerIdx+1)	
}	
SlayerBitDepth=sublayer_bitdepth[sublayerIdx]	
SlayerStepSize=SubLayerStepSize[sublayerIdx]	
PrevPredictedCbookSize=0	
if(sublayerScanOrder == SCAN_CK){	
for(c=0;c< WeightShape[2];c+=MaxCtu3dHeight){	
for(k=0;k< WeightShape[3];k+=MaxCtu3dWidth){	
ctu3d()	
end_of_last_layer_ctu_flag	ae(v)
}	
}	
}	
else if (sublayerScanOrder ==SCAN_KC){	
for(k=0;k< WeightShape[3];k+=MaxCtu3dWidth){	
for(c=0;c< WeightShape[2];c+=MaxCtu3dHeight){	
ctu3d()	
end_of_last_layer_ctu_flag	ae(v)
}	
}	
}	
sublayerIdx+=(includeBias ? 2 : 1)	
}while(sublayerIdx<total_sublayer)	
}	
}	
}	

10.2.2 权重压缩位流头定义

包含训练层总数，码本重排序允许标志，z轴重排序允许标志，三维编码树尺寸改变允许标志，三维编码树最大尺寸索引，一维数组量化样本精度等语法元素。定义见表323。

表 323 权重压缩位流头定义

权重压缩位流头定义	描述符
nnc_header() {	
integer_input	ae(v)
total_trainable_layer	ae(v)
enable_escape_reorder	ae(v)
enable_zdep_reorder	ae(v)
enable_max_ctu3d_size	ae(v)
max_ctu3d_idx	ae(v)
array1d_depth	ae(v)
}	

10.2.3 权重压缩位流层头定义

包含子层总数，最大权重值，权重维度，权重形状，权重偏置一维数组标志，权重扫描顺序，权重样本精度等语法元素。定义见表324。

表 324 权重压缩位流层头定义

权重压缩位流层头定义	描述符
layer_header() {	
includeBnShape = FALSE	
includeBiasArray1d = FALSE	
total_sublayer	ae(v)
for(sublayerIdx=0;sublayerIdx< total_sublayer;++sublayerIdx){	
sublayer_cmaxw[sublayerIdx]	ae(v)
sublayer_shape[sublayerIdx]={1, 1, 1, 1}	
if (!includeBiasArray1d){	
sublayer_dim[sublayerIdx]	ae(v)
if (sublayer_dim[sublayerIdx]!= 1 ‖ ! includeBnShape){	
for (d=4-sublayer_dim[sublayerIdx]; d<4;++d){	
sublayer_shape[sublayerIdx] [d]	ae(v)
}	
}	
else	
sublayer_shape[sublayerIdx] = {1,1,1,sublayer_shape[sublayerIdx-1][3]}	
if(sublayer_dim[sublayerIdx]== 1)	
includeBnShape = TRUE	
}	

表 324 权重压缩位流层头定义（续）

权重压缩位流层头定义	描述符
else{	
includeBiasArray1d = FALSE	
sublayer_shape[sublayerIdx] = {1,1,1,sublayer_shape[sublayerIdx-1][3]}	
sublayer_dim[sublayerIdx] = 1	
}	
if(sublayer_dim[sublayerIdx] != 1){	
if(sublayerIdx+1< total_sublayer){	
include_bias_array1d[sublayerIdx]	ae(v)
includeBiasArray1d = include_bias_array1d[sublayerIdx]	
}	
sublayer_scan_order[sublayerIdx]	ae(v)
sublayer_bitdepth[sublayerIdx]	ae(v)
SubLayerStepSize[sublayerIdx]=integer_input?1.0f: (sublayer_cmaxw[sublayerIdx]/256.0f)/ ((1<<sublayer_bitdepth[sublayerIdx])-1)	
}	
else{	
sublayer_scan_order[sublayerIdx] = 0	
sublayer_bitdepth[sublayerIdx] = array1d_depth	
SubLayerStepSize[sublayerIdx]=integer_input?1.0f: sublayer_cmaxw[sublayerIdx]/((1<<sublayer_bitdepth[sublayerIdx])-1)	
}	
}	
}	

10.2.4 权重压缩位流一维数组定义

包含权重一维数组非零标志，符号位，偏置值绝对值等语法元素。定义见表325。

表 325 权重压缩位流一维数组定义

权重压缩位流一维数组定义	描述符
array1d(sublayerIdx){	
if(sublayer_cmaxw[sublayerIdx]) {	
for(n=0;n<sublayer_shape[sublayerIdx][3];++n){	
bias_nz_flag	ae(v)
if(bias_nz_flag**){**	
bias_sign	ae(v)
bias_abs_q	ae(v)

表 325 权重压缩位流一维数组定义（续）

权重压缩位流一维数组定义	描述符
q = (bias_sign) ? -int(abs_q + 1) : abs_q + 1	
}	
WeightRec1d[n]=dequantw(q, SlayerBitDepth, SlayerStepSize)	
}	
}	
else{	
for(n=0;n< sublayer_shape[sublayerIdx][3]; ++n){	
WeightRec1d[n]=0.0f;	
}	
}	
}	

10.2.5 权重压缩位流三维编码树定义

三维编码树位流由树头位流和三维编码单元位流组成。定义见表326。

表 326 权重压缩位流三维编码树定义

权重压缩位流三维编码树定义	描述符
ctu3d(){	
ctu3d_header()	
cu3d(CbookPredictor, PrevPredictedSize, 0, 0, 0)	
}	

10.2.6 权重压缩位流三维编码树头定义

包含模式标志，起始深度允许标志等语法元素。定义见表327。

表 327 权重压缩位流三维编码树头定义

权重压缩位流三维编码树头定义	描述符
ctu3d_header(){	
SelectMapMode=3	
select_map_mode_flag	ae(v)
if(!select_map_mode_flag){	
map_mode_flag	ae(v)
SelectMapMode= (map_mode_flag) ? 1 : 2	
}	
enable_start_depth	ae(v)
zdep_array(ZdepArray, WeightZdepth, enable_zdep_reorder)	
}	

10.2.7 权重压缩位流RS数组定义

沿[RS]轴重排[C][K]平面:在CTU3D中，RS个[C][K]平面重排序以形成更统一的结构，并用ZdepArray[z]存放重排序的索引值。包含重排序标志，传送标志，索引值减1等语法元素。定义见表328。

表 328 权重压缩位流 RS 数组定义

权重压缩位流 RS 数组定义	描述符
zdep_array(ZdepArray, WeightZdepth, enable_zdep_reorder) {	
reoderFlag = FALSE	
if(enable_zdep_reorder && WeightZdepth >2){	
reorder_flag	ae(v)
reoderFlag = reorder_flag	
}	
ZdepArray.resize(WeightZdepth)	
if(!reorderFlag){	
for(n=0;n< WeightZdepth;++n)	
ZdepArray [n]=n;	
return;	
}	
queue.resize(WeightZdepth)	
queue[0]=-1;	
for(n=1;n< WeightZdepth -1;++n){	ae(v)
signalled_flag	
queue[n]=(signalled_flag)?1:(queue[n-1]>0)?-2:-1	
}	
queue[WeightZdepth -1]=(queue[WeightZdepth -2]>0)?-2:-1	
for(n=0;n< WeightZdepth;++n){	
ZdepArray [n]=-1;	
if(queue[n]==1){	ae(v)
qval_minus_one	
queue[n]=qval_minus_one+1	
}	
}	
qidx=0, zidx=-1	
do{	
while(ZdepArray [++zidx]!=-1)	
if(queue[qidx]==-1){	
ZdepArray [zidx]=zidx	
}	

表 328 权重压缩位流 RS 数组定义（续）

权重压缩位流 RS 数组定义	描述符
else{	
firstZidx=zidx	
while(queue[qidx]!=-2){	
ZdepArray [zidx]=queue[qidx]	
zidx=queue[qidx]	
++qidx	
}	
ZdepArray [zidx]= firstZidx	
zidx= firstZidx;	
}	
++qidx;	
}while(qidx < WeightZdepth);	
}	

10.2.8 权重压缩位流三维编码单元定义

最大尺寸CU3D递归地划分为三维CU3D块。码本是利用原始的权重系数和量化的权重系数生成的。码本预测器包含先前生成的码本。码本包括可在预测器中找到的索引和在预测器中找不到的索引。预测码本的大小使用当前CU3D的预测大小与以前CU3D预测的大小之间的差异进行编码。预测位图一次编码一位。对传送码本的大小进行编码。使用当前值的绝对值和上一个值的绝对值之间的差对传送码本进行编码。包含拆分标志，树图模式，八叉树起始深度差值，八叉树码本逸出模式，同一模式，标签模式，标签树起始深度差值，标签树码本逸出模式等语法元素。定义见表329。

表 329 权重压缩位流三维编码单元定义

权重压缩位流编码单元定义	描述符
cu3d(CbookPredictor, PrevPredictedSize, depth, yIdx, xIdx){	
if(yIdx>= Predictor [depth].size()||xIdx>= Predictor[depth][0].size())	
Return	
Predictor[depth][yIdx][xIdx] = CbookPredictor	
if(depth<Ctu3dDepth-1){	
split_flag	ae(v)
if(split_flag){	
cu3d(CbookPredictor, PrevPredictedSize, depth+1,(yIdx<<1),(xIdx<<1))	
cu3d(CbookPredictor, PrevPredictedSize, depth+1,(yIdx<<1)+1,(xIdx<<1))	
cu3d(CbookPredictor, PrevPredictedSize, depth+1,(yIdx<<1),(xIdx<<1)+1)	
cu3d(CbookPredictor, PrevPredictedSize, depth+1,(yIdx<<1)+1,(xIdx<<1)+1)	

表 329 权重压缩位流三维编码单元定义（续）

权重压缩位流编码单元定义	描述符
Return	
}	
}	
predicted_codebook(Predictor, Predicted, PrevPredictedSize)	
signalled_codebook(Predictor, Predicted, Cbook)	
cu3d_map_mode =(SelectMapMode == 1) ? 0 : 1	
if(SelectMapMode ==3)	
cu3d_map_mode	ae(v)
for (n = 0; n < 2; ++n) {	
NzFlagP[n] = 0	
CoefP[n]=0	
}	
for (z = 0; z < WeightZdepth; ++z)	
for (y = 0; y < Cu3dHeight; ++y)	
for (x = 0; x < Cu3dWidth; ++x)	
Map[z][y][x]=0	
StartDepth = Oct.size() – 1	
if(cu3d_map_mode== 0){	
if(enable_start_depth)	
oct_start_depth_delta	ae(v)
StartDepth -= oct_start_depth_delta	
CbookEscMode = Cbook.size() ? 1 : 0	
if(enable_escape_reorder && CbookSize)	
oct_cbook_esc_mode	ae(v)
CbookEscMode =(oct_cbook_esc_mode == 0) ? 1 : 2	
uni_mode	ae(v)
if(uni_mode)	
unitree3d(StartDepth,0,0,0,0,FALSE)	
Else	
octree3d(StartDepth,0,0,0,0,FALSE)	
}	
else if(cu3d_map_mode== 1){	
tgt_mode	ae(v)
StartDepth= ((tgt_mode) ? Tgtm.size() : Tgt.size()) – 1	
if(enable_start_depth){	
tag_start_depth_delta	ae(v)

表 329 权重压缩位流三维编码单元定义（续）

权重压缩位流编码单元定义	描述符
StartDepth -= tag_start_depth_delta	
}	
CbookEscMode = Cbook.size() ? 1 : 0	
if(enable_escape_reorder && Cbook.Size())	
tag_cbook_esc_mode	ae(v)
CbookEscMode =(tag_cbook_esc_mode == 0) ? 1 : 2	
if(tag_mode)	
tagtree3d(StartDepth,0,0,0,0,FALSE)	
Else	
for(z=0;z<ZdepArray.size();++z)	
uni_tagtree3d(StartDepth,0,z,0,0,FALSE,FALSE)	
}	
escape()	
}	

10.2.9 权重压缩位流预测码本定义

包含预测码本尺寸差值的绝对值，符号位，预测项标志等语法元素。定义见表330。

表 330 权重压缩位流预测码本定义

权重压缩位流预测码本定义	描述符
predicted_codebook(Predictor, Predicted, PrevPredictedSize) {	
PredictedSize = Predictor.size()	
if(!Predictor.size())	
return	
abs_predicted_diff	ae(v)
if(abs_predicted_diff)	
predicted_sign	ae(v)
PredictedSize=(predicted_sign ? -(abs_predicted_diff):abs_predicted_diff)+PrevPredictedSize	
for(n=0;n<MaxPredictorSize;++n) {	
if(Predicted.size() == PredictedSize)	
break	
predicted_flag	ae(v)
if(predicted_flag)	
Predicted.push_back(n)	
}	
}	

10.2.10 权重压缩位流传送码本定义

包含传送码本尺寸，差值非零标志，符号位，差值的绝对值，码本符号位等语法元素。定义见表331。

表 331 权重压缩位流传送码本定义

权重压缩位流传送码本定义	描述符
signalled_codebook(Predictor, Predicted, Cbook){	
PredictedSize = Predicted.size()	
if(PredictedSize<MaxCodebookSize)	
signalled_size	ae(v)
codeBookSize=PredictedSize+signalled_size	
Cbook.resize(codeBookSize)	
for (n = 0; n < PredictedSize; ++n)	
Cbook[n] = Predictor[Predicted[n]]	
prev=(PredictedSize)?abs(Cbook[PredictedSize -1]):0	
for(n= PredictedSize;n< codeBookSize;n++){	
delta=exist=0	
if(n>=2){	
for(m=0;m<n-1;++m){	
if(abs(Cbook[m])==abs(Cbook[n-1])){	
exist=1	
break	
}	
}	
}	
if(!exist)	
nzflag_delta	ae(v)
if(nzflag_delta){	
sign_delta	ae(v)
abs_delta	ae(v)
delta=(sign_delta?-(abs_delta):abs_delta)	
}	
Cbook[n]=delta+prev	
prev=Cbook[n]	
}	
for(n= PredictedSize;n< codeBookSize;++n){	
if (Cbook[n]) {	
cbook_sign	ae(v)
Cbook[n]=(sign?-(abs(Cbook[n])):abs(Cbook[n]))	
}	
}	
}	

10.2.11 权重压缩位流三维同一树定义

包含同一树节点标志，索引值或量化值标志，索引值，量化值，最大深度节点非零标志，索引值，量化值符号位，量化值绝对值，符号位等语法元素。定义见表332。

表 332 权重压缩位流三维同一树定义

权重压缩位流三维同一树定义	描述符
unitree3d(startDepth,treeDepth,zIdx,yIdx,xIdx,bSkip){	
cbookSize=Cbook.size()	
totalDepth= Utree.size()	
zsIdx=(treeDepth== totalDepth -1)?ZdepArray[zIdx]:zIdx	
if(treeDepth < totalDepth -1){	
uni_nzflag=Utree[treeDepth][zsIdx][yIdx][xIdx]=0	
if(treeDepth >= startDepth){	
if(!bSkip){	
uni_nzflag	ae(v)
Utree[treeDepth][zsIdx][yIdx][xIdx]= uni_nzflag	
if(!uni_nzflag){	
uni_map_nzflag	ae(v)
NzFlagP[1] = NzFlag[0]	
NzFlagP[0]= uni_map_nzflag	
if(uni_map_nzflag){	
if(cbookSize){	
uni_cmap_val	ae(v)
mapVal=uni_cmap_val	
}	
else{	
uni_qmap_val	ae(v)
mapVal=uni_qmap_val	
}	
}	
}	
}	
}	
nextZIdx=(zIdx<<1)	
nextYIdx=(yIdx<<1)	

表 332 权重压缩位流三维同一树定义（续）

权重压缩位流三维同一树定义	描述符
nextXIdx=(xIdx<<1)	
bSkip=(treeDepth>=startDepth)?!nzflag:FALSE	
if(bLocation0)	
unitree3d(startDepth, treeDepth +1, nextZIdx, nextYIdx, nextXIdx, bSkip)	
if(bLocation1)	
unitree3d(startDepth, treeDepth +1, nextZIdx, nextYIdx, nextXIdx +1, bSkip)	
if(bLocation2)	
unitree3d(startDepth, treeDepth +1, nextZIdx, nextYIdx +1, nextXIdx, bSkip)	
if(bLocation3)	
unitree3d(startDepth, treeDepth +1, nextZIdx, nextYIdx +1, nextXIdx +1, bSkip)	
if(bLocation4)	
unitree3d(startDepth, treeDepth +1, nextZIdx +1, nextYIdx, nextXIdx, bSkip)	
if(bLocation5)	
unitree3d(startDepth, treeDepth +1, nextZIdx +1, nextYIdx, nextXIdx +1, bSkip)	
if(bLocation6)	
unitree3d(startDepth, treeDepth +1, nextZIdx +1, nextYIdx +1, nextXIdx, bSkip)	
if(bLocation7)	
unitree3d(startDepth, treeDepth +1, nextZIdx +1, nextYIdx +1, nextXIdx +1, bSkip)	
return	
}	
if(startDepth ==totalDepth-1 || Utree[treeDepth -1][zIdx>>1][yIdx>>1][xIdx>>1]){	
uni_map_nzflag	ae(v)
if(uni_map_nzflag){	
if(cbookSize){	
uni_index	ae(v)
Map[zsIdx][yIdx][xIdx]=uni_index	
}	
else{	
uni_sign	ae(v)
uni_abs_q	ae(v)
Map[zsIdx][yIdx][xIdx]=(uni_sign?-(uni_abs_q):uni_abs_q)	
}	
}	

表 332 权重压缩位流三维同一树定义（续）

权重压缩位流三维同一树定义	描述符
else{	
if(!cbookSize && mapVal)	
uni_map_sign	ae(v)
Map[zsIdx][yIdx][xIdx]=(uni_map_sign?-(mapVal):mapVal)	
}	
CoefP[1]=CoefP[0]	
CoefP[0]= Map[zsIdx][yIdx][xIdx]	
}	

10.2.12 权重压缩位流三维八叉树定义

包含索引值非零标志，索引值，量化值符号位，量化值的绝对值等语法元素。定义见表333。

表 333 权重压缩位流三维八叉树定义

权重压缩位流三维八叉树定义	描述符
octree3d(startDepth,treeDepth,zIdx,yIdx,xIdx,bSkip){	
bProceed = TRUE	
cbooSize = Cbook.size()	
totalDepth=Oct.size()	
zsIdx=(treeDepth == totalDepth -1)? ZdepArray [zIdx]:zIdx	
oct_nzflag=Oct[treeDepth][zsIdx][yIdx][xIdx]=1	
if(treeDepth >= startDepth){	
if(!bSkip){	
oct_nzflag	ae(v)
Oct[treeDepth][zsIdx][yIdx][xIdx]=oct_nzflag	
NzFlagP[1]=NzFlagP[0]	
NzFlagP[0]= oct_nzflag	
}	
bProceed=oct_nzflag	
}	
if(bProceed){	
if(treeDepth < totalDepth -1){	
nextZIdx=(zIdx<<1)	
nextYIdx=(yIdx<<1)	
nextXIdx=(xIdx<<1)	

表 333 权重压缩位流三维八叉树定义（续）

权重压缩位流三维八叉树定义	描述符
nzFlag000 = 0, nzFlag001 = 0, nzFlag010 = 0, nzFlag011 = 0, nzFlag100 = 0, nzFlag101 = 0, nzFlag110 = 0	
nextZsIdx0 = (treeDepth + 1 == totalDepth - 1) ? ZdepArray [nextZIdx] : nextZIdx	
nextZsIdx1 = (treeDepth + 1 == totalDepth - 1) ? ((nextZIdx + 1 < zdepSize) ? ZdepArray [nextZIdx + 1] : -1) : nextZIdx + 1	
if(bExist000){	
octree3d(startDepth, treeDepth +1,nextZIdx,nextYIdx,nextXIdx,FALSE)	
nzFlag000 = Oct[treeDepth + 1][nextZsIdx0][nextYIdx][nextXIdx]	
}	
if(bExist001){	
octree3d(startDepth,treeDepth+1,nextZIdx,nextYIdx,nextXIdx+1, (nz1st1==1))	
nzFlag001 = Oct[treeDepth + 1][nextZsIdx0][nextYIdx][nextXIdx+1]	
}	
if(bExist010){	
octree3d(startDepth,treeDepth+1,nextZIdx,nextYIdx+1,nextXIdx, (nz1st2==2))	
nzFlag010 = Oct[treeDepth + 1][nextZsIdx0][nextYIdx+1][nextXIdx]	
}	
if(bExist011){	
octree3d(startDepth,treeDepth+1,nextZIdx,nextYIdx+1,nextXIdx+1, (nz1st3==3))	
nzFlag011 = Oct[treeDepth + 1][nextZsIdx0][nextYIdx+1][nextXIdx+1]	
}	
if(bExist100){	
octree3d(startDepth,treeDept+1,nextZIdx+1,nextYIdx,nextXIdx, (nz1st4==4))	
nzFlag100 = Oct[treeDepth + 1][nextZsIdx1][nextYIdx][nextXIdx]	
}	
if(bExist101){	
octree3d(startDepth,treeDepth+1,nextZIdx+1,nextYIdx,nextXIdx+1, (nz1st5==5))	
nzFlag101 = Oct[treeDepth + 1][nextZsIdx1][nextYIdx][nextXIdx+1]	
}	
if(bExist110){	

表 333 权重压缩位流三维八叉树定义（续）

权重压缩位流三维八叉树定义	描述符
octree3d(startDepth,treeDepth+1,nextZIdx+1,nextYIdx+1,nextXIdx, (nz1st6==6))	
nzFlag110 = Oct[treeDepth + 1][nextZsIdx1][nextYIdx+1][nextXIdx]	
}	
if(bExist111){	
octree3d(startDepth,treeDept+1,nextZIdx+1,nextYIdx+1,nextXIdx+1, (nz1st7==7))	
}	
Return	
}	
if(cbookSize){	
oct_index	ae(v)
Map[zsIdx][yIdx][xIdx]=oct_index	
}	
else{	
oct_sign	ae(v)
oct_abs_q	ae(v)
Map[zsIdx][yIdx][xIdx]=(oct_sign?-(oct_abs_q):oct_abs_q)	
}	
CoefP[1]=CoefP[0]	
CoefP[0]= Map[zsIdx][yIdx][xIdx]	
}	
}	

10.2.13 权重压缩位流三维标签树定义

包含索引值非零标志，索引值，父节点索引值和当前节点索引值之间的差值，量化值非零标志，量化值绝对值，父节点量化值和当前节点量化值之间差值的绝对值，量化值符号位等语法元素。定义见表334。

表 334 权重压缩位流三维标签树定义

权重压缩位流三维标签树定义	描述符
tagtree3d(startDepth,treeDepth,zIdx,yIdx,xIdx,bSkip){	
cbookSize=Cbook.size()	
totalDepth=Tgtm.size()	
zsIdx=(treeDepth ==totalDepth-1)? ZdepArray [zIdx]:zIdx	
if(treeDepth)	

表 334 权重压缩位流三维标签树定义（续）

权重压缩位流三维标签树定义	描述符
Tgtm[treeDepth][zsIdx][yIdx][xIdx]= Tgtm[treeDepth -1] [zIdx>>1] [yIdx>>1][xIdx>>1]	
if(treeDepth >= startDepth){	
if(!bSkip){	
if(cbookSize){	
if(treeDepth == startDepth){	
tgtm_nzflag_index	ae(v)
if(tgtm_nzflag_index)	
tgtm_index	ae(v)
Tgtm[treeDepth][zsIdx][yIdx][xIdx]=tgtm_index	
}	
else{	
tgtm_delta_index	ae(v)
Tgtm[treeDepth][zsIdx][yIdx][xIdx]= Tgtm[treeDepth-1][zsIdx>>1][yIdx>>1][xIdx>>1]+tgtm_delta_index	
}	
}	
else{	
if(treeDepth == startDepth){	
tgtm_nzflag_q	ae(v)
if(tgtm_nzflag_q)	
tgtm_abs_q	ae(v)
Tgtm[treeDepth][zsIdx][yIdx][xIdx]=tgtm_abs_q	
}	
else{	
tgtm_delta_abs_q	ae(v)
Tgtm[treeDepth][zsIdx][yIdx][xIdx]= Tgtm[treeDepth-1][zsIdx>>1][yIdx>>1][xIdx>>1]+tgtm_delta_abs_q	
}	
}	
CoefP[1]=CoefP[0]	
CoefP[0]= Tgtm[treeDepth][zsIdx][yIdx][xIdx]	
}	
nzFlag=!!Tgtm[treeDepth][zsIdx][yIdx][xIdx]	
if(treeDepth == (totalDepth -1)&nzFlag&& cbookSize ==0){	
tgtm_sign_q	ae(v)

表 334 权重压缩位流三维标签树定义（续）

权重压缩位流三维标签树定义	描述符
Tgtm[treeDepth][zsIdx][yIdx][xIdx]= (tgtm_sign_q?-(Tgtm[treeDepth][zsIdx][yIdx][xIdx]): Tgtm[treeDepth][zsIdx][yIdx][xIdx])	
CoefP[0]= Tgtm[treeDepth][zsIdx][yIdx][xIdx]	
}	
}	
if(treeDepth <totalDepth-1){	
nextZIdx=(zIdx<<1)	
nextYIdx=(yIdx<<1)	
nextXIdx=(xIdx<<1)	
eqFlag000 = 0, eqFlag001 = 0, eqFlag010 = 0, eqFlag011 = 0, eqFlag100 = 0, eqFlag101 = 0, eqFlag110 = 0	
nextZsIdx0 = (treeDepth + 1 == TotalDepth - 1) ? ZdepArray [nextZIdx] : nextZIdx	
nextZsIdx1 = (treeDepth + 1 == TotalDepth - 1) ? ((nextZIdx + 1 < zdepSize) ? ZdepArray [nextZIdx + 1] : -1) : nextZIdx + 1	
if(bExist000){	
tagtree3d(startDepth, treeDepth +1, nextZIdx,nextYIdx,nextXIdx,FALSE)	
eqFlag000=(abs(Tgtm[treeDepth][zsIdx][yIdx][xIdx])== abs(Tgtm[treeDepth + 1][nextZsIdx0][nextYIdx][nextXIdx]))	
}	
if(bExist001){	
tagtree3d(startDepth, treeDepth +1, nextZIdx,nextYIdx,nextXIdx+1, (eq1st1 == 1))	
eqFlag001=(abs(Tgtm[treeDepth][zsIdx][yIdx][xIdx])== abs(Tgtm[treeDepth + 1][nextZsIdx0][nextYIdx][nextXIdx+1]))	
}	
if(bExist010){	
tagtree3d(startDepth, treeDepth +1, nextZIdx,nextYIdx+1,nextXIdx, (eq1st2 == 2))	
eqFlag010=(abs(Tgtm[treeDepth][zsIdx][yIdx][xIdx])== abs(Tgtm[treeDepth + 1][nextZsIdx0][nextYIdx+1][nextXIdx]))	
}	
if(bExist011){	
tagtree3d(startDepth, treeDepth +1, nextZIdx,nextYIdx+1,nextXIdx+1, (eq1st3 == 3))	

表 334 权重压缩位流三维标签树定义（续）

权重压缩位流三维标签树定义	描述符
eqFlag011=(abs(Tgtm[treeDepth][zsIdx][yIdx][xIdx])== abs(Tgtm[treeDepth + 1][nextZsIdx0][nextYIdx+1][nextXIdx+1]))	
}	
if(bExist100){	
tagtree3d(startDepth, treeDepth +1, nextZIdx+1,nextYIdx,nextXIdx, (eq1st4 == 4))	
eqFlag100=(abs(Tgtm[treeDepth][zsIdx][yIdx][xIdx])== abs(Tgtm[treeDepth + 1][nextZsIdx1][nextYIdx][nextXIdx]))	
}	
if(bExist101){	
tagtree3d(startDepth, treeDepth +1, nextZIdx+1,nextYIdx,nextXIdx+1, (eq1st5 == 5))	
eqFlag101=(abs(Tgtm[treeDepth][zsIdx][yIdx][xIdx])== abs(Tgtm[treeDepth + 1][nextZsIdx1][nextYIdx][nextXIdx+1]))	
}	
if(bExist110){	
tagtree3d(startDepth, treeDepth +1, nextZIdx+1,nextYIdx+1,nextXIdx, (eq1st6 == 6))	
eqFlag110=(abs(Tgtm[treeDepth][zsIdx][yIdx][xIdx])== abs(Tgtm[treeDepth + 1][nextZsIdx1][nextYIdx+1][nextXIdx]))	
}	
if(bExist111){	
tagtree3d(startDepth, treeDepth +1, nextZIdx+1,nextYIdx+1,nextXIdx+1, (eq1st7 == 7))	
}	
Return	
}	
Map[zsIdx][yIdx][xIdx]= Tgtm[treeDepth][zsIdx][yIdx][xIdx]	
}	

10.2.14 权重压缩位流三维同一及标签树定义

包含节点非零标志，索引值非零标志，索引值，父节点索引值和当前节点索引值之间的差值，量化值非零标志，量化值绝对值，父节点量化值和当前节点量化值之间差值的绝对值，符号位等语法元素。定义见表335。

表 335 权重压缩位流三维同一标签树定义

权重压缩位流三维同一及标签树定义	描述符
uni_tagtree3d(startDepth,treeDepth,zIdx,yIdx,xIdx,uniSkip,tgtSkip){	
cbookSize=Cbook.size()	
totalDepth=Tgt.size()	
zsIdx=(treeDepth ==totalDepth-1)? ZdepArray [zIdx]:zIdx	
uniFlag=Tgt[treeDepth][zsIdx][yIdx][xIdx][0]=0	
if(treeDepth)	
Tgt[treeDepth][zsIdx][yIdx][xIdx] [1]= Tgt[treeDepth-1][zsIdx][yIdx>>1][xIdx>>1] [1]	
if(treeDepth >= startDepth){	
if(!uniSkip){	
if(startDepth < totalDepth -1 || ((yIdx&1)==0 && (xIdx&1)==0)){	
uni_tgt_nzflag	ae(v)
rc = uni_tgt_nzflag_uni	
}	
else{	
rc =tgt[treeDepth][zsIdx][yIdx&(~1)][xIdx&(~1)][0]	
Tgt[treeDepth][zsIdx][yIdx][xIdx] [1]= abs(Tgt[treeDepth][zsIdx][yIdx&(~1)][xIdx&(~1)][1])	
}	
Tgt[treeDepth][zsIdx][yIdx][xIdx] [0]= uniFlag=rc	
if(!tgtSkip && (startDepth < totalDepth -1 || (((yIdx&1)==0 && (xIdx&1)==0) || uniFlag))) {	
if(cbookSize){	
if(treeDepth == startDepth){	
uni_tgt_nzflag_index	ae(v)
if(uni_tgt_nzflag_index)	
uni_tgt_index	ae(v)
Tgt[treeDepth][zsIdx][yIdx][xIdx] [1]=uni_tgt_index	
}	
else{	
uni_tgt_delta_index	ae(v)
Tgt[treeDepth][zsIdx][yIdx][xIdx] [1]= Tgt[treeDepth-1][zsIdx][yIdx>>1][xIdx>>1][1]+ uni_tgt_delta_index	
}	
}	
else{	

表 335 权重压缩位流三维同一标签树定义（续）

权重压缩位流三维同一及标签树定义	描述符
if(treeDepth == startDepth){	
uni_tgt_nzflag_q	ae(v)
if(uni_tgt_nzflag_q)	
uni_tgt_abs_q	ae(v)
Tgt[treeDepth][zsIdx][yIdx][xIdx] [1]=uni_tgt_abs_q	
}	
else{	
uni_tgt_delta_abs_q	ae(v)
Tgt[treeDepth][zsIdx][yIdx][xIdx] [1]= Tgt[treeDepth -1][zsIdx][yIdx>>1][x_idx>>1][1]+ uni_tgt_delta_abs_q	
}	
}	
CoefP[1]=CoefP[0]	
CoefP[0]= abs(Tgt[treeDepth][zsIdx][yIdx][xIdx] [1])	
}	
}	
nzFlag=!!Tgt[treeDepth][zsIdx][yIdx][xIdx] [1]	
if(treeDepth ==(totalDepth-1)&&nzFlag&&!cbookSize){	
uni_tgt_sign_q	ae(v)
Tgt[treeDepth][zsIdx][yIdx][xIdx] [1]= (uni_tgt_sign_q?-(Tgt[treeDepth][zsIdx][yIdx][xIdx][1]): Tgt[treeDepth][zsIdx][yIdx][xIdx][1])	
CoefP[0]= Tgt[treeDepth][zsIdx][yIdx][xIdx] [1]	
}	
}	
if(treeDepth < totalDepth -1){	
nextYIdx=(yIdx<<1)	
nextIdx=(xIdx<<1)	
uSkip=(treeDepth >=startDepth)?!uniFlag:FALSE	
eqFlag00 = 0, eqFlag01 = 0, eqFlag10 = 0	
zsIdx0 = (treeDepth + 1 == totalDepth - 1) ? ZdepArray [zIdx] : zIdx	
if(bExist00){	
uni_tagtree3d(startDepth, treeDepth +1,zIdx,nextYIdx,nextXIdx,uSkip,FALSE)	
eqFlag00=(abs(Tgt[treeDepth][zIdx][yIdx][xIdx][1])== abs(Tgt[treeDepth + 1][zsIdx0][nextYIdx][nextXIdx][1]))	
}	

表 335 权重压缩位流三维同一标签树定义（续）

权重压缩位流三维同一及标签树定义	描述符
if(bExist01){	
uni_tagtree3d(startDepth,treeDepth+1,zIdx,nextYIdx,nextXIdx+1,uSkip, (eq1st1==1))	
eqFlag01=(abs(Tgt[treeDepth][zIdx][yIdx][xIdx][1])== abs(Tgt[treeDepth + 1][zsIdx0][nextYIdx][nextXIdx+1][1]))	
}	
if(bExist10){	
uni_tagtree3d(startDepth,treeDepth+1,zIdx,nextYIdx+1,nextXIdx,uSkip, (eq1st2==2))	
eqFlag10=(abs(Tgt[treeDepth][zIdx][yIdx][xIdx][1])== abs(Tgt[treeDepth + 1][zsIdx0][nextYIdx+1][nextXIdx][1]))	
}	
if(bExist11){	
uni_tagtree3d(startDepth,treeDepth+1,zIdx,nextYIdx+1,nextXIdx+1,uSkip, (eq1st3==3))	
}	
Return	
}	
Map[zsIdx][yIdx][xIdx]=Tgt[treeDepth][zsIdx][yIdx][xIdx][1]	
}	

10.2.15 权重压缩位流三维逸出定义

包含量化值非零标志，不使用码本量化值符号位和绝对值等语法元素。定义见表336。

表 336 权重压缩位流逸出定义

权重压缩位流逸出定义	描述符
escape(CbookEscMode){	
escapeIndex=(CbookEscMode==0)?-1:(CbookEscMode == 1) ? Cbook.size() : 0	
if(Cbook.size() || CbookEscMode!=0) {	
for(z=0;z<WeightZdepth;++z){	
zs=ZdepArray[z]	
for(y=0;y<Cu3dHeight;++y){	
for(x=0;x<Cu3dWidth:++x){	
if (CbookEscMode != 0) {	
if(Map[zs][y][x]== escapeIndex){	
q=0	
esc_nzflag	ae(v)
if(esc_nzflag){	
esc_sign	ae(v)
esc_abs_q	ae(v)
q=(esc_sign?-esc_abs_q:esc_abs_q)	
}	
else{	
q=(CbookEscMode ==1)?Cbook[Map[zs][y][x]]: Cbook[Map[zs][y][x]-1]	
}	
}	
else {	
q = Map[zs][y][x]	
}	
}	
else{	
q = Map[zs][y][x]	
}	
dQ=dequantw(q, SlayerBitDepth, SlayerStepSize)	
WeightRec3d[zs][y+Cu3dYOff][x+Cu3dXOff]=dQ	
}	
}	
}	
}	

10.3 权重压缩位流的语义描述

10.3.1 初始化

语法元素和变量的初始值设定为0，除非明确定义。

10.3.2 权重压缩位流语义

最后子层标志 **end_of_last_layer_flag**

二值变量。值为‘1’表示最后的子层；值为‘0’表示非最后的子层。

end_of_last_layer_flag必须与(subLayerIdx == total_sublayer – 1)相同。

最后一个CTU3D标志 **end_of_last_layer_ctu_flag**

二值变量。值为‘1’表示子层最后的一个CTU3D；值为‘0’表示非最后的一个CTU3D。

end_of_last_layer_ctu_flag必须与

(MaxCtu3dHeight>=WeightShape[2]-c&&MaxCtu3dWidth>=WeightShape[3]-k)相同。

10.3.3 权重压缩位流头语义

整数标志 **integer_input**

二值变量。值为‘1’表示整数输入；值为‘0’表示小数输入。

训练层总数 **total_trainable_layer**

16位无符号整数。

码本重排序允许标志 **enable_escape_reorder**

二值变量。值为‘1’表示允许escape重排序。值为‘0’表示不允许。

ZdepArray重排序允许标志 **enable_zdep_reorder**

二值变量。值为‘1’表示允许ZdepArray重排序。值为‘0’表示不允许。

三维编码树尺寸改变允许标志 **enable_max_ctu3d_size**

二值变量。值为‘1’表示允许基于核的尺寸改变ctu3d_height/ctu3d_width。值为‘0’表示不允许。

三维编码树最大尺寸索引 **max_ctu3d_idx**

2位无符号整数。maxCtu3dWidth3d和maxCtu3dHeight3d按下面的方法计算：

maxCtu3dWidth3d=maxCtu3dHeight3d=

(max_ctu3d_idx==0)?64:(max_ctu3d_idx==1)?32:(max_ctu3d_idx==2)?16:8

一维数组量化样本精度 **array1d_depth**

5位无符号整数。一维数组的最大量化样本精度BitDepth。

10.3.3 权重压缩位流层头语义

子层总数 **total_sublayer**

4位无符号整数。各层的子层总数。

子层最大权重值 **sublayer_cmaxw**[sublayerIdx]

32位无符号整数。

子层权重维度 **sublayer_dim**[sublayerIdx]

2位无符号整数。

子层权重形状 **sublayer_shape**[sublayerIdx]

16位无符号整数。 表示[R][S][C][K]的值。

R = sublayer_shape[sublayerIdx][0]

S = sublayer_shape[sublayerIdx][1]

C = sublayer_shape[sublayerIdx][2]

K = sublayer_shape[sublayerIdx][3]

子层权重偏置一维数组标志 **include_bias_array1d**[sublayerIdx]

二值变量。值为‘1’表示该子层权重是偏置一维数组。值为‘0’表示不是。

子层权重扫描顺序 **sublayer_scan_order**[sublayerIdx]

二值变量。值为‘1’表示该子层权重使用垂直扫描(SCAN_KC=1)。

值为‘0’表示使用水平扫描(SCAN_CK=0)。

子层权重样本精度 **sublayer_bitdepth**[sublayerIdx]

5位无符号整数。

10.3.4 权重压缩位流一维数组语义

子层权重偏置值非零标志 **bias_nz_flag**

二值变量。值为‘1’表示该子层权重偏置值非零。值为‘0’表示是零。

子层权重偏置值符号位 **bias_sign**

二值变量。值为‘1’表示该子层权重偏置值为负。值为‘0’表示为正。

子层权重偏置值绝对值 **bias_abs_q**

表示该子层权重偏置值的绝对值。

10.3.5 权重压缩位流三维编码树语义

表示三维编码树位流的组成。首先是三维编码树树头位流，其次是三维编码单元位流。

10.3.6 权重压缩位流三维编码树头语义

三维编码树图模式标志 **select_map_mode_flag**

二值变量。值为‘1’表示该三维编码树中所有的三维编码单元使用相同的MapMode。值为‘0’表示每一个三维编码单元使用自己的MapMode。

图模式标志 **map_mode_flag**

二值变量。值为‘1’表示SelectMapMode的值等于1；值为‘0’表示SelectMapMode的值等于2。SelectMapMode的缺省值等于3。

起始深度允许标志 **enable_start_depth**

二值变量。值为‘1’表示三维编码单元可以从任何深度开始编码；值为‘0’表示总是从最大深度开始编码。

10.3.7 权重压缩位流 RS 数组语义

重排序标志 **reorder_flag**

二值变量。值为‘1’表示RS数组ZdepArray允许重排序；值为‘0’表示不允许。

传送标志 **signalled_flag**

二值变量。值为‘1’表示queue[n]是传送的；值为‘0’表示queue[n]是推断的。

索引值减1 **qval_minus_one**

表示传送的queue[n] 减1值。

10.3.8 权重压缩位流三维编码单元语义

拆分标志 **split_flag**

二值变量。值为‘1’表示三维编码单元进行拆分；值为‘0’表示不应拆分。

图模式 **cu3d_map_mode**

二值变量。值为‘1’表示选择三维标签树 或三维同一标签树；值为‘0’选择三维同一树或三维八叉树。

八叉树起始深度差值 **oct_start_depth_delta**

StartDepth = total_depth – 1 – oct_start_depth_delta

八叉树码本逸出模式 **oct_cbook_esc_mode**

二值变量。值为‘1’表示逸出，使用索引idx-1；值为‘0’表示非逸出，使用索引idx。

同一模式 **uni_mode**

二值变量。值为‘1’表示选择三维同一树方法；值为‘0’表示选择三维八叉树方法。

标签模式 **tag_mode**

二值变量。值为‘1’表示选择三维标签树方法；值为‘0’表示选择三维同一标签树方法。

标签树起始深度差值 **tag_start_depth_delta**

StartDepth = total_depth – 1 – tag_start_depth_delta

标签树码本逸出模式 **tag_cbook_esc_mode**

二值变量。值为‘1’表示逸出，使用索引idx-1；值为‘0’表示非逸出，使用索引idx。

10.3.9 权重压缩位流预测码本语义

预测尺寸差值的绝对值 **abs_predicted_diff**

符号位 **predicted_sign**

二值变量。值为‘1’表示预测尺寸差值为负；值为‘0’表示预测尺寸差值为正。

预测项标志 **predicted_flag**

二值变量。值为‘1’表示预测位置n为预测项；值为‘0’表示预测位置n为非预测项。

10.3.10 权重压缩位流传送码本语义

传送码本尺寸 **signalled_size**

表示传送码本尺寸。

差值非零标志 **nzflag_delta**

二值变量。值为‘1’表示差值非零为非零；值为‘0’表示差值非零为零。

差值符号位 **sign_delta**

二值变量。值为‘1’表示差值为负；值为‘0’表示差值为正。

差值的绝对值 **abs_delta**

delta = sign_delta? -abs_delta: abs_delta

码本符号位 **cbook_sign**

二值变量。值为‘1’表示码本位置n为负；值为‘0’表示码本位置n为正。

Cbook[n]=sign?-abs(Cbook[n]):abs(Cbook[n])

10.3.11 权重压缩位流三维同一树语义

同一树节点标志 **uni_nzflag**

二值变量。值为‘1’表示同一树节点值为1;值为‘0’表示同一树节点值为0。

图非零标志 **uni_map_nzflag**

二值变量。值为‘1’表示同一树节点值的索引值;值为‘0’表示同一树节点绝对值的量化值。

在定义之内的变量按下面的方法计算：

zdepth = Utree[treeDepth + 1].size()

height = Utree[treeDepth + 1][0].size()

width = Utree[treeDepth + 1][0][0].size()

bLocation0= (nextZIdx < zdepth && nextYIdx < height && nextXIdx < width)

bLocation1= (nextZIdx < zdepth && nextYIdx < height && nextXIdx + 1 < width)

bLocation2 = (nextZIdx < zdepth && nextYIdx + 1 < height && nextXIdx < width)

bLocation3= (nextZIdx < zdepth && nextYIdx + 1 < height && nextXIdx + 1 < width)

bLocation4= (nextZIdx + 1 < zdepth && nextYIdx < height && nextXIdx < width)

bLocation5= (nextZIdx + 1 < zdepth && nextYIdx < height && nextXIdx + 1 < width)

bLocation6= (nextZIdx + 1 < zdepth && nextYIdx + 1 < height && nextXIdx < width)

bLocation7= (nextZIdx + 1 < zdepth && nextYIdx + 1 < height && nextXIdx + 1 < width)

图值的索引值 **uni_cmap_val**

表示图值的索引值。

mapVal = uni_cmap_val

图值的量化值 **uni_qmap_val**

表示图值的量化值。

mapVal = uni_qmap_val

以下的是同一树最大深度节点语义。

图非零标志 **uni_map_nzflag**

二值变量。值为‘1’表示同一树最大深度节点图值为索引值;值为‘0’表示同一树节点图值为量化值。

索引值 **uni_index**

量化值符号位 **uni_sign**

二值变量。值为‘1’表示量化值为负;值为‘0’表示量化值为正。

量化值绝对值 **uni_abs_q**

表示量化值的绝对值。

符号位 **uni_map_sign**

二值变量。值为‘1’表示量化值为负;值为‘0’表示量化值为正。

使用直接量化，则对符号位进行编码。

10.3.12 权重压缩位流三维八叉树语义

八叉树索引值非零标志 **oct_nzflag**

二值变量。值为‘1’表示八叉树索引值为非零；值为‘0’表示八叉树索引值为零。

在定义之内的变量按下面的方法计算：

zdepth = Oct[treeDepth + 1].size()

height = Oct[treeDepth + 1][0].size()

width = Oct[treeDepth + 1][0][0].size()

nz1st1= nz1st2= nz1st3= nz1st4= nz1st5= nz1st6= nz1st7= -1

bExist000 = (nextZIdx < zdepth && nextYIdx < height && nextXIdx < width)

bExist001 = (nextZIdx < zdepth && nextYIdx < height && nextXIdx + 1 < width)

bExist010 = (nextZIdx < zdepth && nextYIdx + 1 < height && nextXIdx < width)

bExist011 = (nextZIdx < zdepth && nextYIdx + 1 < height && nextXIdx + 1 < width)

bExist100 = (nextZIdx + 1 < zdepth && nextYIdx < height && nextXIdx < width)

bExist101 = (nextZIdx + 1 < zdepth && nextYIdx < height && nextXIdx + 1 < width)

bExist110 = (nextZIdx + 1 < zdepth && nextYIdx + 1 < height && nextXIdx < width)

bExist111 = (nextZIdx + 1 < zdepth && nextYIdx + 1 < height && nextXIdx + 1 < width)

当(treeDepth >= startDepth)的时候，按下面的方法计算变量:

nz1st1 =

bExist111?(!(nzflag000||nzflag001||nzflag010||nzflag011||nzflag100||nzflag101||nzflag110)?7:-1) :
bExist110 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100 || nzflag101) ? 6 : -1) :
bExist101 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100) ? 5 : -1) :
bExist100 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011) ? 4 : -1) :
bExist011 ? (!(nzflag000 || nzflag001 || nzflag010) ? 3 : -1) :
bExist010 ? (!(nzflag000 || nzflag001) ? 2 : -1) :
bExist001 ? (!(nzflag000) ? 1 : -1) :-1

nz1st2 =

bExist111 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100 || nzflag101 || nzflag110) ? 7 : -1) :
bExist110 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100 || nzflag101) ? 6 : -1) :
bExist101 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100) ? 5 : -1) :
bExist100 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011) ? 4 : -1) :
bExist011 ? (!(nzflag000 || nzflag001 || nzflag010) ? 3 : -1) :
bExist010 ? (!(nzflag000 || nzflag001) ? 2 : -1) :
bExist001 ? (!(nzflag000) ? 1 : -1) : -1

nz1st3 =

bExist111 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100 || nzflag101 || nzflag110) ? 7 : -1) :
bExist110 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100 || nzflag101) ? 6 : -1) :
bExist101 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100) ? 5 : -1) :
bExist100 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011) ? 4 : -1) :
bExist011 ? (!(nzflag000 || nzflag001 || nzflag010) ? 3 : -1) :
bExist010 ? (!(nzflag000 || nzflag001) ? 2 : -1) :
bExist001 ? (!(nzflag000) ? 1 : -1) : -1

nz1st4 =

bExist111 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100 || nzflag101 || nzflag110) ? 7 : -1) :
bExist110 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100 || nzflag101) ? 6 : -1) :
bExist101 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100) ? 5 : -1) :
bExist100 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011) ? 4 : -1) :
bExist011 ? (!(nzflag000 || nzflag001 || nzflag010) ? 3 : -1) :
bExist010 ? (!(nzflag000 || nzflag001) ? 2 : -1) :
bExist001 ? (!(nzflag000) ? 1 : -1) : -1

nz1st5 =

bExist111 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100 || nzflag101 || nzflag110) ? 7 : -1) :
bExist110 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100 || nzflag101) ? 6 : -1) :

bExist101 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100) ? 5 : -1) :
bExist100 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011) ? 4 : -1) :
bExist011 ? (!(nzflag000 || nzflag001 || nzflag010) ? 3 : -1) :
bExist010 ? (!(nzflag000 || nzflag001) ? 2 : -1) :
bExist001 ? (!(nzflag000) ? 1 : -1) : -1

nz1st6 =
bExist111 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100 || nzflag101 || nzflag110) ? 7 : -1) :
bExist110 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100 || nzflag101) ? 6 : -1) :
bExist101 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100) ? 5 : -1) :
bExist100 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011) ? 4 : -1) :
bExist011 ? (!(nzflag000 || nzflag001 || nzflag010) ? 3 : -1) :
bExist010 ? (!(nzflag000 || nzflag001) ? 2 : -1) :
bExist001 ? (!(nzflag000) ? 1 : -1) : -1

nz1st7 =
bExist111 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100 || nzflag101 || nzflag110) ? 7 : -1) :
bExist110 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100 || nzflag101) ? 6 : -1) :
bExist101 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100) ? 5 : -1) :
bExist100 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011) ? 4 : -1) :
bExist011 ? (!(nzflag000 || nzflag001 || nzflag010) ? 3 : -1) :
bExist010 ? (!(nzflag000 || nzflag001) ? 2 : -1) :
bExist001 ? (!(nzflag000) ? 1 : -1) : -1

八叉树索引值 **oct_index**

表示八叉树索引值，使用码本。

八叉树量化值符号位 **oct_sign**

二值变量。值为‘1’表示量化值为负；值为‘0’表示量化值为正。

八叉树量化值的绝对值 **oct_abs_q**

表示八叉树量化值的绝对值，不使用码本。

10.3.13 权重压缩位流三维标签树语义

索引值非零标志 **tgtm_nzflag_index**

二值变量。值为‘1’表示标签树索引值为非零；值为‘0’表示标签树索引值为零。

索引值 **tgtm_index**

表示标签树索引值。

差值索引值 **tgtm_delta_index**

表示父节点索引值和当前节点索引值之间的差值。

量化值非零标志 **tgtm_nzflag_q**

二值变量。值为‘1’表示量化值为非零；值为‘0’表示量化值为零。

量化值绝对值 **tgtm_abs_q**

表示八叉树量化值的绝对值。

量化值差值绝对值 **tgtm_delta_abs_q**

表示父节点量化值和当前节点量化值之间差值的绝对值。

量化值符号位 **tgtm_sign_q**

使用直接量化，则对其相应的非零系数的符号位进行编码。

在定义之内的变量按下面的方法计算：

zdepth = Tgtm[treeDepth + 1].size()
height = Tgtm[treeDepth + 1][0].size()
width = Tgtm[treeDepth + 1][0][0].size()
eq1st1= eq1st2= eq1st3= eq1st1st4= eq1st5= eq1st6= eq1st7= -1
bExist000 = (nextZIdx < zdepth && nextYIdx < height && nextXIdx < width)
bExist001 = (nextZIdx < zdepth && nextYIdx < height && nextXIdx + 1 < width)
bExist010 = (nextZIdx < zdepth && nextYIdx + 1 < height && nextXIdx < width)
bExist011 = (nextZIdx < zdepth && nextYIdx + 1 < height && nextXIdx + 1 < width)
bExist100 = (nextZIdx + 1 < zdepth && nextYIdx < height && nextXIdx < width)
bExist101 = (nextZIdx + 1 < zdepth && nextYIdx < height && nextXIdx + 1 < width)
bExist110 = (nextZIdx + 1 < zdepth && nextYIdx + 1 < height && nextXIdx < width)
bExist111 = (nextZIdx + 1 < zdepth && nextYIdx + 1 < height && nextXIdx + 1 < width)

当(treeDepth >= startDepth)的时候，按下面的方法计算变量:

```
eq1st1 =
  bExist111?(!(nzflag000||nzflag001||nzflag010||nzflag011||nzflag100||nzflag101||nzflag110)?7:-1) :
  bExist110 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100 || nzflag101) ? 6 : -1) :
  bExist101 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100) ? 5 : -1) :
  bExist100 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011) ? 4 : -1) :
  bExist011 ? (!(nzflag000 || nzflag001 || nzflag010) ? 3 : -1) :
  bExist010 ? (!(nzflag000 || nzflag001) ? 2 : -1) :
  bExist001 ? (!(nzflag000) ? 1 : -1) :-1

eq1st2 =
  bExist111 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100 || nzflag101 || nzflag110) ? 7 : -1) :
  bExist110 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100 || nzflag101) ? 6 : -1) :
  bExist101 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100) ? 5 : -1) :
  bExist100 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011) ? 4 : -1) :
  bExist011 ? (!(nzflag000 || nzflag001 || nzflag010) ? 3 : -1) :
  bExist010 ? (!(nzflag000 || nzflag001) ? 2 : -1) :
  bExist001 ? (!(nzflag000) ? 1 : -1) : -1

eq1st3 =
  bExist111 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100 || nzflag101 || nzflag110) ? 7 : -1) :
  bExist110 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100 || nzflag101) ? 6 : -1) :
  bExist101 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100) ? 5 : -1) :
  bExist100 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011) ? 4 : -1) :
```

```
    bExist011 ? (!(nzflag000 || nzflag001 || nzflag010) ? 3 : -1) :
    bExist010 ? (!(nzflag000 || nzflag001) ? 2 : -1) :
    bExist001 ? (!(nzflag000) ? 1 : -1) : -1

  eq1st4 =
    bExist111 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100 || nzflag101 || nzflag110) ? 7 : -1) :
    bExist110 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100 || nzflag101) ? 6 : -1) :
    bExist101 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100) ? 5 : -1) :
    bExist100 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011) ? 4 : -1) :
    bExist011 ? (!(nzflag000 || nzflag001 || nzflag010) ? 3 : -1) :
    bExist010 ? (!(nzflag000 || nzflag001) ? 2 : -1) :
  bExist001 ? (!(nzflag000) ? 1 : -1) : -1

  eq1st5 =
    bExist111 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100 || nzflag101 || nzflag110) ? 7 : -1) :
    bExist110 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100 || nzflag101) ? 6 : -1) :
    bExist101 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100) ? 5 : -1) :
    bExist100 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011) ? 4 : -1) :
    bExist011 ? (!(nzflag000 || nzflag001 || nzflag010) ? 3 : -1) :
    bExist010 ? (!(nzflag000 || nzflag001) ? 2 : -1) :
    bExist001 ? (!(nzflag000) ? 1 : -1) : -1

  eq1st6 =
    bExist111 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100 || nzflag101 || nzflag110) ? 7 : -1) :
    bExist110 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100 || nzflag101) ? 6 : -1) :
    bExist101 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100) ? 5 : -1) :
    bExist100 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011) ? 4 : -1) :
    bExist011 ? (!(nzflag000 || nzflag001 || nzflag010) ? 3 : -1) :
    bExist010 ? (!(nzflag000 || nzflag001) ? 2 : -1) :
    bExist001 ? (!(nzflag000) ? 1 : -1) : -1

  eq1st7 =
    bExist111 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100 || nzflag101 || nzflag110) ? 7 : -1) :
    bExist110 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100 || nzflag101) ? 6 : -1) :
    bExist101 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011 || nzflag100) ? 5 : -1) :
    bExist100 ? (!(nzflag000 || nzflag001 || nzflag010 || nzflag011) ? 4 : -1) :
    bExist011 ? (!(nzflag000 || nzflag001 || nzflag010) ? 3 : -1) :
    bExist010 ? (!(nzflag000 || nzflag001) ? 2 : -1) :
    bExist001 ? (!(nzflag000) ? 1 : -1) : -1
```

10.3.14 权重压缩位流三维同一及标签树语义

节点非零标志 **uni_tgt_nzflag**

二值变量。值为‘1’表示节点值为非零；值为‘0’表示节点值为零。

索引值非零标志 **uni_tgt_nzflag_index**

二值变量。值为‘1’表示标签树索引值为非零；值为‘0’表示标签树索引值为零。

索引值 **uni_tgt_index**

表示标签树索引值。

差值索引值 **uni_tgt_delta_index**

表示父节点索引值和当前节点索引值之间的差值。

量化值非零标志 **uni_tgt_nzflag_q**

二值变量。值为‘1’表示量化值为非零；值为‘0’表示量化值为零。

量化值绝对值 **uni_tgt_abs_q**

表示量化值的绝对值。

量化值差值绝对值 **uni_tgt_delta_abs_q**

表示父节点量化值和当前节点量化值之间差值的绝对值。

量化值符号位 **uni_tgt_sign_q**

使用直接量化，则对其相应的非零系数的符号位进行编码。

在定义之内的变量按下面的方法计算：

```
height = Tgt[treeDepth + 1][0].size()
width = Tgt[treeDepth + 1][0][0].size()
eq1st1= eq1st2= eq1st3=-1
bExist00 = (nextYIdx < height && nextXIdx < width)
bExist01 = (nextYIdx < height && nextXIdx + 1 < width)
bExist10 = (nextYIdx + 1 < height && nextXIdx < width)
bExist11 = (nextYIdx + 1 < height && nextXIdx + 1 < width)
```

当(treeDepth >= startDepth)的时候，按下面的方法计算变量:

```
eq1st1 =
  bExist11 ? (!(eqflag00 || eqflag01 || eqflag10) ? 3 : -1) :
  bExist10 ? (!(eqflag00 || eqflag01) ? 2 : -1) :
  bExist01 ? (!(eqflag00) ? 1 : -1) : -1
eq1st2 =
  bExist11 ? (!(eqflag00 || eqflag01 || eqflag10) ? 3 : -1) :
  bExist10 ? (!(eqflag00 || eqflag01) ? 2 : -1) :
  bExist01 ? (!(eqflag00) ? 1 : -1) : -1
eq1st3 =
  bExist11 ? (!(eqflag00 || eqflag01 || eqflag10) ? 3 : -1) :
  bExist10 ? (!(eqflag00 || eqflag01) ? 2 : -1) :
  bExist01 ? (!(eqflag00) ? 1 : -1) : -1
```

10.3.15 权重压缩位流三维逸出语义

量化值非零标志 **esc_nzflag**

值变量。值为‘1’表示量化值为非零；值为‘0’表示量化值为零。

量化值符号位 **esc_sign**

二值变量。值为‘1’表示量化值为负；值为‘0’表示量化值为正。

量化值绝对值 **esc_abs_q**

表示量化值的绝对值。

10.4 权重压缩位流解析过程

10.4.1 概述

权重压缩位流中所有的语法元素均为ae(v)型。进行解析前，首先初始化所有二元符号模型和熵编码解码器，见10.4.2。然后从位流中依次解析得到二元符号串，见10.4.3.3。最后根据二元符号串得到ae(v)描述的语法元素的值，见10.4.3.4。

10.4.2 初始化

10.4.2.1 初始化二元符号模型

解码器应保存全部二元符号模型，每个二元符号模型ctx需要初始化三个变量mps、cycno和lgPmps。mps的位宽为1位，cycno的位宽为2位，lgPmps的位宽为10位。mps和cycno的值应初始化为0；lgPmps的值应初始化为1023。

10.4.2.2 初始化熵编码解码器

rS1、rT1、bFlag、cFlag、valueS、boundS、valueT和valueD是用于熵编码解码器的变量。boundS是一个大于0的整数。valueD的值为0或1，bFlag的值为0或1，cFlag的值为0或1。valueS和boundS的位宽是大于或等于Log(boundS+1)的最小整数，rS1的位宽是大于或等于Log(boundS+2)的最小整数，rT1的位宽是8位，valueT的位宽是9位。rS1的值应初始化为0；rT1的值应初始化为0xFF。如果boundS的值为254，则valueS和rS1的位宽是8位。

注：valueS记录了预先连续读进多少位才会使valueT的最高位为1。这一功能可用于快速读取位流及并行解码。在极端的状况下，valueS的值有可能超过16位可表示的范围。boundS限制了连续读进‘0’的个数，从而对valueS的位宽进行合理的控制。

valueS、valueT和valueD的初始化过程用伪代码描述如下：

```
valueS = 0
valueT = read_bits(9)
valueD = 1
```

10.4.3 二元符号串解析

10.4.3.1 概述

解析二元符号串的步骤如下。

a) 设二元符号的索引号 binIdx 的值为-1，二元符号串为空。

b) binIdx 的值加 1，然后进行以下操作：

如果当前二元符号是以下语法元素的二元符号串之一，置 BypassFlag 的值为 1：

```
end_of_last_layer_flag;
end_of_last_layer_ctu_flag;
integer_input;
total_trainable_layer;
enable_escape_reorder;
```

enable_zdep_reorder;
enable_max_ctu3d_size;
max_ctu3d_idx;
array1d_depth;
total_sublayer;
sublayer_cmaxw;
sublayer_dim;
sublayer_shape;
include_bias_array1d;
sublayer_scan_order;
sublayer_bitdepth;
bias_abs_q

c)解析当前二元符号（见10.4.3.2 ）。

d)将由步骤 c 得到的二元符号加入二元符号串的尾部，得到更新的二元符号串。

e)将由步骤 d 得到的二元符号串与 10.4.3.3 中对应的进行比较。如果该二元符号串与表格中某个二元符号串相匹配，则完成二元符号串的解析；否则回到步骤 b，继续解析下一个二元符号。

10.4.3.2 导出二元符号模型

二元符号模型ctx等于ctxArray[ctxIdx]，其中ctxArray是保存二元符号模型的数组，ctxIdx是数组的索引值。语法元素的每个二元符号的ctxIdx等于ctxIdxInc加ctxIdxStart。各语法元素对应的ctxIdxStart及每个二元符号对应的ctxIdxInc见表337。

表 337 语法元素对应的 ctxIdxStart 和 ctxIdxInc

语法元素	ctxIdxInc	ctxIdxStart	ctx数量
bias_sign	0	0	3
bias_nz_flag	0	3	3
split_flag	0	6	3
ctu3d_map_mode_flag	0	9	3
map_mode_flag	1	9	3
enable_start_depth	0	12	3
cu3d_map_mode	0	15	3
abs_predicted_diff	binIdx = (binIdx < 23) ? binIdx + 1 : 23	18	48
predicted_sign	0	66	3
predicted_flag	0	69	3
signalled_size	0	72	3
nzflag_delta	0	75	3
sign_delta	0	78	3

表 337 语法元素对应的 ctxIdxStart 和 ctxIdxInc（续）

语法元素	ctxIdxInc	ctxIdxStart	ctx数量
abs_delta	sign = (sign_delta == 1) binIdx = (binIdx < 46 + sign) ? binIdx + 2 : 46 + sign	81	48
oct_cbook_esc_mode	0	129	3
uni_mode	0	132	3
oct_start_depth_delta	binIdx=(binIdx)?1 : 0	135	3

oct_nzflag	if (treeDepth < Oct.size() - 1){
	binIdx = (NzFlagP[0] == NzFlagP[1]) ?
	NzFlagP[0] : 2
	}
	else{
	binIdx = (CoefP[0] < 0) ? 0 :
	(CoefP[0] == 0) ? 1 : 2
	}
		138	9

oct_sign	binIdx = (CoefP[0] != 0 && CoefP[1] != 0) ? 0 : (CoefP[0] == 0 && CoefP[1] == 0) ? 1 : 2	147	3
oct_index	binIdx = (binIdx < 46) ? binIdx + 2 : 46	150	48
oct_abs_q	sign = (oct_sign == 1) binIdx = (binIdx < 46 + sign) ? binIdx + 2 : 46 + sign	198	48
uni_nzflag	0	246	3
uni_map_nzflag	binIdx = (NzFlagP[0] == NzFlagP[1]) ? NzFlagP[0] : 2	249	9
uni_sign	binIdx = (CoefP[0] < 0) ? 0 : (CoefP[0] == 0) ?1 : 2	258	3
uni_cmap_val	sign = 0 binIdx = (binIdx < 46 + sign) ? binIdx + 2 : 46 + sign	261	48
uni_qmap_val	sign = 0 binIdx = (binIdx < 46 + sign) ? binIdx + 2 : 46 + sign	309	48
tag_cbook_esc_mode	0	357	3
tgt_mode	0	360	3
tag_start_depth_delta	binIdx= binIdx?1:0	360	3
tgtm_nzflag_index	binIdx = (CoefP[0] != 0 && CoefP[1] != 0) ? 0 : (CoefP[0] == 0 && CoefP[1] == 0) ? 1 : 2	363	9
tgtm_sign_q	0	372	3

表 337 语法元素对应的 ctxIdxStart 和 ctxIdxInc（续）

语法元素	ctxIdxInc	ctxIdxStart	ctx数量
tgtm_index	sign = 0 binIdx = (binIdx < 46 + sign) ? binIdx + 2 : 46 + sign	375	48
tgtm_delta_index	binIdx = (binIdx < 23) ? binIdx + 1 : 23	375	48
tgtm_abs_q	sign = 0 binIdx = (binIdx < 46 + sign) ? binIdx + 2 : 46 + sign	423	48
tgtm_delta_abs_q	binIdx = (binIdx < 23) ? binIdx + 1 : 23	423	48
uni_tgt_nzflag	0	471	3
uni_tgt_nzflag_q	0	474	9
uni_tgt_sign_q	0	483	3
uni_tgt_index	sign = 0 binIdx = (binIdx < 46 + sign) ? binIdx + 2 : 46 + sign	486	48
uni_tgt_delta_index	binIdx = (binIdx < 23) ? binIdx + 1 : 23	486	48
uni_tgt_abs_q	sign = 0 binIdx = (binIdx < 46 + sign) ? binIdx + 2 : 46 + sign	534	48
uni_tgt_delta_abs_q	binIdx = (binIdx < 23) ? binIdx + 1 : 23	534	48
esc_nzflag	0	582	3
esc_sign	0	585	3
esc_abs_q	sign = (esc_sign == 1) binIdx = (binIdx < 46 + sign) ? binIdx + 2 : 46 + sign	588	3
reorder_flag	0	591	3
signalled_flag	0	594	48
qval_minus_one	binIdx = (binIdx < 23) ? binIdx + 1 : 23	642	48

10.4.3.3 二元符号解析

10.4.3.3.1 二元符号的解析过程

过程如下：

a) 首先，解析二元符号值 binVal

1)如果 BypassFlag 的值为 1，执行 decode_bypass 过程（见 10.4.3.3.4）；

2)否则，如果 StuffingBitFlag 的值为 1，则执行 decode_aec_stuffing_bit 过程（见10.4.3.3.3）；

3)否则，执行 decode_decision 过程（见 10.4.3.3.2）。

b) 第二步，如果 binVal 的值为 0，则二元符号为‘0’；如果 binVal 的值为 1，则二元符号为‘1’。

10.4.3.3.2 decode_decision

decode_decision过程的输入是bFlag、cFlag、rS1、rT1、valueS、valueT、valueD以及上下文模型ctx。decode_decision过程的输出是二元符号值binVal。令cFlag等于1，decode_decision过程用伪代码描述如下：

```
decode_decision()
{
    predMps = ctx->mps
    lgPmps = ctx->lgPmps >> 2

    if (valueD || (bFlag == 1 && rS1 == boundS ) ) {
        rS1 = 0
        valueS = 0
        while ( valueT < 0x100 && valueS < boundS ) {
            valueS++
            valueT = (valueT << 1 ) | read_bits(1)
        }
        if ( valueT < 0x100 )
            bFlag = 1
        else
            bFlag = 0
        valueT = valueT & 0xFF
    }

        if ( rT1 >= lgPmps ) {
            rS2 = rS1
            rT2 = rT1 - lgPmps
            sFlag = 0
        }
        else {
            rS2 = rS1 + 1
            rT2 = 256 + rT1 - lgPmps
            sFlag = 1
        }

        if( rS2 > valueS || (rS2 == valueS && valueT >= rT2) && bFlag == 0 ) {
            binVal = ! predMps
            if ( sFlag == 0 )
                tRlps = lgPmps
            else
                tRlps = rT1 + lgPmps

        if ( rS2 == valueS )
            valueT = valueT - rT2
        else
            valueT = 256 + ((valueT << 1 ) | read_bits(1)) - rT2
```

```
            while ( tRlps < 0x100 ) {
                tRlps = tRlps << 1
                valueT = (valueT << 1 ) | read_bits(1)
            }
            rT1 = tRlps & 0xFF
            valueD = 1
        }
        else {
            binVal = predMps rS1 = rS2
            rT1 = rT2
            valueD = 0
        }

        if ( cFlag )
        ctx = update_ctx(binVal, ctx)
        return (binVal)
    }
```

10.4.3.3.3 decode_aec_stuffing_bit

decode_aec_stuffing_bit过程的输入是bFlag、cFlag、rS1、rS2、valueS、valueT和valueD。decode_aec_stuffing_bit过程的输出是二元符号值binVal。

令cFlag等于0，ctx->lgPmps等于4，ctx->mps等于0，带入decode_decision过程实现decode_aec_stuffing_bit过程。

10.4.3.3.4 decode_bypass

decode_bypass过程的输入是bFlag、cFlag、rS1、rS2、valueS、valueT和valueD。decode_bypass 过程的输出是二元符号值binVal。

令cFlag等于0，ctx->lgPmps等于1024，ctx->mps等于0，带入decode_decision过程实现decode_bypass过程。

10.4.3.3.5 update_ctx

update_ctx过程的输入是binVal和ctx。update_ctx过程的输出是更新后的ctx。

update_ctx过程用伪代码描述如下：

```
update_ctx( )
{
    if ( ctx->cycno <= 1 )
        cwr = 3
    else if ( ctx->cycno == 2 )
        cwr = 4
    else
```

```
        cwr = 5
    if ( binVal != ctx->mps ) {
        if ( ctx->cycno <= 2 )
            ctx->cycno = ctx->cycno + 1
        else
            ctx->cycno = 3
    }
    else if ( ctx->cycno == 0)
        ctx->cycno = 1
    if ( binVal == ctx->mps )
        ctx->lgPmps = ctx->lgPmps – (ctx->lgPmps >> cwr) - (ctx->lgPmps >> (cwr+2))
    else {
        switch (cwr) {
            case 3:
                ctx->lgPmps = ctx->lgPmps + 197
                break
            case 4:
                ctx->lgPmps = ctx->lgPmps + 95
                break
            default:
                ctx->lgPmps = ctx->lgPmps + 46
        }
        if ( ctx->lgPmps > 1023 ) {
            ctx->lgPmps = 2047 - ctx->lgPmps
            ctx->mps = ! (ctx->mps)
        }
    }
    return (ctx)
}
```

10.4.3.4 反二值化方法

10.4.3.4.1 概述

本条定义语法元素的反二值化方法，见表338。

表 338 语法元素的反二值化方法

语法元素	反二值化方法	
	条	输入参数
integer_input	FL，见10.4.3.4.2	fixedLength = 1
total_trainable_layer	FL，见10.4.3.4.2	fixedLength = 16
enable_escape_reorder	FL，见10.4.3.4.2	fixedLength = 1
enable_zdep_reorder	FL，见10.4.3.4.2	fixedLength = 1
enable_max_ctu3d_size	FL，见10.4.3.4.2	fixedLength = 1
max_ctu3d_idx	FL，见10.4.3.4.2	fixedLength = 2
array1d_depth	FL，见10.4.3.4.2	fixedLength = 5
total_sublayer	FL，见10.4.3.4.2	fixedLength = 4
sublayer_cmaxw	FL，见10.4.3.4.2	fixedLength = 32
sublayer_dim	FL，见10.4.3.4.2	fixedLength = 2
sublayer_shape	FL，见10.4.3.4.2	fixedLength = 16
include_bias_array1d	FL，见10.4.3.4.2	fixedLength = 1
sublayer_scan_order	FL，见10.4.3.4.2	fixedLength = 1
sublayer_bitdepth	FL，见10.4.3.4.2	fixedLength = 5
bias_abs_q	FL，见10.4.3.4.2	fixedLength = array1d_depth
bias_sign	FL，见10.4.3.4.2	fixedLength = 1
bias_nz_flag	FL，见10.4.3.4.2	fixedLength = 1
split_flag	FL，见10.4.3.4.2	fixedLength = 1
ctu3d_map_mode_flag	FL，见10.4.3.4.2	fixedLength = 1
map_mode_flag	FL，见10.4.3.4.2	fixedLength = 1
enable_start_depth	FL，见10.4.3.4.2	fixedLength = 1
cu3d_map_mode	FL，见10.4.3.4.2	fixedLength = 1
abs_predicted_diff	UEGk，见10.4.3.4.6	cMax = 6，k = 0
predicted_sign	FL，见10.4.3.4.2	fixedLength = 1
predicted_flag	FL，见10.4.3.4.2	fixedLength = 1
signalled_size	U，见10.4.3.4.3	
nzflag_delta	FL，见10.4.3.4.2	fixedLength = 1
sign_delta	FL，见10.4.3.4.2	fixedLength = 1
abs_delta	UEGk，见10.4.3.4.6	cMax=6，k=0
oct_cbook_esc_mode	FL，见10.4.3.4.2	fixedLength = 1
uni_mode	FL，见10.4.3.4.2	fixedLength = 1
oct_start_depth_delta	U，见10.4.3.4.3	
oct_nzflag	FL，见10.4.3.4.2	fixedLength = 1
oct_sign	FL，见10.4.3.4.2	fixedLength = 1
oct_index	UEGk，见10.4.3.4.6	cMax=16，k=0
oct_abs_q	UEGk，见10.4.3.4.6	cMax=16，k=0

表 338 语法元素的反二值化方法（续）

语法元素	反二值化方法	
	条	输入参数
uni_nzflag	FL，见10.4.3.4.2	fixedLength = 1
uni_map_nzflag	FL，见10.4.3.4.2	fixedLength = 1
uni_sign	FL，见10.4.3.4.2	fixedLength = 1
uni_cmap_val	UEGk，见10.4.3.4.6	cMax=16，k=0
uni_qmap_val	UEGk，见10.4.3.4.6	cMax=16，k=0
tag_cbook_esc_mode	FL，见10.4.3.4.2	fixedLength = 1
tgt_mode	FL，见10.4.3.4.2	fixedLength = 1
tag_start_depth_delta	U，见10.4.3.4.3	
tgtm_nzflag_index	FL，见10.4.3.4.2	fixedLength = 1
tgtm_sign_q	FL，见10.4.3.4.2	fixedLength = 1
tgtm_index	UEGk，见10.4.3.4.6	cMax=16，k=0
tgtm_delta_index	UEGk，见10.4.3.4.6	cMax=0，k=5
tgtm_abs_q	UEGk，见10.4.3.4.6	cMax=16，k=4
tgtm_delta_abs_q	UEGk，见10.4.3.4.6	cMax=0，k=8
uni_tgt_nzflag	FL，见10.4.3.4.2	fixedLength = 1
uni_tgt_nzflag_q	FL，见10.4.3.4.2	fixedLength = 1
uni_tgt_sign_q	FL，见10.4.3.4.2	fixedLength = 1
uni_tgt_index	UEGk，见10.4.3.4.6	cMax=16，k=0
uni_tgt_delta_index	UEGk，见10.4.3.4.6	cMax=0，k=5
uni_tgt_abs_q	UEGk，见10.4.3.4.6	cMax=16，k=4
uni_tgt_delta_abs_q	UEGk，见10.4.3.4.6	cMax=0，k=8
esc_nzflag	FL，见10.4.3.4.2	fixedLength = 1
esc_sign	FL，见10.4.3.4.2	fixedLength = 1
esc_abs_q	UEGk，见10.4.3.4.6	cMax=16，k=4
reorder_flag	FL，见10.4.3.4.2	fixedLength = 1
signalled_flag	FL，见10.4.3.4.2	fixedLength = 1
qval_minus_one	UEGk，见10.4.3.4.6	cMax=8，k=8

10.4.3.4.2 采用固定长度(FL)的反二值化方法

语法元素的值由fixedLength位无符号整数二元符号串得到。

10.4.3.4.3 采用一元码(U)的反二值化方法

由二元符号串根据表339得到语法元素的值(synElVal)。

表 339 synElVal 的值与二元符号串的关系（一元码）

synElVal	二元符号串					
0	0					
1	1	0				
2	1	1	0			
3	1	1	1	0		
4	1	1	1	1	0	
5	1	1	1	1	1	0
„						
binIdx	0	1	2	3	4	5

10.4.3.4.4 采用截断一元码(TU)的反二值化方法

由二元符号串根据表340得到synElVal的值。

表 340 synElVal 与二元符号串的关系（截断一元码）

synElVal	二元符号串							
0	0							
1	1	0						
2	1	1	0					
3	1	1	1	0				
4	1	1	1	1	0			
5	1	1	1	1	1	0		
„	1	1	1	1	1	1	„	
cMax-1	1	1	1	1	1	1	„	0
cMax	1	1	1	1	1	1	„	1
binIdx	0	1	2	3	4	5	„	cMax-1

10.4.3.4.5 采用 k 阶指数哥伦布码(EGk) 的反二值化方法

解析k阶指数哥伦布码时，首先从位流的当前位置开始寻找第一个‘0’位，并将找到的‘1’位个数记为leadingOneBits，然后根据leadingOneBits和k计算CodeNum。用伪代码描述如下:

```
leadingOneBits = -1;
for ( b = 1; b; leadingOneBits++ )
    b = read_bits(1)
CodeNum = 2^(leadingZeroBits + k) – 2^k + read_bits(leadingOneBits + k)
```

表341给出了0阶、1阶、2阶和3阶指数哥伦布码的结构。指数哥伦布码的位串分为“前缀”和“后缀”两部分。前缀由leadingOneBit个连续的‘1’和一个‘0’构成。后缀由leadingOneBits+k个位构成，即表中的xi串，xi的值为‘0’或‘1’。具体定义见表341。

表 341 k 阶指数哥伦布码表

阶数	码字结构	CodeNum 取值范围
k=0	0	0
	1 0 x0	1～2
	1 1 0 x1 x0	3～6
	1 1 0 x2 x1 x0	7～14
	.....	.....
k=1	0 x0	0～1
	1 0 x1 x0	2～5
	1 1 0 x2 x1 x0	6～13
	1 1 1 0 x3 x2 x1 x0	14～29
	.....	.....
k=2	0 x1 x0	0～3
	1 0 x2 x1 x0	4～11
	1 1 0 x3 x2 x1 x0	12～27
	1 1 1 0 x4 x3 x2 x1 x0	28～59
	.....	.....
k=3	0 x2 x1 x0	0～7
	1 0 x3 x2 x1 x0	8～23
	11 0 x4 x3 x2 x1 x0	24～55
	1 1 0 x5 x4 x3 x2 x1 x0	56～119
	.....	.....

10.4.3.4.6 采用截断一元码串联 k 阶指数哥伦布码(UEGk) 的反二值化方法

解析UEGk时，首先从位流的当前位置开始寻找第一个‘0’位，并将找到的‘1’位个数记为leadingOneBits，然后根据leadingOneBit， k和cMax计算CodeNum。用伪代码描述如下:

```
leadingOneBits = -1;
for ( b = 1; b; leadingOneBits++ )
   b = read_bits(1)
if (leadingOneBits <= cMax)
   CodeNum = leadingOneBits
else {
   leadingOneBits=leadingOneBits-cMax
   CodeNum = 2^(leadingZeroBits + k) – 2^k + read_bits(leadingOneBits + k) + cMax
}
```

10.5 权重压缩位流解码

10.5.1 解码过程

第一步，解码序列头。见 10.5.2。
第二步，依次解码各层：
a) 步骤 2.1，解码层头。见 10.5.3。
b) 步骤 2.2，依次解码各子层。见 10.5.4。
c) 步骤 2.3，如果所有子层已完成，执行第三步；否则，继续执行第 2.2 步。
第三步，如果所有层已完成，结束解码过程；否则，继续执行第二步。

10.5.2 解码序列头

根据max_ctu3d_idx的值， maxCtu3dWidth3d和maxCtu3dHeight3d按下面的方法计算：

```
maxCtu3dWidth3d=maxCtu3dHeight3d=
        (max_ctu3d_idx==0)?64:(max_ctu3d_idx==1)?32:(max_ctu3d_idx==2)?16:8
```

10.5.3 解码层头

1)根据sublayer_shape[sublayerIdx]的值:

WeightShape= sublayer_shape[sublayerIdx]，
表示当前子层权重形状([R][S][C][K])的值。

```
R = sublayer_shape[sublayerIdx][0]
S = sublayer_shape[sublayerIdx][1]
C = sublayer_shape[sublayerIdx][2]
K = sublayer_shape[sublayerIdx][3]
WeightZdepth = WeightShape[0]* WeightShape[1],
```

表示当前子层RS值。

2）根据integer_input，sublayer_cmaxw和sublayer_bitdepth的值:

```
sublayer_stepsize[sublayerIdx]=integer_input ? 1.0f :
    (sublayer_cmaxw[sublayerIdx] / ((1 <<sublayer_bitdepth[sublayerIdx]) - 1))
```

记SlayerBitDepth=sublayer_bitdepth[sublayerIdx]，
记SlayerStepSize =SlayerStepSize [sublayerIdx]， 权重w的反量化值dqw:

```
clipMinimum = -(1 << SlayerBitDepth)
clipMaximum = (1 << SlayerBitDepth) - 1
sign = (w < 0) ? -1 : 1
dqw = abs(w) * SlayerStepSize
dqw = sign * dqw
```

10.5.4 解码各子层

```
记weightDim = sublayer_dim[sublayerIdx]
记sublayerScanOrder = sublayer_scan_order[sublayerIdx]
记includeBias = (sublayerIdx+1<total_sublayer&&WeightDim!=1
                    &&sublayer_dim[sublayerIdx+1]==1)
```

第一步，如果weightDim=1，解码sublayer子层权重偏置一维数组。见10.5.5。
第二步，否则，如果 includeBias=1，
2.1步，首先解码sublayer+1子层权重偏置一维数组。见10.5.5。
2.2步， MaxCtu3dHeight和MaxCtu3dWidth按下面的方法计算：

```
if (enable_max_ctu3d_size) {
```

```
            if(weightDim==4&&WeightShape[2]==1){
                MaxCtu3dWidth=1
                MaxCtu3dHeight=
                    1<<(get_bitdepth(maxCtu3dWidth3d * maxCtu3dHeight3d /
                                        WeightZdepth)-1)
            }
            else if(WeightDim==4&&WeightShape[3]==1){
                MaxCtu3dWidth=
                    1<<(get_bitdepth(maxCtu3dWidth3d * maxCtu3dHeight3d /
                                        WeightZdepth)-1)
                MaxCtu3dHeight=1
            }
            else{
                MaxCtu3dWidth=1<<(get_bitdepth(maxCtu3dWidth3d /
                                                    WeightShape[0])-1)
                MaxCtu3dHeight=1<<(get_bitdepth(maxCtu3dWidth3d /
                                                    WeightShape[1])-1)
            }
        }
        else{
            MaxCtu3dWidth= maxCtu3dWidth3d
            MaxCtu3dHeight= maxCtu3dHeight3d
        }
```

2.3步，然后遍历所有三维编码树CTU3D。

用伪代码描述如下：

```
if(sublayerScanOrder == SCAN_CK){
    for(c=0;c< WeightShape[2];c+=MaxCtu3dHeight){
        for(k=0;k< WeightShape[3];k+=MaxCtu3dWidth){
            初始化三维编码树CTU3D。见10.5.6。
            解码三维编码树CTU3D。见10.5.7。
        }
    }
}
else if (sublayerScanOrder ==SCAN_KC){
    for(k=0;k< WeightShape[3];k+=MaxCtu3dWidth){
        for(c=0;c< WeightShape[2];c+=MaxCtu3dHeight){
            初始化三维编码树CTU3D。见10.5.6。
            解码三维编码树CTU3D。见10.5.7。
        }
    }
}
```

10.5.5 解码子层权重偏置一维数组

记WeightRec1d[n]，表示权重重建值数组。

第一步，根据bias_sign和bias_abs_q，确定权重量化值：

q = bias_sign ? -(bias_abs_q + 1) : bias_abs_q + 1

第二步，反量化确定权重重建值:

WeightRec1d[n]=dequantw(q， SlayerBitDepth， SlayerStepSize)

完整伪代码见表325。

10.5.6 初始化三维编码树 CTU3D

MaxCtuDepth=get_bitdepth(max(MaxCtu3dWidth， MaxCtu3dHeight) / 8)

表示CTU3D的最大bitdepth。

Ctu3dYOff = c

Ctu3dXOff = k

Ctu3dHeight = min(maxCtu3dHeight， WeightShape[2] - Ctu3dYOff)

Ctu3dWidth = min(maxCtu3dWidth， WeightShape[3] - Ctu3dXOff)

MaxCbookSize=31

表示码本最大尺寸。

MaxPredictorSize=64

表示码本预测器最大尺寸。

CTU3D和CU3D的数据结构定义:

[cu3dDepth][cu3dYIdx][cu3dXIdx]

用来定位一个CU3D。

Split[cu3dDepth][cu3dYIdx][cu3dXIdx]

用来存放拆分标志值。

CbookCentroid [cu3dDepth][cu3dYIdx][cu3dXIdx]

表示CU3D一维码本，它的别名是Cbook。

CodeBookPredictor[cu3dDepth][cu3dYIdx][cu3dXIdx]

表示CU3D一维码本预测器，它的别名是Predictor。

CbookPredictor[n]

表示一维码本预测器。

PredictedCbook[cu3dDepth][cu3dYIdx][cu3dXIdx]

表示CU3D预测的一维码本，它的别名是Predicted。

IndexMap[cu3dDepth][cu3dYIdx][cu3dXIdx]

表示CU3D内的三维索引或量化值，它的别名是Map。

MapOctree3d[cu3dDepth][cu3dYIdx][cu3dXIdx]

表示八叉树CU3D内的四维索引或量化值，它的别名是Oct。

MapUnitree3d[cu3dDepth][cu3dYIdx][cu3dXIdx]

表示同一树CU3D内的四维索引或量化值，它的别名是Utree。

MapTagtree3dm[cu3dDepth][cu3dYIdx][cu3dXIdx]

表示标签树CU3D内的四维索引或量化值，它的别名是Tgtm。

MapTagtree3d[cu3dDepth][cu3dYIdx][cu3dXIdx]

表示同一标签树CU3D内的五维索引或量化值，它的别名是Tgt。

这些数据结构需要按下面的方法初始化:

```
minCu3dHeight = MaxCtu3dHeight >> (MaxCtu3dDepth - 1)
minCu3dWidth = MaxCtu3dWidth >> (MaxCtu3dDepth - 1)

create_tensor3d(Split, minCu3dHeight, minCu3dWidth, Ctu3dHeight, Ctu3dWidth)
Ctu3dDepth = Split.size()
create_tensor3d(CbookCentroid, minCu3dHeight, minCu3dWidth, Ctu3dHeight, Ctu3dWidth)
create_tensor3d(CodeBookPredictor,minCu3dHeight,minCu3dWidth,Ctu3dHeight, Ctu3dWidth)
create_tensor3d(PredictedCbook, minCu3dHeight, minCu3dWidth, Ctu3dHeight, Ctu3dWidth)
create_tensor6d(IndexMap,minCu3dHeight,minCu3dWidth,Ctu3dHeight,
                Ctu3dWidth,WeightZdepth)
create_tree3d(MapOctree3d,minCu3dHeight,minCu3dWidth,Ctu3dHeight,
                Ctu3dWidth,WeightZdepth, FALSE)
create_tree3d(MapUnitree3d, minCu3dHeight, minCu3dWidth, Ctu3dHeight,
                Ctu3dWidth,WeightZdepth, FALSE)
create_tree3d(MapTagtree3dm, minCu3dHeight, minCu3dWidth, Ctu3dHeight,
                Ctu3dWidth,WeightZdepth, FALSE)
create_tree3d(MapTagtree3d, minCu3dHeight, minCu3dWidth, Ctu3dHeight,
                Ctu3dWidth,WeightZdepth, TRUE)
```

10.5.7 解码三维编码树 CTU3D

第一步，解码RS数组ZdepArray。见10.5.8。

第二步，三维编码单元CU3D解码。见10.5.9。

10.5.8 解码 RS 数组 ZdepArray

10.5.8.1 概述

对CTU3D中RS个2D[C][K]平面进行重新排序，形成更统一的结构，并用ZdepArray阵列存储重新排序的索引。遍历ZdepArray数组并迭代地分配nextIdx=ZdepArray[nextIidx]。定义一个队列来存储下一个idx的序列。

10.5.8.2 解码过程

第一步，如果 reorder_flag=1， ZdepArray[zidx] = zidx;

第二步，否则，

步2.1，用queue[n]来存储WeightZdepth个signalled_flag。

步2.2，用queue[n]来存储当signalled_flag=1时的索引qval_minus_one+1。

步2.3，遍历ZdepArray数组并迭代地分配nextIdx=ZdepArray[nextIidx]，用queue来存储下一个idx的序列。

完整伪代码见表328。

10.5.9 三维编码单元 CU3D 解码

10.5.9.1 码本生成和编码概述

码本是利用原始的权重系数和量化的权重系数生成的。码本预测器包含先前生成的码本。码本包括可在预测器中找到的索引和在预测器中找不到的索引。预测码本的大小使用当前CU3D的预测大小与以前CU3D预测的大小之间的差异进行编码。预测位图一次编码一位。传送码本的大小进行编码。使用当前值的绝对值和上一个值的绝对值之间的差对传送码本进行编码。

10.5.9.2 解码过程

第一步，解码预测码本见10.5.10。
第二步，解码传送码本见10.5.11。
第三步，如果cu3d_map_mode=1，
步3.1，如果uni_mode = 1，解码三维同一树(unitree3d)见10.5.12。
步3.2，否则，解码三维八叉树(octree3d)见10.5.13。
第四步，否则，
步4.1，如果tag_mode = 1，解码三维标签树(tagtree3d)见10.5.14。
步4.2，否则，解码三维同一及标签树(uni_tagtree3d)见10.5.15。
第五步，解码逸出重建见10.5.16。
完整伪代码见表329。

10.5.10 解码预测码本

第一步，根据abs_predicted_diff，predicted_sign和以前预测码本的大小PrevPredictedSize确定当前预测码本的大小PredictedSize:

```
PredictedSize=(predicted_sign?-(abs_predicted_diff):
                    abs_predicted_diff)+ PrevPredictedSize
```

第二步，根据predicted_flag，使用预测码本来存储预测位图。
完整伪代码见表335。

10.5.11 解码传送码本

第一步，根据传送码本的大小signalled_size，确定当前码本的大小codeBookSize:

```
codeBookSize=PredictedSize+signalled_size
```

第二步，确定当前码本Cbook的预测码本:

```
for (n = 0; n < PredictedSize; ++n)
    Cbook[n] = Predictor[Predicted[n]]
```

第三步，
步3.1，prev=(PredictedSize)?abs(Cbook[PredictedSize -1]):0
步3.2，根据sign_delta和abs_delta，确定当前码本的传送码本中的索引绝对值:

```
Cbook[n]=delta+prev
prev=Cbook[n]
```

步3.3，根据cbook_sign，确定当前码本的传送码本中的索引值:

```
Cbook[n]=(cbook_sign?-(abs(Cbook[n])):abs(Cbook[n]))
```

完整伪代码见表330。

10.5.12 解码三维同一树(unitree3d)

10.5.12.1 三维同一树编码概述

深度优先搜索用于从顶部节点开始扫描三维单元树。编码从第一个深度startDepth开始。如果节点值为0，则对相同值进行编码，仍然扫描其子节点，但跳过对节点值的编码。如果最大深度的节点值为1，则对节点的索引值或量化值进行编码，否则，如果使用直接量化，则对符号位进行编码。

10.5.12.2 解码过程

解码从第一个深度startDepth开始。

第一步，非最大深度节点:

步1.1，根据uni_nzflag，确定三维同一树Utree节点值:

Utree[treeDepth][zsIdx][yIdx][xIdx]= uni_nzflag

步1.2，根据uni_map_nzflag和cbookSize，确定三维同一树Utree节点索引值或量化值mapVal:

如果cbookSize != 0， mapVal=uni_cmap_val表示索引值。

否则，mapVal= uni_qmap_val表示量化值。

第二步，最大深度节点:

根据uni_map_nzflag，cbookSize和uni_index，确定三维同一树Utree节点值:

如果 cbookSize != 0，

Map[zsIdx][yIdx][xIdx]=uni_index表示索引值。

否则，根据uni_sign和uni_abs_q:

Map[zsIdx][yIdx][xIdx]=(uni_sign?-(uni_abs_q):uni_abs_q)表示量化值

第三步，如果使用直接量化，则:

Map[zsIdx][yIdx][xIdx]=(uni_map_sign?-(mapVal):mapVal)

完整伪代码见表331。

10.5.13 解码三维八叉树(octree3d)

10.5.13.1 三维八叉树编码概述

对编码开始深度startDepth，参与编码过程的第一个深度，进行编码。从顶部节点开始，深度优先搜索用于扫描和编码三维八叉树。如果节点值为0，则跳过对子节点的扫描和编码。如果节点值为1，则逐个扫描其子节点。如果除最后一个子节点值外的所有值都是0，则即使扫描最后一个子节点及其子节点，也会跳过最后一个子节点值的编码。如果最大深度的节点值为1，则对节点的索引值或量化值进行编码。

10.5.13.2 解码过程

解码从第一个深度startDepth开始。

第一步，非最大深度节点:

根据oct_nzflag， 确定三维八叉树Oct节点值:

Oct[treeDepth][zsIdx][yIdx][xIdx]=oct_nzflag

第二步，最大深度节点:

根据cbookSize和oct_index，确定三维八叉树Map值:

如果 cbookSize != 0，

Map[zsIdx][yIdx][xIdx]=oct_index表示索引值。

否则， 根据oct_sign和oct_abs_q:

Map[zsIdx][yIdx][xIdx]=(oct_sign?-(oct_abs_q):oct_abs_q)表示量化值

完整伪代码见表332。

10.5.14 解码三维标签树(tagtree3d)

10.5.14.1 三维标签树编码概述

对编码开始深度startDepth，参与编码过程的第一个深度，进行编码。从顶部节点开始，深度优先搜索用于扫描和编码三维标签树。如果节点值为0，则跳过对子节点的扫描和编码。如果节点值为X，则逐个扫描其子节点。如果除最后一个子节点值外的所有值小于X，则即使扫描最后一个子节点及其子节点，也会跳过最后一个子节点值的编码。在编码开始深度，为节点编码节点值，否则，编码父节点值和当前节点值之间的差值。如果节点在最大深度的值为非零且使用直接量化，则对其对应的非零系数的符号位进行编码。

10.5.14.2 解码过程

解码从第一个深度startDepth开始。

第一步，非最大深度节点:

步1.1，如果cbookSize != 0，

步1.1.1，如果深度是startDepth，则根据tgtm_index，确定标签树Tgtm索引值:

Tgtm[treeDepth][zsIdx][yIdx][xIdx]=tgtm_index

步1.1.2，否则，根据tgtm_delta_index，确定标签树Tgtm量化值:

Tgtm[treeDepth][zsIdx][yIdx][xIdx]=

Tgtm[treeDepth-1][zsIdx>>1][yIdx>>1][xIdx>>1]+tgtm_delta_index

步1.2，否则，

步1.2.1，如果深度是startDepth，则根据tgtm_abs_q，确定标签树Tgtm量化值:

Tgtm[treeDepth][zsIdx][yIdx][xIdx]=tgtm_abs_q

步1.2.2，否则，根据tgtm_delta_abs_q，

Tgtm[treeDepth][zsIdx][yIdx][xIdx]=

Tgtm[treeDepth-1][zsIdx>>1][yIdx>>1][xIdx>>1]+tgtm_delta_abs_q

步1.3，如果使用直接量化，则根据tgtm_sign_q:

Tgtm[treeDepth][zsIdx][yIdx][xIdx]=tgtm_sign_q?

-Tgtm[treeDepth][zsIdx][yIdx][xIdx]:Tgtm[treeDepth][zsIdx][yIdx][xIdx]

第二步，最大深度节点，确定三维标签树Map值:

Map[zsIdx][yIdx][xIdx]= Tgtm[treeDepth][zsIdx][yIdx][xIdx]

完整伪代码见表333。

10.5.15 解码三维同一及标签树(uni_tagtree3d)

10.5.15.1 三维同一及标签树编码概述

对语法元素编码开始深度(encoding_start_depth)，参与编码过程的第一个深度，进行编码。从顶部节点开始，深度优先搜索用于扫描和编码二维四叉树。如果unitree3d节点值为0，则使用tagtree3d编码方法对tagtree3d节点值进行编码，仍然扫描其子节点，但跳过节点值的编码。如果unitree3d节点值为1，则使用tagtree3d编码方法对tagtree3d节点值进行编码。如果tagtree3d节点值为0，则跳过对子节点的扫描和编码。如果节点值为X，则逐个扫描其子节点。如果除最后一个子节点值外的所有值小于X，则即使扫描最后一个子节点及其子节点，也会跳过最后一个子节点值的编码。如果节点在最大深度的值为非零且使用直接量化，则对其对应的非零系数的符号位进行编码。重复上述步骤，直到完成二维矩阵序列的编码。

10.5.15.2 解码过程

解码从第一个深度startDepth开始。

第一步，非最大深度节点:

步1.1，根据uniFlag，确定标签树Tgt非零标志:

Tgt[treeDepth][zsIdx][yIdx][xIdx][0]= uniFlag

步1.2，如果cbookSize != 0，

步1.2.1，如果深度是startDepth，则根据uni_tgt_index，确定标签树Tgt索引值:

Tgt[treeDepth][zsIdx][yIdx][xIdx][1]=uni_tgt_index

步1.2.2，否则，根据uni_tgt_delta_index 确定标签树Tgt索引值:

Tgt[treeDepth][zsIdx][yIdx][xIdx] [1] =

tgt[treeDepth -1][zsIdx][yIdx>>1][xIdx>>1][1]+uni_tgt_delta_index

步1.3，否则，

步1.3.1，如果深度是startDepth，则根据uni_tgt_abs_q，确定标签树Tgt量化值:

Tgt[treeDepth][zsIdx][yIdx][xIdx][1]=uni_tgt_abs_q

步1.3.2，否则，根据uni_tgt_delta_abs_q，

Tgt[treeDepth][zsIdx][yIdx][xIdx][1]=

Tgt[treeDepth -1][zsIdx][yIdx>>1][x_idx>>1][1]+uni_tgt_delta_abs_q

步1.4，如果使用直接量化，则根据uni_tgt_sign_q:

Tgt[treeDepth][zsIdx][yIdx][xIdx][1]=Uni_tgtm_sign_q?

-Tgt[treeDepth][zsIdx][yIdx][xIdx][1]:Tgt[treeDepth][zsIdx][yIdx][xIdx][1]

第二步，最大深度节点，确定三维同一及标签树Map值:

Map[zsIdx][yIdx][xIdx]= Tgt[treeDepth][zsIdx][yIdx][xIdx][1]

完整伪代码见表334。

10.5.16 解码逸出重建

记WeightRec3d[zs][y+Cu3dYOff][x+Cu3dXOff]，表示CTU3D权重重建值。

记[z][y][x]，表示CU3D中的坐标。

记zs=ZdepArray[z]，表示重排序的坐标。

记[zs][y+Cu3dYOff][x+Cu3dXOff]，等于CTU3D中的坐标。

CbookEscMode等于0表示没有码本。

CbookEscMode等于1表示码本使用索引Map[zs][y][x]。

CbookEscMode等于2表示码本使用索引Map[zs][y][x]-1。

记escapeIndex=(CbookEscMode==0)?-1:(CbookEscMode==1)?Cbook.size():0

第一步，如果CbookEscMode等于0，确定权重量化值:

q = Map[zs][y][x]

第二步，否则，

步2.1，如果Map[zs][y][x]等于escapeIndex，表示不使用码本:

q=(esc_sign?-esc_abs_q:esc_abs_q)

步2.2，否则使用码本:

q=(CbookEscMode==1)?Cbook[Map[zs][y][x]]:Cbook[Map[zs][y][x]-1]

第三步，反量化确定权重重建值:

WeightRec3d[zs][y+Cu3dYOff][x+Cu3dXOff]=

dequantw(q, SlayerBitDepth, SlayerStepSize)

完整伪代码见表335。

11 模型保护

11.1 模型保护定义

本部分针对模型压缩后的加密方法和过程进行定义。模型加密对应于多个场景，包括但不限于：模型由云端分发传输到边缘域设备或端侧设备；模型在单个设备中由非易失性器件部署到易失性器件。

模型保护为可选方案，流程图如图13,14所示。本章节将对其进行详细描述。

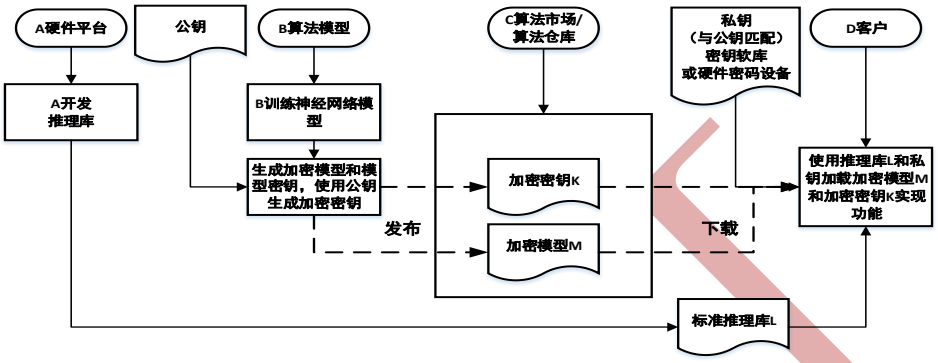


图 13 模型云端分发传输中的加密保护方案

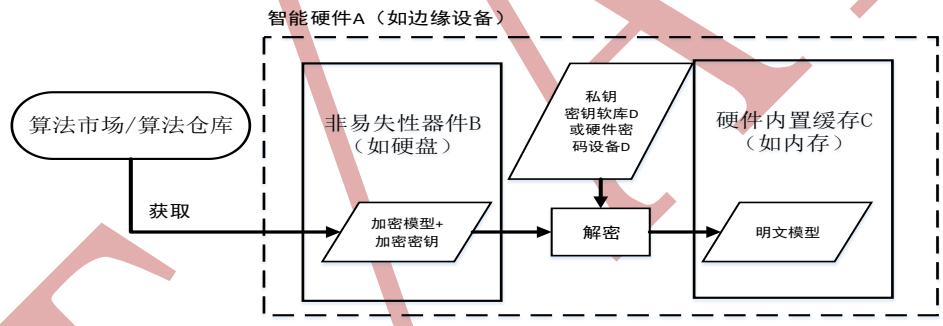


图 14 模型硬件上部署中的加密保护方案

11.2 模型加密过程

模型加密过程定义如下：

a) 采用随机数生成方式生成对称加密所需的工作密钥明文（plainkey）以及初始向量（IV）；

b) 根据输入的公钥和非对称加密方法选择参数（AsymAlgNO）中选择的标准非对称加解密方案，对工作密钥明文进行加密，生成工作密钥密文（key）；

c) 根据使用工作密钥明文和初始向量，采用对称加密方法选择参数（SymAlgNO）中选择的标准对称加解密方案对明文模型数据（PlainModeData）进行加密，生成加密中间数据（MidData）；

d) 将工作密钥密文，初始向量，对称加密方法选择参数，非对称加密方法选择参数，中间数据按照规定格式（详见 11.3）打包成密文模型数据（CipherModelData）输出。具体定义见表 342、343 和表 344。其中，模型加密操作数据定义见表 342。

表 342 模型加密操作数据定义

操作	描述	字段	关键字	定义	数据类型	数据格式
模型加密	对模型文件进行加密	Input	PubKey	公钥	value	Char[]
			PlainModelData	明文模型数据	value	Char[]
			SymAlgNO	对称加解密方法选项	value	Int
			AsymAlgNO	非对称加解密方法选项	value	Int
		Output	CipherModelData	密文模型数据	value	Char[]

对称加密方法选择参数定义见表343。

表 343 对称加密方法选择参数定义

SymAlgNO标号	类型定义
1	AES
2	SM4

非对称加密方法选择参数定义见表344。

表 344 非对称加密方法选择参数定义

AsymAlgNO	类型定义
1	RSA
2	SM2
注1：RSA参见：PKCS#1：RSA加密标准 注2：SM2参见GB/T 35276-2017《信息安全技术 SM2密码算法使用规范》	

模型加密过程伪代码见表345。

表 345 非对称加密方法选择参数定义

模型加密过程	描述符
model_enc(pub_key, plain_model_data, symalgno, asymalgno){	
plainkey = gen_random_data()	
iv = gen_random_data()	
if asymalgno == 1 {	
key = rsa_enc(plainkey)	
}	
elif asymalgno == 2 {	

表 345 非对称加密方法选择参数定义（续）

模型加密过程	描述符
key = sm2_enc(plainkey)	
}	
if symalgno == 1 {	
mid_data = aes_enc(plainkey, iv, plain_model_data)	
}	
elif symalgno == 2 {	
mid_data = sm4_enc(plainkey, iv, plain_model_data)	
}	
encrypt_info.key = key	
encrypt_info.iv = iv	
encrypt_info. symalgno = symalgno	
encrypt_info. asymalgno = asymalgno	
cipher_model_data. encrypt_info = encrypt_info	
cipher_model_data. mid_data = mid_data	
return cipher_model_data	
}	

11.3 模型解密过程

模型解密过程定义如下：

a) 根据规定格式（详见 11.3）从输入的密文模型数据中提取出工作密钥密文，初始向量，对称加密方法选择参数，非对称加密方法选择参数，中间数据；

b) 根据输入的私钥（PrivKey）和非对称加密方法选择参数中选择的标准非对称加解密方案，将工作密钥密文解密生成工作密钥明文；

c) 根据工作密钥明文和初始向量，采用对称加密方法选择参数中选择的标准对称加解密方案中间数据进行解密，得到明文模型数据输出。

具体定义见表 346。

表 346 解密数据定义

操作	描述	字段	关键字	定义	数据类型	数据格式
模型解密	对密文模型文件进行解密	Input	PrivKey	私钥	value	Char[]
			CipherModelData	密文模型数据	value	Char[]
		Output	PlainModelData	明文模型数据	value	Char[]

模型解密过程伪代码见表347。

表 347 模型解密过程伪代码

模型解密过程	描述符
model_enc(priv_key, cipher_model_data){	
key = cipher_model_data.encrypt_info.key	
iv = cipher_model_data.encrypt_info.iv	
asymalgno= cipher_model_data.encrypt_info.asymalgno	
symalgno = cipher_model_data.encrypt_info.symalgno	
mid_data = cipher_model_data.mid_data	
if asymalgno == 1 {	
plainkey = rsa_dec(key)	
}	
elif asymalgno == 2 {	
plainkey = sm2_dec(key)	
}	
if symalgno == 1 {	
plain_model_data = aes_dec(plainkey, iv, mid_data)	
}	
elif symalgno == 2 {	
plain_model_data = sm4_dec(plainkey, iv, mid_data)	
}	
return plain_model_data	
}	

11.4 密文模型数据结构定义

密文模型数据结构定义见表348。

表 348 密文模型数据结构定义

密文模型数据格式定义	描述符
Struct CipherModelData {	
EncryptInfo encrypt_info	
Unsigned char mid_data[MODEL_SIZE]	
}	
Struct EncryptInfo {	
Unsigned char magic[MAGIC_SIZE]	
Unsigned int version	
Unsigned int symalgno	
Unsigned int asymalgno	
Unsigned char iv[IV_SIZE]	
Char key[KEY_SIZE]	
}	