

IEEE Standard for Artificial Intelligence (AI) Model Representation, Compression, Distribution, and Management

IEEE Computer Society

Developed by the
Data Compression Standards Committee

IEEE Std 2941™-2021



STANDARDS

IEEE Standard for Artificial Intelligence (AI) Model Representation, Compression, Distribution, and Management

Developed by the

Data Compression Standards Committee
of the
IEEE Computer Society

Approved 8 December 2021

IEEE SA Standards Board

Abstract: The AI development interface, AI model interoperable representation, coding format, and model encapsulated format for efficient AI model inference, storage, distribution, and management are discussed in this standard.

Keywords: coding format, compact representation, convolution neural networks, encapsulated format, IEEE 2941™, interoperable representation, model compression, neural network representation

The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

Copyright © 2022 by The Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 18 March 2022. Printed in the United States of America.

IEEE is a registered trademark in the U.S. Patent & Trademark Office, owned by The Institute of Electrical and Electronics Engineers, Incorporated.

PDF: ISBN 978-1-5044-8307-0 STD25176
Print: ISBN 978-1-5044-8308-7 STDPD25176

IEEE prohibits discrimination, harassment, and bullying.

For more information, visit <https://www.ieee.org/about/corporate/governance/p9-26.html>.

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

Important Notices and Disclaimers Concerning IEEE Standards Documents

IEEE Standards documents are made available for use subject to important notices and legal disclaimers. These notices and disclaimers, or a reference to this page (<https://standards.ieee.org/ipr/disclaimers.html>), appear in all standards and may be found under the heading “Important Notices and Disclaimers Concerning IEEE Standards Documents.”

Notice and Disclaimer of Liability Concerning the Use of IEEE Standards Documents

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE SA) Standards Board. IEEE develops its standards through an accredited consensus development process, which brings together volunteers representing varied viewpoints and interests to achieve the final product. IEEE Standards are documents developed by volunteers with scientific, academic, and industry-based expertise in technical working groups. Volunteers are not necessarily members of IEEE or IEEE SA, and participate without compensation from IEEE. While IEEE administers the process and establishes rules to promote fairness in the consensus development process, IEEE does not independently evaluate, test, or verify the accuracy of any of the information or the soundness of any judgments contained in its standards.

IEEE does not warrant or represent the accuracy or completeness of the material contained in its standards, and expressly disclaims all warranties (express, implied and statutory) not included in this or any other document relating to the standard, including, but not limited to, the warranties of: merchantability; fitness for a particular purpose; non-infringement; and quality, accuracy, effectiveness, currency, or completeness of material. In addition, IEEE disclaims any and all conditions relating to results and workmanlike effort. In addition, IEEE does not warrant or represent that the use of the material contained in its standards is free from patent infringement. IEEE Standards documents are supplied “AS IS” and “WITH ALL FAULTS.”

Use of an IEEE standard is wholly voluntary. The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard.

In publishing and making its standards available, IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity, nor is IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing any IEEE Standards document, should rely upon his or her own independent judgment in the exercise of reasonable care in any given circumstances or, as appropriate, seek the advice of a competent professional in determining the appropriateness of a given IEEE standard.

IN NO EVENT SHALL IEEE BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO: THE NEED TO PROCURE SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE PUBLICATION, USE OF, OR RELIANCE UPON ANY STANDARD, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE AND REGARDLESS OF WHETHER SUCH DAMAGE WAS FORESEEABLE.

Translations

The IEEE consensus development process involves the review of documents in English only. In the event that an IEEE standard is translated, only the English version published by IEEE is the approved IEEE standard.

Official statements

A statement, written or oral, that is not processed in accordance with the IEEE SA Standards Board Operations Manual shall not be considered or inferred to be the official position of IEEE or any of its committees and shall not be considered to be, nor be relied upon as, a formal position of IEEE. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that the presenter's views should be considered the personal views of that individual rather than the formal position of IEEE, IEEE SA, the Standards Committee, or the Working Group.

Comments on standards

Comments for revision of IEEE Standards documents are welcome from any interested party, regardless of membership affiliation with IEEE or IEEE SA. However, **IEEE does not provide interpretations, consulting information, or advice pertaining to IEEE Standards documents.**

Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Since IEEE standards represent a consensus of concerned interests, it is important that any responses to comments and questions also receive the concurrence of a balance of interests. For this reason, IEEE and the members of its Societies and Standards Coordinating Committees are not able to provide an instant response to comments, or questions except in those cases where the matter has previously been addressed. For the same reason, IEEE does not respond to interpretation requests. Any person who would like to participate in evaluating comments or in revisions to an IEEE standard is welcome to join the relevant IEEE working group. You can indicate interest in a working group using the Interests tab in the Manage Profile & Interests area of the [IEEE SA myProject system](#). An IEEE Account is needed to access the application.

Comments on standards should be submitted using the [Contact Us](#) form.

Laws and regulations

Users of IEEE Standards documents should consult all applicable laws and regulations. Compliance with the provisions of any IEEE Standards document does not constitute compliance to any applicable regulatory requirements. Implementers of the standard are responsible for observing or referring to the applicable regulatory requirements. IEEE does not, by the publication of its standards, intend to urge action that is not in compliance with applicable laws, and these documents may not be construed as doing so.

Data privacy

Users of IEEE Standards documents should evaluate the standards for considerations of data privacy and data ownership in the context of assessing and using the standards in compliance with applicable laws and regulations.

Copyrights

IEEE draft and approved standards are copyrighted by IEEE under US and international copyright laws. They are made available by IEEE and are adopted for a wide variety of both public and private uses. These

include both use, by reference, in laws and regulations, and use in private self-regulation, standardization, and the promotion of engineering practices and methods. By making these documents available for use and adoption by public authorities and private users, IEEE does not waive any rights in copyright to the documents.

Photocopies

Subject to payment of the appropriate licensing fees, IEEE will grant users a limited, non-exclusive license to photocopy portions of any individual standard for company or organizational internal use or individual, non-commercial use only. To arrange for payment of licensing fees, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400; <https://www.copyright.com/>. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Updating of IEEE Standards documents

Users of IEEE Standards documents should be aware that these documents may be superseded at any time by the issuance of new editions or may be amended from time to time through the issuance of amendments, corrigenda, or errata. An official IEEE document at any point in time consists of the current edition of the document together with any amendments, corrigenda, or errata then in effect.

Every IEEE standard is subjected to review at least every 10 years. When a document is more than 10 years old and has not undergone a revision process, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE standard.

In order to determine whether a given document is the current edition and whether it has been amended through the issuance of amendments, corrigenda, or errata, visit [IEEE Xplore](#) or [contact IEEE](#). For more information about the IEEE SA or IEEE's standards development process, visit the IEEE SA Website.

Errata

Errata, if any, for all IEEE standards can be accessed on the [IEEE SA Website](#). Search for standard number and year of approval to access the web page of the published standard. Errata links are located under the Additional Resources Details section. Errata are also available in [IEEE Xplore](#). Users are encouraged to periodically check for errata.

Patents

IEEE Standards are developed in compliance with the [IEEE SA Patent Policy](#).

IMPORTANT NOTICE

IEEE Standards do not guarantee or ensure safety, security, health, or environmental protection, or ensure against interference with or from other devices or networks. IEEE Standards development activities consider research and information presented to the standards development group in developing any safety recommendations. Other information about safety practices, changes in technology or technology implementation, or impact by peripheral systems also may be pertinent to safety considerations during implementation of the standard. Implementers and users of IEEE Standards documents are responsible for determining and complying with all appropriate safety, security, environmental, health, and interference protection practices and all applicable laws and regulations.

Participants

At the time this IEEE standard was completed, the AI Model Representation, Compression, Distribution and Management Working Group had the following membership:

Yonghong Tian, *Chair*

Chao Yang, *Vice Chair*

Guangyao Chen
Jian Cheng
Wenhong Duan
Wen Gao
Haoji Hu
Xiaoguang Hu
Tiejun Huang
Rongrong Ji
Jiajun Jiang
Huixia Li
Xiang Li

Yilin Li
Zheyang Li
Mingbao Lin
Shaohui Lin
Xiaoxu Luan
Yang Luo
Jincheng Ma
Bo Peng
Shiliang Pu
Wenming Tan
Peisong Wang

Wei Wang
Xiaojuan Wang
Yixing Xu
Zhaoyi Yan
Fan Yang
Hengrui Zhao
Yi Zhao
Huihuang Zheng
Xiawu Zheng
Gang Zhong
Wenwu Zhu

The following members of the individual Standards Association balloting group voted on this standard. Balloters may have voted for approval, disapproval, or abstention.

Diego Chiozzi
Piotr Karocki
Alexis Kokkonos
Jun Li
Ting Li
Xiaoru Li
Guojie Luo

Siwei Ma
Cesar Negri
Joanna Olszewska
Sivaraman P
R. K. Rannow
Pablo Rivas Perea
Barnaby Simkin
Walter Struppler

Yonghong Tian
Yaowei Wang
Yu Wang
Feng Wu
Lu Yu
Yu Yuan
Yuyi Zhang

When the IEEE SA Standards Board approved this standard on 8 December 2021, it had the following membership:

Gary Hoffman, *Chair*

Jon Walter Rosdahl, *Vice Chair*

John D. Kulick, *Past Chair*

Konstantinos Karachalios, *Secretary*

Edward A. Addy
Doug Edwards
Ramy Ahmed Fathy
J. Travis Griffith
Thomas Koshy
Joseph L. Koepfinger*
David J. Law

Howard Li
Daozhuang Lin
Kevin Lu
Daleep C. Mohla
Chenhui Niu
Damir Novosel
Annette Reilly
Dorothy Stanley

Mehmet Ulema
Lei Wang
F. Keith Waters
Karl Weber
Sha Wei
Howard Wolfman
Daidi Zhong

*Member Emeritus

Introduction

This introduction is not part of IEEE Std 2941-2021, IEEE Standard for Artificial Intelligence (AI) Model Representation, Compression, Distribution, and Management

The standard satisfies the demand of the unified representation of heterogeneous computing platforms and frameworks, higher efficiency of inference and compression, and secure and credible distribution and management of models, especially on resource-constrained devices and large-scale model distribution learning. Standardization is essential for improving the efficiency of large-scale AI model distribution management and application and resource-constrained devices.

Contents

1. Overview	10
1.1 Scope	10
1.2 Purpose	10
1.3 Word usage	10
2. Normative references.....	11
3. Definitions, acronyms, and abbreviations	11
3.1 Definitions	11
3.2 Acronyms and abbreviations	13
4. Symbols and operators	13
4.1 Arithmetic operators	13
4.2 Logical operator.....	14
4.3 Relational operators	14
4.4 Bitwise operators	15
4.5 Assignment operators	15
5. Framework of convolutional neural network representation and model compression	15
6. Syntax and semantics of neural network models.....	16
6.1 Data structure.....	16
6.2 Syntax description	18
6.3 Semantic description.....	23
6.4 Definition of training operator	70
7. Compression process	74
7.1 Multiple models.....	74
7.2 Quantization	80
7.3 Pruning	96
7.4 Structured matrix	99
8. Decompression process	106
8.1 Multiple models.....	106
8.2 Dequantization.....	112
8.3 Inverse sparsity/inverse pruning operator	120
8.4 Structured matrix	122
9. Data generation.....	130
9.1 Definition.....	130
9.2 Training data generation method	130
9.3 Multiple models.....	137
9.4 Quantization	143
9.5 Pruning	158
9.6 Structured matrix	166
10. Compressed representation of neural network.....	174
10.1 Specification of syntax and semantics	174
10.2 Syntax	179
10.3 Semantics.....	193
10.4 Parsing process	203
10.5 Decoding process.....	214

11. Model protection	221
11.1 Model protection definition	221
11.2 Model encryption process.....	222
11.3 Model decryption process.....	223
11.4 Cipher model data structure definition	224

IEEE Standard for Artificial Intelligence (AI) Model Representation, Compression, Distribution, and Management

1. Overview

1.1 Scope

This standard defines AI development interface, AI model interoperable representation, coding format, and model encapsulated format for efficient AI model inference, storage, distribution, and management.

1.2 Purpose

This standard provides efficient toolsets for inference, compression, decompression, coding, and packaging of the AI model that breaks down the AI model's barriers between different computing architecture and algorithm frameworks, storage cost, as well as protects business-sensitive information. The target applications and services include but are not limited to AI education, AI healthcare, and smart city.

1.3 Word usage

The word *shall* indicates mandatory requirements strictly to be followed in order to conform to the standard and from which no deviation is permitted (*shall* equals *is required to*).^{1, 2}

The word *should* indicates that among several possibilities one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required (*should* equals *is recommended that*).

The word *may* is used to indicate a course of action permissible within the limits of the standard (*may* equals *is permitted to*).

The word *can* is used for statements of possibility and capability, whether material, physical, or causal (*can* equals *is able to*).

¹ The use of the word *must* is deprecated and cannot be used when stating mandatory requirements, *must* is used only to describe unavoidable situations.

² The use of *will* is deprecated and cannot be used when stating mandatory requirements, *will* is only used in statements of fact.

2. Normative references

The following referenced documents are indispensable for the application of this document (i.e., they must be understood and used, so each referenced document is cited in text and its relationship to this document is explained). For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or corrigenda) applies.

IEEE Std 754™, IEEE Standard for Floating-Point Arithmetic.^{3, 4}

IETF RFC 2437, PKCS #1: RSA Cryptography Specifications.⁵

ISO/IEC 8859 Series, Standards for 8-bit character encodings.⁶

ISO/IEC 14888-3, IT Security Techniques—Part 3.

Protocol Buffers: Encoding: Signed Integers.⁷

3. Definitions, acronyms, and abbreviations

3.1 Definitions

For the purposes of this document, the following terms and definitions apply. The *IEEE Standards Dictionary Online* should be consulted for terms not defined in this clause.⁸

basic symbol vector: Including the basic symbol vector coexisting in the entire network, and the basic symbol vector corresponding to each layer, where the basic symbol vector corresponding to each layer is obtained from the basic symbol vector of the network according to a certain agreed rule.

block circulant matrix: The block circulant matrix is a matrix in which each block is a circulant matrix.

block structured matrix: Refers to a matrix that can be divided into multiple blocks, and each block is arranged according to a certain rule.

circulant matrix: The matrix arrangement of a regular structure for the current row/column that is the same as the previous row/column cycle.

compact representation: Expenses are compressed by a variety of neural networks in a compact representation of the serialized format after the algorithm is compressed. Examples: structured matrix, weight sharing among multiple models, sparse matrix of sparse coding, weighted quantization table, decomposition matrix of low-rank decomposition, and bit representation of binary neural network.

NOTE—Refer to the compression process, decompression process, and data generation method for specific definitions. See Clause 7 through Clause 9.

³ The IEEE standards or products referred to in Clause 2 are trademarks owned by The Institute of Electrical and Electronics Engineers, Incorporated.

⁴ IEEE publications are available from The Institute of Electrical and Electronics Engineers (<https://standards.ieee.org/>).

⁵ IETF publications are available from the IETF Administration LLC (<https://ietf.org/>).

⁶ ISO publications are available from the International Organization for Standardization (<https://www.iso.org/>) and the American National Standards Institute (<https://www.ansi.org/>). IEC publications are available from the International Electrotechnical Commission (<https://www.iec.ch>) and the American National Standards Institute (<https://www.ansi.org/>).

⁷ Protocol buffers are available from the Google Developers Group (<https://developers.google.com/protocol-buffers>).

⁸ *IEEE Standards Dictionary Online* is available at: <http://dictionary.ieee.org>. An IEEE Account is required for access to the dictionary, and one can be created at no charge on the dictionary sign-in page.

compressed representation of neural network: The representation of a neural network compressed by compression techniques to reduce the size of the model.

NOTE—Refer to the Compressed Representation of Neural Network for specific definitions (see Clause 10).⁹

convolution kernel granularity and quantization: A quantization method. For the weight of the convolutional layer, the output channel is divided and the three-dimensional convolution kernel is quantized one by one.

convolutional neural network: A neural network with deep structure including convolution calculations

NOTE—It is a common deep learning algorithm in artificial intelligence applications.

correct input vector: The element in the input vector of the network layer is multiplied by the element at the corresponding position in the disturbance vector to obtain the vector.

disturbance vector: Extend the random vector so that its dimension is the dimension of the input vector of the network layer, and the resulting symbol vector.

encapsulation representation: Expenditure interfaces such as security information and identity verification.

NOTE—The specific definition of the reference model protection; see Clause 1.

high precision: The high-precision data or calculations generally are based on float16, float32, float64, or higher precision floating number encoding.

interoperable representation: The definition of the grammar of the neural network, the definition of supported operators, and the definition of the weight format (integer, floating-point and fixed-point).

NOTE—Refer to the syntax and semantics of the neural network model for specific definitions (see Clause 6).

layer: Hierarchical structure in neural network.

NOTE—Each layer of the neural network contains multiple operators, such as the input layer, convolutional layer, and fully connected layer.

list of unique offset vectors: A neural network is used to process multiple different tasks. In addition to the shared weight value, it is also a combination of weight parts that are not shared by multiple tasks.

low precision: The low-precision data or calculations generally are based on 1 to 8 bits integer encoding.

model: Corresponding to a neural network that completes single or multiple tasks.

model compression: A method to reduce the scale of the neural network model and improve the operator and transmission efficiency of the neural network model.

multiple INT4 quantization: A quantization method that quantizes a tensor into a combination of multiple INT4 tensors.

neural network representation: Represents the basic syntax, semantics, operators, hyperparameters, and code stream definition methods of neural networks.

⁹ Notes in text, tables, and figures of a standard are given for information only and do not contain requirements needed to implement this standard.

offset, offset vector: One of the inherent parameters of neural networks.

quantitative scale factor: The scaling ratio between the original tensor and the quantized tensor.

random vector: The basic symbol vector corresponding to a single network layer.

reference random vector: The basic symbol vector for the coexistence of the entire network.

shared bias: The bias shared for multiple tasks, where one neural network handles multiple different tasks.

shared weight, shared weight value, weight shared weight tensor: The weights shared for multiple tasks, where one neural network handles multiple different tasks.

specific offset vector: A neural network is used to process multiple different tasks, and for any task, it does not share the bias part.

structured matrix: A special kind of matrix can form a complete matrix by using less data and certain arrangement rules.

task: Use neural networks to achieve certain functions. Examples: image super resolution and image noise reduction.

unique weight, unique weight value, unique weight tensor: A neural network is used to process multiple different tasks, and for any task, it does not share the weight part.

unique weight list: A neural network is used to process multiple different tasks. In addition to the shared weight value, it is also a combination of weight parts that are not shared by multiple tasks.

weight, weight tensor: The strength of the connection between the different units.

NOTE—Weight is one of the inherent parameters of a neural network.

weight aggregation dimension: Shared weight tensor and special weight vector for concatenation of dimensions.

3.2 Acronyms and abbreviations

CNN	convolutional neural network
DNN	depth neural network (deep neural network)
P	product quantization

4. Symbols and operators

4.1 Arithmetic operators

The following arithmetic operators are adapted to this document:

+	plus
–	minus (binary operator) or neg (unary prefix operator)
×	multiply
a^b	power, representing b th power of a , also representing a superscript

$\wedge$	power
$/$	divisible operations, truncate along the value direction of 0. For example: $7/4$ and $-7/-4$ shall be truncated to 1, $-7/4$ and $7/-4$ shall be truncated to -1
$\frac{a}{b}$	division, without truncation or rounding; Division, without rounding or half adjust
$ $	absolute value, when $x \geq 0$, $ x = x$, when $x < 0$, $ x = -x$
<code>abs()</code>	absolute value
<code>sign()</code>	take sign $\text{sign}(x) = \begin{cases} 1; x \geq 0 \\ -1; x < 0 \end{cases}$
$\sqrt{\quad}$	square root
$\sum_{i=a}^b f(i)$	cumulative sum of the function $f(i)$ when the independent variable i takes all the integer values from a to b (including b)
$\log_{10}$	denary logarithm
$\log_2$	binary logarithm
$\lfloor \quad \rfloor$	round down
<code>exp</code>	exponential function based on natural constant e
residual	refers to the difference between the actual observed value and the estimated value
$*$	conjugate operation

4.2 Logical operator

The following logical operators are adapted to this document:

$ $	logical OR
$\&\&$	logical AND
$!$	logical NOT

4.3 Relational operators

The following relational operators are adapted to this document:

$>$	greater than
$>=$	greater than or equal to
$<$	less than
$<=$	less than or equal to
$=$	equal to
$!=$	not equal to
$\max[,...,]$	maximum value of all the value
$\min[,...,]$	minimum value of all the value

4.4 Bitwise operators

The following bitwise operators are adapted to this document:

&	AND
	OR
~	neg
a>>b	move a expressed in the form of two's complement integer rightward for b places. This operation is defined only when b takes a positive number.
a<<b	move a expressed in the form of two's complement integer leftward for b places. This operation is defined only when b takes a positive number.

4.5 Assignment operators

The following assignment operators are adapted to this document:

=	assignment operator
$x = a...b$	x takes the value from a to b (including b), of which x , a , and b are integers
++	automatic plus, $x++$ is equivalent to $x = x + 1$. When it is used in array subscript, first calculate the variable value before automatic plus operation
--	automatic minus, $x--$ is equivalent to $x = x - 1$. When it is used in array subscript, first calculate the variable value before automatic minus operation
+=	automatic plus assigned value; Example 1: $x += 3$ is equivalent to $x = x + 3$ Example 2: $x += (-3)$ is equivalent to $x = x + (-3)$
-=	automatic minus assigned value Example 1: $x -= 3$ is equivalent to $x = x - 3$ Example 2: $x -= (-3)$ is equivalent to $x = x - (-3)$

5. Framework of convolutional neural network representation and model compression

The framework of convolutional neural network representation and model compression is a four-layer framework to break the barriers between various deep learning algorithm frameworks and promote the development and application of deep learning in resource-constrained end-to-end computing platforms and wearable devices (Figure 1). The four-layer architecture of the neural network representation standard includes the following:

- Interoperable Representation Format:** Clause 6 describes the definition of the syntax and semantics of the convolutional neural network.
- Compact Representation Format:** Clause 7 through Clause 9 describe the method for the compressed neural network through a variety of training data sets, such as the original data set, some data in the original data set, data in other data sets, and generated training data, as well as the serialization format compressed by a variety of neural network compact representation algorithms, such as knowledge distillation, structured matrix, weight sharing among multiple models, a sparse matrix of sparse coding, quantization table of weight quantization, decomposition matrix of low-rank decomposition, and bit representation of binary neural network. Clause 7 describes the compression process of neural networks. Clause 8 describes the decompression process of neural networks. Clause 9 includes methods of data generation for different compression methods.

- c) **Coding Representation Format:** Clause 9 describes the compressed representation of a neural network and its encoding and decoding technology to reduce the size of the model.
- d) **Encapsulation Format:** Clause 10 describes model protection that supports security information and the authentication interface.

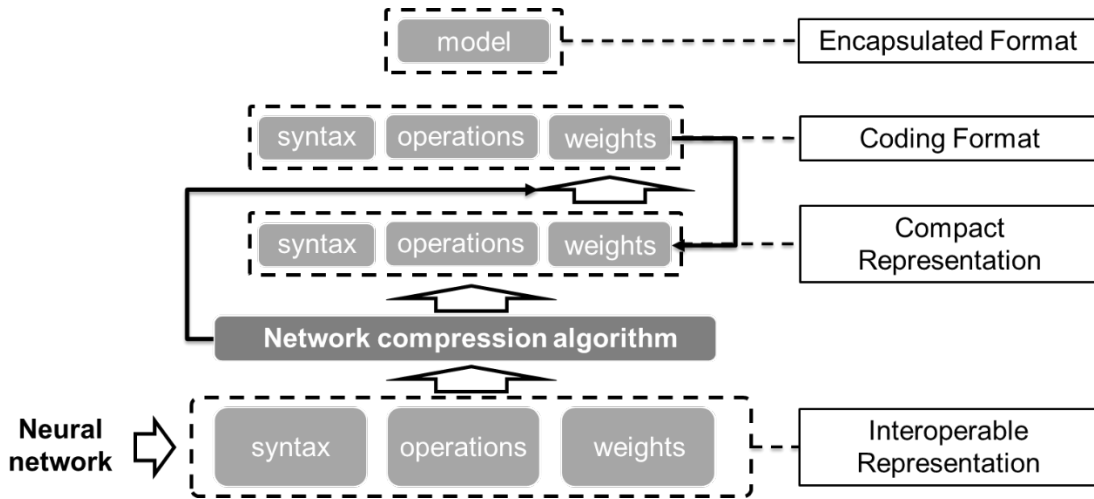


Figure 1—Structure schematic of convolutional neural network representation and model compression

6. Syntax and semantics of neural network models

6.1 Data structure

6.1.1 Data structure of neural network structure

This clause defines the data structure of the neural network structure, where the encoding structure of the calculation graph and model information are shown in Figure 2. Multiple fields are compressed and stored sequentially or nested, where each field uses *tag* and *value* to describe all the information. There are *required*, *repeated*, and *optional* formats for a field, where *required* indicates that the field shall exist, *repeated* indicates that the field could repeat, and *optional* indicates that the field is optional.

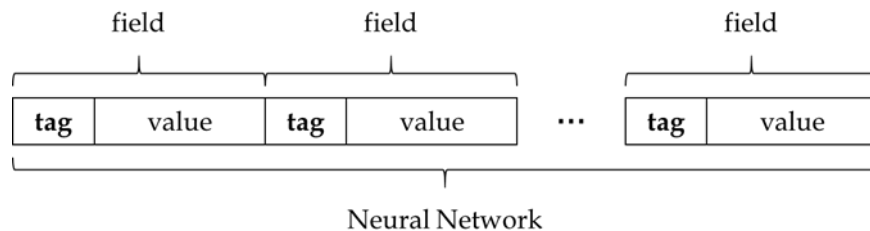


Figure 2—Coding structure of neural network structure

The field for different data types adopt different encoding methods is shown in Figure 3.

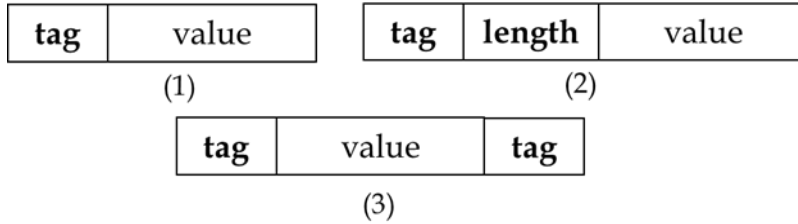


Figure 3—Different types of encoding

a) *Tag coding:*

The coding of the *Tag* adopts $field_number \ll 3 \mid wire_type$, where *field_number* is the label defined for each field and *wire_type* is the data type of the field. Finally, ZigZag encoding for signed integers is adopted as value encoding. ZigZag encoding refers to *Protocol Buffers: Encoding: Signed Integers*.¹⁰

b) *Int32/int64/uint32/uint64/bool/enum encoding:*

These types use the encoding method (1) in Figure 3. The *Tag* is stored in the first byte by the *Tag* encoding method. The *value* is variant encoded and stored in the subsequent bytes.

c) *String encoding:*

This type uses the encoding method (2) in Figure 3. The first byte encodes the *Tag*, the second byte encodes the *length* in characters of the string in a value by int32 or int64, and the remaining bytes store the *value* encoded by UTF-8. UTF-8 refers to ISO/IEC 8859.

d) *Float32/Float64 encoding:*

These types use the encoding method (1) in Figure 3. The first byte encodes the *tag*, and the following bytes directly copy the float memory encoded by IEEE Std 754.

e) *Nested structure coding:*

This type uses the encoding method (1) in Figure 3. The first byte encodes the *tag*, and the *value* uses encoding serialization regardless of whether it is a unit or a tuple.

f) *Repeated coding:*

This type uses the coding scheme (3) in Figure 3. The overall number is calculated by the number of repeated tags. The first byte encodes the tag, and the value uses encoding serialization regardless of whether it is a unit or a tuple.

6.1.2 Data structure of neural network parameters

The parameters of neural network are encoded and stored independently, which is composed of a series of operator parameters. The parameter coding method is shown in Figure 4.

¹⁰ Information on references can be found in Clause 2.

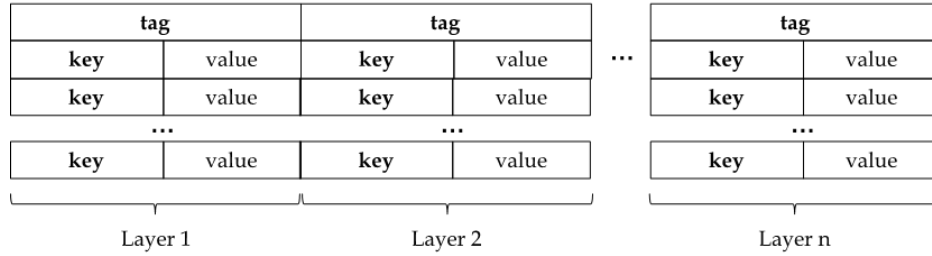


Figure 4—Encoding method of neural network parameters

For each operator or each node of the neural network, the *tag* corresponds to the custom name of each layer. Each operator or node corresponds to multiple fields, representing multiple parameters. The *key* of the field corresponds to the keyword or name of the parameter, and the *value* stores the corresponding array or data.

For the *value* in Figure 4, the encoding method for the array is shown in Figure 5. It includes data types (*dtypes*), matrix dimensions (*dim count*), the size of each dimension (*dimensions*), the stride of each dimension (*strides*), and specific data (*data*). The corresponding value could be searched by index, data type, and the stride of each dimension.

dtype	float32	
dim count	2	
dimensions	3	3
strides	12	4
data	*	

Figure 5—Data structure of array

6.2 Syntax description

6.2.1 Overview

This part defines the grammar of neural network representation, using definitions ranging from coarse grained to fine grained. The basic grammatical description of the neural network representation is constructed from structure definition, computational graph definition, node definition, and so on, to tensor definition and basic data type definition.

Overview of the principles of use: Neural network representation grammar should be implemented in accordance with standard specifications. When completed by a specific computing system (deep learning platform and related software and hardware), the following principles shall be followed:

- a) The definition of the necessary levels should be considered, including the following:
 - 1) Model structure definition
 - 2) Definition of computational graph
 - 3) Node definition
 - 4) Tensor definition
 - 5) Definition of basic data types

- b) When the computing system is implemented, the grammatical elements can be adjusted according to actual needs, including but not limited to, the following:
- 1) Keyword (parameter) naming
 - 2) Operator naming
 - 3) Data type

6.2.2 Definition of model structure

The model structure definition is shown in Table 1.

Table 1—Definition of model structure

Model	Parameter	
Field	Type	Definition
Version	int64	The standard version
contributors	ContributorsList	The list of contributors
framework_name	string	The name of initial training framework
framework_version	string	The version of initial training framework
model_name	string	Model name
model_version	string	Model version
doc_url	string	The document link of model description
graph	graph	The calculation graph of the model

6.2.3 Definition of contributor list

The list of contributors is defined in Table 2.

Table 2—Definition of contributor list

Contributor list	Parameter	
Field	Type	Definition
name	string (repeated)	Contributor's name
email	string (repeated)	Contributor's Email
institute	string (repeated)	Contributor's institution

6.2.4 Definition of computational graph

The definition of the computational graph is shown in Table 3.

The *id* of the computational graph represents the unique serial number of the computational graph. The computational graph may contain several subcomputational graphs, as follows:

- a) The conditional statement, internal operator, and variable node of the loop statement in the neural network representation constitute a subcomputation graph.
- b) Several operator nodes in the neural network may be merged into a new operator node and handed over to a third-party efficient calculation engine (such as *Nvidia TensorRT*, *Intel Ngraph*, etc.) to execute. The new operator node and the corresponding variable node can be expressed by a subcomputation graph.

The *parent_graph_id* of each computational graph points to its parent computational graph, and the *forward_graph_id* points to its forward computational graph. The main computational graph has no corresponding parent computational graph, and its *parent_graph_id* is expressed as -1 . If the computational graph does not have a corresponding forward computational graph, its *forward_graph_id* is expressed as -1 .

Each computational graph contains its own operator nodes and variable nodes. The operator nodes correspond to the calculation operators in the network, and the variable nodes correspond to the variables in the network, including network parameters and temporary variables. The operator node receives a series of variable nodes as input, and outputs a series of variable nodes after one operator.

Table 3—Definition of computational graph

Graph	Parameter	
Field	Type	Definition
Operator_node	OperatorNode (repeated)	Operator nodes of the computational graph
variable_node	VariableNode (repeated)	Variable nodes of the computational graph
id	int64	The unique serial number of the computational graph
parent_graph_id	int64	The sequence number of the parent computational graph corresponding to the child computational graph
forward_graph_id	int64	The sequence number of the forward computational graph corresponding to the subcomputation graph
sub_graphs	Graph (repeated)	List of subgraphs included in the computational graph

6.2.5 Definition of operator node

The operator node definition is shown in Table 4. The *operator* defines the name of the operator node, such as *abs*, *relu*, and so on. *Input* and *output* are the input variable node and output variable node of the operator node. *attribute* is the attributes of the operator node.

Table 4—Definition of operator node

OperatorNode	Parameter	
Field	Type	Definition
name	string	Node's name
operator	string	Operator's name
input	map<string, list(VariableNode)>	Node's input
output	map<string, list(VariableNode)>	Node's output
attribute	map<string, Attribute>	Node's attributes
doc_string	string	Node's description
definition	string	Node's definition

6.2.6 Definition of variable node

Variable node as defined in Table 5. The name of the variable node represents the name of the variable; *variable_type* represents the type of the variable, such as dense tensor or sparse tensor, and so on, which are represented by integer values.

Table 5—Definition of variable node

VariableNode	Parameter	
Field	Type	Definition
name	string	Node's name
variable_type	int64	Node's type

6.2.7 Definition of attribute

The attribute is shown in Table 6.

Table 6—Definition of attribute

Attribute	Specify parameter	
Field	Type	Definition
val	Type	Attribute value
list	list (Type)	Attribute sequence

6.2.8 Definition of other type

Other types are shown in Table 7. *float* and *double* refers to IEEE Std 754.

Table 7—Definition of other type

Type	Parameter	
Field	Type	Definition
i	int64	Integer type
f	float	Single floating point type
d	double	Double floating point type
s	bytes	Bit type
b	bool	Boolean type
vt	vector	Array type
c	callable	Call type
v	ValueType	Tensor data type
t	Tensor	Tensor type
shape	TensorShape	Tensor size type

6.2.9 Definition of tensor type

The definition of tensor data type is shown in Table 8.

Table 8—Definition of tensor type

ValueType	
Label	Type
1	UNDEFINED
2	BINARY
3	INT2
4	INT4
5	INT8
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
twenty one	COMPLEX128

6.2.10 Definition of tensor

The definition of tensor is shown in Table 9.

Table 9—Definition of tensor

Tensor	Specify parameter	
Field	Type	Definition
name	string	Tensor's name
version	string	Tensor's version information
shape	TensorShape	Tensor's size
data_type	ValueType	Tensor's data type
content	ValueType(repeated)	Tensor's data area
int32_data	int32_data	Tensor 32 bit signed integer data
uint32_data	uint32_data	Tensor 32 bit unsigned integer data
int64_data	int64_data	Tensor 64 bit signed integer data
uint64_data	uint64_data	Tensor 64 bit unsigned integer data
float_data	float_data	Tensor single-precision floating point data
double_data	double_data	Tensor double-precision floating-point data
bool_data	bool_data	Tensor boolean data
string_data	string_data	Tensor string data

6.2.11 Definition of tensor size

Tensor size is shown in Table 10.

Table 10—Definition of tensor size

TensorShape	Parameter	
Field	Type	Definition
unknown	bool	Whether there is size information
dim	dimension	Dimension

6.2.12 Definition of dimension

The definition of dimension is given in Table 11.

Table 11—Definition of dimension

Dimension	Parameter	
Field	Type	Definition
Size	int64	Specify dimension
Name	string	Dimension's name

6.3 Semantic description

This subclause defines the operators of the neural network. Each operator includes input, output, type, and attribute, and all the definitions are described in detail.

Principle of use:

The operators contained in the neural network model are implemented by a specific computing system, and the following principles should be followed:

- a) The elements involved in the calculation should be considered, including the following:
 - 1) Supportable parameters
 - 2) Supportable types
- a) Adjustments should be made according to actual needs, including but not limited to, the following:
 - 1) Keyword (parameter) naming
 - 2) Operator naming
 - 3) Data type support range

This subclause defines the operator of the neural network, and specific definitions are shown in Table 12 to Table 149.

Table 12—abs operator

Operator	Description	Field	Keyword	Definition	Data type
abs	Calculate the absolute value of the input data and apply it to each element	Input	X	Input tensor	float16, float32, float64
		Output	Y	Output tensor	float16, float32, float64

Table 13—accuracy operator

Operator	Description	Field	Keyword	Definition	Data type
accuracy	Use inputs and labels to calculate accuracy	Input	X	Input tensor	float32 float64
			label	Data set label	int32 int64
		Output	Y	Output tensor	float32
		Attributes	k	Take K predicted values in each category for calculation	int32 int64
			correct	Number of correct predictions	int32 int64
			total	Total predicted value	int32 int64

Table 14—add operator

Operator	Description	Field	Keyword	Definition	Data type
add	Perform the addition of two inputs and broadcast the parameters on the right to match the size of the input on the left if necessary	Input	X	Input operand 1	float16, float32, float64
			Y	Input operand 2	float16, float32, float64
		Output	Z	Output tensor	float16, float32, float64
		Attributes	axis	Calculated dimension	int
			broadcast	Whether to allow broadcasting	bool

Table 15—affine_channel operator

Operator	Description	Field	Keyword	Definition	Data type
affine_channel	Apply a separate affine transformation to each input channel	Input	X	Input tensor	float32 float64
		Output	Y	Output tensor	float32 float64
		Attributes	scale	Scale factor for affine transformation	float32 float64
			bias	Bias for affine transformation	float32 float64
			data_format	Optional, image channel format, default is “NCHW”	string
			name	Optional, the prefix identifier of the network layer output and weight, the default is None	string
			act	Optional, activation function, default is None	string

Table 16—and operator

Operator	Description	Field	Keyword	Definition	Data type
and	Perform logical operators and operators on the input X and Y	Input	X	Input operand 1	bool
			Y	Input operand 2	bool
		Output	Z	Output tensor	bool
		Attributes	axis	Calculated dimension	int
			broadcast	Whether to allow broadcasting	bool

Table 17—argmax operator

Operator	Description	Field	Keyword	Definition	Data type
argmax	Calculate the index value of the largest element of the input tensor according to the provided dimensions	Input	X	Input tensor	float16, float32, float64
		Output	Y	Output tensor	int64
		Attributes	axis	Calculated dimension	int

Table 18—argmin operator

Operator	Description	Field	Keyword	Definition	Data type
argmin	Calculate the index value of the smallest element of the input tensor according to the provided dimension	Input	X	Input tensor	float16, float32, float64
		Output	Y	Output tensor	int64
		Attributes	axis	Calculated dimension	int

Table 19—argsort operator

Operator	Description	Field	Keyword	Definition	Data type
argsort	Arrange the input variables in ascending order along a given axis, and output the sorted data and the corresponding index, the dimensions of which are the same as the input	Input	X	Input tensor	float32 float64
			Y	Output tensor	float32 float64
		Output	Z	Output index	int64
			axis	Operate the input tensor on the axis	int
		Attributes	descending	Optional, algorithm sorting direction	bool
			name	Optional, the prefix identifier of the network layer output and weight, the default is None	string

Table 20—erf operator

Operator	Description	Field	Keyword	Definition	Data type
erf	Calculate the error function value of each element of the input tensor	Input	X	Input tensor	float32 float64
		Output	Y	Input tensor	float32 float64
		Attributes	name	Optional, the prefix identifier of the network layer output and weight, the default is None	string

Table 21—size operator

Operator	Description	Field	Keyword	Definition	Data type
size	Count the number of elements of the input tensor	Input	X	Input tensor	float32 float64 int32 int64 bool
		Output	Y	Output tensor	int64
		Attributes	name	Optional, the prefix identifier of the network layer output and weight, the default is None	string

Table 22—assign operator

Operator	Description	Field	Keyword	Definition	Data type
assign	Copy input tensor to output tensor	Input	X	Input tensor or numpy array	float32 float64 int32 int64 bool
		Output	Y	Output tensor	float32 float64 int32 int64 bool

Table 23—auc operator

Operator	Description	Field	Keyword	Definition	Data type
auc	Calculation model evaluation index	Input	X	Input tensor	float32 float64
			label	Training data label	int32 int64
		Output	Y	Output tensor	float32 float64
		Attributes	curve	Curve type, the default is ROC	string
			num_thresholds	The number of critical values used when discretizing the ROC curve, the default is 200	int
			topk	Predicted output topk	int
			slide_steps	When calculating batch auc, use not only the current step but also the previous step	int

Table 24—average_pool operator

Operator	Description	Field	Keyword	Definition	Data type
average_pool	Apply average pooling on the input tensor according to kernel_shape, strides, and pad	Input	X	Input tensor	float16, float32, float64
		Output	Y	Output tensor	float16, float32, float64
		Attributes	auto_pad	Valid or Same, and can not be pads used with	string
			kernel_shape	Core size	list of ints
			pads	For the beginning and end, the padding of each dimension	list of ints
			strides	Step size for each dimension	list of ints
			data_format	Input and output data format	string

Table 25—case operator

Operator	Description	Field	Keyword	Definition	Data type
case	Implement if-else statement	Output	Y	Output tensor	float 32 float64 int uint bool
		Attributes	pred_fn_pairs	Pred a Boolean tensor, the Fn is callable a Callable, return tensor	list of (pred, fn)
			default	Optional, callable object	callable
			name	Optional, the prefix identifier of the network layer output and weight, the default is None	string

Table 26—cast operator

Operator	Description	Field	Keyword	Definition	Data type
cast	Convert the input tensor element to the data type pointed to by to	Input	X	Input tensor	all
		Output	Y	Output tensor	all
		Attributes	to	Target type of conversion	string

Table 27—ceil operator

Operator	Description	Field	Keyword	Definition	Data type
ceil	Rounded up	Input	X	Input tensor	float16, float32, float64
		Output	Y	Output tensor	float16, float32, float64

Table 28—concat operator

Operator	Description	Field	Keyword	Definition	Data type
concat	Concatenate the list of input tensors	Input	X	Input tensor list	float16, float32, float64
		Output	Y	Output tensor	float16, float32, float64
		Attributes	axis	Merge dimensions	int

Table 29—constant operator

Operator	Description	Field	Keyword	Definition	Data type
constant	Output a constant value tensor	Output	Output	Output tensor	float16, float32, float64
		Attributes	value	The value of the output tensor element	tensor

Table 30—cond operator

Operator	Description	Field	Keyword	Definition	Data type
cond	If pred is True, return true_fn(), otherwise return false_fn()	Input	pred	Input tensor	bool
		Output	true_fn	Called when pred is true	callable
			false_fn	Called when pred is false	callable
		Attributes	name	Optional, the prefix identifier of the network layer output and weight, the default is None	string

Table 31—conv2d operator

Operator	Description	Field	Keyword	Definition	Data type
conv2d	Construct a two-dimensional convolutional layer, and calculate the output feature map according to the input, filter parameters, step size, filling, and other parameters	Input	X	Input tensor	float16 float32 float64
			W	Weight tensor	float16 float32 float64
			B	Bias tensor	float16 float32 float64
		Output	Y	Output tensor	float16 float32 float64
		Attributes	num_channels	The number of channels of the image	int
			num_filters	Number of filters	int
			filter_size	Filter size	list of ints int
			stride	Optional, step size	list of ints int
			padding	Optional, fill size	list of ints int
			dilation	Optional, expansion coefficient size	list of ints int
			groups	Optional, the number of groups of the two-dimensional convolutional layer	int
			dtype	Optional, output tensor data type	string

Table 32—conv3d operator

Operator	Description	Field	Keyword	Definition	Data type
conv3d	Construct a three-dimensional convolutional layer, and calculate the output feature map according to the input, filter parameters, step size, filling, and other parameters	Input	X	Input tensor	float16 float32 float64
			W	Weight tensor	float16 float32 float64
			B	Bias tensor	float16 float32 float64
		Output	Y	Output tensor	float16 float32 float64
		Attributes	num_channels	The number of channels of the input image	int
			num_filters	Number of filters	int
			filter_size	Filter size	int list of ints
			stride	Optional, step size	int list of ints
			padding	Optional, fill size	int list of ints
			dilation	Optional, expansion ratio	int list of ints
			groups	Optional, the number of groups of the 3D convolutional layer	int
			dtype	Optional, input and output tensor format	string

Table 33—conv2d_transpose operator

Operator	Description	Field	Keyword	Definition	Data type
conv2d_transpose	Construct a two-dimensional convolution transposed layer in the neural network, and output the feature map according to the input, filter parameters, step size, and other parameters	Input	X	Input tensor	float16 float32 float64
			W	Weight tensor	float16 float32 float64
			B	Bias tensor	float16 float32 float64
		Output	Y	Output tensor	float16 float32 float64
		Attributes	num_channels	The number of channels of the image	int
			num_filters	Number of filters	int
			filter_size	Filter size	int list of ints
		Attributes	output_size	Optional, the size of the output feature map	int list of ints
			padding	Optional, fill size	int list of ints
			stride	Optional, step size	int list of ints
			dilation	Optional, expansion ratio	int list of ints
			groups	Optional, the number of groups of the two-dimensional convolutional layer	int
			dtype	Optional, input and output tensor format	string

Table 34—conv3d_transpose operator

Operator	Description	Field	Keyword	Definition	Data type
conv3d_transpose	Construct a three-dimensional convolution transposed layer in the neural network, and output the feature map according to the input, filter parameters, step size, and other parameters	Input	X	Input tensor	float16 float32 float64
			W	Weight tensor	float16 float32 float64
			B	Bias tensor	float16 float32 float64
		Output	Y	Output tensor	float16 float32 float64
		Attributes	num_channels	The number of channels of the input image	int
			num_filters	Number of filters	int
			filter_size	Filter size	int list of ints
			output_size	Optional, the size of the output image	int list of ints
			padding	Padding size	int list of ints
			stride	Step size	int list of ints
			dilation	Optional, expansion ratio	int list of ints
			groups	Optional, the number of groups of the 3D convolutional layer	int
			name	Optional, the prefix identifier of the network layer output and weight, the default is None	string
			dtype	Optional, input and output tensor format	string

Table 35—cos operator

Operator	Description	Field	Keyword	Definition	Data type
cos	Cosine function	Input	X	Input tensor	float32 float64
		Output	Y	Output tensor	float32 float64
		Attributes	name	Optional, the prefix identifier of the network layer output and weight, the default is None	string

Table 36—cos_sim operator

Operator	Description	Field	Keyword	Definition	Data type
cos_sim	Cosine similarity operator	Input	X	Input tensor	float32
			Y	Input tensor	float32
		Output	Z	Output tensor	int float

Table 37—cross_entropy operator

Operator	Description	Field	Keyword	Definition	Data type
cross_entropy	Calculate the cross entropy between the input tensor and the label label	Input	X	Input tensor	float32 float64
			label	Enter the label tensor value corresponding to X	int64
		Output	Y	Output tensor	float32 float64
		Attributes	soft_label	Specify whether the label is a soft label	bool
			ignore_index	Specify an ignored label value, this label value does not participate in the calculation	int

Table 38—crop operator

Operator	Description	Field	Keyword	Definition	Data type
crop	According to the offset and shape, crop the input tensor	Input	X	Input tensor	float32 float64
		Output	Y	Output tensor	float32 float64
		Attributes	shape	Dimensions of the output tensor	list of ints
			offsets	Optional, the offset of the crop in each dimension	list of ints
			name	Optional, used when developers print debugging information	string

Table 39—greater_equal operator

Operator	Description	Field	Keyword	Definition	Data type
greater_equal	Apply the logical operator greater_equal to input X and Y	Input	X	Input tensor	float32 float64 int32 int64
			Y	Input tensor	float32 float64 int32 int64
		Output	Z	Output tensor	bool
		Attributes	axis	Calculated dimension	int64
			broadcast	Whether to allow broadcasting	bool

Table 40—data operator

Operator	Description	Field	Keyword	Definition	Data type
data	The input tensor of the graph, which can be accessed by the operators in the calculation graph. This tensor can be used as a placeholder for data input	Output	Y	Output tensor	bool float16 float32 float64 int8 int16 int32 uint8
		Attributes	shape	Dimensions of the output tensor	list of ints
			dtype	Type of output tensor	string

Table 41—reciprocal operator

Operator	Description	Field	Keyword	Definition	Data type
reciprocal	Calculate the reciprocal of each element in the input tensor	Input	X	Input tensor	float16, float32, float64
		Output	Y	Output tensor	float16, float32, float64
		Attributes	name	Optional, the prefix identifier of the output weight of the network layer, the default is Non e	string

Table 42—depth_to_space operator

Operator	Description	Field	Keyword	Definition	Data type
depth_to_space	Rearrange the input tensor from depth to spatial data	Input	X	Input tensor	float16, float32, float64
		Output	Y	Output tensor	float16, float32, float64
		Attributes	blocksize	The size of the block being moved	int

Table 43—diag operator

Operator	Description	Field	Keyword	Definition	Data type
diag	Create a square matrix and use the input tensor to specify the values of the diagonal elements of the square matrix	Input	X	Input tensor	float32 float64 int32 int64
		Output	Y	Output tensor	float32 float64 int32 int64

Table 44—div operator

Operator	Description	Field	Keyword	Definition	Data type
div	Arithmetic division of input tensors X and Y	Input	X	Input operand 1	float16 float32 float64
			Y	Input operand 2	float16 float32 float64
		Output	Z	Output tensor	float16, float32, float64
		Attributes	axis	Calculated dimension	int
			broadcast	Whether to allow broadcasting	bool

Table 45—dropout operator

Operator	Description	Field	Keyword	Definition	Data type
dropout	Randomly lose part of the input tensor or simply output the input tensor	Input	X	Input tensor	float16, float32, float64
		Output	Y	Output tensor	float16, float32, float64
		Attributes	is_training	Whether to train	bool
			ratio	Random probability (default is 0.5)	float

Table 46—embedding operator

Operator	Description	Field	Keyword	Definition	Data type
embedding	Query the corresponding embedded information from the embedded tensor according to the input information. The function generates a two-dimensional embedded tensor according to the input tensor dimension and data type	Input	X	Input tensor	int64
			W	Weight tensor	float32 float64
		Output	Y	Output tensor	float32 float64
		Attributes	size	Dimensions of the embedding matrix	list of ints
			is_sparse	Whether to use sparse update method	bool
			is_distributed	Whether to store the embedding tensor in a distributed manner	bool
			padding_idx	Set the embedding information corresponding to padding_idx in the input tensor to 0, the default is None	int long none
			dtype	Type of output tensor	string

Table 47—floordiv operator

Operator	Description	Field	Keyword	Definition	Data type
floordiv	Divide the operator element by element, and save the output element at each position to the returned result	Input	X	Input tensor	float32 float64 int32 int64
			Y	Input tensor	float32 float64 int32 int64
		Output	Z	Output tensor	float32 float64 int32 int64
		Attributes	axis	Optional, the index when the Y dimension corresponds to the X dimension	int32
			name	Optional, the prefix identifier of the network layer output and weight, the default is None	string

Table 48—elementwise_max operator

Operator	Description	Field	Keyword	Definition	Data type
elementwise_max	Compare the two input multidimensional tensors element by element, and save the elements with larger positions in the returned result	Input	X	Input tensor	float32 float64 int32 int64
			Y	Input tensor	float32 float64 int32 int64
		Output	Z	Output tensor	float32 float64 int32 int64
		Attributes	axis	The index when the Y dimension corresponds to the X dimension	int32
			name	The prefix identifier of the network layer output and weight, the default is None	string

Table 49—mod operator

Operator	Description	Field	Keyword	Definition	Data type
mod	Take the modulo operator element by element, and save the output element at each position to the returned result	Input	X	Input tensor	float32 float64 int32 int64
			Y	Input tensor	float32 float64 int32 int64
		Output	Z	Output tensor	float32 float64 int32 int64
		Attributes	axis	Optional, the index when the Y dimension corresponds to the X dimension	int32
			name	Optional, the prefix identifier of the network layer output and weight, the default is None	string

Table 50—elementwise_min operator

Operator	Description	Field	Keyword	Definition	Data type
elementwise_min	Compare the two input multidimensional tensors element by element, and save the smaller elements in each position to the returned result	Input	X	Input tensor	float32 float64 int32 int64
			Y	Input tensor	float32 float64 int32 int64
		Output	Z	Output tensor	float32 float64 int32 int64
		Attributes	axis	Optional, the index when the Y dimension corresponds to the X dimension	int32
			name	Optional, the prefix identifier of the network layer output and weight, the default is None	string

Table 51—equal operator

Operator	Description	Field	Keyword	Definition	Data type
equal	Apply the logical operator equal to the input tensors X and Y	Input	X	Input operand 1	float16, float32, float64
			Y	Input operand 2	float16, float32, float64
		Output	Z	Output tensor	bool
		Attributes	axis	Calculated dimension	Int
			broadcast	Whether to allow broadcasting	bool

Table 52—exp operator

Operator	Description	Field	Keyword	Definition	Data type
exp	Return the exp index according to the input tensor element	Input	X	Input tensor	float16, float32, float64
		Output	Y	Output tensor	float16, float32, float64

Table 53—expand operator

Operator	Description	Field	Keyword	Definition	Data type
expand	Copy each dimension of the input according to the parameters, and set the number of times of copying for each dimension of the input.	Input	X	Input tensor	float32 float64 int32 int64 bool
			expand_times	The number of times to expand on each dimension, tensor or constant List	int32
		Output	Y	Output tensor	int32 int64 list of ints
		Attributes	name	Optional, the prefix identifier of the network layer output and weight, the default is None	string

Table 54—flatten operator

Operator	Description	Field	Keyword	Definition	Data type
flatten	Flatten the input tensor into a two-dimensional matrix	Input	X	Input tensor	float16, float32, float64
		Output	Y	Output tensor	float16, float32, float64
		Attributes	axis	Calculated dimension	int

Table 55—floor operator

Operator	Description	Field	Keyword	Definition	Data type
floor	Round down	Input	X	Input tensor	float16, float32, float64
		Output	Y	Output tensor	float16, float32, float64

Table 56—gather operator

Operator	Description	Field	Keyword	Definition	Data type
gather	Return input tensor X Content indexed as indices	Input	X	Input tensor	float16, float32, float64
			indices	Input integer tensor index	int32,int64
		Output	Y	Output tensor	float16, float32, float64
		Attributes	axis	Calculated dimension	int

Table 57—gather_nd operator

Operator	Description	Field	Keyword	Definition	Data type
gather_nd	High-dimensional promotion of gather operator, support multiaxis simultaneous indexing	Input	X	Input tensor	int32 int64 float32 float64 bool
			index	Index tensor	int32 int64
		Output	Y	Output tensor	int32 int64 float32 float64 bool
		Attributes	name	The name of the layer, default is None	string

Table 58—greater operator

Operator	Description	Field	Keyword	Definition	Data type
greater	Apply logical operator greater to input X and Y	Input	X	Input operand 1	Tensor
			Y	Input operand 2	Tensor
		Output	Z	Output tensor	bool
		Attributes	axis	Calculated dimension	int
			broadcast	Whether to allow broadcasting	bool

Table 59—group_norm operator

Operator	Description	Field	Keyword	Definition	Data type
group_norm	Group normalization	Input	X	Input tensor	float32 float64
			W	Weight tensor	float32 float64
			B	Bias tensor	float32 float64
		Output	Y	Output tensor	float32 float64
		Attributes	groups	Number of groups separated from the channel	int32
			epsilon	Optional to help prevent the variance from dividing by zero	float32
			act	Optional, activation function	string
			data_format	Optional, image channel format, default is "NCHW"	string
			name	Optional, the prefix identifier of the network layer output and weight, the default is None	string

Table 60—hard_max operator

Operator	Description	Field	Keyword	Definition	Data type
hard_max	Returns the corresponding size map of the maximum value of a given dimension (1 means the maximum value, 0 means other values)	Input	X	Input tensor	float16, float32, float64
		Output	Y	Output tensor	float16, float32, float64
		Attributes	axis	Calculated dimension	int

Table 61—less operator

Operator	Description	Field	Keyword	Definition	Data type
less	Apply logical operator less to the input tensors X and Y	Input	X	Input operand 1	float16, float32, float64
			Y	Input operand 2	float16, float32, float64
		Output	Z	Output tensor	bool
		Attributes	axis	Calculated dimension	int
			broadcast	Whether to allow broadcasting	bool

Table 62—less_equal operator

Operator	Description	Field	Keyword	Definition	Data type
less_equal	Apply logical operator less_equal to input X and Y	Input	X	Input tensor	float32 float64 int32 int64
			Y	Input tensor	float32 float64 int32 int64
		Output	Z	Output tensor	bool
		Attributes	axis	Calculated dimension	int
			broadcast	Whether to allow broadcasting	bool

Table 63—is_empty operator

Operator	Description	Field	Keyword	Definition	Data type
is_empty	Test if the input tensor is empty	Input	X	Input tensor	float16 float32 float64 int32 int64
		Output	Y	Output tensor	bool
		Attributes	cond	Optional, if this parameter is passed in, the test result of the given x is stored in this parameter	int float

Table 64—`isfinite` operator

Operator	Description	Field	Keyword	Definition	Data type
isfinite	Test whether the input tensor contains infinite values	Input	X	Input tensor	int32 float float64
		Output	Y	Output tensor	bool

Table 65—`label_smooth` operator

Operator	Description	Field	Keyword	Definition	Data type
label_smooth	Label smoothing function	Input	X	Input tensor	float32 float64
		Output	Y	Output tensor	float32 float64
		Attributes	prior_dist	Optional, used to smooth the prior distribution of labels	int float
			epsilon	Optional, mix the weights of the original true distribution and the fixed distribution	float
			dtype	Optional, input tensor type	string
			name	Optional, the prefix identifier of the network layer output and weight, the default is None	string

Table 66—`layer_norm` operator

Operator	Description	Field	Keyword	Definition	Data type
layer_norm	Layer normalization	Input	X	Input tensor	float32 float64
			W	Weight tensor	float32 float64
			B	Bias tensor	float32 float64
		Output	Y	Output tensor	float32 float64
		Attributes	scale	Optional, whether to learn adaptive gain after normalization	bool
			shift	Optional, whether to learn adaptive deviation after normalization	bool
			begin_norm_axis	Optional, normalized execution dimension	int
			epsilon	Optional to help prevent the variance from dividing by zero	float32
			act	Optional, activation function	string
			name	Optional, the prefix identifier of the output weight of the network layer, the default is None	string

Table 67—scatter_nd operator

Operator	Description	Field	Keyword	Definition	Data type
scatter_nd	The index index value in the updated data is added to a new amount of sheets	Input	index	index	int32 int64
			updates	Update tensor	float32 float64
		Output	Y	Output tensor	float32 float64
		Attributes	shape	The shape of the output tensor	List of int
			name	Optional, the prefix identifier of the output weight of the network layer, the default is Non	string

Table 68—log operator

Operator	Description	Field	Keyword	Definition	Data type
log	Calculate the natural logarithm of a given input tensor by element	Input	X	Input tensor	float16, float32, float64
		Output	Y	Output tensor	float16, float32, float64

Table 69—log_loss operator

Operator	Description	Field	Keyword	Definition	Data type
log_loss	Calculate the input prediction result and target label, and return the negative log loss value	Input	X	Input tensor	float32
			label	Enter the label tensor value corresponding to X	float32
		Output	Y	Output tensor	float32
		Attributes	epsilon	Prevent the variance from dividing by zero	float32
			name	Optional, the prefix identifier of the output weight of the network layer, the default is None	string

Table 70—lpnorm operator

Operator	Description	Field	Keyword	Definition	Data type
lpnorm	Apply Lp normalization according to a given dimension (the p value is limited to 1 or 2)	Input	X	Input tensor	float16, float32, float64
		Output	Y	Output tensor	float16, float32, float64
		Attributes	axis	Calculated dimension	int
			p	Normalized p-value (only 1 and 2 are supported)	int

Table 71—lstm operator

Operator	Description	Field	Keyword	Definition	Data type
lstm	Long- and short-term memory operators	Input	X	Input tensor	float32 float64
			weight_ih	Input weight tensor	float32 float64
			weight_hh	Hidden state weight tensor	float32 float64
			bias_ih	Input bias tensor	float32 float64
			bias_hh	Hidden state bias tensor	float32 float64
		Output	rnn_out	LSTM hidden output	float32 float64
			last_h	The hidden state of the last step of LSTM	float32 float64
			last_c	Cell state in the last step of LSTM	float32 float64
		Attributes	max_len	Maximum length of LSTM	int
			input_size	The dimensions of the input tensor	int
			hidden_size	LSTM hidden state dimensions	int
			num_layers	The total number of LSTM layers	int
			dropout_prob	Optional, ignore the ratio	float
			is_bidirectional	Optional, whether it is a two-way LSTM	bool
			time_major	Optional, whether the time dimension is the first	bool
			is_test	Optional, whether in the testing phase	bool

Table 72—mat_mul operator

Operator	Description	Field	Keyword	Definition	Data type
mat_mul	Matrix multiplication	Input	X	Input tensor 1	float16, float32, float64
			Y	Input tensor 2	float16, float32, float64
		Output	Z	Output tensor	float16, float32, float64
		Attributes	transpose_x	Whether to transpose X	bool
			transpose_y	Whether to transpose Y	bool

Table 73—max_pool operator

Operator	Description	Field	Keyword	Definition	Data type
max_pool	Apply the maximum pooling operator to the input tensor X according to kernel_shape, pads and strides	Input	X	Input tensor	float16, float32, float64
		Output	Y	Output tensor	float16, float32, float64
		Attributes	auto_pad	valid or Same, and can not be pads used with	string
			kernel_shape	Convolution kernel size	list of ints
			pads	For the beginning and end, the padding of each dimension	list of ints
			strides	Convolution step size per dimension	list of ints
			return_indices	If equal to True, the serial number of the maximum output value is returned	bool
			date_format	channel_first or channel_last	string

Table 74—max_roi_pool operator

Operator	Description	Field	Keyword	Definition	Data type
max_roi_pool	According to the input tensor X and the region of interest, maximum pooling is applied to each region of interest	Input	X	Input tensor	float16, float32, float64
			rois	Area of interest	float16, float32, float64
		Output	Y	Output tensor	float16, float32, float64
		Attributes	pooled_shape	ROI pool output size	list of ints
			spatial_scale	Size of space	float

Table 75—mul operator

Operator	Description	Field	Keyword	Definition	Data type
mul	Perform arithmetic multiplication	Input	X	Input tensor 1	float16, float32, float64
			Y	Input tensor 2	float16, float32, float64
		Output	Z	Output tensor	float16, float32, float64
		Attributes	broadcast	Whether to allow broadcasting	bool
			axis	Calculated dimension	int

Table 76—negative operator

Operator	Description	Field	Keyword	Definition	Data type
negative	Invert each element of the input tensor	Input	X	Input tensor	float16, float32, float64
		Output	Y	Output tensor	float16, float32, float64

Table 77—not operator

Operator	Description	Field	Keyword	Definition	Data type
not	Return the negation of each input element	Input	X	Input tensor	bool
		Output	Y	Output tensor	bool

Table 78—one_hot operator

Operator	Description	Field	Keyword	Definition	Data type
one_hot	Convert each id in the input to a one_hot tensor	Input	X	Input tensor	int32 int64
		Output	Y	Output tensor	float32
		Attributes	depth	Define the length of the output tensor	int
			allow_out_of_range	Whether the id value contained in the input tensor can be greater than the depth value	bool

Table 79—or operator

Operator	Description	Field	Keyword	Definition	Data type
or	Perform logical operator or to each input element X and Y	Input	X	Input tensor 1	bool
			Y	Input tensor 2	bool
		Output	Z	Output tensor	bool
		Attributes	axis	Calculated dimension	int
			broadcast	Whether to allow broadcasting	bool

Table 80—pad operator

Operator	Description	Field	Keyword	Definition	Data type
pad	Expand the input tensor according to the given dimensions and attributes	Input	X	Input tensor	float16, float32, float64
		Output	Y	Output tensor	float16, float32, float64
		Attributes	mode	Three modes: constant(default), reflect, edge	string
			pads	A list of integers represents the count of fill elements at the beginning and end of each axis	list of ints
			value	Floating point number, which means the value to be filled in	float

Table 81—print operator

Operator	Description	Field	Keyword	Definition	Data type
print	Create a print operator to print the content of the tensor being accessed	Input	X	Input tensor	float16 float32 float64 int32 int64 bool
		Output	Y	Output tensor	float16 float32 float64 int32 int64 bool
		Attributes	summarize	Print the number of elements in the tensor, if the value is –1, print all elements	int
			message	Custom string type message before printing tensor information	string
			first_n	Print the first n values of the tensor	int
			print_tensor_name	Optional, specify whether to print the tensor name, the default is True	bool
			print_tensor_type	Optional, specify whether to print the tensor type, the default is True	bool

Table 82—print operator (*continued*)

Operator	Description	Field	Keyword	Definition	Data type
print	Create a print operator to print the content of the tensor being accessed	Attributes	print_tensor_shape	Optional, specify whether to print the tensor dimension, the default is True	bool
			print_phase	Specify the printing stage, including forward, backward and both, the default is both	string

Table 83—pool2d operator

Operator	Description	Field	Keyword	Definition	Data type
pool2d	Construct a two-dimensional pooling layer in the neural network, and get the output according to the parameter configuration	Input	X	Input tensor	float32 float64
		Output	Y	Output tensor	float32 float64
		Attributes	pool_size	Optional, pooled core size	int list of ints
			pool_type	Optional, pooling type	string
			pool_stride	Optional, the step size of the pooling layer	int list of ints
			pool_padding	Optional, fill size	int list of ints
			global_pooling	Optional, whether to use global pooling	bool
			ceil_mode	Optional, whether to use the ceil function to calculate the output height and width	bool
			exclusive	Optional, whether to ignore the fill value in the average pooling model	bool