## First Implementation

## Method

* Pretrain a resnet56 baseline to a level as good as possible.
* Pruning the network using the criteria:
  + Random method without constraint
  + Random method with constraint of 70±2.5% sparsity
  + TPE method without constraint
  + TPE method with constraint of 70±2.5% sparsity

## Train a resnet56 Baseline Model

### Command

|  |
| --- |
| python3 ../classifier\_compression/compress\_classifier.py --arch resnet56\_cifar --lr 0.03 -p 50 ../../../data.cifar10 -b 128 -j 1 --vs 0.1 --deterministic --epochs 200 --compress=./resnet56\_cifar\_baseline.yaml |

### YAML File

*lr\_schedulers:*

*training\_lr:*

*class: MultiStepMultiGammaLR*

*milestones: [5, 10, 20, 30, 50, 150]*

*gammas: [0.7, 0.5, 0.5, 0.5, 0.5, 0.5]*

*policies:*

*- lr\_scheduler:*

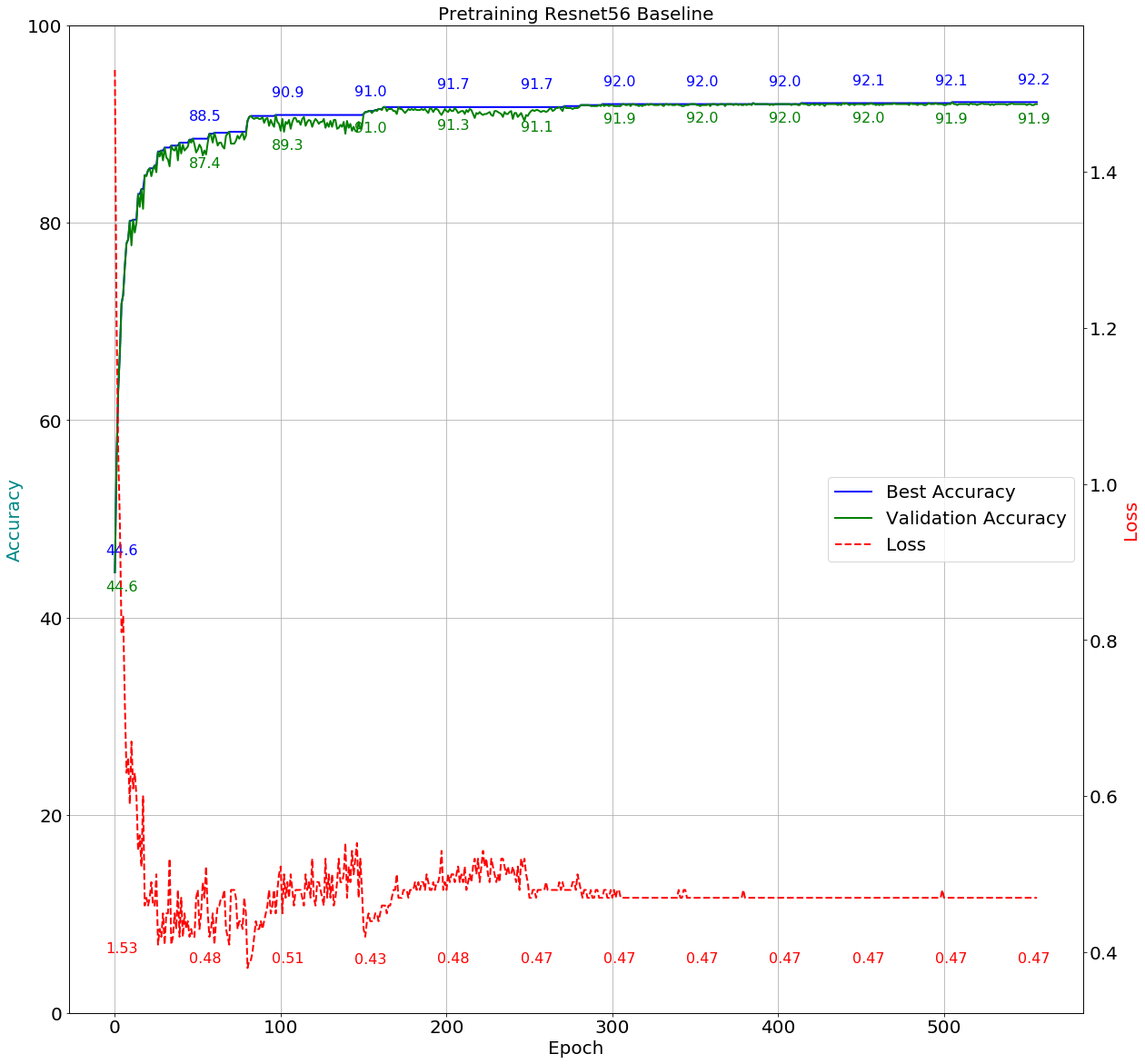
*instance\_name: training\_lr*

*starting\_epoch: 0*

*ending\_epoch: 560*

*frequency: 1*

### Training Result



## Search Space

The search space is the uniform distribution between minimum value and maximum value on each convolutional weight.

def get\_space():

space = {}

for name, parameter in model.named\_parameters():

if 'conv' in name and 'weight' in name:

space[name] = hp.uniform(name, 0.30, 0.99)

return space

## Score Function

score = (1-(val\_accuracy/100.)) + (alpha \* (1-sparsity/100.))

## Suggest Method

For TPE:

best = fmin(objective, space, algo=tpe.suggest, max\_evals=args.rounds)

For Random:

best = fmin(objective, space, algo=hyperopt.rand.suggest, max\_evals=args.rounds)

## Pruning Result

### Random method without constraint

|  |
| --- |
| Command:  python auto.py --arch resnet56\_cifar --gpus=0 --lr=0.003 ../../../data.cifar10 -b=128 -j=1 --deterministic --resume='./pertrain base line/2018.11.08-213504/best.pth.tar' --rounds=20 --epochs=30 --method=’random’ |

Search Space:

def get\_space():

space = {}

for name, parameter in model.named\_parameters():

if 'conv' in name and 'weight' in name:

space[name] = hp.uniform(name, 0.30, 0.99)

return space

Training:

if(args.pruner\_constraint == True):

PrunerConstraint = True

else:

PrunerConstraint = False

...

Expected\_Sparsity\_Level\_High = 72.5

Expected\_Sparsity\_Level\_Low = 67.5

...

def objective(space):

…

train\_accuracy = train(i,criterion, optimizer, compression\_scheduler)

val\_accuracy = validate() # Validate hyperparameter setting

t, sparsity = distiller.weights\_sparsity\_tbl\_summary(model, return\_total\_sparsity=True)

…

if (PrunerConstraint == True and i >= PrunerEpoch and (sparsity < Expected\_Sparsity\_Level\_Low or sparsity > Expected\_Sparsity\_Level\_High)):

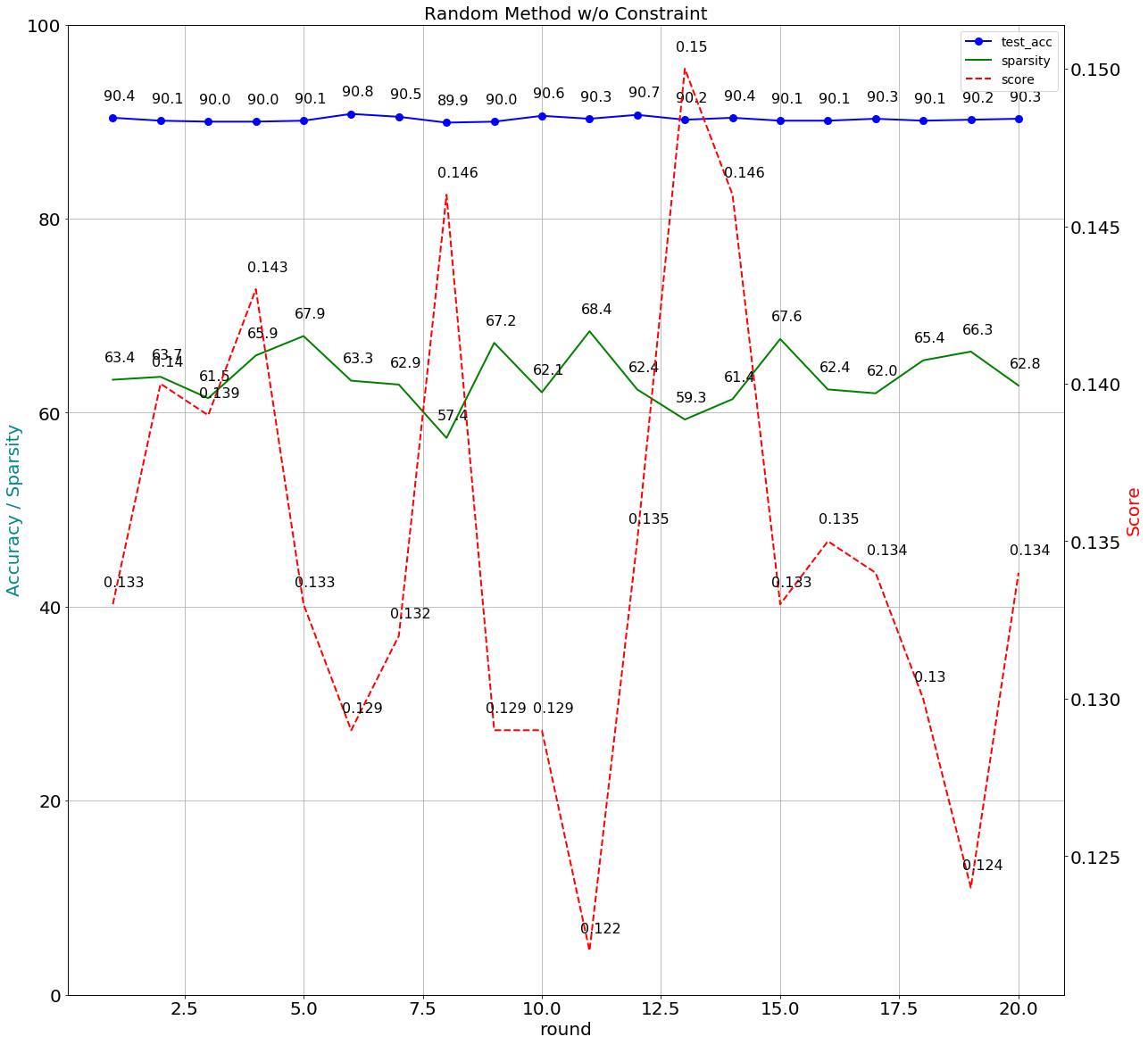
break

…

return local\_min\_score

round:1, score:0.1329, train\_acc:99.2178, val\_acc:97.5400, test\_acc:90.3700, sparsity:63.4339  
round:2, score:0.1397, train\_acc:99.2000, val\_acc:96.9200, test\_acc:90.0700, sparsity:63.6942  
round:3, score:0.1390, train\_acc:99.2111, val\_acc:97.5600, test\_acc:90.0100, sparsity:61.5247  
round:4, score:0.1428, train\_acc:98.5911, val\_acc:95.2600, test\_acc:90.0000, sparsity:65.9488  
round:5, score:0.1333, train\_acc:98.7778, val\_acc:96.1600, test\_acc:90.1200, sparsity:67.9112  
round:6, score:0.1291, train\_acc:99.5467, val\_acc:97.8800, test\_acc:90.7800, sparsity:63.3144  
round:7, score:0.1315, train\_acc:99.5200, val\_acc:97.8000, test\_acc:90.5300, sparsity:62.8851  
round:8, score:0.1456, train\_acc:99.5111, val\_acc:98.2200, test\_acc:89.8900, sparsity:57.4099  
round:9, score:0.1288, train\_acc:99.0044, val\_acc:96.9600, test\_acc:89.9700, sparsity:67.1856  
round:10, score:0.1292, train\_acc:99.6444, val\_acc:98.2200, test\_acc:90.6200, sparsity:62.0543  
round:11, score:0.1221, train\_acc:99.2667, val\_acc:97.1200, test\_acc:90.3000, sparsity:68.3670  
round:12, score:0.1345, train\_acc:99.4578, val\_acc:97.8400, test\_acc:90.7100, sparsity:62.3675  
round:13, score:0.1497, train\_acc:99.1933, val\_acc:96.9400, test\_acc:90.2200, sparsity:59.3125  
round:14, score:0.1459, train\_acc:99.1444, val\_acc:96.5000, test\_acc:90.3800, sparsity:61.4334  
round:15, score:0.1333, train\_acc:98.9800, val\_acc:96.3800, test\_acc:90.1100, sparsity:67.6239  
round:16, score:0.1347, train\_acc:99.4556, val\_acc:97.6200, test\_acc:90.1300, sparsity:62.4456  
round:17, score:0.1341, train\_acc:99.5222, val\_acc:97.8800, test\_acc:90.2900, sparsity:62.0389  
round:18, score:0.1298, train\_acc:99.2333, val\_acc:96.8400, test\_acc:90.1000, sparsity:65.4029  
round:19, score:0.1240, train\_acc:99.3911, val\_acc:97.6200, test\_acc:90.2000, sparsity:66.3396  
round:20, score:0.1339, train\_acc:99.2667, val\_acc:97.1600, test\_acc:90.2900, sparsity:62.8307

’module.conv1.weight': 0.519453534851879,  
’module.layer1.0.conv1.weight': 0.601271639735327,  
’module.layer1.0.conv2.weight': 0.726769891797648,  
’module.layer1.1.conv1.weight': 0.508005745340092,  
’module.layer1.1.conv2.weight': 0.758294720818404,  
’module.layer1.2.conv1.weight': 0.646381526901049,  
’module.layer1.2.conv2.weight': 0.937157476348435,  
’module.layer1.3.conv1.weight': 0.767553787554462,  
’module.layer1.3.conv2.weight': 0.76934172597243,  
’module.layer1.4.conv1.weight': 0.763386259515083,  
’module.layer1.4.conv2.weight': 0.908742731915059,  
’module.layer1.5.conv1.weight': 0.805953997682668,  
’module.layer1.5.conv2.weight': 0.379755691920525,  
’module.layer1.6.conv1.weight': 0.368141891071684,  
’module.layer1.6.conv2.weight': 0.343499338701901,  
’module.layer1.7.conv1.weight': 0.393293204931277,  
’module.layer1.7.conv2.weight': 0.595538741654875,  
’module.layer1.8.conv1.weight': 0.525133091388918,  
’module.layer1.8.conv2.weight': 0.695164216626768,  
’module.layer2.0.conv1.weight': 0.472049933125105,  
’module.layer2.0.conv2.weight': 0.386504612864878,  
’module.layer2.1.conv1.weight': 0.914350048478572,  
’module.layer2.1.conv2.weight': 0.397557517400476,  
’module.layer2.2.conv1.weight': 0.379297200027578,  
’module.layer2.2.conv2.weight': 0.920091448446573,  
’module.layer2.3.conv1.weight': 0.972311697355797,  
’module.layer2.3.conv2.weight': 0.690845119425622,  
’module.layer2.4.conv1.weight': 0.920073490319871,  
’module.layer2.4.conv2.weight': 0.615004502439158,  
’module.layer2.5.conv1.weight': 0.798276007973996,  
’module.layer2.5.conv2.weight': 0.985300850012866,  
’module.layer2.6.conv1.weight': 0.586577168162162,  
’module.layer2.6.conv2.weight': 0.550362174640348,  
’module.layer2.7.conv1.weight': 0.826528166270262,  
’module.layer2.7.conv2.weight': 0.391337063866534,  
’module.layer2.8.conv1.weight': 0.521855154043542,  
’module.layer2.8.conv2.weight': 0.539052580992194,  
’module.layer3.0.conv1.weight': 0.574417946579844,  
’module.layer3.0.conv2.weight': 0.840507684201953,  
’module.layer3.1.conv1.weight': 0.725209367141896,  
’module.layer3.1.conv2.weight': 0.886489269126627,  
’module.layer3.2.conv1.weight': 0.724017752889759,  
’module.layer3.2.conv2.weight': 0.300666443850314,  
’module.layer3.3.conv1.weight': 0.903523709266426,  
’module.layer3.3.conv2.weight': 0.464714018583245,  
’module.layer3.4.conv2.weight': 0.570831766815719,  
’module.layer3.5.conv1.weight': 0.743720363557057,  
’module.layer3.5.conv2.weight': 0.744680627513436,  
’module.layer3.6.conv1.weight': 0.886918610669229,  
’module.layer3.6.conv2.weight': 0.839630135202238,  
’module.layer3.7.conv1.weight': 0.545559014171576,  
’module.layer3.7.conv2.weight': 0.562193763806044,  
’module.layer3.8.conv1.weight': 0.481831252331312,  
’module.layer3.8.conv2.weight': 0.934626398180086,  
‘module.layer3.4.conv1.weight': 0.717924244814645,



### Random method with constraint of 70±2.5% sparsity

Version 1:

|  |
| --- |
| Command:  python auto.py --arch resnet56\_cifar --gpus=0 --lr=0.003 ../../../data.cifar10 -b=128 -j=1 --deterministic --resume='./pertrain base line/2018.11.08-213504/best.pth.tar' --rounds=20 --epochs=30 --method=’random’ --pruner-constraint |

Search Space:

def get\_space():

space = {}

for name, parameter in model.named\_parameters():

if 'conv' in name and 'weight' in name:

space[name] = hp.uniform(name, 0.30, 0.99)

return space

Training:

if(args.pruner\_constraint == True):

PrunerConstraint = True

else:

PrunerConstraint = False

...

Expected\_Sparsity\_Level\_High = 72.5

Expected\_Sparsity\_Level\_Low = 67.5

...

def objective(space):

…

train\_accuracy = train(i,criterion, optimizer, compression\_scheduler)

val\_accuracy = validate() # Validate hyperparameter setting

t, sparsity = distiller.weights\_sparsity\_tbl\_summary(model, return\_total\_sparsity=True)

…

if (PrunerConstraint == True and i >= PrunerEpoch and (sparsity < Expected\_Sparsity\_Level\_Low or sparsity > Expected\_Sparsity\_Level\_High)):

break

…

return local\_min\_score

round:1, score:0.1645, train\_acc:91.4511, val\_acc:93.5800, test\_acc:88.2200, sparsity:66.5504  
round:2, score:0.2011, train\_acc:89.0778, val\_acc:92.6400, test\_acc:86.9400, sparsity:57.5066  
round:3, score:0.1676, train\_acc:92.9022, val\_acc:95.2800, test\_acc:88.7800, sparsity:59.8809  
round:4, score:0.1566, train\_acc:94.2422, val\_acc:95.9200, test\_acc:88.5700, sparsity:61.4135  
round:5, score:0.1747, train\_acc:92.0444, val\_acc:94.1800, test\_acc:88.4200, sparsity:61.1647  
round:6, score:0.1752, train\_acc:88.4511, val\_acc:92.3200, test\_acc:87.2300, sparsity:67.2153  
round:7, score:0.1759, train\_acc:92.1822, val\_acc:94.6600, test\_acc:88.4100, sparsity:59.1565  
round:8, score:0.1444, train\_acc:95.2178, val\_acc:96.6200, test\_acc:89.2400, sparsity:63.1227  
round:9, score:0.2006, train\_acc:88.6067, val\_acc:91.5600, test\_acc:86.8200, sparsity:61.2630  
round:10, score:0.1539, train\_acc:92.2044, val\_acc:94.5600, test\_acc:88.3500, sparsity:66.8386  
round:11, score:0.1585, train\_acc:91.5978, val\_acc:94.3400, test\_acc:87.7700, sparsity:66.0233  
round:12, score:0.1715, train\_acc:90.4511, val\_acc:93.2200, test\_acc:87.1500, sparsity:65.4194  
round:13, score:0.1564, train\_acc:91.9178, val\_acc:94.1600, test\_acc:88.0500, sparsity:67.3181  
round:14, score:0.1497, train\_acc:92.7689, val\_acc:95.0400, test\_acc:88.4800, sparsity:66.6444  
round:15, score:0.1152, train\_acc:98.9000, val\_acc:96.6000, test\_acc:90.3800, sparsity:72.1488  
round:16, score:0.1879, train\_acc:91.4178, val\_acc:92.5400, test\_acc:87.0300, sparsity:62.2461  
round:17, score:0.1554, train\_acc:92.5511, val\_acc:94.7200, test\_acc:88.1800, sparsity:65.7879  
round:18, score:0.1340, train\_acc:98.8422, val\_acc:95.8800, test\_acc:89.8200, sparsity:67.9245  
round:19, score:0.1711, train\_acc:91.4289, val\_acc:94.5800, test\_acc:88.4100, sparsity:61.0408  
round:20, score:0.1574, train\_acc:92.8156, val\_acc:94.8800, test\_acc:88.8300, sparsity:64.5942

‘module.conv1.weight': 0.906957785929862,  
‘module.layer1.0.conv1.weight': 0.326796506341287,  
‘module.layer1.0.conv2.weight': 0.846398861018247,  
‘module.layer1.1.conv1.weight': 0.84533834417431,  
‘module.layer1.1.conv2.weight': 0.489599625347857,  
‘module.layer1.2.conv1.weight': 0.3435984722888,  
‘module.layer1.2.conv2.weight': 0.775349634247014,  
‘module.layer1.3.conv1.weight': 0.660027847705321,  
‘module.layer1.3.conv2.weight': 0.405484578959248,  
‘module.layer1.4.conv1.weight': 0.684085323094024,  
‘module.layer1.4.conv2.weight': 0.580759493177218,  
‘module.layer1.5.conv1.weight': 0.792632410740443,  
‘module.layer1.5.conv2.weight': 0.434046149274458,  
‘module.layer1.6.conv1.weight': 0.675729759703671,  
‘module.layer1.6.conv2.weight': 0.783211679910042,  
‘module.layer1.7.conv1.weight': 0.935708394829227,  
‘module.layer1.7.conv2.weight': 0.401720611171903,  
‘module.layer1.8.conv1.weight': 0.500830930153686,  
‘module.layer1.8.conv2.weight': 0.355737348044877,  
‘module.layer2.0.conv1.weight': 0.941925888522003,  
‘module.layer2.0.conv2.weight': 0.455280907435438,  
‘module.layer2.1.conv1.weight': 0.718546590315616,  
‘module.layer2.1.conv2.weight': 0.732413919670848,  
‘module.layer2.2.conv1.weight': 0.617701686995196,  
‘module.layer2.2.conv2.weight': 0.623592830554754,  
‘module.layer2.3.conv1.weight': 0.851018665793473,  
‘module.layer2.3.conv2.weight': 0.336900943081801,  
‘module.layer2.4.conv1.weight': 0.609943863136227,  
'module.layer2.4.conv2.weight': 0.327545881052538,  
‘module.layer2.5.conv1.weight': 0.755912537582466,  
‘module.layer2.5.conv2.weight': 0.300216150608542,  
‘module.layer2.6.conv1.weight': 0.447068219324498,  
‘module.layer2.6.conv2.weight': 0.858982744969615,  
‘module.layer2.7.conv1.weight': 0.548628940887679,  
‘module.layer2.7.conv2.weight': 0.346923473882068,  
‘module.layer2.8.conv1.weight': 0.979553238973532,  
‘module.layer2.8.conv2.weight': 0.955200089147754,  
‘module.layer3.0.conv1.weight': 0.750567483899369,  
‘module.layer3.0.conv2.weight': 0.833317773191016,  
‘module.layer3.1.conv1.weight': 0.799399440599761,  
‘module.layer3.1.conv2.weight': 0.621054440673115,  
‘module.layer3.2.conv1.weight': 0.568551543026765,  
‘module.layer3.2.conv2.weight': 0.601257045189569,  
‘module.layer3.3.conv1.weight': 0.955131451386637,  
‘module.layer3.3.conv2.weight': 0.855316557550849,  
‘module.layer3.4.conv1.weight': 0.765216557745791,  
‘module.layer3.4.conv2.weight': 0.734353016590695,  
‘module.layer3.5.conv1.weight': 0.483217672352786,  
‘module.layer3.5.conv2.weight': 0.940586614836248,  
‘module.layer3.6.conv1.weight': 0.313836980644055,  
‘module.layer3.6.conv2.weight': 0.983817708659908,  
‘module.layer3.7.conv1.weight': 0.909327867847061,  
‘module.layer3.7.conv2.weight': 0.825570975564672,  
‘module.layer3.8.conv1.weight': 0.947950635327218,  
‘module.layer3.8.conv2.weight': 0.730800428932419,

Version 2:

|  |
| --- |
| Command:  python auto.py --arch resnet56\_cifar --gpus=0 --lr=0.003 ../../../data.cifar10 -b=128 -j=1 --deterministic --resume='./pertrain base line/2018.11.08-213504/best.pth.tar' --rounds=20 --epochs=30 --method=’random’ --pruner-constraint |

Search Space:

def get\_space():

space = {}

for name, parameter in model.named\_parameters():

if 'conv' in name and 'weight' in name:

space[name] = hp.uniform(name, 0.55, 0.85)

return space

Training:

if(args.pruner\_constraint == True):

PrunerConstraint = True

else:

PrunerConstraint = False

...

Expected\_Sparsity\_Level\_High = 72.5

Expected\_Sparsity\_Level\_Low = 67.5

...

def objective(space):

…

train\_accuracy = train(i,criterion, optimizer, compression\_scheduler)

val\_accuracy = validate() # Validate hyperparameter setting

t, sparsity = distiller.weights\_sparsity\_tbl\_summary(model, return\_total\_sparsity=True)

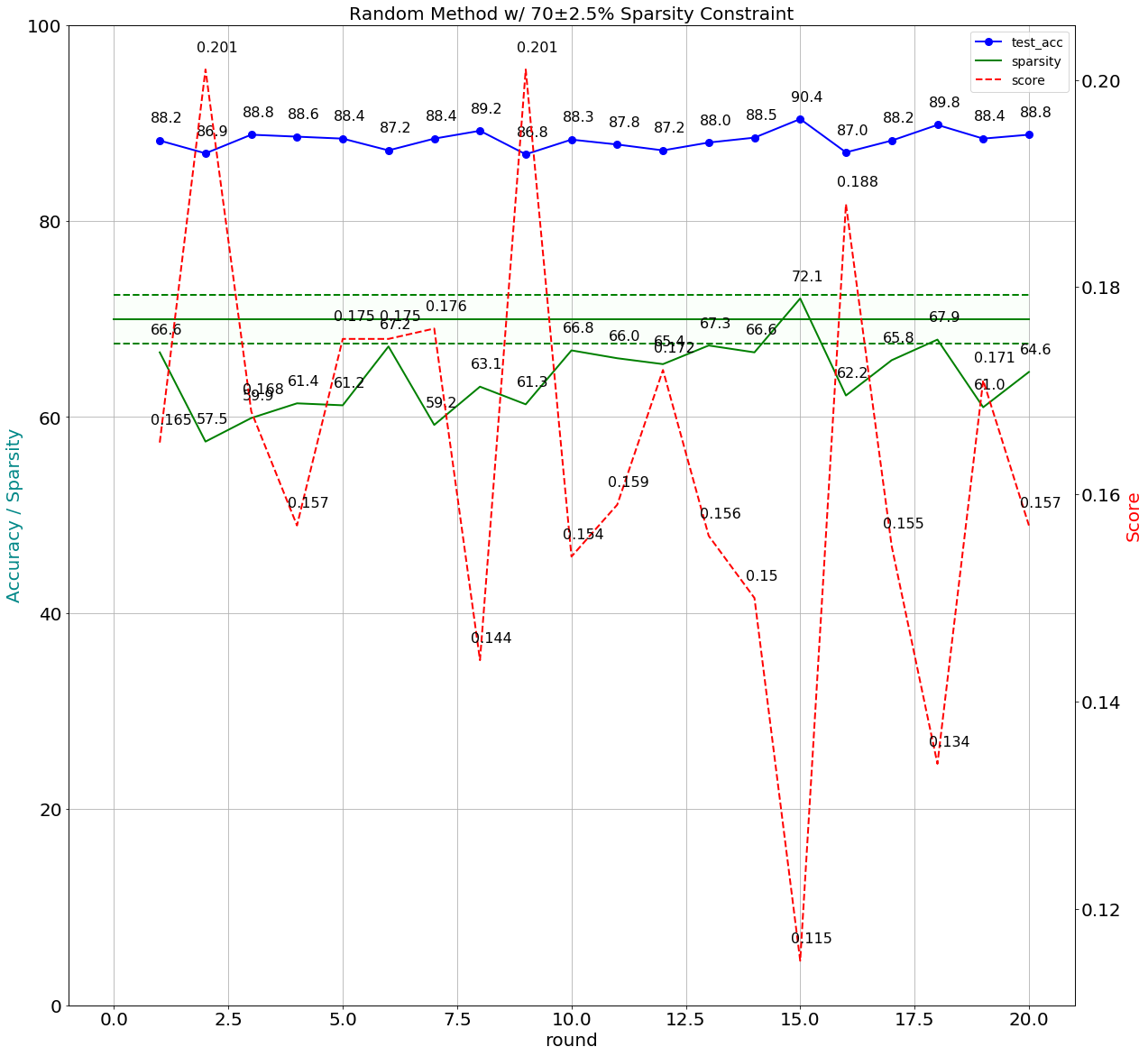
…

if (PrunerConstraint == True and i >= PrunerEpoch and (sparsity < Expected\_Sparsity\_Level\_Low or sparsity > Expected\_Sparsity\_Level\_High)):

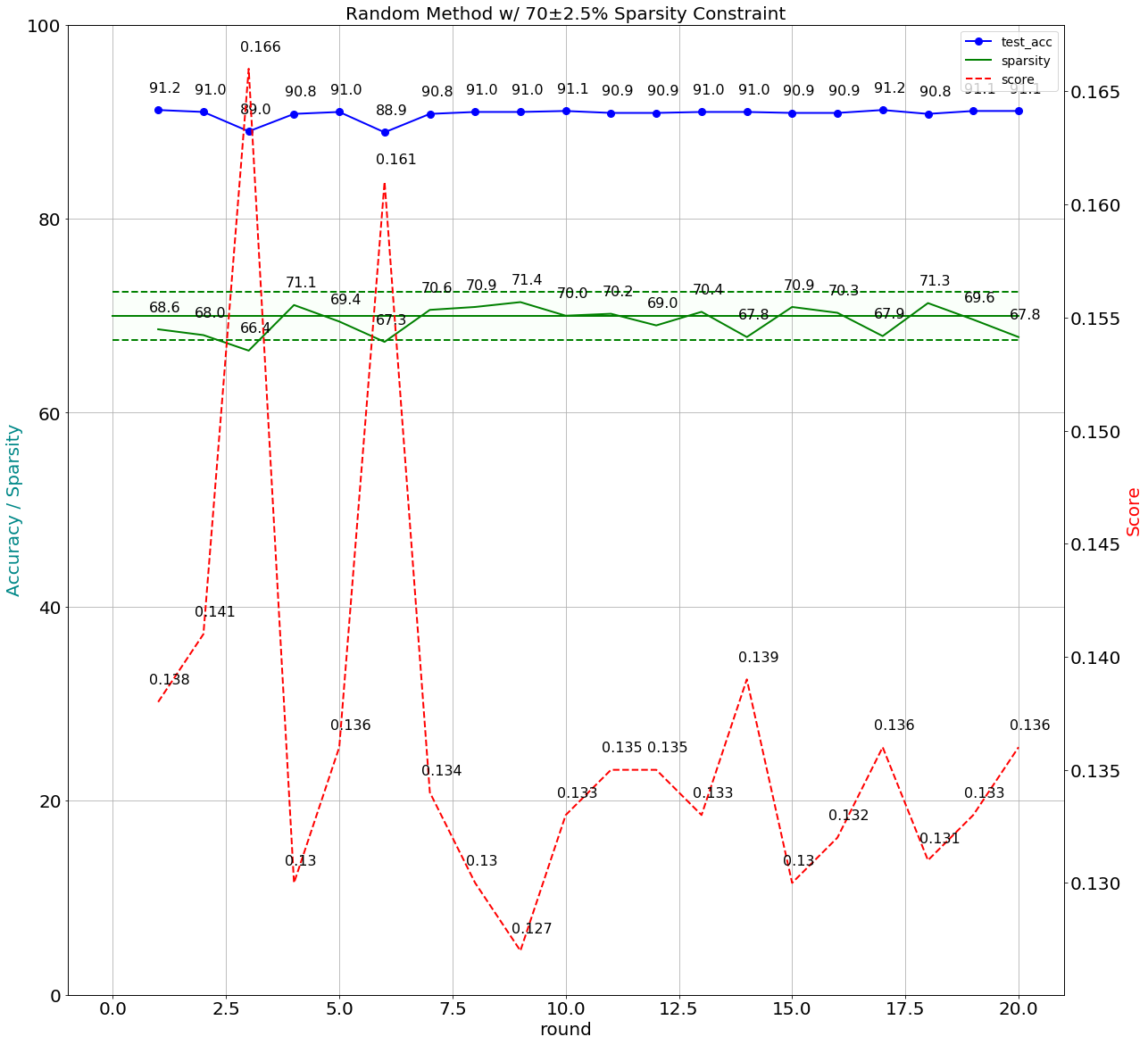
break

…

return local\_min\_score



round:1, score:0.1375, train\_acc:97.6578, val\_acc:95.0000, test\_acc:91.1600, sparsity:68.5825  
round:2, score:0.1407, train\_acc:97.6556, val\_acc:95.0000, test\_acc:90.9500, sparsity:67.9800  
round:3, score:0.1664, train\_acc:93.6911, val\_acc:93.4400, test\_acc:89.0100, sparsity:66.3998  
round:4, score:0.1301, train\_acc:97.7444, val\_acc:95.2600, test\_acc:90.8300, sparsity:71.1025  
round:5, score:0.1355, train\_acc:97.6422, val\_acc:95.6000, test\_acc:90.9800, sparsity:69.3519  
round:6, score:0.1610, train\_acc:93.5622, val\_acc:93.7000, test\_acc:88.9200, sparsity:67.3438  
round:7, score:0.1343, train\_acc:97.4778, val\_acc:94.8000, test\_acc:90.8500, sparsity:70.6276  
round:8, score:0.1298, train\_acc:97.6578, val\_acc:95.5000, test\_acc:91.0200, sparsity:70.9180  
round:9, score:0.1270, train\_acc:97.6822, val\_acc:95.2200, test\_acc:90.9700, sparsity:71.4041  
round:10, score:0.1328, train\_acc:97.5867, val\_acc:94.9800, test\_acc:91.0700, sparsity:69.9838  
round:11, score:0.1345, train\_acc:97.4867, val\_acc:95.1400, test\_acc:90.9300, sparsity:70.1533  
round:12, score:0.1347, train\_acc:97.7356, val\_acc:95.3200, test\_acc:90.8900, sparsity:69.0207  
round:13, score:0.1327, train\_acc:97.8044, val\_acc:95.2000, test\_acc:90.9500, sparsity:70.3825  
round:14, score:0.1387, train\_acc:97.9378, val\_acc:95.7200, test\_acc:91.0100, sparsity:67.7771  
round:15, score:0.1304, train\_acc:97.5133, val\_acc:95.3800, test\_acc:90.8800, sparsity:70.9170  
round:16, score:0.1316, train\_acc:97.6422, val\_acc:95.2400, test\_acc:90.9200, sparsity:70.2825  
round:17, score:0.1363, train\_acc:97.7711, val\_acc:95.7600, test\_acc:91.2500, sparsity:67.9026  
round:18, score:0.1311, train\_acc:97.5956, val\_acc:95.3800, test\_acc:90.7800, sparsity:71.3114  
round:19, score:0.1333, train\_acc:97.7267, val\_acc:95.3600, test\_acc:91.1400, sparsity:69.5795  
round:20, score:0.1359, train\_acc:97.8689, val\_acc:96.0600, test\_acc:91.1300, sparsity:67.8443



### TPE method without constraint

|  |
| --- |
| Command:  python auto.py --arch resnet56\_cifar --gpus=0 --lr=0.003 ../../../data.cifar10 -b=128 -j=1 --deterministic --resume='./pertrain base line/2018.11.08-213504/best.pth.tar' --rounds=20 --epochs=30 --method=’tpe’ |

Search Space:

def get\_space():

space = {}

for name, parameter in model.named\_parameters():

if 'conv' in name and 'weight' in name:

space[name] = hp.uniform(name, 0.30, 0.99)

return space

Training:

if(args.pruner\_constraint == True):

PrunerConstraint = True

else:

PrunerConstraint = False

...

Expected\_Sparsity\_Level\_High = 72.5

Expected\_Sparsity\_Level\_Low = 67.5

...

def objective(space):

…

train\_accuracy = train(i,criterion, optimizer, compression\_scheduler)

val\_accuracy = validate() # Validate hyperparameter setting

t, sparsity = distiller.weights\_sparsity\_tbl\_summary(model, return\_total\_sparsity=True)

…

if (PrunerConstraint == True and i >= PrunerEpoch and (sparsity < Expected\_Sparsity\_Level\_Low or sparsity > Expected\_Sparsity\_Level\_High)):

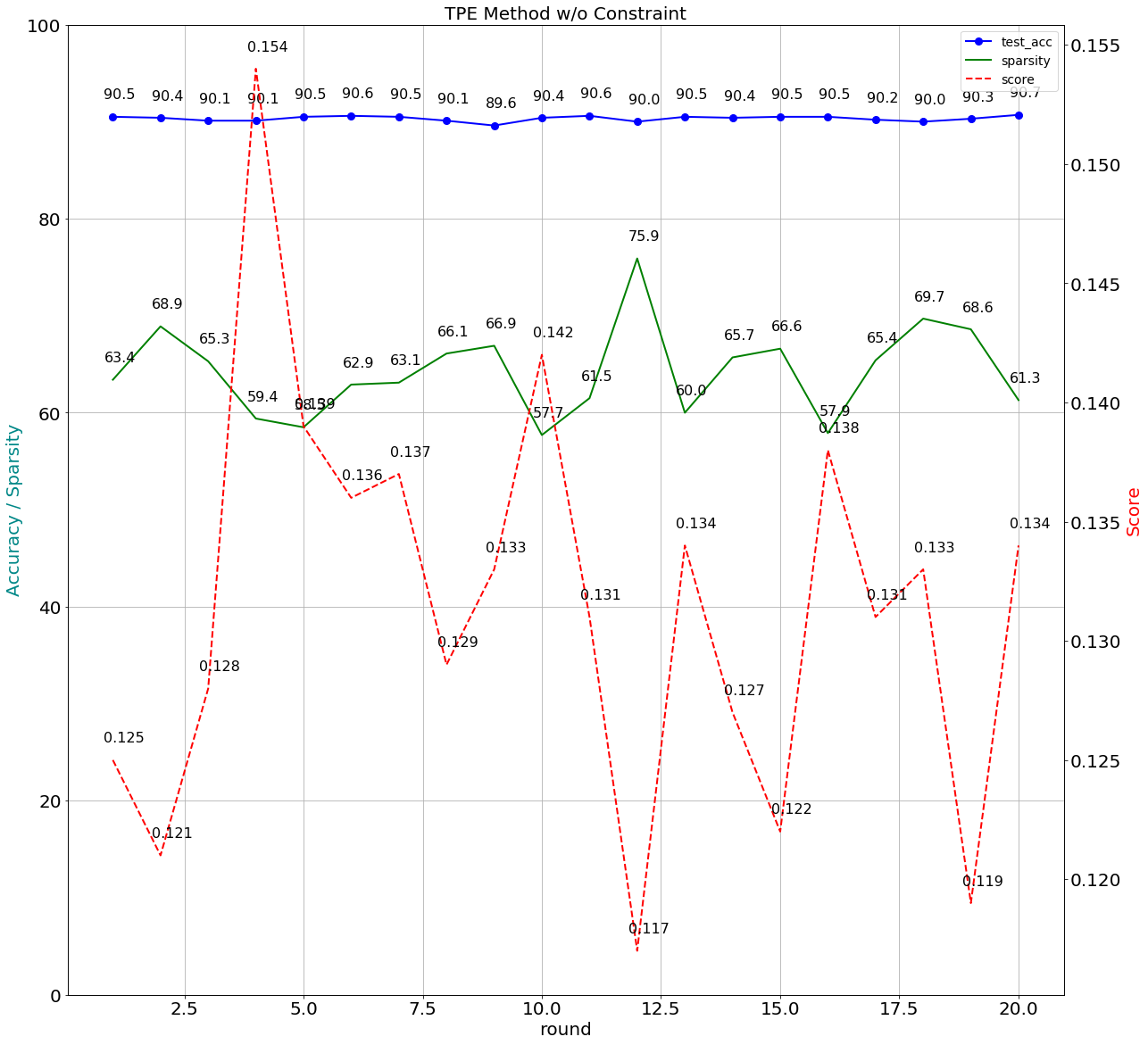
break

…

return local\_min\_score

round:1, score:0.1250, train\_acc:99.6578, val\_acc:97.9800, test\_acc:90.5200, sparsity:63.3868  
round:2, score:0.1214, train\_acc:99.2489, val\_acc:97.1000, test\_acc:90.4100, sparsity:68.9222  
round:3, score:0.1284, train\_acc:99.3200, val\_acc:97.5600, test\_acc:90.0800, sparsity:65.3271  
round:4, score:0.1541, train\_acc:98.7867, val\_acc:96.4400, test\_acc:90.1400, sparsity:59.4185  
round:5, score:0.1394, train\_acc:99.6600, val\_acc:98.5200, test\_acc:90.5300, sparsity:58.4627  
round:6, score:0.1358, train\_acc:99.4089, val\_acc:97.2000, test\_acc:90.5900, sparsity:62.8610  
round:7, score:0.1365, train\_acc:99.2667, val\_acc:97.2600, test\_acc:90.5400, sparsity:63.0949  
round:8, score:0.1289, train\_acc:99.2022, val\_acc:97.2000, test\_acc:90.0800, sparsity:66.1005  
round:9, score:0.1333, train\_acc:98.7867, val\_acc:96.6000, test\_acc:89.5800, sparsity:66.8992  
round:10, score:0.1418, train\_acc:99.6622, val\_acc:97.9600, test\_acc:90.4300, sparsity:57.6561  
round:11, score:0.1308, train\_acc:99.6756, val\_acc:98.3000, test\_acc:90.5700, sparsity:61.5354  
round:12, score:0.1165, train\_acc:98.0978, val\_acc:95.4200, test\_acc:89.9500, sparsity:75.8952  
round:13, score:0.1343, train\_acc:99.5933, val\_acc:98.1800, test\_acc:90.4500, sparsity:59.9705  
round:14, score:0.1268, train\_acc:99.3889, val\_acc:97.5600, test\_acc:90.4000, sparsity:65.6596  
round:15, score:0.1221, train\_acc:99.3422, val\_acc:97.6200, test\_acc:90.5400, sparsity:66.6363  
round:16, score:0.1381, train\_acc:99.7467, val\_acc:98.5200, test\_acc:90.4900, sparsity:57.9126  
round:17, score:0.1313, train\_acc:99.2244, val\_acc:97.2400, test\_acc:90.2200, sparsity:65.4473  
round:18, score:0.1328, train\_acc:98.5089, val\_acc:95.8000, test\_acc:89.9600, sparsity:69.7375  
round:19, score:0.1192, train\_acc:99.3600, val\_acc:97.3400, test\_acc:90.3300, sparsity:68.5837  
round:20, score:0.1337, train\_acc:99.5911, val\_acc:97.9400, test\_acc:90.6500, sparsity:61.2959

‘module.conv1.weight': 0.868803883740578,  
‘module.layer1.0.conv1.weight': 0.847789658066599,  
‘module.layer1.0.conv2.weight': 0.574262829428587,  
‘module.layer1.1.conv1.weight': 0.392838908394838,  
‘module.layer1.1.conv2.weight': 0.680772868968364,  
‘module.layer1.2.conv1.weight': 0.944061828783238,  
‘module.layer1.2.conv2.weight': 0.905331099352333,  
‘module.layer1.3.conv1.weight': 0.806255849987785,  
‘module.layer1.3.conv2.weight': 0.561129289872814,  
‘module.layer1.4.conv1.weight': 0.80870261405486,  
‘module.layer1.4.conv2.weight': 0.494439995536961,  
‘module.layer1.5.conv1.weight': 0.907794705131917,  
‘module.layer1.5.conv2.weight': 0.673254583352526,  
‘module.layer1.6.conv1.weight': 0.932986084663167,  
‘module.layer1.6.conv2.weight': 0.761617052260549,  
‘module.layer1.7.conv1.weight': 0.547482128369246,  
‘module.layer1.7.conv2.weight': 0.699383705527818,  
‘module.layer1.8.conv1.weight': 0.899599497588079,  
‘module.layer1.8.conv2.weight': 0.380596443607256,  
‘module.layer2.0.conv1.weight': 0.542018286462394,  
‘module.layer2.0.conv2.weight': 0.779463219134289,  
‘module.layer2.1.conv1.weight': 0.686097140833618,  
‘module.layer2.1.conv2.weight': 0.526496910263335,  
‘module.layer2.2.conv1.weight': 0.343402055615396,  
‘module.layer2.2.conv2.weight': 0.839446047866604,  
‘module.layer2.3.conv1.weight': 0.821880369099761,  
'module.layer2.3.conv2.weight': 0.726773437911085,  
‘module.layer2.4.conv1.weight': 0.760749743114005,  
‘module.layer2.4.conv2.weight': 0.855588601189668,  
‘module.layer2.5.conv1.weight': 0.653141249648415,  
‘module.layer2.5.conv2.weight': 0.677469203391921,  
‘module.layer2.6.conv1.weight': 0.881792294234138,  
‘module.layer2.6.conv2.weight': 0.970271867851428,  
‘module.layer2.7.conv1.weight': 0.577934191090879,  
‘module.layer2.7.conv2.weight': 0.807236660616409,  
‘module.layer2.8.conv1.weight': 0.562945173426366,  
‘module.layer2.8.conv2.weight': 0.794507914223084,  
‘module.layer3.0.conv1.weight': 0.633372452157346,  
‘module.layer3.0.conv2.weight': 0.961327722069813,  
‘module.layer3.1.conv1.weight': 0.69362934055276,  
‘module.layer3.1.conv2.weight': 0.729874360314567,  
‘module.layer3.2.conv1.weight': 0.935577549770274,  
‘module.layer3.2.conv2.weight': 0.618159617557541,  
‘module.layer3.3.conv1.weight': 0.65537874423706,  
‘module.layer3.3.conv2.weight': 0.902453270508724,  
‘module.layer3.4.conv1.weight': 0.462919919275933,  
‘module.layer3.4.conv2.weight': 0.620307351161171,  
‘module.layer3.5.conv1.weight': 0.946742021222744,  
‘module.layer3.5.conv2.weight': 0.785683598242341,  
‘module.layer3.6.conv1.weight': 0.878266822408936,  
‘module.layer3.6.conv2.weight': 0.978773204613939,  
‘module.layer3.7.conv1.weight': 0.757007521384302,  
‘module.layer3.7.conv2.weight': 0.896653749518336,  
‘module.layer3.8.conv1.weight': 0.472719983269247,  
‘module.layer3.8.conv2.weight': 0.973763667266204,



### TPE method with constraint of 70±2.5% sparsity

Version 1:

|  |
| --- |
| Command:  python auto.py --arch resnet56\_cifar --gpus=0 --lr=0.03 ../../../data.cifar10 -b=128 -j=1 --deterministic --resume='./pertrain base line/2018.11.12-210049/best.pth.tar' --rounds=20 --epochs=20 --method=tpe --pruner-constraint |

Search Space:

def get\_space():

space = {}

for name, parameter in model.named\_parameters():

if 'conv' in name and 'weight' in name:

space[name] = hp.uniform(name, 0.30, 0.99)

return space

Training:

if(args.pruner\_constraint == True):

PrunerConstraint = True

else:

PrunerConstraint = False

...

Expected\_Sparsity\_Level\_High = 72.5

Expected\_Sparsity\_Level\_Low = 67.5

...

def objective(space):

…

train\_accuracy = train(i,criterion, optimizer, compression\_scheduler)

val\_accuracy = validate() # Validate hyperparameter setting

t, sparsity = distiller.weights\_sparsity\_tbl\_summary(model, return\_total\_sparsity=True)

…

if (PrunerConstraint == True and i >= PrunerEpoch and (sparsity < Expected\_Sparsity\_Level\_Low or sparsity > Expected\_Sparsity\_Level\_High)):

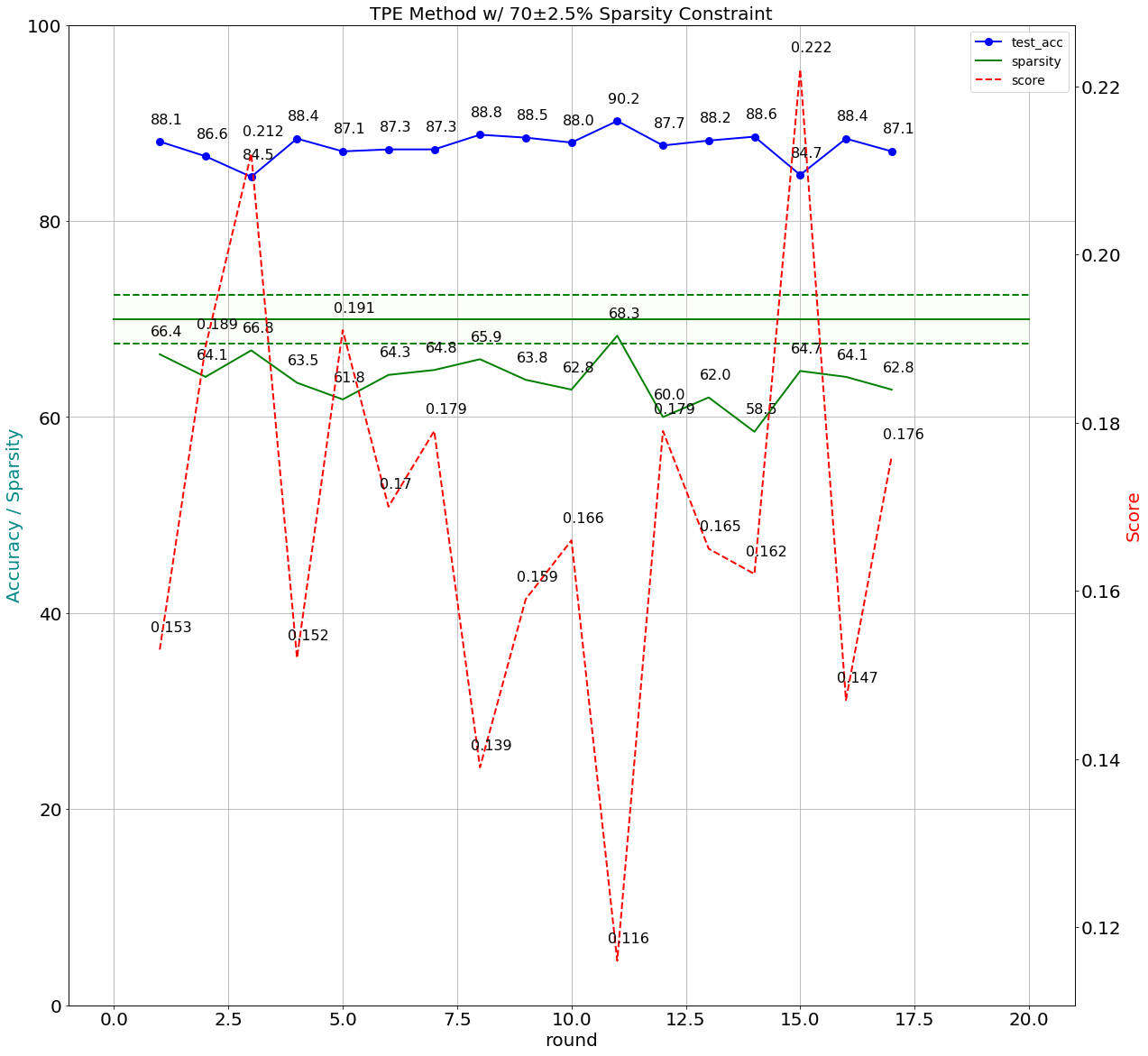
break

…

return local\_min\_score

round:1, score:0.1535, train\_acc:92.5667, val\_acc:94.7400, test\_acc:88.1000, sparsity:66.3634  
round:2, score:0.1888, train\_acc:87.6733, val\_acc:91.8800, test\_acc:86.6000, sparsity:64.1261  
round:3, score:0.2123, train\_acc:82.1822, val\_acc:88.7400, test\_acc:84.5100, sparsity:66.7591  
round:4, score:0.1523, train\_acc:92.8400, val\_acc:95.7200, test\_acc:88.3900, sparsity:63.4922  
round:5, score:0.1908, train\_acc:87.8933, val\_acc:92.3800, test\_acc:87.0800, sparsity:61.7990  
round:6, score:0.1702, train\_acc:91.5444, val\_acc:93.7000, test\_acc:87.2700, sparsity:64.2736  
round:7, score:0.1793, train\_acc:88.6244, val\_acc:92.6400, test\_acc:87.3000, sparsity:64.7719  
round:8, score:0.1388, train\_acc:94.4311, val\_acc:96.3400, test\_acc:88.8200, sparsity:65.9181  
round:9, score:0.1591, train\_acc:92.4044, val\_acc:94.9600, test\_acc:88.4700, sparsity:63.7628  
round:10, score:0.1657, train\_acc:91.4022, val\_acc:94.6000, test\_acc:88.0300, sparsity:62.7523  
round:11, score:0.1156, train\_acc:99.6956, val\_acc:97.1600, test\_acc:90.1500, sparsity:68.3384  
round:12, score:0.1795, train\_acc:91.3956, val\_acc:94.0600, test\_acc:87.7200, sparsity:59.9692  
round:13, score:0.1647, train\_acc:91.2689, val\_acc:94.9400, test\_acc:88.1900, sparsity:61.9591  
round:14, score:0.1624, train\_acc:93.7533, val\_acc:96.2000, test\_acc:88.5700, sparsity:58.5260  
round:15, score:0.2224, train\_acc:82.6756, val\_acc:88.3600, test\_acc:84.7200, sparsity:64.6548  
round:16, score:0.1472, train\_acc:94.2289, val\_acc:96.0400, test\_acc:88.4400, sparsity:64.1218  
round:17, score:0.1759, train\_acc:89.8044, val\_acc:93.5600, test\_acc:87.1200, sparsity:62.8186

‘module.conv1.weight': 0.498775841148591,  
‘module.layer1.0.conv1.weight': 0.851414562718911,  
‘module.layer1.0.conv2.weight': 0.191314114784789,  
‘module.layer1.1.conv1.weight': 0.878179117364064,  
‘module.layer1.1.conv2.weight': 0.51522309849354,  
‘module.layer1.2.conv1.weight': 0.89392584263347,  
‘module.layer1.2.conv2.weight': 0.633952902240188,  
‘module.layer1.3.conv1.weight': 0.91672235662212,  
‘module.layer1.3.conv2.weight': 0.822525927620518,  
‘module.layer1.4.conv1.weight': 0.925430459774281,  
‘module.layer1.4.conv2.weight': 0.0374609799121243,  
‘module.layer1.5.conv1.weight': 0.286095419799372,  
‘module.layer1.5.conv2.weight': 0.44288127741843,  
‘module.layer1.6.conv1.weight': 0.873420372245792,  
‘module.layer1.6.conv2.weight': 0.651433321901078,  
‘module.layer1.7.conv1.weight': 0.107739230425684,  
‘module.layer1.7.conv2.weight': 0.946693765902945,  
‘module.layer1.8.conv1.weight': 0.241864107941752,  
‘module.layer1.8.conv2.weight': 0.621881652697629,  
‘module.layer2.0.conv1.weight': 0.834728391946025,  
‘module.layer2.0.conv2.weight': 0.509128606201432,  
‘module.layer2.1.conv1.weight': 0.302084662379377,  
‘module.layer2.1.conv2.weight': 0.370861991679644,  
‘module.layer2.2.conv1.weight': 0.233955018746964,  
‘module.layer2.2.conv2.weight': 0.579624557558419,  
‘module.layer2.3.conv1.weight': 0.750318832664843,  
‘module.layer2.3.conv2.weight': 0.694026373138848,  
‘module.layer2.4.conv1.weight': 0.0747055063269187,  
‘module.layer2.4.conv2.weight': 0.485257308590196,  
‘module.layer2.5.conv1.weight': 0.881231861785112,  
‘module.layer2.5.conv2.weight': 0.204359009938375,  
‘module.layer2.6.conv1.weight': 0.848518681238732,  
‘module.layer2.6.conv2.weight': 0.929941251534044,  
‘module.layer2.7.conv1.weight': 0.915616750724122,  
‘module.layer2.7.conv2.weight': 0.21819395933601,  
‘module.layer2.8.conv1.weight': 0.491796788656375,  
‘module.layer2.8.conv2.weight': 0.779708715125794,  
‘module.layer3.0.conv1.weight': 0.474080568874371,  
‘module.layer3.0.conv2.weight': 0.610143430745419,  
‘module.layer3.1.conv1.weight': 0.461582617476816,  
‘module.layer3.1.conv2.weight': 0.52042652940282,  
‘module.layer3.2.conv1.weight': 0.916363432986558,  
‘module.layer3.2.conv2.weight': 0.424169051911123,  
‘module.layer3.3.conv1.weight': 0.392011117698078,  
‘module.layer3.3.conv2.weight': 0.247932260806036,  
‘module.layer3.4.conv1.weight': 0.466589411144595,  
‘module.layer3.4.conv2.weight': 0.881303139191081,  
‘module.layer3.5.conv1.weight': 0.974922864103847,  
‘module.layer3.5.conv2.weight': 0.767357319726048,  
‘module.layer3.6.conv1.weight': 0.0853981292582127,  
‘module.layer3.6.conv2.weight': 0.190372060612402,  
‘module.layer3.7.conv1.weight': 0.151529233650744,  
‘module.layer3.7.conv2.weight': 0.695678522742824,  
‘module.layer3.8.conv1.weight': 0.857640961753601,  
‘module.layer3.8.conv2.weight': 0.150235196440312,



Version 2:

|  |
| --- |
| Command:  python auto.py --arch resnet56\_cifar --gpus=0 --lr=0.03 ../../../data.cifar10 -b=128 -j=1 --deterministic --resume='./pertrain base line/2018.11.12-210049/best.pth.tar' --rounds=20 --epochs=20 --method=tpe --pruner-constraint |

Search Space:

def get\_space():

space = {}

for name, parameter in model.named\_parameters():

if 'conv' in name and 'weight' in name:

space[name] = hp.uniform(name, 0.55, 0.85)

return space

Training:

if(args.pruner\_constraint == True):

PrunerConstraint = True

else:

PrunerConstraint = False

...

Expected\_Sparsity\_Level\_High = 72.5

Expected\_Sparsity\_Level\_Low = 67.5

...

def objective(space):

…

train\_accuracy = train(i,criterion, optimizer, compression\_scheduler)

val\_accuracy = validate() # Validate hyperparameter setting

t, sparsity = distiller.weights\_sparsity\_tbl\_summary(model, return\_total\_sparsity=True)

…

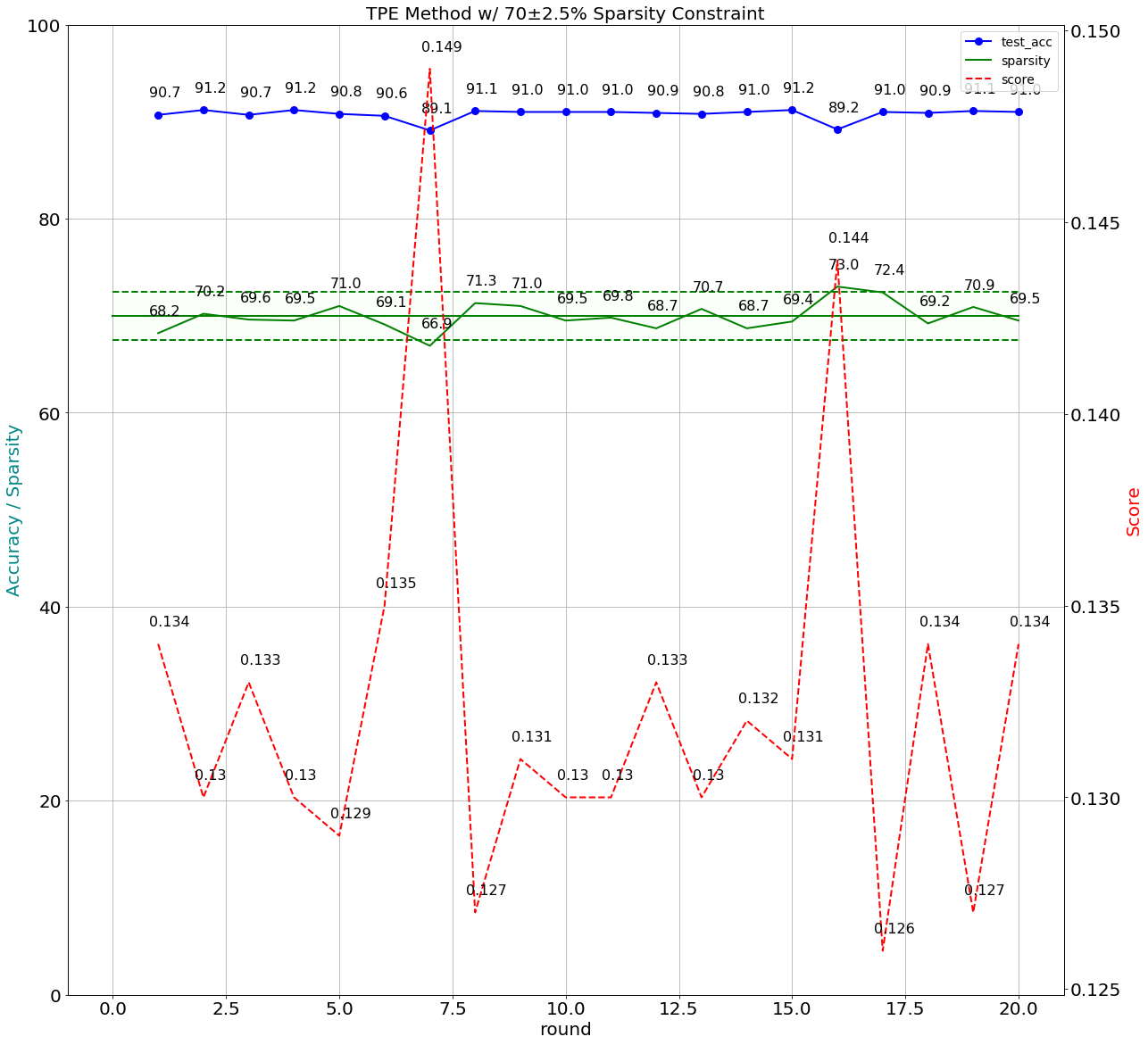
if (PrunerConstraint == True and i >= PrunerEpoch and (sparsity < Expected\_Sparsity\_Level\_Low or sparsity > Expected\_Sparsity\_Level\_High)):

break

…

return local\_min\_score

round:1, score:0.1341, train\_acc:97.6156, val\_acc:95.7600, test\_acc:90.7400, sparsity:68.1798  
round:2, score:0.1301, train\_acc:97.7556, val\_acc:95.8200, test\_acc:91.1500, sparsity:70.2498  
round:3, score:0.1326, train\_acc:97.6667, val\_acc:95.7400, test\_acc:90.7400, sparsity:69.6132  
round:4, score:0.1304, train\_acc:97.7600, val\_acc:95.7000, test\_acc:91.2000, sparsity:69.5233  
round:5, score:0.1285, train\_acc:97.3822, val\_acc:95.3200, test\_acc:90.7800, sparsity:71.0469  
round:6, score:0.1352, train\_acc:97.5956, val\_acc:95.7200, test\_acc:90.6400, sparsity:69.0773  
round:7, score:0.1494, train\_acc:94.1178, val\_acc:95.0000, test\_acc:89.0900, sparsity:66.8788  
round:8, score:0.1267, train\_acc:97.7778, val\_acc:95.7000, test\_acc:91.0600, sparsity:71.2922  
round:9, score:0.1305, train\_acc:97.4067, val\_acc:95.6400, test\_acc:90.9800, sparsity:71.0287  
round:10, score:0.1298, train\_acc:97.6311, val\_acc:95.8000, test\_acc:91.0500, sparsity:69.4719  
round:11, score:0.1299, train\_acc:97.5444, val\_acc:96.0600, test\_acc:91.0200, sparsity:69.8307  
round:12, score:0.1330, train\_acc:97.7733, val\_acc:95.8600, test\_acc:90.8700, sparsity:68.7279  
round:13, score:0.1302, train\_acc:97.3244, val\_acc:95.4000, test\_acc:90.7700, sparsity:70.7307  
round:14, score:0.1320, train\_acc:97.5644, val\_acc:95.8400, test\_acc:91.0100, sparsity:68.7455  
round:15, score:0.1305, train\_acc:97.6089, val\_acc:95.6400, test\_acc:91.1800, sparsity:69.4272  
round:16, score:0.1436, train\_acc:93.3844, val\_acc:93.7400, test\_acc:89.2200, sparsity:73.0056  
round:17, score:0.1257, train\_acc:97.4622, val\_acc:95.6600, test\_acc:90.9600, sparsity:72.3522  
round:18, score:0.1343, train\_acc:97.5467, val\_acc:95.4200, test\_acc:90.9100, sparsity:69.1600  
round:19, score:0.1268, train\_acc:97.6222, val\_acc:95.2800, test\_acc:91.1200, sparsity:70.8529  
round:20, score:0.1336, train\_acc:97.7489, val\_acc:95.7400, test\_acc:90.9500, sparsity:69.4609



​

# Second Implementation

* Build the baseline model

This procedure was as same as the first one, but the YAML file was just using the StepLR policies and listed below

*lr\_schedulers:*

*training\_lr:*

*class: StepLR*

*step\_size: 45*

*gamma: 0.10*

*policies:*

*- lr\_scheduler:*

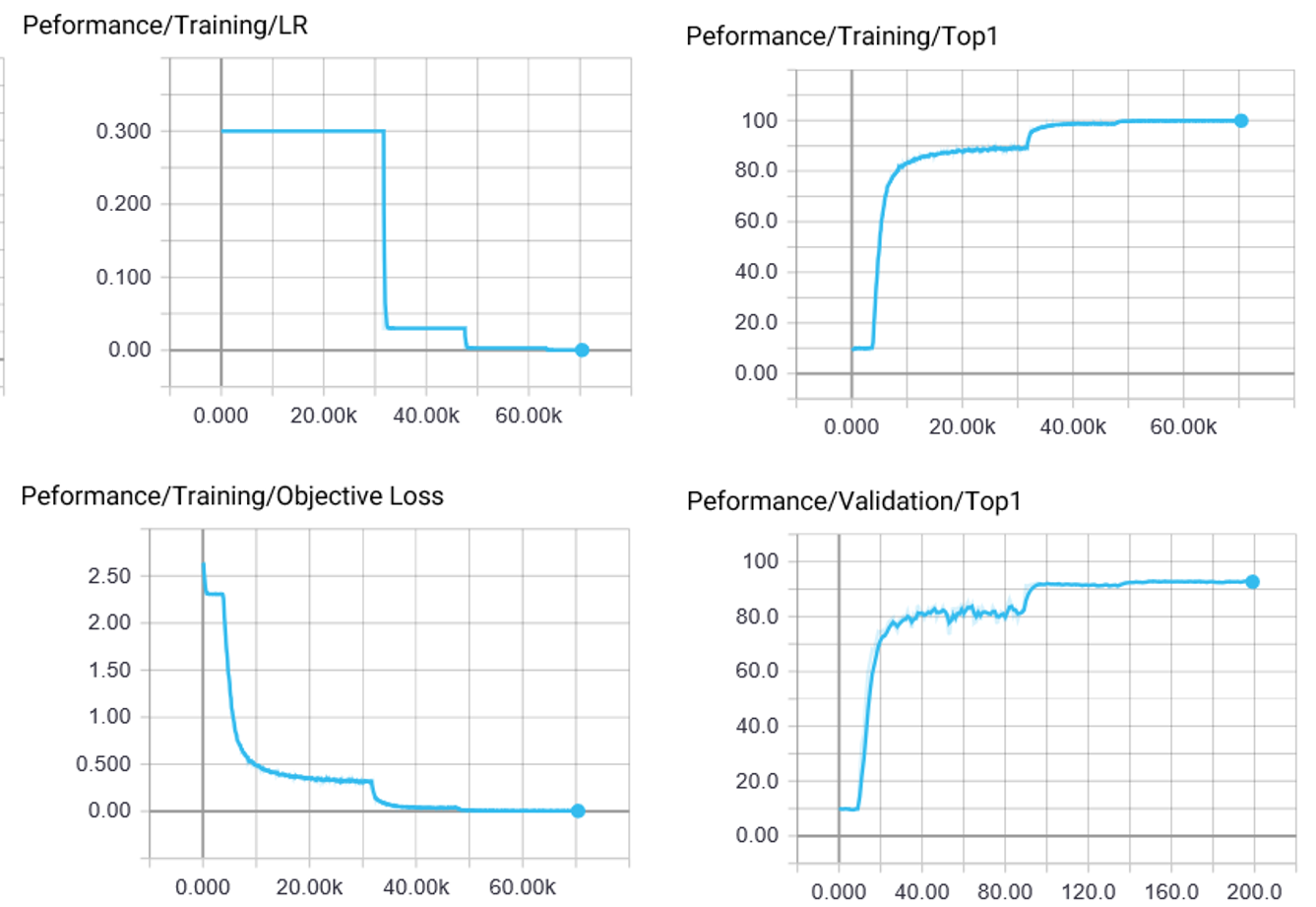
*instance\_name: training\_lr*

*starting\_epoch: 45*

*ending\_epoch: 200*

*frequency: 1*

The best top1 accuracy was 93.1% on epoch 195. Of course, the sparsity was 0.0%. Baseline model was saved in ‘./logs/2018.11.10-021811/best.pth.tar’



* Search Space

Define the search space was the sparsity of the weights of all the FC and CONV layers. min\_v and max\_v were the minimal and maximal value with uniform distribution of the sparsities. max\_v was always less than 1 and near to 1. If the sparsity constraint was 70%, min\_v would be 0.7 or higher. Otherwise, if there was no sparsity constraint, min\_v would be 0.

*def get\_space(min\_v, max\_v):*

*space = {}*

*for name, parameter in model.named\_parameters():*

*if 'conv' in name and 'weight' in name or 'fc' in name and 'weight' in name:*

*space[name] = hp.uniform(name, min\_v, max\_v)*

*return space*

* Score function

There were two Interdependent parameters, sparsity and accuracy. The optimization strategy was getting as high as possible sparsity and the accuracy is higher than 90%.

And the score function would be

*score = -((val\_accuracy/100.)\*\*2-0.9\*\*2 + alpha \* ((sparsity/100.)\*\*2-0.5\*\*2))*

alpha was the weight of these two parameters. In this experiment, alpha was setting to 1.

* Training procedure

There were four cases, randomization algorithm *tpe.suggest* and *rand.suggest* with and without sparsity constraint.

*#Tpe + constraint*

*main(0.7, 0.99, tpe.suggest)*

*#Random + constraint*

*main(0.7, 0.99, rand.suggest)*

*#Tpe + no constraint*

*main(0.1, 0.99, tpe.suggest)*

*#Random + no constraint*

*main(0.1, 0.99, rand.suggest)*

Each case ran 20 epoches and 30 rounds. The command was

*python LAB4-2.py --epochs=20 -r=30 --resume='./logs/2018.11.10-021811/best.pth.tar'*

* Training results

The case of tpe.suggest with constraint

===========================================

The best accuracy and sparsity is occurred by:

trials :27

score :-0.5759

train acc :96.13%

val acc :94.54%

test acc :93.92%

sparsity :86.14%

module.conv1.weight : 0.7895

module.fc.weight : 0.7639

module.layer1.0.conv1.weight : 0.9797

module.layer1.0.conv2.weight : 0.8745

module.layer1.1.conv1.weight : 0.9539

module.layer1.1.conv2.weight : 0.8849

module.layer1.2.conv1.weight : 0.9800

module.layer1.2.conv2.weight : 0.8898

module.layer1.3.conv1.weight : 0.8104

module.layer1.3.conv2.weight : 0.9857

module.layer1.4.conv1.weight : 0.8776

module.layer1.4.conv2.weight : 0.9426

module.layer1.5.conv1.weight : 0.8687

module.layer1.5.conv2.weight : 0.9778

module.layer1.6.conv1.weight : 0.7361

module.layer1.6.conv2.weight : 0.7532

module.layer1.7.conv1.weight : 0.8093

module.layer1.7.conv2.weight : 0.7250

module.layer1.8.conv1.weight : 0.7295

module.layer1.8.conv2.weight : 0.7157

module.layer2.0.conv1.weight : 0.9154

module.layer2.0.conv2.weight : 0.8121

module.layer2.1.conv1.weight : 0.7593

module.layer2.1.conv2.weight : 0.7991

module.layer2.2.conv1.weight : 0.9250

module.layer2.2.conv2.weight : 0.8930

module.layer2.3.conv1.weight : 0.9544

module.layer2.3.conv2.weight : 0.7855

module.layer2.4.conv1.weight : 0.9438

module.layer2.4.conv2.weight : 0.7055

module.layer2.5.conv1.weight : 0.7925

module.layer2.5.conv2.weight : 0.7860

module.layer2.6.conv1.weight : 0.8248

module.layer2.6.conv2.weight : 0.7994

module.layer2.7.conv1.weight : 0.9143

module.layer2.7.conv2.weight : 0.7744

module.layer2.8.conv1.weight : 0.9603

module.layer2.8.conv2.weight : 0.7458

module.layer3.0.conv1.weight : 0.8060

module.layer3.0.conv2.weight : 0.8725

module.layer3.1.conv1.weight : 0.8552

module.layer3.1.conv2.weight : 0.8656

module.layer3.2.conv1.weight : 0.7762

module.layer3.2.conv2.weight : 0.9084

module.layer3.3.conv1.weight : 0.8798

module.layer3.3.conv2.weight : 0.8519

module.layer3.4.conv1.weight : 0.8062

module.layer3.4.conv2.weight : 0.8479

module.layer3.5.conv1.weight : 0.9816

module.layer3.5.conv2.weight : 0.8229

module.layer3.6.conv1.weight : 0.7111

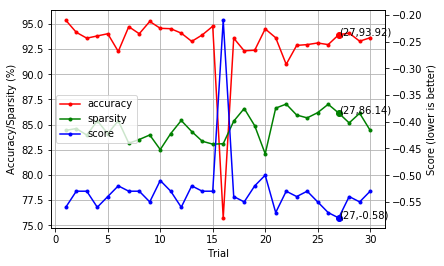
module.layer3.6.conv2.weight : 0.8610

module.layer3.7.conv1.weight : 0.9690

module.layer3.7.conv2.weight : 0.9896

module.layer3.8.conv1.weight : 0.9153

module.layer3.8.conv2.weight : 0.9325



The case of rand.suggest with constraint

===========================================

The best accuracy and sparsity is occurred by:

trials :18

score :-0.5835

train acc :95.20%

val acc :92.54%

test acc :92.28%

sparsity :88.72%

module.conv1.weight : 0.7776

module.fc.weight : 0.8970

module.layer1.0.conv1.weight : 0.8344

module.layer1.0.conv2.weight : 0.9175

module.layer1.1.conv1.weight : 0.8889

module.layer1.1.conv2.weight : 0.7183

module.layer1.2.conv1.weight : 0.9101

module.layer1.2.conv2.weight : 0.9079

module.layer1.3.conv1.weight : 0.9879

module.layer1.3.conv2.weight : 0.7057

module.layer1.4.conv1.weight : 0.8514

module.layer1.4.conv2.weight : 0.8985

module.layer1.5.conv1.weight : 0.7277

module.layer1.5.conv2.weight : 0.8572

module.layer1.6.conv1.weight : 0.9812

module.layer1.6.conv2.weight : 0.9329

module.layer1.7.conv1.weight : 0.8755

module.layer1.7.conv2.weight : 0.9558

module.layer1.8.conv1.weight : 0.9781

module.layer1.8.conv2.weight : 0.9408

module.layer2.0.conv1.weight : 0.7302

module.layer2.0.conv2.weight : 0.7648

module.layer2.1.conv1.weight : 0.9433

module.layer2.1.conv2.weight : 0.9273

module.layer2.2.conv1.weight : 0.9013

module.layer2.2.conv2.weight : 0.9510

module.layer2.3.conv1.weight : 0.9154

module.layer2.3.conv2.weight : 0.9534

module.layer2.4.conv1.weight : 0.8810

module.layer2.4.conv2.weight : 0.8259

module.layer2.5.conv1.weight : 0.7444

module.layer2.5.conv2.weight : 0.9813

module.layer2.6.conv1.weight : 0.8816

module.layer2.6.conv2.weight : 0.9252

module.layer2.7.conv1.weight : 0.8375

module.layer2.7.conv2.weight : 0.7573

module.layer2.8.conv1.weight : 0.8446

module.layer2.8.conv2.weight : 0.8013

module.layer3.0.conv1.weight : 0.7369

module.layer3.0.conv2.weight : 0.9233

module.layer3.1.conv1.weight : 0.9066

module.layer3.1.conv2.weight : 0.9610

module.layer3.2.conv1.weight : 0.8966

module.layer3.2.conv2.weight : 0.7625

module.layer3.3.conv1.weight : 0.9080

module.layer3.3.conv2.weight : 0.9378

module.layer3.4.conv1.weight : 0.8813

module.layer3.4.conv2.weight : 0.9783

module.layer3.5.conv1.weight : 0.8324

module.layer3.5.conv2.weight : 0.9604

module.layer3.6.conv1.weight : 0.9062

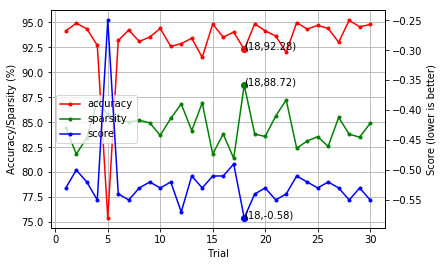
module.layer3.6.conv2.weight : 0.9734

module.layer3.7.conv1.weight : 0.8846

module.layer3.7.conv2.weight : 0.8520

module.layer3.8.conv1.weight : 0.9358

module.layer3.8.conv2.weight : 0.8090



The case of tpe.suggest without constraint

===========================================

The best accuracy and sparsity is occurred by:

trials :20

score :-0.2657

train acc :97.60%

val acc :95.78%

test acc :95.86%

sparsity :63.90%

module.conv1.weight : 0.9370

module.fc.weight : 0.1123

module.layer1.0.conv1.weight : 0.4538

module.layer1.0.conv2.weight : 0.7014

module.layer1.1.conv1.weight : 0.4281

module.layer1.1.conv2.weight : 0.5650

module.layer1.2.conv1.weight : 0.3714

module.layer1.2.conv2.weight : 0.1742

module.layer1.3.conv1.weight : 0.6974

module.layer1.3.conv2.weight : 0.7587

module.layer1.4.conv1.weight : 0.1171

module.layer1.4.conv2.weight : 0.5168

module.layer1.5.conv1.weight : 0.1576

module.layer1.5.conv2.weight : 0.1587

module.layer1.6.conv1.weight : 0.8939

module.layer1.6.conv2.weight : 0.5503

module.layer1.7.conv1.weight : 0.8929

module.layer1.7.conv2.weight : 0.4923

module.layer1.8.conv1.weight : 0.7034

module.layer1.8.conv2.weight : 0.2777

module.layer2.0.conv1.weight : 0.5722

module.layer2.0.conv2.weight : 0.4510

module.layer2.1.conv1.weight : 0.8902

module.layer2.1.conv2.weight : 0.9800

module.layer2.2.conv1.weight : 0.7725

module.layer2.2.conv2.weight : 0.3761

module.layer2.3.conv1.weight : 0.2478

module.layer2.3.conv2.weight : 0.9426

module.layer2.4.conv1.weight : 0.8435

module.layer2.4.conv2.weight : 0.9246

module.layer2.5.conv1.weight : 0.9034

module.layer2.5.conv2.weight : 0.3818

module.layer2.6.conv1.weight : 0.6839

module.layer2.6.conv2.weight : 0.8532

module.layer2.7.conv1.weight : 0.7384

module.layer2.7.conv2.weight : 0.5213

module.layer2.8.conv1.weight : 0.4331

module.layer2.8.conv2.weight : 0.1610

module.layer3.0.conv1.weight : 0.9362

module.layer3.0.conv2.weight : 0.2133

module.layer3.1.conv1.weight : 0.4460

module.layer3.1.conv2.weight : 0.8533

module.layer3.2.conv1.weight : 0.9893

module.layer3.2.conv2.weight : 0.3674

module.layer3.3.conv1.weight : 0.9032

module.layer3.3.conv2.weight : 0.7727

module.layer3.4.conv1.weight : 0.4253

module.layer3.4.conv2.weight : 0.4637

module.layer3.5.conv1.weight : 0.5036

module.layer3.5.conv2.weight : 0.7766

module.layer3.6.conv1.weight : 0.7453

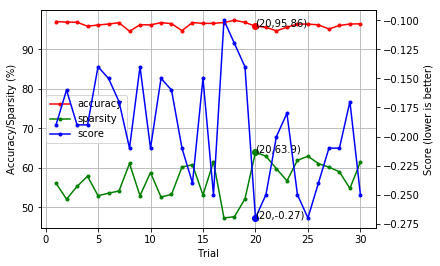
module.layer3.6.conv2.weight : 0.9881

module.layer3.7.conv1.weight : 0.2252

module.layer3.7.conv2.weight : 0.8423

module.layer3.8.conv1.weight : 0.8976

module.layer3.8.conv2.weight : 0.4628



The case of rand.suggest without constraint

===========================================

The best accuracy and sparsity is occurred by:

trials :10

score :-0.2465

train acc :98.17%

val acc :95.80%

test acc :96.26%

sparsity :62.35%

module.conv1.weight : 0.1293

module.fc.weight : 0.2342

module.layer1.0.conv1.weight : 0.1409

module.layer1.0.conv2.weight : 0.3423

module.layer1.1.conv1.weight : 0.3039

module.layer1.1.conv2.weight : 0.7220

module.layer1.2.conv1.weight : 0.5978

module.layer1.2.conv2.weight : 0.4936

module.layer1.3.conv1.weight : 0.6383

module.layer1.3.conv2.weight : 0.9051

module.layer1.4.conv1.weight : 0.4139

module.layer1.4.conv2.weight : 0.5391

module.layer1.5.conv1.weight : 0.1048

module.layer1.5.conv2.weight : 0.7531

module.layer1.6.conv1.weight : 0.1417

module.layer1.6.conv2.weight : 0.4241

module.layer1.7.conv1.weight : 0.9791

module.layer1.7.conv2.weight : 0.5851

module.layer1.8.conv1.weight : 0.3300

module.layer1.8.conv2.weight : 0.1471

module.layer2.0.conv1.weight : 0.6467

module.layer2.0.conv2.weight : 0.9783

module.layer2.1.conv1.weight : 0.5493

module.layer2.1.conv2.weight : 0.8116

module.layer2.2.conv1.weight : 0.5126

module.layer2.2.conv2.weight : 0.4778

module.layer2.3.conv1.weight : 0.1325

module.layer2.3.conv2.weight : 0.7142

module.layer2.4.conv1.weight : 0.7432

module.layer2.4.conv2.weight : 0.8882

module.layer2.5.conv1.weight : 0.3783

module.layer2.5.conv2.weight : 0.2084

module.layer2.6.conv1.weight : 0.2061

module.layer2.6.conv2.weight : 0.9044

module.layer2.7.conv1.weight : 0.2546

module.layer2.7.conv2.weight : 0.2484

module.layer2.8.conv1.weight : 0.8758

module.layer2.8.conv2.weight : 0.3500

module.layer3.0.conv1.weight : 0.7044

module.layer3.0.conv2.weight : 0.4056

module.layer3.1.conv1.weight : 0.8055

module.layer3.1.conv2.weight : 0.7078

module.layer3.2.conv1.weight : 0.3490

module.layer3.2.conv2.weight : 0.6563

module.layer3.3.conv1.weight : 0.2188

module.layer3.3.conv2.weight : 0.9267

module.layer3.4.conv1.weight : 0.8811

module.layer3.4.conv2.weight : 0.9674

module.layer3.5.conv1.weight : 0.2417

module.layer3.5.conv2.weight : 0.7765

module.layer3.6.conv1.weight : 0.1243

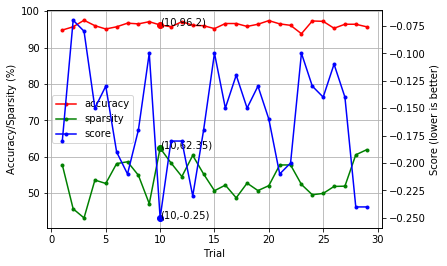
module.layer3.6.conv2.weight : 0.9185

module.layer3.7.conv1.weight : 0.3753

module.layer3.7.conv2.weight : 0.9708

module.layer3.8.conv1.weight : 0.8950

module.layer3.8.conv2.weight : 0.8998



* Q&A

1. Explain how you deal with constraints

* Search space
* Objective function

There were two ideas, one was focused on the weights of all CONV layers and tried to minimize the score below.

def get\_space():

space = {}

for name, parameter in model.named\_parameters():

if 'conv' in name and 'weight' in name:

space[name] = hp.uniform(name, 0.30, 0.99)

return space

score = (1-(val\_accuracy/100.)) + (alpha \* (1-sparsity/100.))

The other one searched the weights of both CONV and FC layers and used the second order score function.

def get\_space(min\_v, max\_v):

space = {}

for name, parameter in model.named\_parameters():

if 'conv' in name and 'weight' in name or 'fc' in name and 'weight' in name:

space[name] = hp.uniform(name, min\_v, max\_v)

return space

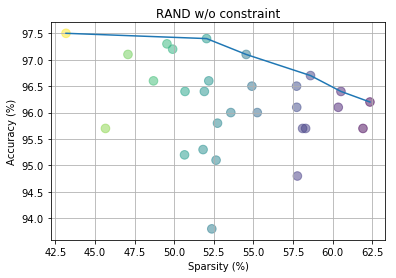
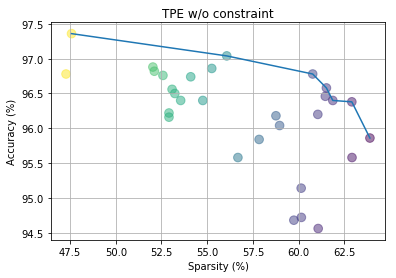
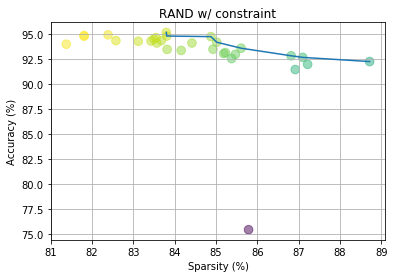
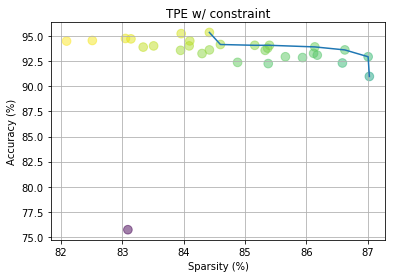
score = -((val\_accuracy/100.)\*\*2-0.9\*\*2 + alpha \* ((sparsity/100.)\*\*2-0.5\*\*2))

Search algorithm would be tpe.suggest and rand.suggest. Search duration depended on the target sparsity and accuracy.

2. Compare with different algorithms

* Draw and explain the Pareto curve

The Pareto curves for the second ideas were shown below. Generally speaking, the Pareto curves were bounded and balanced by sparsity and accuracy. Higher sparsity would get a lower accuracy.



3. Give a global constraint

* Can hyper-parameters optimization easily fit constraints?
* Can human-design space easily cover all possibility?

By the large, it was hard to fit the constraints by hyper-parameter optimization. It took very long time and many computation power to search an appropriate set of parameters. Human didn’t have the exactly knowledge about which parameters changed would get higher performance. In the end, it just used a smarter brute force method to find a better result.

In this experiment, using a baseline pretrained model was a good way to shorten the search time and got a not bad performance.

4. How do we avoid curse of dimensionality with the increase of the number of layers?

In general, more layers got higher performance but higher complexity. We could increase the sparsity to simplify the network structure. In this experiment results, sparsity could be 88% or higher and the accuracy 92.5%. The full set baseline accuracy was 93.1%.

We also could use quantization to quantize the weights from floating-point to 8-bit or lower. Finally, we could train a smaller network, for example, 5 layers, to learn the behavior of the original model by Knowledge Distillation.

# 