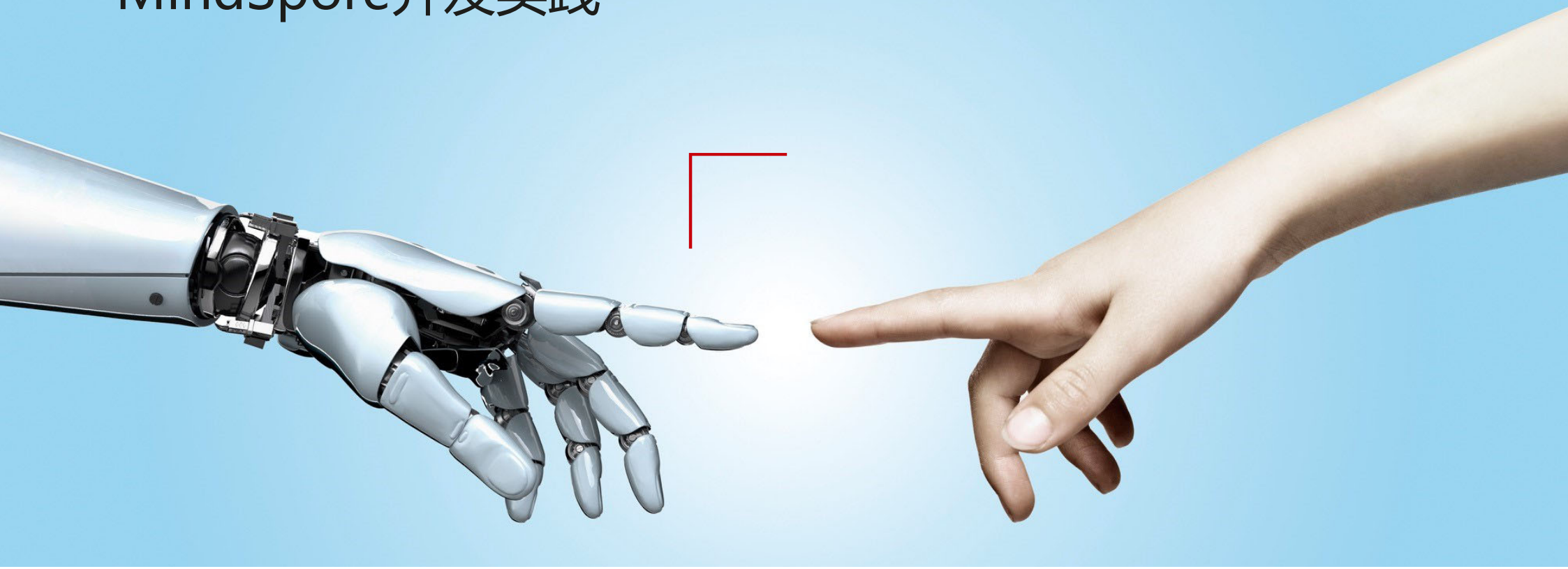
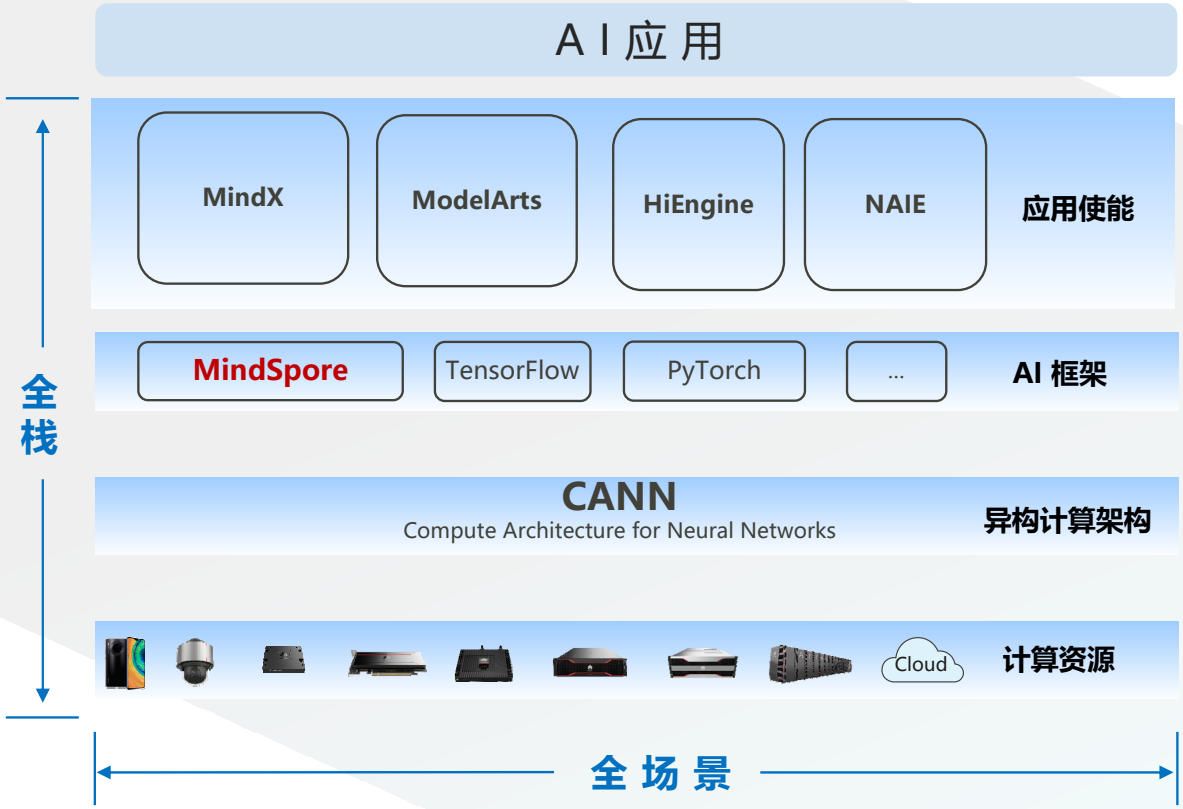
MindSpore开发实践



Security Level:

HUAWEI

华为AI解决方案



应用使能：提供全流程服务和预集成方案

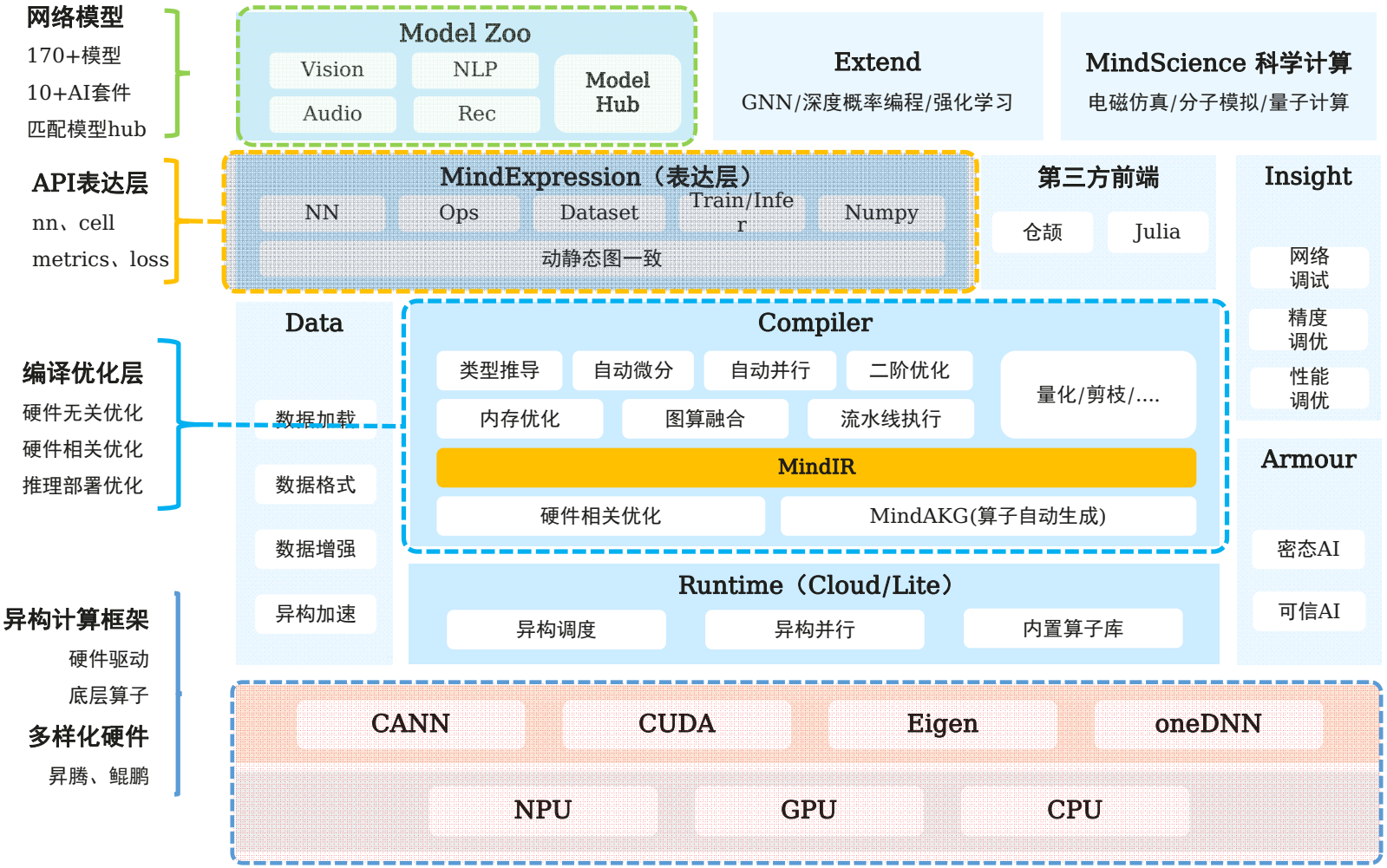
MindSpore
支持端、边、云独立的和协同的统一训练和推理框架

CANN
提供统一高效编程接口，支持用户快速构建基于昇腾平台的AI应用和业务

计算资源
基于统一、可扩展架构的系列化AI IP 和 芯片

MindSpore

MindSpore 全场景AI计算框架架构图



全场景AI计算框架MindSpore

MindSpore 架构特点：

用户态易用；
运行态高效；
部署态灵活；

MindSpore 特性

自动并行：通过自动并行机制、数据pipeline处理等手段降低超大模型训练门槛。

AI+科学计算：支持AI+科学计算的高阶/高纬、多范式编程。

通用计算+DSA：通过图算融合对性能进行优化，自动算子生成技术简化异构（DSA）编程，发挥多样性算力的性能。

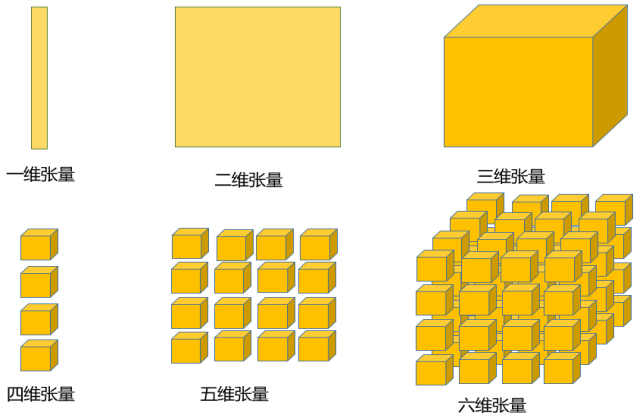
端边云统一的可信架构：解决企业级部署和可信的挑战。

HUAWEI

目录

编程概念：张量（Tensor）

- MindSpore中最基础的数据结构是张量 (Tensor)，是计算的基础：
  - > 是Parameter（权重和偏置）的载体，Feature Map的载体；
  - > 可与numpy.ndarray无缝转换。



1. Tensor声明和使用

```
t1 = Tensor(np.zeros([1, 2, 3]), ms.float32)
assert isinstance(t1, Tensor)
assert t1.shape == (1, 2, 3)
assert t1.dtype == ms.float32
```

2. Tensor是Parameter的数据载体

Parameter的属性：

- default_input: Tensor
- name: str
- requires_grad: bool
- layerwise_parallel: bool

3. Tensor常见的操作

```
// 转换成Numpy数组
asnumpy()
// 获取Tensor的大小
size()
// 获取Tensor的维数
dim()
// 获取Tensor的数据类型
dtype
// 设置Tensor的数据类型
set_dtype()
// 获取Tensor的形状.
shape
```

编程概念：算子（Operation）

常用的Operation：

- array: Array相关的算子
  - ExpandDims
  - Concat
  - OSqueeze
  - nesLike
  - Select
  - StridedSlice
  - ScatterNd
  - ……
- math: 数学计算相关的算子
  - AddN
  - Cos
  - Sub
  - Sin
  - Mul
  - LogicalAnd
  - MatMul
  - LogicalNot
  - RealDiv
  - Less
  - ReduceMean
  - Greater …..
- nn: 网络类算子
  - Conv2d
  - MaxPool
  - Flatten
  - AvgPool
  - Softmax
  - TopK
  - ReLU
  - SoftmaxCrossEntropy
  - Sigmoid
  - SmoothL1Loss
  - Pooling
  - SGD
  - BatchNorm
  - SigmoidCrossEntropy …..
- control: 控制类算子
  - ControlDepend
- other: 其他

```
>>> data1 = Tensor(np.array([[0, 1], [2, 1]]).astype(np.int32))
>>> data2 = Tensor(np.array([[0, 1], [2, 1]]).astype(np.int32))
>>> op = P.Concat()
>>> output = op((data1, data2))
```

```
>>> cos = P.Cos()
>>> input_x = Tensor(np.array([0.24, 0.83, 0.31, 0.09]),
mindspore.float32)
>>> output = cos(input_x)
```

```
>>> input_x = Tensor(np.array([-1, 2, -3, 2, -1]),
mindspore.float16)
>>> relu = nn.ReLU()
>>> relu(input_x)
[0.  2.  0.  2.  0.]
```

编程概念：Cell

- Cell是MindSpore核心编程结构，定义了执行计算的基本模块。Cell对象有以下成员方法：
  > __init__，初始化参数（Parameter），子模块（Cell），算子（Primitive）等组件，进行初始化的校验；
  > construct，定义执行的过程，有一些语法限制。图模式时，会被编译成图来执行；
  > bprop（可选），自定义模块的反向。未定义时，框架会自动生成反向图，计算construct的反向。
- 预定义的Cell
  > nn算子：ReLU, Dense, Conv2d
  > Loss, Optimizer, Metrics
  > Wrapper：TrainOneStepCell, WithLossCell, WithGradCell,

```
class MyNet(nn.Cell):
    def __init__(self, in_channel, out_channel):
        super(MyNet, self).__init__()
        self.fc = nn.Dense(in_channel, out_channel, weight_init='normal',  bias_init='zero', has_bias=True)
        self.relu = nn.ReLU()
    def construct(self, x):
        x = self.fc(x)
        x = self.relu(x)
        return x
```

常用模块

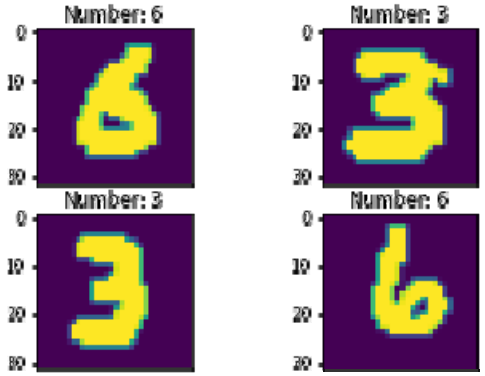
模块	描述
mindspore.dataset	常见Dataset加载和处理接口，如MNIST, CIFAR-10, VOC, COCO, ImageFolder, CelebA等； transforms提供基于OpenCV、PIL及原生实现的数据处理、数据增强接口； text提供文本处理接口；
mindspore.common	Tensor（张量），Parameter（权重、偏置等参数），dtype （数据类型）及Initializer（Parameter初始化）等接口
mindspore.context	设置上下文，如设置Graph和PyNative模式、设备类型、中间图或数据保存、Profiling、设备内存、自动并行等
mindspore.nn	Neural Network常用算子，如Layer（层）、Loss（损失函数）、Optimizer（优化器）、Metrics（验证指标）以及基类Cell和Wrapper
mindspore.ops	原语算子operations（需初始化），组合算子composite，功能算子（已初始化的原语算子）
mindspore.train	Model（训练、验证、推理接口）、callback（损失监控、Checkpoint保存、Summary记录等）， serialization（加载Checkpoint、导出模型等），quant（量化）

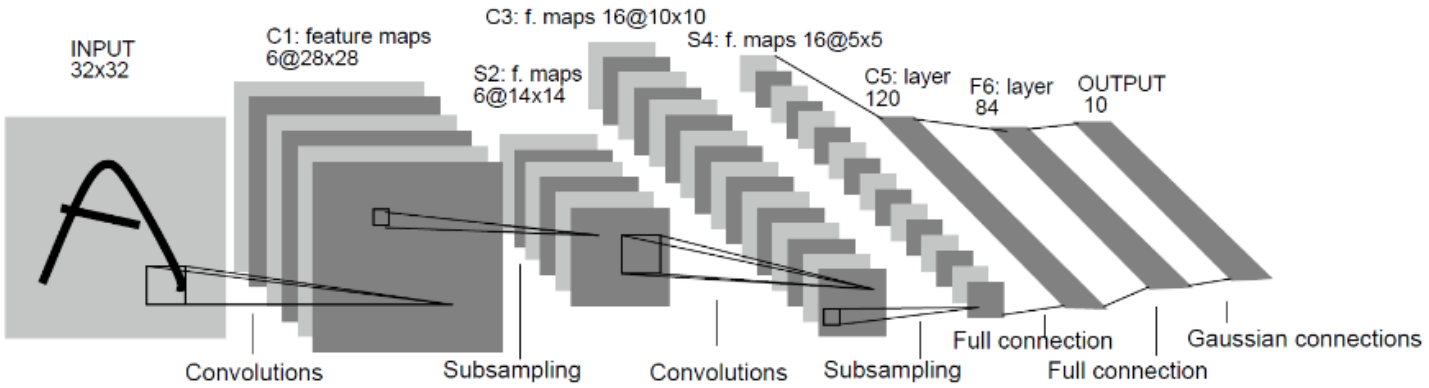
API文档： https://www.mindspore.cn/docs/api/zh-CN/master/index.html

目录

基于LeNet-5的手写数字识别

- MNIST是一个手写数字数据集，训练集包含60000张手写数字，测试集包含10000张手写数字，共10类。
- 卷积神经网络（Convolutional Neural Networks, CNN）的研究始于二十世纪80至90年代，LeNet是最早出现的卷积神经网络之一。
- 在LeNet的基础上，1998年Yann LeCun等构建了卷积神经网络LeNet-5并在手写数字的识别问题中取得成功 。LeNet-5及其后产生的变体定义了现代卷积神经网络的基本结构，可谓入门级神经网络模型。

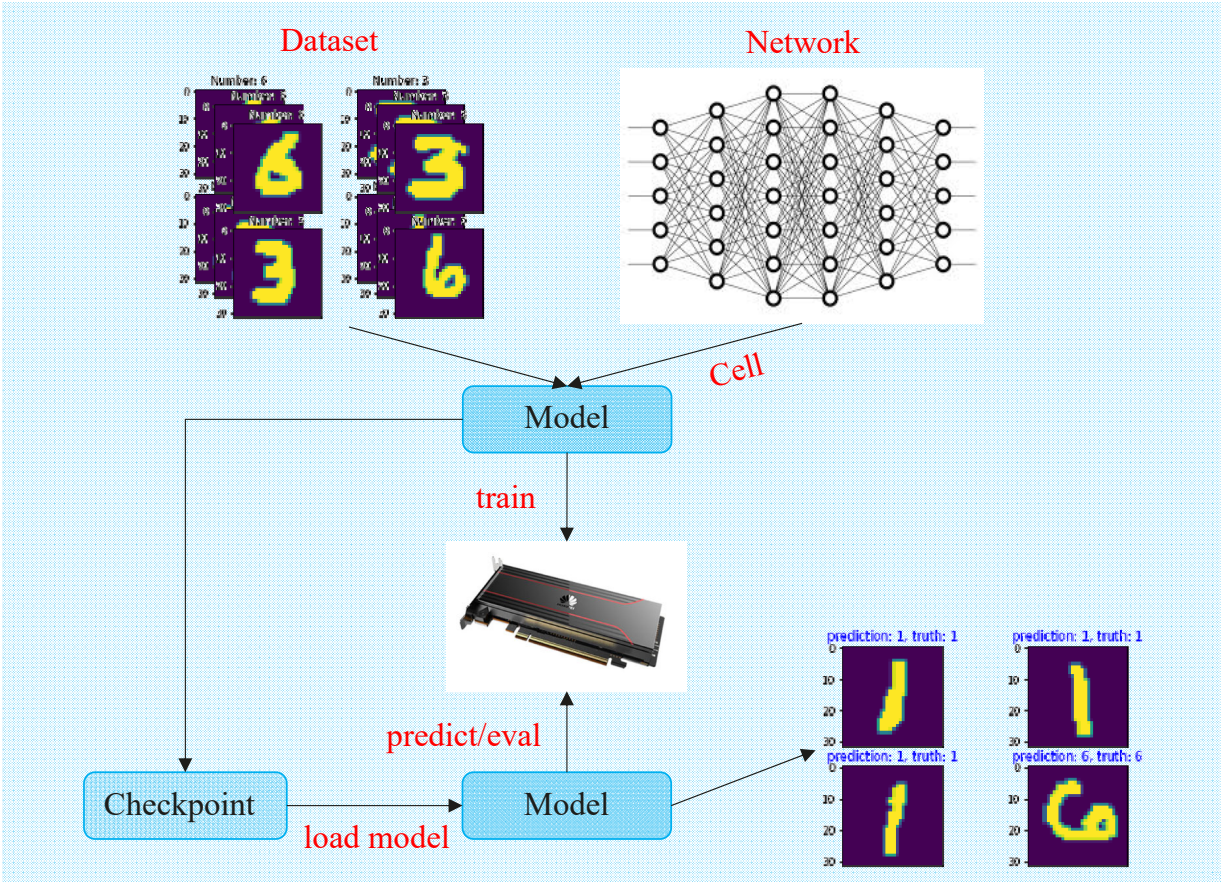




MNIST官网：http://yann.lecun.com/exdb/mnist/
LeNet-5论文：http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

开发流程

- 数据处理
  - > 数据加载
  - > 数据增强
- 模型定义
  - > 定义网络
  - > 损失函数
  - > 优化器
- 模型训练
  - > Loss监控
  - > 验证
  - > 保存Checkpoint
- 模型推理
  - > 推理
  - > 部署



HUAWEI

代码实例

```
import os

import mindspore as ms
import mindspore.context as context
import mindspore.dataset.transforms.c_transforms as C
import mindspore.dataset.transforms.vision.c_transforms as CV

from mindspore import nn
from mindspore.train import Model
from mindspore.train.callback import LossMonitor
```

① 导入标准库、第三方库和MindSpore模块。

```
context.set_context(mode=context.GRAPH_MODE, device_target="Ascend")
```

② 设置上下文：GRAPH模式、Ascend设备。

```
def create_dataset(data_dir, training=True, batch_size=32, resize=(32, 32), rescale=1/(255*0.3081), shift=-0.1307/0.3081,
buffer_size=64):
    data_train = os.path.join(data_dir, 'train')
    data_test = os.path.join(data_dir, 'test')
    ds = ms.dataset.MnistDataset(data_train if training else data_test)

    ds = ds.map(input_columns=["image"], operations=[CV.Resize(resize), CV.Rescale(rescale, shift), CV.HWC2CHW()])
    ds = ds.map(input_columns=["label"], operations=C.TypeCast(ms.int32))
    ds = ds.shuffle(buffer_size=buffer_size).batch(batch_size, drop_remainder=True)

    return ds
```

② 数据处理：数据集加载、缩放、归一化、格式转换、洗牌、批标准化。

```
class LeNet5(nn.Cell):
    def __init__(self):
        super(LeNet5, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, 5, stride=1, pad_mode='valid')
        self.conv2 = nn.Conv2d(6, 16, 5, stride=1, pad_mode='valid')
        self.relu = nn.ReLU()
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.flatten = nn.Flatten()
        self.fc1 = nn.Dense(400, 120)
        self.fc2 = nn.Dense(120, 84)
        self.fc3 = nn.Dense(84, 10)

    def construct(self, x):
        x = self.relu(self.conv1(x))
        x = self.pool(x)
        x = self.relu(self.conv2(x))
        x = self.pool(x)
        x = self.flatten(x)
        x = self.fc1(x)
        x = self.fc2(x)
        x = self.fc3(x)

        return x
```

② 模型定义：算子初始化（参数设置），网络构建。

```
def train(data_dir, lr=0.01, momentum=0.9, num_epochs=3):
    ds_train = create_dataset(data_dir)
    ds_eval = create_dataset(data_dir, training=False)

    net = LeNet5()
    loss = nn.loss.SoftmaxCrossEntropyWithLogits(is_grad=False, sparse=True, reduction='mean')
    opt = nn.Momentum(net.trainable_params(), lr, momentum)
    loss_cb = LossMonitor(per_print_times=ds_train.get_dataset_size())

    model = Model(net, loss, opt, metrics={'acc', 'loss'})
    model.train(num_epochs, ds_train, callbacks=[loss_cb], dataset_sink_mode=False)
    metrics = model.eval(ds_eval, dataset_sink_mode=False)
    print('Metrics:', metrics)
```

② 模型训练和验证：实例化net, loss, optimizer, loss_monitor，调用Model接口进行训练和验证。

目录

数据处理Pipeline

- 数据是深度学习的基础，高质量的数据输入会在整个深度神经网络中起到积极作用。
- 在训练开始之前，由于数据量有限，或者为了得到更好的结果，通常需要进行数据处理与数据增强，以获得能使网络受益的数据输入。

- 数据经过处理和增强，像流经管道的水一样源源不断的流向训练系统。

加载数据集

Dataset structure	使用简介

ImageFolderDatasetV2	# 读取图目录名作为类别标签，每个目录下的图片为同一类
	# Example: cat/image1.jpg, dog/image2.jpg
	ds = ds.ImageFolderDatasetV2(DATA_DIR, class_indexing={"cat":0,"dog":1})

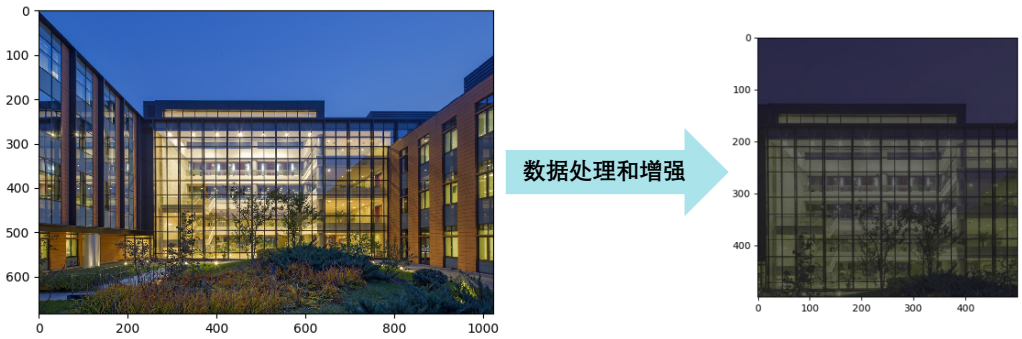
MnistDataset	ds = ms.dataset.MnistDataset(DATA_DIR)
Cifar10Dataset	ds = ms.dataset.Cifar10Dataset(DATA_DIR)
CocoDataset	ds = ds.CocoDataset(DATA_DIR, annotation_file=annotation_file, task='Detection')
MindRecord	ds = ds.MindDataset(FILE_NAME)
GeneratorDataset	ds = ds.GeneratorDataset(GENERATOR, [“image", “label"])

TextFileDataset	dataset_files = ["/path/to/1", "/path/to/2"] # contains 1 or multiple text files
	ds = ds.TextFileDataset(dataset_files=dataset_files)

GraphData	ds = ds.GraphData(DATA_DIR, 2) nodes = ds.get_all_edges(0)

数据处理

- 数据处理
  - 以图片为例：将原始图片进行裁剪、大小缩放、归一化、格式转换等，以便深度学习模型能够使用。
- 数据增强
  - 一种创造 “新”数据的方法，从有限数据中生成“更多数据”，防止过拟合现象



数据增强：有限数据和复杂模型→过拟合

格式转换：MindSpore支持channel first


```
operations = [
    CV.Resize(resize),
    CV.Rescale(rescale, shift),
    CV.HWC2CHW()
]
ds = ds.map(input_columns=["image"],
operations=operations)
ds =
ds.shuffle(buffer_size=buffer_size).batch(batch_s
ize, drop_remainder=True)
```

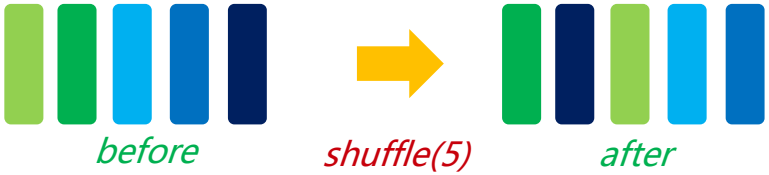

HUAWEI

常用操作

(1)shuffle

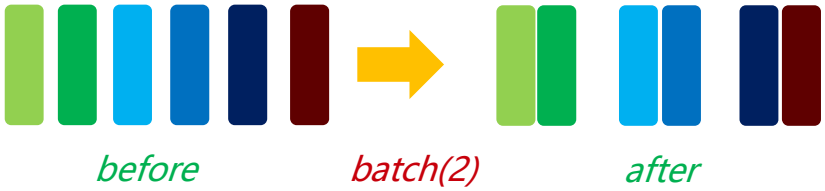
shuffle操作用来打乱数据集中的数据排序。

越大的buffer_size意味着越高的混洗度，但也意味着会花费更多的时间和计算资源。



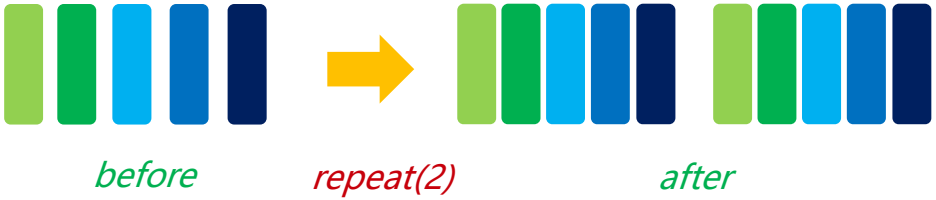
(2)batch

在训练时，数据可能需要分批batch处理。



(3)repeat

可以通过repeat的方式增加训练数据量，通常放在batch操作之后。

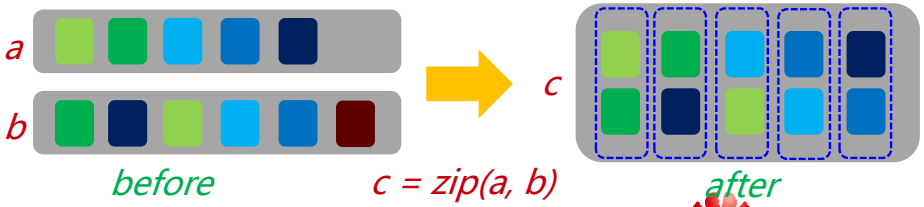


(4)zip

将多个数据集合并成一个。

若两个数据集的列的名字相同，则不能合并。

若两个数据集的行数不同，合并后的行数以较小的为准。



代码实例

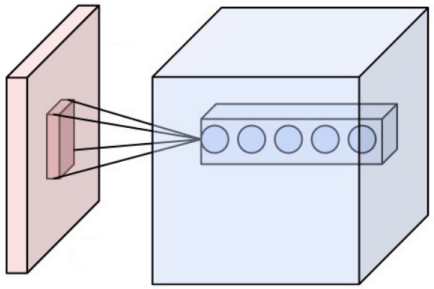
```
def create_dataset(data_dir, training=True, batch_size=32, resize=(32, 32),
                   rescale=1/(255*0.3081), shift=-0.1307/0.3081, buffer_size=64):
    # 训练集和测试集路径
    data_train = os.path.join(data_dir, 'train')
    data_test = os.path.join(data_dir, 'test')
    # 加载MNIST数据集
    ds = ms.dataset.MnistDataset(data_train if training else data_test)

    # 通过map方法对每张图片应用数据处理操作：
    # Resize：缩放图片大小
    # Rescale：将每个像素的灰度值由[0, 255]标准化到[-1, 1]
    # HWC2CHW：将张量格式从NHWC转换成NCHW
    ds = ds.map(input_columns=["image"], operations=[CV.Resize(resize), CV.Rescale(rescale, shift),
CV.HWC2CHW()])
    # 将每个标签的数据类型转换为int32
    ds = ds.map(input_columns=["label"], operations=C.TypeCast(ms.int32))
    # 当在Ascend环境上使用数据下沉模型时，设置`dataset_sink_mode=True`
    ds = ds.shuffle(buffer_size=buffer_size).batch(batch_size, drop_remainder=True)

    return ds
```

目录

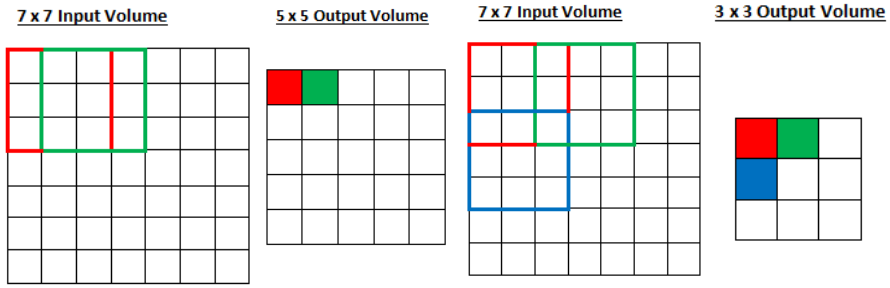
卷积（Convolution）

```
conv1 = nn.Conv2d(in_channels, out_channels,  ①
                  ②kernel_size=3, stride=1, has_bias=False, weight_init='normal', bias_init='zeros',
                  ③pad_mode='same', padding=0)
```

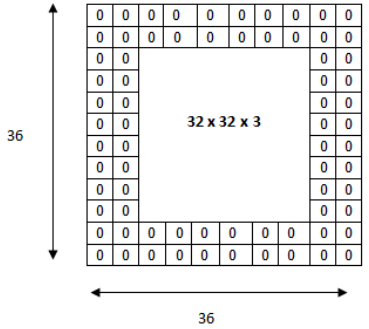
① Input and Output Channels



② Kernel Size and Stride



③ Padding



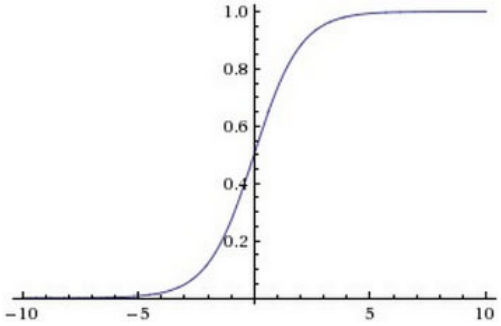
y = conv1(x)

接受一个4维的张量作为输入，返回一个4维的张量作为输出，4维即batch_size x channels x height x width

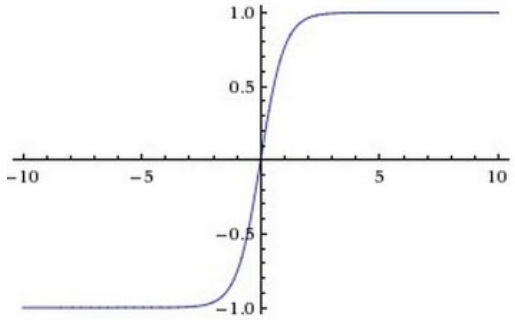
设定的参数决定了输出的特征图的大小：h_out = (h_in - kernel_size + 2*padding)/strides + 1

激活函数

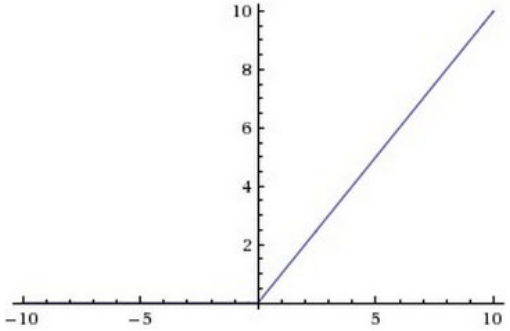
```
relu = nn.Sigmoid()
relu = nn.Tanh()
relu = nn.ReLU()
```



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



$$Tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

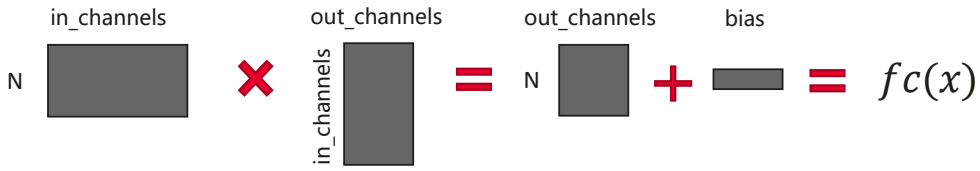


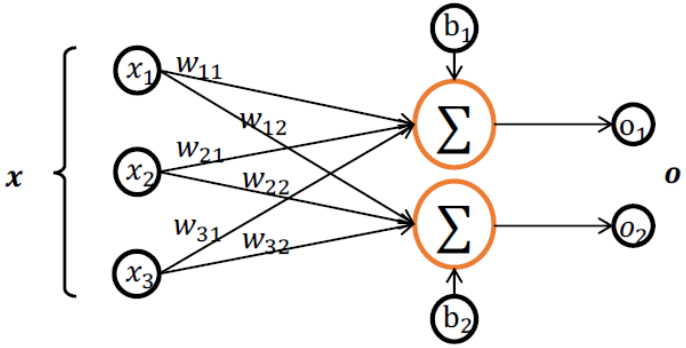
$$ReLU(x) = \max(0, x)$$

在卷积层之后引入激活函数进行非线性变换，可以提高模型的拟合能力。

全连接（Dense）

```
fc1 = nn.Dense(in_channels, out_channels, weight_init='normal', bias_init='zeros', has_bias=True,
activation=None)
```





y = fc1(x)：

- 输入：Tensor shape(batch_size, in_channels).
- 输出：Tensor shape(batch_size, out_channels).

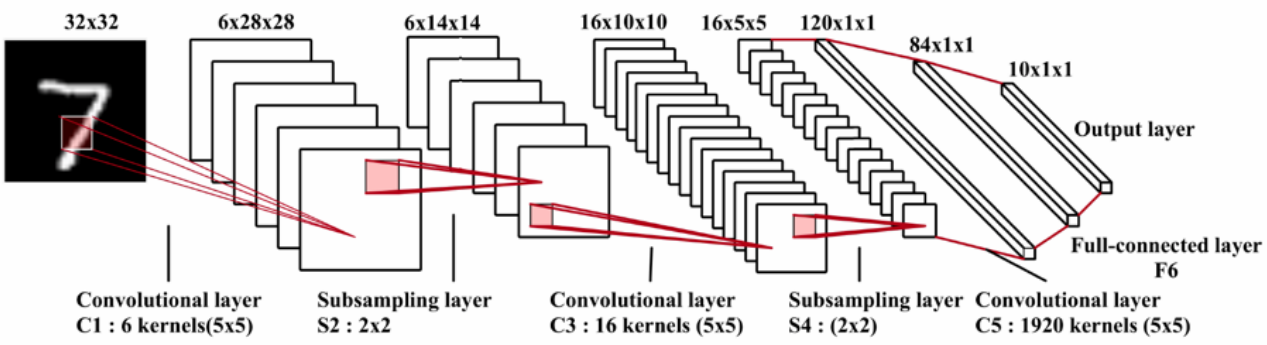
接受一个2维的张量作为输入，返回一个2维的张量作为输出，2维即batch_size x channels

网络定义

```
class LeNet5(nn.Cell):①
    def __init__(self):
        super(LeNet5, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, 5, stride=1, pad_mode='valid')
        self.conv2 = nn.Conv2d(6, 16, 5, stride=1, pad_mode='valid')
        self.relu = nn.ReLU()
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.flatten = nn.Flatten()
        self.fc1 = nn.Dense(400, 120)
        self.fc2 = nn.Dense(120, 84)
        self.fc3 = nn.Dense(84, 10)

    def construct(self, x):
        x = self.relu(self.conv1(x))
        x = self.pool(x)
        x = self.relu(self.conv2(x))
        x = self.pool(x)
        x = self.flatten(x)
        x = self.fc1(x)
        x = self.fc2(x)
        x = self.fc3(x)

        return x
```

② ③

① 网络定义必须继承基类nn.Cell；
② __init__()中的语句由Python解析执行；
③ construct()中的语句由MindSpore接管，有语法限制；



(a) LeNet-5 network

损失函数（Loss）

```
loss = nn.loss.SoftmaxCrossEntropyWithLogits(is_grad=False, sparse=True, reduction='mean')
```

$$Z_i = W \cdot X_i + b$$

$$P_i = \frac{e^{Z_i}}{\sum_j e^{Z_i}}$$

$$l(Z_i, Y_i) = -\log\left(P_{iY_i}\right)$$

对于每个样本N_i，X_i是3D Tensor（CHW），Z_i是3D Tensor，Y_i是真实类别（10类中的一个），P_i是3D Tensor（含10个元素，分别表示属于相应的概率，值域为[0, 1]），$l(Z_i, Y_i)$是标量。

损失函数用于描述模型预测值与真实值的误差，用于指导模型的更新方向。MindSpore支持的损失函数有：L1Loss、MSELoss、SoftmaxCrossEntropyWithLogits、CosineEmbeddingLoss等。
参数解释：

- is_grad，是否仅计算梯度，默认为True；
- sparse，默认为False，为True时对Label数据做one_hot处理；
- reduction，支持mean、sum。

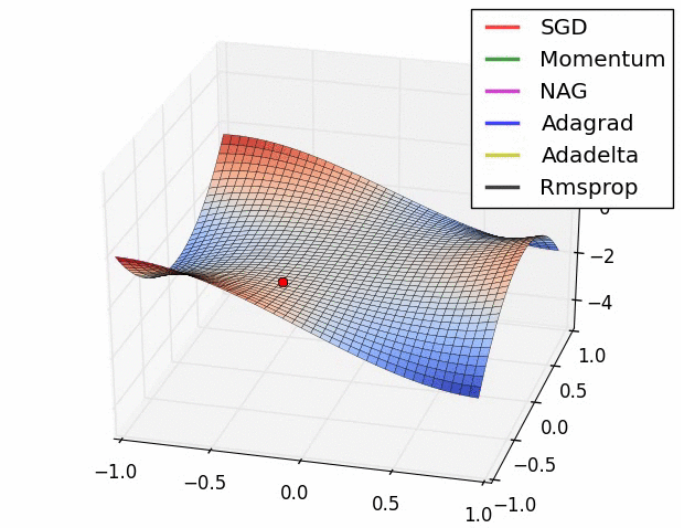
优化器（Optimizer）

```
opt = nn.Momentum(net.trainable_params(), lr, momentum)
```

- 有了优化目标（Loss）之后，我们还需要定义优化的策略，即优化器
- MindSpore支持的优化器有：SGD、Momentum、RMSProp、Adam、AdamWeightDecay、LARS、FTRL、ProximalAdagrad等。

SGD with Momentum:

$$v_{t+1} \leftarrow \rho v_t + \nabla_\theta \mathcal{L}(\theta)$$
$$\theta_j \leftarrow \theta_j - \epsilon v_{t+1}$$



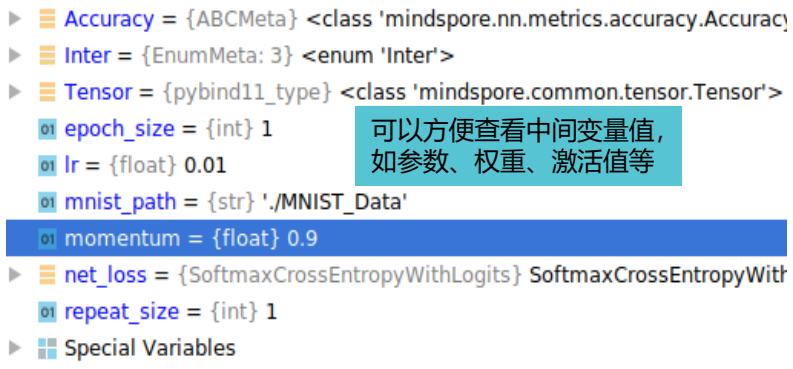
目录

代码调试

MindSpore支持Graph和PyNative 2种运行模式，**PYNATIVE**模式易调试，支持Python原生调试方式。

```
net_loss = SoftmaxCrossEntropyWithLogits(is_grad=False, sparse=True, reduction='
repeat_size = epoch_size  repeat_size: 1
# create the network
network = LeNet5()
# define the optimizer
net_opt = nn.Momentum(network.trainable_params(), lr, momentum)
config_ck = CheckpointConfig(save_checkpoint_steps=1875, keep_checkpoint_max=10)
# save the network model and parameters for subsequence fine-tuning
ckpoint_cb = ModelCheckpoint(prefix="checkpoint_lenet", config=config_ck)
# group layers into an object with training and evaluation features
model = Model(network, net_loss, net_opt, metrics={"Accuracy": Accuracy()})
```

支持Python原生的调试方式，如设置断点



可以方便查看中间变量值，如参数、权重、激活值等

```
def construct(self, x):
    x = self.relu(self.conv1(x))
    x = self.pool(x)
    print(x.shape)
    x = self.relu(self.conv2(x))
    x = self.pool(x)
    x = self.flatten(x)
    x = self.fc1(x)
    x = self.fc2(x)
    x = self.fc3(x)

    return x
```

打印某层算子输出的shape，方便设置下一层算子的参数

GRAPH模式性能高，可调用**Print**算子打印Tensor或字符串。

```
self.print = P.Print()
self.print(x)
Tensor shape:[[const vector][2, 1]]Int32
val:[[1] [1]]
```


HUAWEI

模型调优

- mindspore.train.callback可用于训练/验证过程中执行特定的任务。常用的Callback如下：
  > LossMonitor：监控loss值，当loss值为Nan或Inf时停止训练；
  > SummaryStep：把训练过程中的信息存储到文件中，用于后续查看或可视化展示。
  > ModelCheckpoint：保存模型文件和参数，用于再训练或推理；

```
loss_cb = LossMonitor(per_print_times=ds_train.get_dataset_size())
# 打印
epoch: 2 step 1875, loss is 0.4737345278263092
```

- mindspore.nn提供了多种Metric评估指标，如accuracy、loss、precision、recall、F1。
  > 定义一个metrics字典/元组，里面包含多种指标，传递给Model；
  > 然后调用model.eval接口来计算这些指标。
  > model.eval会返回一个字典，包含各个指标及其对应的值。

```
metrics={'acc', 'loss'}
# 输出
{'loss': 0.10531254443608654, 'acc': 0.9701522435897436}
```

模型训练和验证

- mindspore.train.Model提供了高阶模型训练和验证接口。
  - 传入net、loss和opt，并完成Model的初始化；
  - 然后调用train接口，指定迭代次数（num_epochs）、数据集（dataset）、回调（callback）；
- 训练包含两层循环：外层为数据集的多代（epoch）循环，内层为epoch内多步（batch/step）的迭代；

```
def train(data_dir, lr=0.01, momentum=0.9, num_epochs=3):
    ds_train = create_dataset(data_dir)
    ds_eval = create_dataset(data_dir, training=False)

    net = LeNet5()
    loss = nn.loss.SoftmaxCrossEntropyWithLogits(is_grad=False, sparse=True, reduction='mean')
    opt = nn.Momentum(net.trainable_params(), lr, momentum)
    loss_cb = LossMonitor(per_print_times=ds_train.get_dataset_size())

    model = Model(net, loss, opt, metrics={'acc', 'loss'})
    # dataset_sink_mode can be True when using Ascend
    model.train(num_epochs, ds_train, callbacks=[loss_cb], dataset_sink_mode=False)
    metrics = model.eval(ds_eval, dataset_sink_mode=False)
    print('Metrics:', metrics)
```

LeNet5完整代码示例：https://gitee.com/mindspore/course/tree/master/checkpoint

模型保存和加载

- ModelCheckpoint会生成模型（.meta）和Chekpoint（.ckpt）文件，如每个epoch结束时，都保存一次checkpoint。

```
ckpt_cfg = CheckpointConfig(save_checkpoint_steps=steps_per_epoch, keep_checkpoint_max=5)
ckpt_cb = ModelCheckpoint(prefix=ckpt_name, directory='ckpt', config=ckpt_cfg)
model.train(num_epochs, ds_train, callbacks=[ckpt_cb, loss_cb], dataset_sink_mode=dataset_sink)
# 输出：
lenet-1_1875.ckpt
lenet-2_1875.ckpt
lenet-graph.meta
```

- 如果训练中断后想继续做训练，或者加载模型做微调（Fine-tuning），先使用mindspore.train.serialization提供的load_checkpoint、load_param_into_net功能，再行训练。

```
CKPT_1 = ‘ckpt/lenet-2_1875.ckpt’
# 加载模型，返回参数字典
param_dict = load_checkpoint(CKPT_1)
# 分别将参数加载到网络和优化器上
load_param_into_net(net, param_dict)
load_param_into_net(opt, param_dict)
```

Checkpoint完整代码示例：https://gitee.com/mindspore/course/tree/master/checkpoint

目录

模型推理

- 使用MindSpore做推理
  - > 加载Checkpoint到网络中；
  - > 调用model.predict()接口进行推理。

```
CKPT_2 = 'ckpt/lenet_1-2_1875.ckpt'
def infer(data_dir):
    ds = create_dataset(data_dir, training=False).create_dict_iterator()
    data = ds.get_next()
    images = data['image']
    labels = data['label']
    net = LeNet5()
    load_checkpoint(CKPT_2, net=net)
    model = Model(net)
    output = model.predict(Tensor(data['image']))
    preds = np.argmax(output.asnumpy(), axis=1)
```

- 使用其他平台做推理
  - > 加载Checkpoint到网络中；
  - > 调用mindspore.train.serialization提供的export()接口，导出ONNX等格式的模型文件；
  - > 在任何支持ONNX模型文件的平台上进行推理；

```
input = np.random.uniform(0.0, 1.0, size = [1, 3, 224, 224]).astype(np.float32)
export(resnet, Tensor(input), file_name = 'resnet50-2_32.pb', file_format = 'ONNX')
```

目录

编程接口对比

MindSpore高阶API和编程风格与tf.keras和Pytorch类似，上手和迁移容易

```
class Net(tf.keras.Model):
  def __init__(self):
    super(MyModel, self).__init__()
    self.conv1 = tf.keras.layers.Conv2D()
    self.relu = tf.keras.layers.ReLU()
    self.fc1 = tf.keras.layers.Dense()

  def call(self, x):
    x = self.conv1(x)
    x = self.relu(x)
    x = self.fc1(x)
    return x

model = Net()
loss_fn =
tf.keras.losses.SparseCategoricalCrossentropy(
)
opt = tf.keras.optimizers.Adam()

data = tf.keras.datasets.cifar10.load_data()

model.compile(optimizer=opt, loss=loss_fn,
metrics=['accuracy'])
model.fit(data[0], data[1], 5)
model.evaluate(data[0], data[1])
```

Tensorflow.keras tial

```
class Net(nn.Cell):
  def __init__(self):
    super(Net, self).__init__()
    self.conv1 = nn.Conv2d()
    self.relu = nn.ReLU()
    self.fc1 = nn.Dense()

  def construct(self, x):
    x = self.conv1(x)
    x = self.relu(x)
    x = self.fc1(x)
    return x

net = Net()
loss_fn =
nn.loss.SoftmaxCrossEntropyWithLogits()
opt = nn.Momentum()

dataset = dataset.Cifar10Dataset()

model = train.Model(net, loss_fn, opt,
metrics={'acc'})
model.train(5, dataset)
model.eval(dataset)
```

MindSpore

```
class Net(nn.Module):
  def __init__(self):
    super(Net, self).__init__()
    self.conv1 = nn.Conv2d()
    self.fc1 = nn.Linear()

  def forward(self, x):
    x = self.conv1(x)
    x = F.relu(x)
    x = self.fc1(x)
    return x

net = Net()
loss_fn = nn.CrossEntropyLoss()
opt = optim.SGD(momentum=0.9)

dataset = torchvision.datasets.CIFAR10()
loader = torch.utils.data.DataLoader(dataset)

for epoch in range(5):
  for i, data in enumerate(loader, 0):
    opt.zero_grad()
    outputs = net(data[0])
    loss = loss_fn(outputs, data[1])
    loss.backward()
    opt.step()

with torch.no_grad():
  for data in dataloader:
    outputs = net(data[0])
    _, pred = torch.max(outputs.data, 1)
    total += data[1].size(0)
    correct += (pred == labels).sum().item()
    acc = 100 * correct / total
```

Pytorch

HUAWEI

网络迁移

文档 (r1.3) > 算子匹配 > API映射

PyTorch算子: torch.abs

PyTorch 1.5.0

`torch.abs(input, out=None) → Tensor`

Computes the element-wise absolute value of the given `input` tensor.

$$\text{out}_i = |\text{input}_i|$$

Parameters

input (Tensor) – the input tensor.

out (Tensor, optional) – the output tensor.

Example:

```
>>> torch.abs(torch.tensor([-1, -2, 3]))
tensor([ 1,  2,  3])
```

MindSpore 1.3.0

`class mindspore.ops.Abs(*args, **kwargs)`

Returns absolute value of a tensor element-wise.

$$out_i = |x_i|$$

Inputs:

x (Tensor) - The input tensor. The shape of tensor is $(x_1, x_2, ..., x_R)$.

Outputs:

Tensor, has the same shape as the *x*.

Raises

TypeError – If *x* is not a Tensor.

Supported Platforms:

`Ascend` `GPU` `CPU`

Examples

```
>>> x = Tensor(np.array([-1.0, 1.0, 0.0]), mindspore.float32)
>>> abs = ops.Abs()
```

文档反馈

API映射： https://www.mindspore.cn/docs/note/zh-CN/r1.3/index.html#operator_api
网络迁移教程： https://www.mindspore.cn/docs/migration_guide/zh-CN/r1.3/index.html

参与社区

官方网站

MindSpore

https://www.mindspore.cn

代码托管平台

gitee

https://gitee.com/mindspore

- MindSpore社区：https://gitee.com/mindspore/community
- 优秀开发者及布道师招募：https://www.mindspore.cn/evangelist

- 高校开发者微信群：添加微信mindspore0328，备注“开发者”

- 官方交流QQ群：871543426
- 官方微信公众号：MindSpore

小助手mindspore0328

HUAWEI

Thank you.

把数字世界带入每个人、每个家庭、
每个组织，构建万物互联的智能世界。

Bring digital to every person, home and organization for a fully connected, intelligent world.

HUAWEI