MindSpore

昇思金箍棒

模型压缩算法集

作者：韩刚强

目录

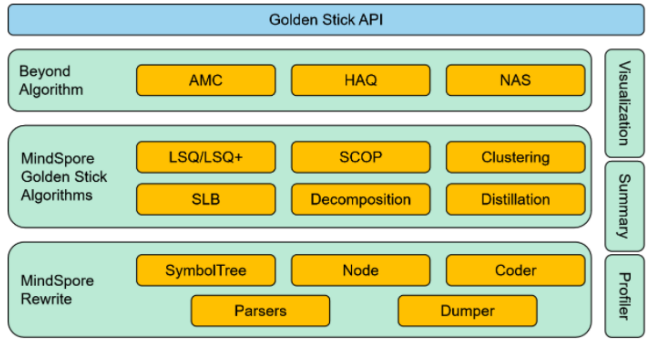
contents

MindSpore

昇思科学计算发展历程

全场景AI计算框架MindSpore正式发布

2019.8.23

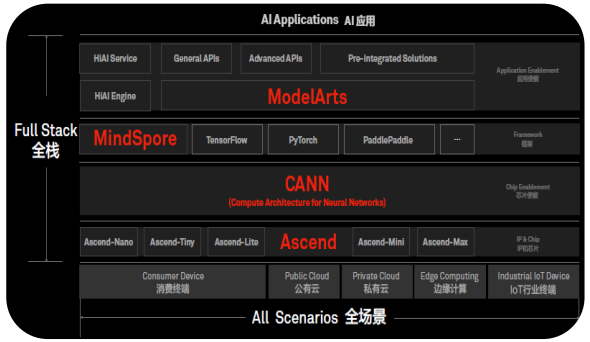


MindSpore发布 昇思金箍棒模型压缩套件

2022.7.27

2018.10.10

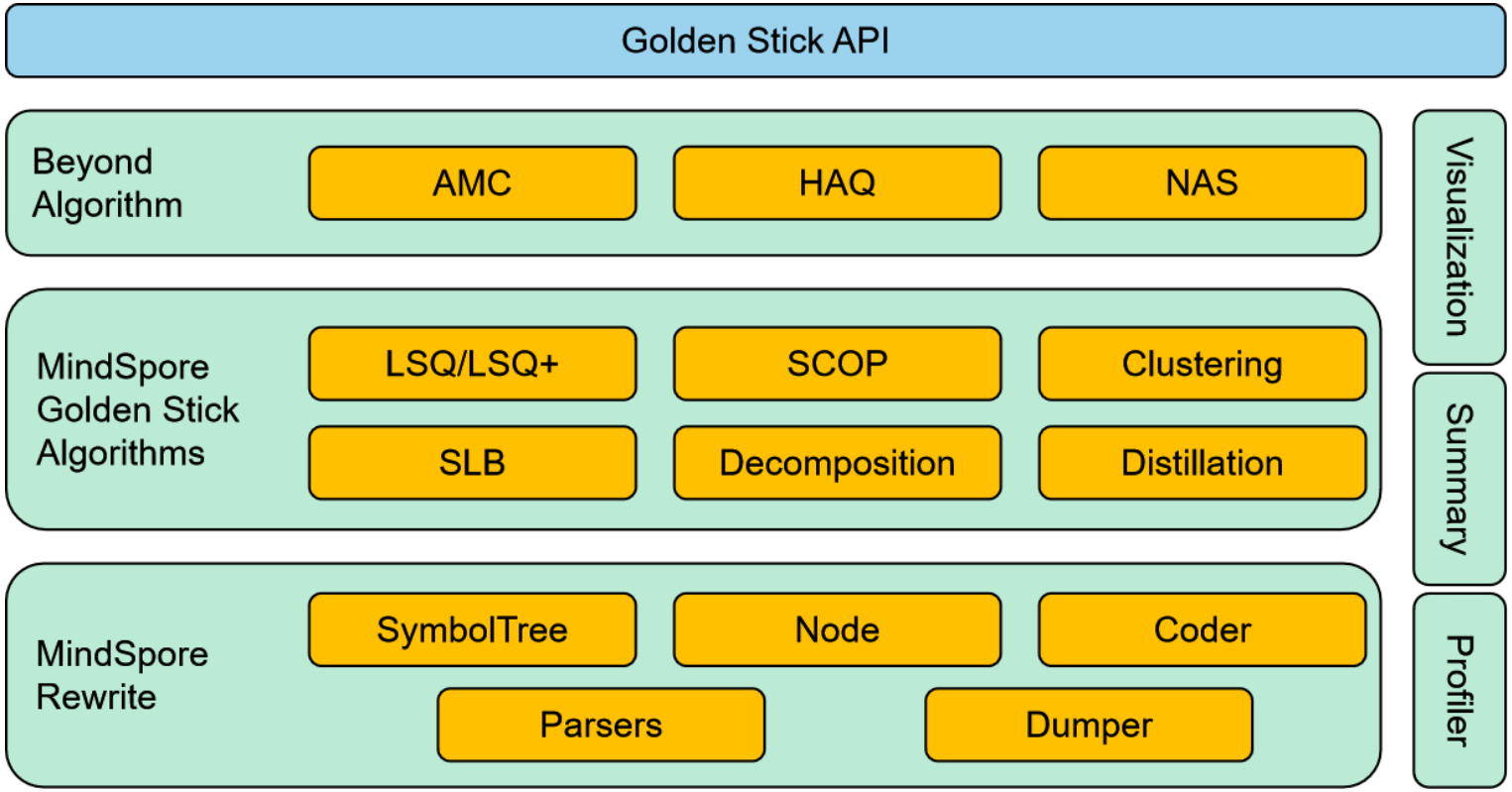
华为全栈全场景AI解决方案发布



2020.3.28

HDC大会正式开源MindSpore

昇思金箍棒 打破AI端边侧部署最后一个屏障



- **面向用户：**1）提供一套统一的算法应用接口，降低算法使用成本；2）提供丰富的模型压缩算法选择，满足用户不同场景的各种指标要求；
- **面向开发者：**1）提供“动静统一”的网络修改能力，保留网络动态性的同时，提供给用户静态图形式的接口；2）提供一系列网络调测工具，如Dumper、可视化工具、压缩算法效果分析工具等

目录

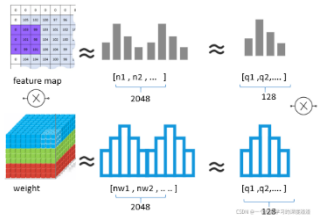
contents

MindSpore

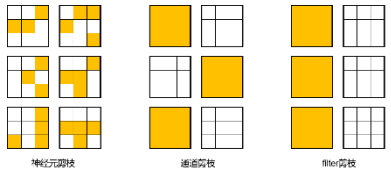
统一的算法应用接口

模型压缩算法种类多样， AI框架选择繁多，同一框架中的不同算法或者同种算法在不同框架中使用方式各不相同，让算法使用者望而却步。

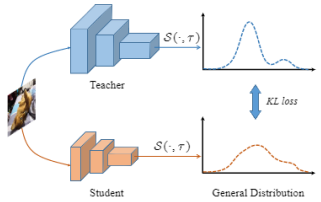
多样的模型压缩算法



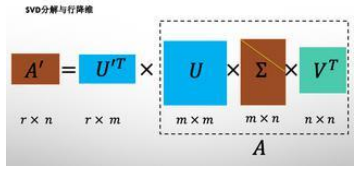
量化算法



剪枝算法



知识蒸馏算法



矩阵压缩算法

...

各种AI计算框架

昇思 MindSpore

PyTorch

TensorFlow

Keras

飞桨 PaddlePaddle

mxnet

ONNX

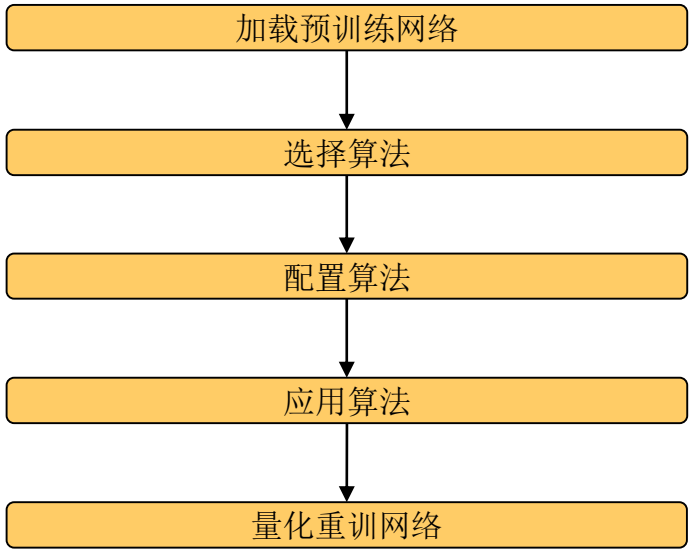
Neural Network Intelligence

...

统一的算法应用接口

```python
class CompAlgo:
    def __init__(self, config):
        self._config = config

    def apply(self, network: Cell) -> Cell:
        return network

    def callback(self) -> Callback: ...

    def loss(self, loss_fn): ...
```

```python
net = LeNet5()

algo = QATCompressAlgo({"bit_num": 8, "per_channel": True})
net_opt = algo.apply(net)

loss = algo.loss(SoftmaxCrossEntropyWithLogits())
optimizer = Momentum(params=net.trainable_params(), learning
model = Model(net_opt, loss_fn=loss, optimizer=optimizer, me
dataset = {}
model.train(2, dataset, algo.callbacks())
```



侵入性修改少，图中红框就是应用一个QAT算法需要新增的代码：

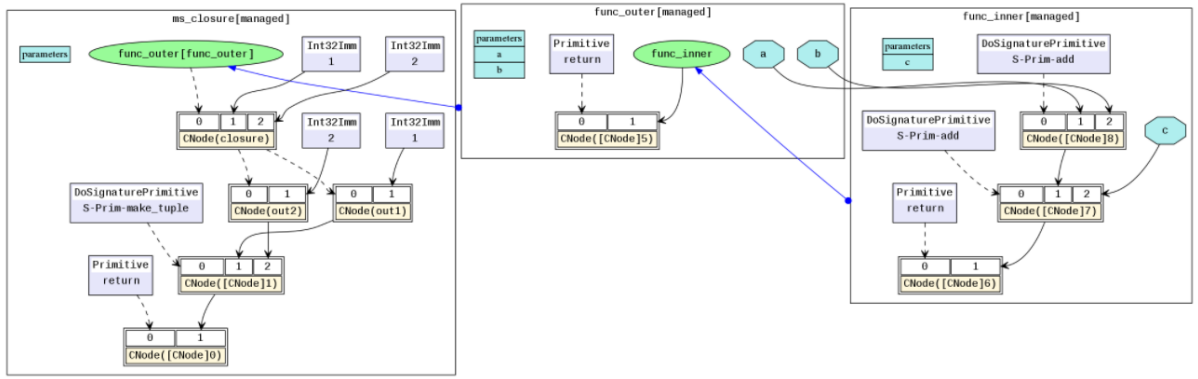
- 无需手动修改网络定义
- 对训练脚本修改小

动态图与静态图

PyTorch的横空出世，充分证明了动态图的用户友好型：书写自由、调试方便、代码即公式。这其实是使用了动态图执行方式，即采用Python的编程风格，直接解析式地执行每一行网络代码，并同时返回计算结果。

```
# 设置运行模式为动态图模式
ms.set_context(mode=ms.PYNATIVE_MODE)

x = ms.Tensor(np.array([1.0, 2.0, 3.0]).astype(np.float32))
y = ms.Tensor(np.array([4.0, 5.0, 6.0]).astype(np.float32))

output = ops.mul(x, y)

print(output.asnumpy())
```

```
[ 4. 10. 18.]
```



灵活易用：按照用户编写的代码逐行执行，更容易进行网络修改和调试。且基本上Python的语法都支持，对于动态模型，比如动态shape、有复杂控制流等，尤其有利；

性能差：动态图的执行性能基本上取决于单算子的性能，缺乏算子间的融合优化，无法充分发挥AI芯片的算力。

部署存在gap：AI在部署的时候，为了性能和功耗，一般要需要生成一个部署模型，部署模型本质是静态图的表达，所以这里就存在一个动静态转换的问题，太灵活的动态图训练完后，想转换变成一个部署模型同样存在很大的挑战。

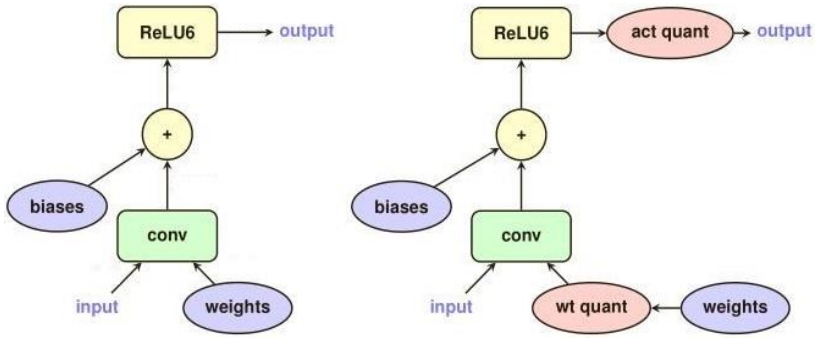
性能佳：静态图允许编译器对执行图进行编译优化，能够提供更好的执行性能。

部署方便：静态图本身就是一个图结构，可以方便地转化为部署模型。

限制多：支持的语法十分有限。

易用性差：图编译和图优化同时也带来了编译器实际执行的代码和原始代码之间存在很大的差距，导致静态图很难调试。比如我用python定义了一个网络，报错却是一堆底层C++堆栈信息。

“动静统一”的网络修改能力



模型压缩算法通常需要修改网络结构以实现算法逻辑，比如简单量化感知训练算法中伪量化节点的插入

```
class MyConv(nn.Cell):
    def __init__(self, channel):
        super().__init__()
        self.conv = nn.Conv2d(channel, channel, 3)

    def construct(self, x):  self: MyConv<\n  (conv): Conv2d<input_
        x = self.conv(x)
        return x
```

ug: pytest for test_tf.test_rewrite

Debugger | Console

Frames | Variables

MainThread

construct, test_tf.py:29

_run_construct, cell.py:412

type(stree._symbol_tree._nodes['sub_net_1'].s

self = {MyConv} MyConv<\n (conv): Conv2d

x = {Tensor: 1} [[[[-0.7094384 -0.4964752

改写后的网络代码可见可调试。用户可以不关注整体网络，直接跟踪到自己定义的子网内部。

```
class Net(nn.Cell):
    def __init__(self):
        super().__init__()
        self.conv = nn.Conv2d(5, 5, 3)
        self.bn = nn.BatchNorm2d(5)
        self.relu = nn.ReLU()

    def construct(self, x):
        x = self.conv(x)
        x = self.bn(x)
        return self.relu(x)
```

原始网络

```
def create_node(op, targets: [Argument], target_type: str = None, args: [Argume
def nodes(self) -> [Node]: ...
def find_node(self, full_name_with_scope: str) -> Node: ...
def node_inputs(self, full_name_with_scope: str) -> [Node]: ...
def node_outputs(self, full_name_with_scope: str) -> [Node]: ...
def insert_node(self, new_node: Node) -> Node: ...
def remove_node(self, full_name_with_scope: str) -> Node: ...
def replace_node(self, full_name_with_scope: str, new_node: Node) -> Node: ...
def pattern_transform(self, pattern_engine: PatternEngine) -> bool: ...
```

图的形式修改网络更加便捷（拓扑关系等）

```
class Net(nn.Cell):
    def __init__(self):
        super().__init__()
        self.conv = nn.Conv2dBnFoldQuant(5, 5, 3)

    def construct(self, x):
        x = self.conv(x)
        return x
```

修改后的网络依然是源码，保留动态执行的可能

其他调试工具

算法应用到具体网络上往往需要调试调优工作，昇思金箍棒提供了一些调试工具希望能够提升算法开发者的效率

```
#SymbolTree entry    : @construct
#Inputs num     : 1
%input_x

Symbol Tree @construct {
  %0(x_1) = Conv2d(%input_x){instance name: conv1} attributes {kernel_size: (5, 5), stride: (1, 1), pad_
      : (null) -> (null)
      # In file /mnt/BigBoy/software/anaconda3/envs/py39-ms/lib/python3.9/site-packages/mindspore/nn/lay
      # In file /mnt/data/workspace/MindSpore/mindspore/tests/ut/python/rewrite/test_net_simple.py
  %1(x_2) = NoneType(x_1){instance name: name_assign} attributes {}
      : (null) -> (null)
      # In file None
      # In file /mnt/data/workspace/MindSpore/mindspore/tests/ut/python/rewrite/test_net_simple.py
  %2(None) = NoneType{instance name: Assign_1} attributes {}
      : (null) -> (null)
      # In file None
      # In file /mnt/data/workspace/MindSpore/mindspore/tests/ut/python/rewrite/test_net_simple.py
  %3(x_3) = ReLU(x_2){instance name: relu} attributes {}
      : (null) -> (null)
      # In file /mnt/BigBoy/software/anaconda3/envs/py39-ms/lib/python3.9/site-packages/mindspore/nn/lay
      # In file /mnt/data/workspace/MindSpore/mindspore/tests/ut/python/rewrite/test_net_simple.py
  %4(x_4) = MaxPool2d(x_3){instance name: max_pool2d} attributes {}
      : (null) -> (null)
      # In file /mnt/BigBoy/software/anaconda3/envs/py39-ms/lib/python3.9/site-packages/mindspore/nn/lay
      # In file /mnt/data/workspace/MindSpore/mindspore/tests/ut/python/rewrite/test_net_simple.py
  %5(x_5) = Conv2d(x_4){instance name: conv2} attributes {kernel_size: (5, 5), stride: (1, 1), pad_mode:
      : (null) -> (null)
      # In file /mnt/BigBoy/software/anaconda3/envs/py39-ms/lib/python3.9/site-packages/mindspore/nn/lay
      # In file /mnt/data/workspace/MindSpore/mindspore/tests/ut/python/rewrite/test_net_simple.py
```

Dump能力：将网络中的拓扑关系，属性等信息打印到屏幕或者存储到文件，方便算法开发者针对性地对网络进行优化。

Profiling能力：展示逐层的精度损失、稀疏度、均值、方差等，不同算法也可以自定义需要展示的数据。解决用户选用何种算法，如何配置算法的问题。

还有一些调试工具，比如可视化、模型压缩效果分析器等还在开发中，敬请期待

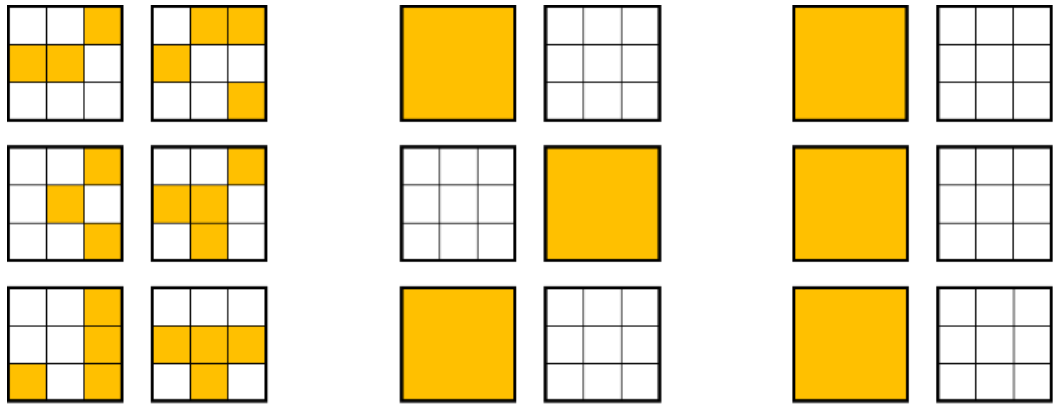
目录

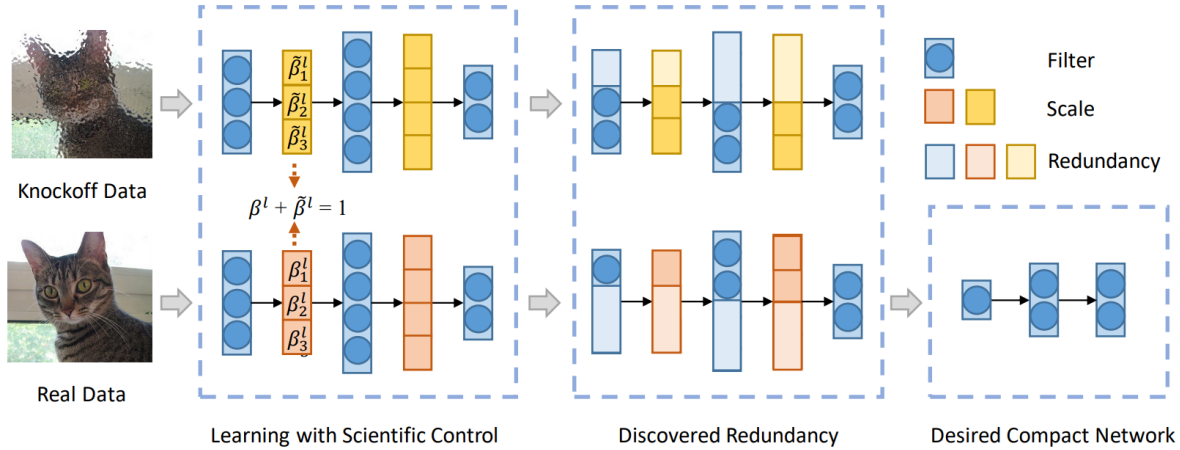
contents

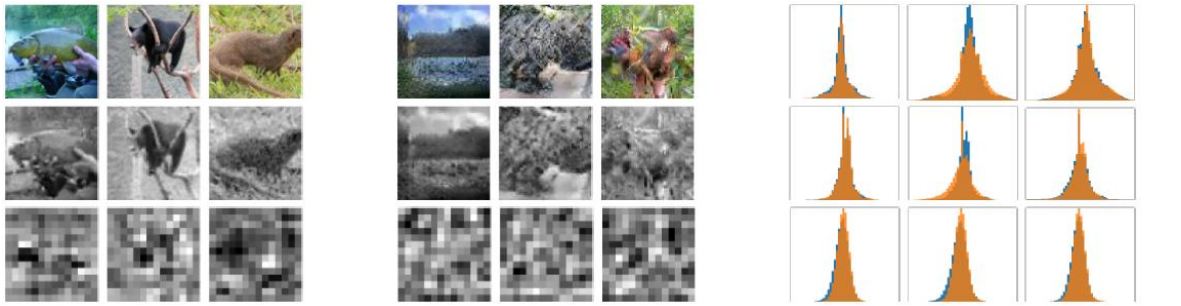
MindSpore

SCOP结构化剪枝算法

基于科学控制法的神经网络剪枝算法，数据驱动的结构化剪枝算法，精度更高，且无需软硬件适配







(a) Real data and features (b) Knockoff data and features (c) Frequency distribution histogram

假数据：独立同分布且标签无关

原始特征: $\mathcal{A}^l$　　高仿特征: $\tilde{\mathcal{A}}^l$

目标函数： $\min_{\mathcal{A}^l, \tilde{\mathcal{A}}^l} E([\mathcal{A}^l, \tilde{\mathcal{A}}^l], Y), \quad s.t. \quad \|[\mathcal{A}^l, \tilde{\mathcal{A}}^l]\|_0 \leq \kappa^l,$

控制因子： $\beta^l, \tilde{\beta}^l \in [0,1]^{M^l}, \beta^l + \tilde{\beta}^l = 1$

特征选择层： $\mathcal{A}^{l+1} = \phi(\mathcal{W}^{l+1} * (\boldsymbol{\beta}^l \odot \mathcal{A}^l + \tilde{\boldsymbol{\beta}}^l \odot \tilde{\mathcal{A}}^l)),$

优化： $\min_{\boldsymbol{\beta}, \tilde{\boldsymbol{\beta}}} E(X, \tilde{X}, Y, \boldsymbol{\beta}, \tilde{\boldsymbol{\beta}}), \quad s.t. \quad \boldsymbol{\beta}^l + \tilde{\boldsymbol{\beta}}^l = \mathbf{1}, l \in [1, 2, \cdots, L],$

SCOP结构化剪枝算法

MindSpore

ImageNet上的分类实验效果

Model	Method	Top-1 Error (%) Orig.	Pruned	Gap	Top-5 Error (%) Orig.	Pruned	Gap	Params. ↓ (%)	FLOPs ↓ (%)
Res18	MIL (2017) [5]	30.02	33.67	3.65	10.76	13.06	2.30	N/A	33.3
	SFP (2018) [9]	29.72	32.90	3.18	10.37	12.22	1.85	39.3	41.8
	FPGM (2019) [10]	29.72	31.59	1.87	10.37	11.52	1.15	39.3	41.8
	PFP-A (2020) [21]	30.26	32.62	2.36	10.93	12.09	1.16	43.8	29.3
	PFP-B (2020) [21]	30.26	34.35	4.09	10.93	13.25	2.32	60.5	43.1
	SCOP-A (Ours)	30.24	**30.82**	**0.58**	10.92	**11.11**	**0.19**	39.3	38.8
	SCOP-B (Ours)	30.24	31.38	1.14	10.92	11.55	0.63	43.5	**45.0**
Res34	SFP(2018) [9]	26.08	28.17	2.09	8.38	9.67	1.29	39.8	41.1
	FPGM(2019) [10]	26.08	27.46	1.38	8.38	8.87	0.49	39.8	41.1
	Taylor (2019) [27]	26.69	27.17	0.48	N/A	N/A	N/A	22.1	24.2
	SCOP-A (Ours)	26.69	**27.07**	**0.38**	8.58	**8.80**	**0.22**	39.7	39.1
	SCOP-B (Ours)	26.69	27.38	0.69	8.58	9.02	0.44	**45.6**	**44.8**
Res50	CP (2017) [11]	N/A	N/A	N/A	7.80	9.20	1.40	N/A	50.0
	ThiNet (2017) [26]	27.12	27.96	0.84	8.86	9.33	0.47	33.72	36.8
	SFP (2018) [9]	23.85	25.39	1.54	7.13	7.94	0.81	N/A	41.8
	Autopruner(2018) [25]	23.85	25.24	1.39	7.13	7.85	0.72	N/A	48.7
	FPGM (2019) [10]	23.85	24.41	0.56	7.13	7.73	0.24	37.5	42.2
	Taylor (2019) [27]	23.82	25.50	1.68	N/A	N/A	N/A	44.5	44.9
	C-SGD (2019) [4]	24.67	25.07	0.40	7.44	7.73	0.29	N/A	46.2
	GAL (2019) [23]	23.85	28.05	4.20	7.13	9.06	1.93	16.9	43.0
	RRBP (2019) [44]	23.90	27.00	3.10	7.10	9.00	1.90	N/A	54.5
	Hrank (2020) [22]	23.85	25.02	1.17	7.13	7.67	0.51	36.7	43.7
	PFP-A (2020) [21]	23.87	24.09	0.22	7.13	7.19	0.06	18.1	10.8
	PFP-B (2020) [21]	23.87	24.79	0.92	7.13	7.57	0.45	30.1	44.0
	SCOP-A (Ours)	23.85	**24.05**	**0.20**	7.13	**7.21**	**0.08**	42.8	45.3
	SCOP-B (Ours)	23.85	24.74	0.89	7.13	7.47	0.34	**51.8**	**54.6**
Res101	SFP (2018) [9]	22.63	22.49	-0.14	6.44	6.29	-0.20	38.8	42.2
	FPGM (2019) [10]	22.63	22.68	0.05	6.44	6.44	0.00	38.8	42.2
	Taylor (2019) [27]	22.63	22.65	0.02	N/A	N/A	N/A	30.2	39.7
	PFP-A (2020) [21]	22.63	23.22	0.59	6.45	6.74	0.29	33.0	29.4
	PFP-B (2020) [21]	22.63	23.57	0.94	6.45	6.89	0.44	50.4	45.1
	SCOP-A (Ours)	22.63	**22.25**	**-0.32**	6.44	**6.16**	**-0.28**	46.8	48.6
	SCOP-B (Ours)	22.63	22.64	0.01	6.44	6.43	-0.01	**57.8**	**60.2**

CIFAR-10上的分类实验效果

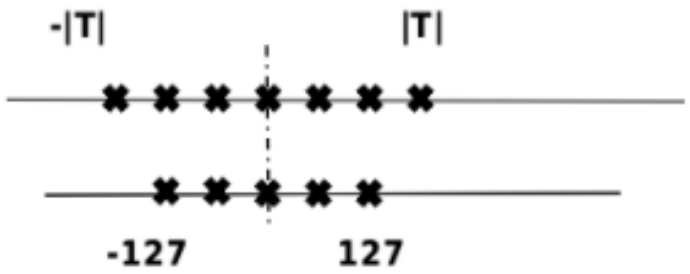
Model	Method	Error (%) Original	Pruned	Gap	Params. ↓ (%)	FLOPs ↓ (%)
ResNet-20	SFP (2018) [9]	7.80	9.17	1.37	39.9	42.2
	FPGM (2019) [10]	7.80	9.56	1.76	51.0	54.0
	SCOP (Ours)	7.78	**9.25**	**1.44**	**56.3**	**55.7**
ResNet-32	MIL (2017) [5]	7.67	9.26	1.59	N/A	31.2
	SFP (2018) [9]	7.37	7.92	0.55	39.7	41.5
	FPGM (2019) [10]	7.37	8.07	0.70	50.8	53.2
	SCOP (Ours)	7.34	**7.87**	**0.53**	**56.2**	**55.8**
ResNet-56	CP (2017) [11]	7.20	8.20	1.00	N/A	50.0
	SFP (2018) [9]	6.41	7.74	1.33	50.6	52.6
	GAL (2019) [23]	6.74	7.26	0.52	44.8	48.5
	FPGM (2019) [10]	6.41	6.51	0.10	50.6	52.6
	HRank (2020) [22]	6.74	6.83	0.09	42.4	50.0
	SCOP (Ours)	6.30	**6.36**	**0.06**	**56.3**	**56.0**
MobileNetV2	DCP (2018) [45]	5.53	5.98	0.45	23.6	26.4
	SCOP (Ours)	5.52	**5.76**	**0.24**	**36.1**	**40.3**

通道剪枝，无需后端软硬件配合，top1掉点0.6%以内，剪掉40%+的权重；

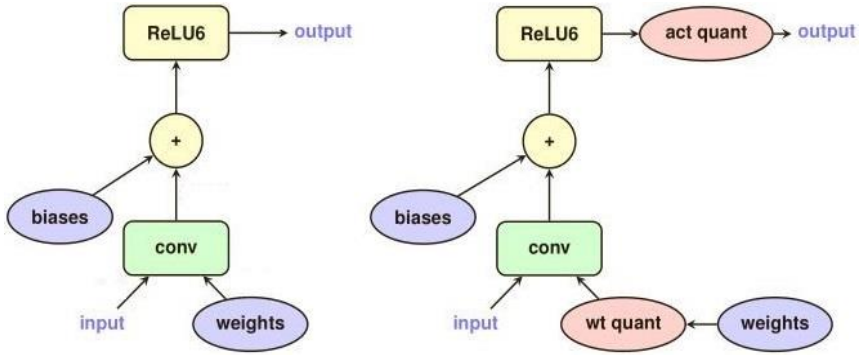
Tang, Yehui, et al. "Scop: Scientific control for reliable neural network pruning." Advances in Neural Information Processing Systems 33 (2020): 10936-10947.

LSQ量化训练算法

可微线性量化算法，基于谷歌提出的伪量化量化感知训练算法改进



AI网络量化是指将网络中的参数或者激活值，从浮点域映射到整型域的过程。在这个过程中尽可能保证数据的分布不变。目的是通过降低数值的表达能力来缩减网络的规模、降低推理的功耗以及加速推理。但是量化是一种有损压缩，可能会导致网络的准确率下降。



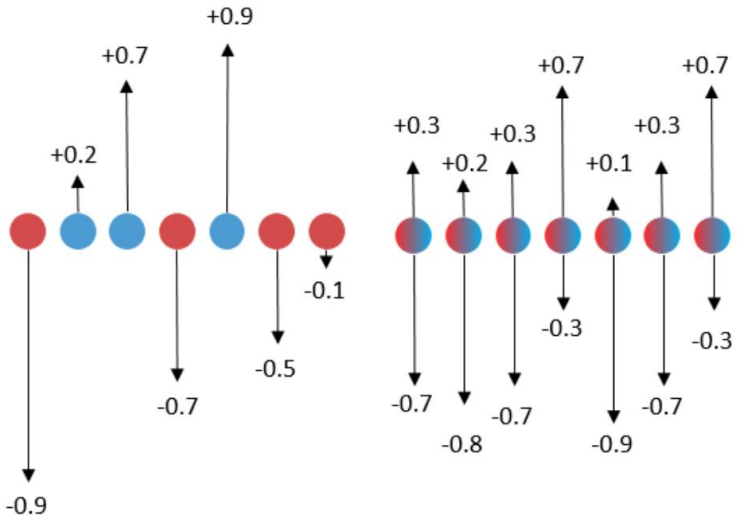
线性量化是指浮点域到整型域的映射关系可以用一个线性函数表示的量化方法。
伪量化节点是一种用来模拟量化损失的算子，通常用于在网络训练中，在前向中模拟量化损失，从而实现量化训练。值得注意的是伪量化节点无法求导，在反向时使用STE规避这个问题，通过引入损失影响loss从而更新权重，进一步根据权重更新伪量化节点中的参数。

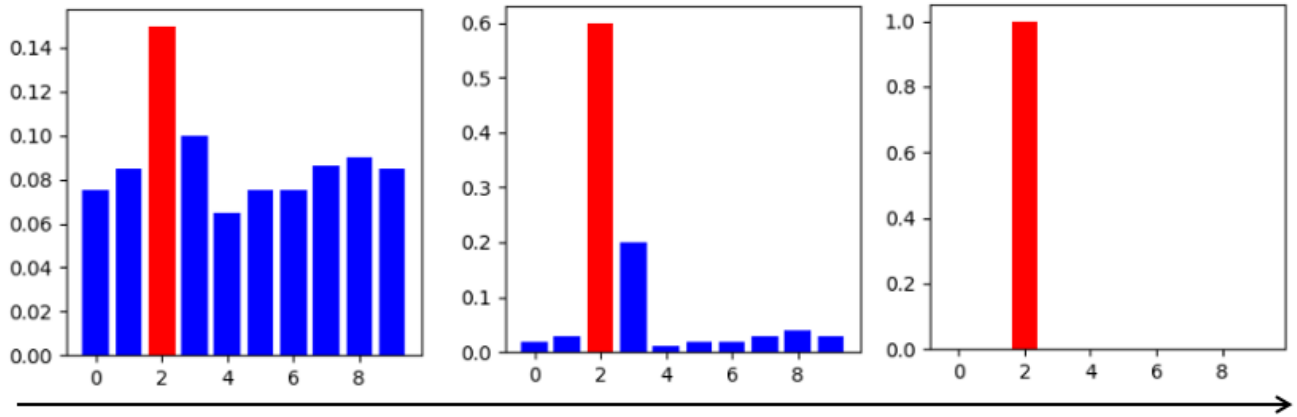
Network	Method	Top-1 Accuracy @ Precision 2	3	4	8	Top-5 Accuracy @ Precision 2	3	4	8
ResNet-18		*Full precision: 70.5*				*Full precision: 89.6*			
	LSQ (Ours)	**67.6**	**70.2**	**71.1**	**71.1**	**87.6**	**89.4**	**90.0**	**90.1**
	QIL	65.7	69.2	70.1					
	FAQ			69.8	70.0			89.1	89.3
	LQ-Nets	64.9	68.2	69.3		85.9	87.9	88.8	
	PACT	64.4	68.1	69.2		85.6	88.2	89.0	
	NICE		67.7	69.8			87.9	89.21	
	Regularization	61.7		67.3	68.1	84.4		87.9	88.2
ResNet-34		*Full precision: 74.1*				*Full precision: 91.8*			
	LSQ (Ours)	**71.6**	**73.4**	**74.1**	**74.1**	**90.3**	**91.4**	**91.7**	**91.8**
	QIL	70.6	73.1	73.7					
	LQ-Nets	69.8	71.9			89.1	90.2		
	NICE		71.7	73.5			90.8	91.4	
	FAQ			73.3	73.7			91.3	91.6
ResNet-50		*Full precision: 76.9*				*Full precision: 93.4*			
	LSQ (Ours)	**73.7**	**75.8**	**76.7**	**76.8**	**91.5**	**92.7**	93.2	**93.4**
	PACT	72.2	75.3	76.5		90.5	92.6	93.2	
	NICE		75.1	76.5			92.3	**93.3**	
	FAQ			76.3	76.5			92.9	93.1
	LQ-Nets	71.5	74.2	75.1		90.3	91.6	92.4	
ResNet-101		*Full precision: 78.2*				*Full precision: 94.1*			
	LSQ (Ours)	**76.1**	**77.5**	**78.3**	**78.1**	**92.8**	**93.6**	**94.0**	**94.0**
ResNet-152		*Full precision: 78.9*				*Full precision: 94.3*			
	LSQ (Ours)	**76.9**	**78.2**	**78.5**	**78.5**	**93.2**	**93.9**	**94.1**	**94.2**
	FAQ			78.4	**78.5**			**94.1**	94.1
VGG-16bn		*Full precision: 73.4*				*Full precision: 91.5*			
	LSQ (Ours)	**71.4**	**73.4**	**74.0**	73.5	**90.4**	**91.5**	**92.0**	**91.6**
	FAQ			73.9	**73.7**			91.7	**91.6**

不同量化方法在ImageNet上的分类实验结果

SLB非线性量化算法

基于权值搜索的低比特神经网络量化，可训非线性量化算法，精度更高，支持低比特量化





左：传统量化算法，训练时量化浮点权重，并用不准确的梯度更新权重，最后对浮点权重做量化

右：SLB量化算法，利用连续松弛策略搜索离散权重，训练时优化离散权重的分布，最后根据概率挑选离散权重实现量化

改造softmax $\mathrm{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=0}^{n-1}\exp(x_j)} \longrightarrow \mathrm{softmax}(x_i) = \frac{\exp(x_i * T)}{\sum_{j=0}^{n-1}\exp(x_j * T)}$

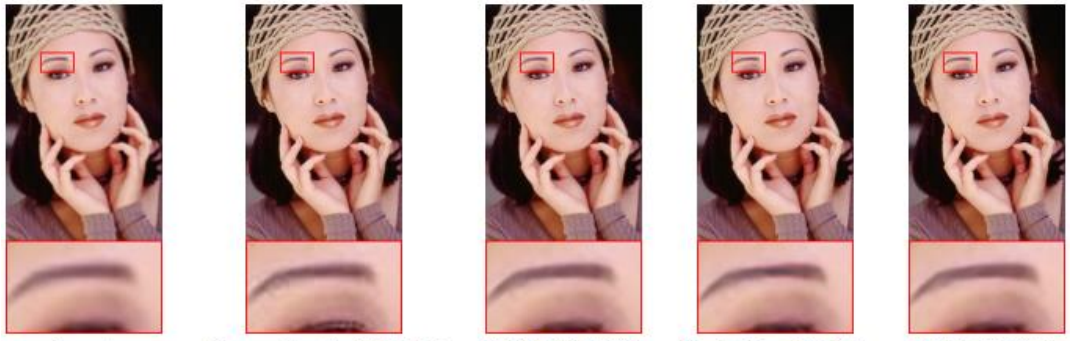
调节温度系数 $\mathrm{softmax}(x_i) = \frac{\exp(x_i * T)}{\sum_{j=0}^{n-1}\exp(x_j * T)} \longrightarrow I = \mathrm{onehot}(\arg\max_i(P_i))$

SLB非线性量化算法

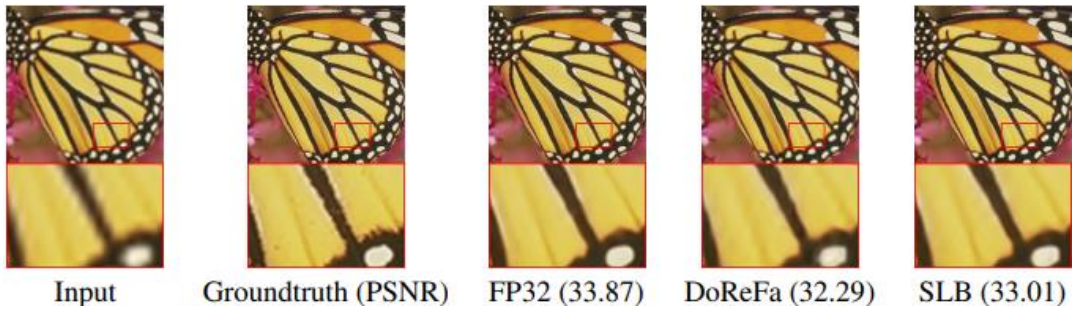
Methods	Bit-width (W/A)	Top-1 (%)	Top-5 (%)
FP32 [20]	32/32	69.3	89.2
BNN [23]	1/1	42.2	67.1
ABCNet [32]	1/1	42.7	67.6
XNORNet [39]	1/1	51.2	73.2
BiRealNet [35]	1/1	56.4	79.5
PCNN [14]	1/1	57.3	80.0
SQ [13]	1/1	53.6	75.3
ResNetE [2]	1/1	58.1	80.6
BONN [15]	1/1	59.3	81.6
SLB	1/1	**61.3**	**83.1**
SLB (w/o SBN)	1/1	61.0	82.9
DoReFa [56]	1/2	53.4	-
LQ-Net [54]	1/2	62.6	84.3
HWGQ [4]	1/2	59.6	82.2
TBN [45]	1/2	55.6	79.0
HWGQ [4]	1/2	59.6	82.2
SLB	1/2	**64.8**	**85.5**
DoReFa [56]	1/4	59.2	-
SLB	1/4	**66.0**	**86.4**
SYQ [10]	1/8	62.9	84.6
SLB	1/8	**66.2**	**86.5**

Methods	Bit-width (W/A)	Top-1 (%)	Top-5 (%)
FP32 [20]	32/32	69.3	89.2
BWN [39]	1/32	60.8	83.0
DSQ [12]	1/32	63.7	-
SQ [13]	1/32	66.5	87.3
SLB	1/32	**67.1**	**87.2**
PACT [5]	2/2	64.4	-
LQ-Net [54]	2/2	64.9	85.9
DSQ [12]	2/2	65.2	-
SLB	2/2	**66.1**	**86.3**
SLB	2/4	**67.5**	**87.4**
SYQ [10]	2/8	67.7	87.8
SLB	2/8	**68.2**	**87.7**
TTQ [57]	2/32	66.6	87.2
LQ-Net [54]	2/32	68.0	88.0
SLB	2/32	**68.4**	**88.1**

ImageNet上的分类实验效果，在低比特量化上优势更大



Set5数据集，1bit量化2倍超分任务效果



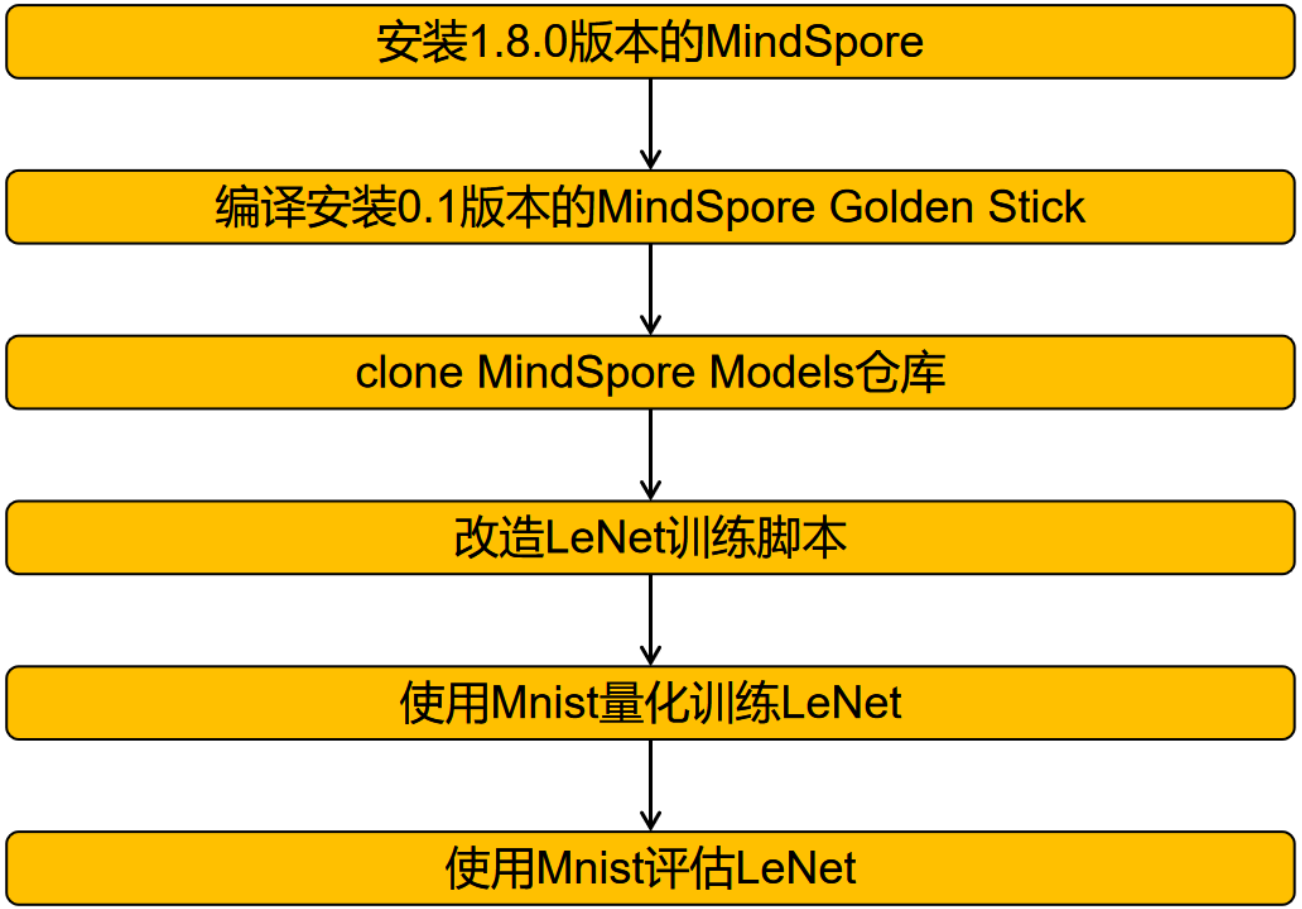
Set5数据集，1bit量化3倍超分任务效果

目录

contents

MindSpore

简单的Example


安装1.8.0版本的MindSpore
编译安装0.1版本的MindSpore Golden Stick
clone MindSpore Models仓库
改造LeNet训练脚本
使用Mnist量化训练LeNet
使用Mnist评估LeNet