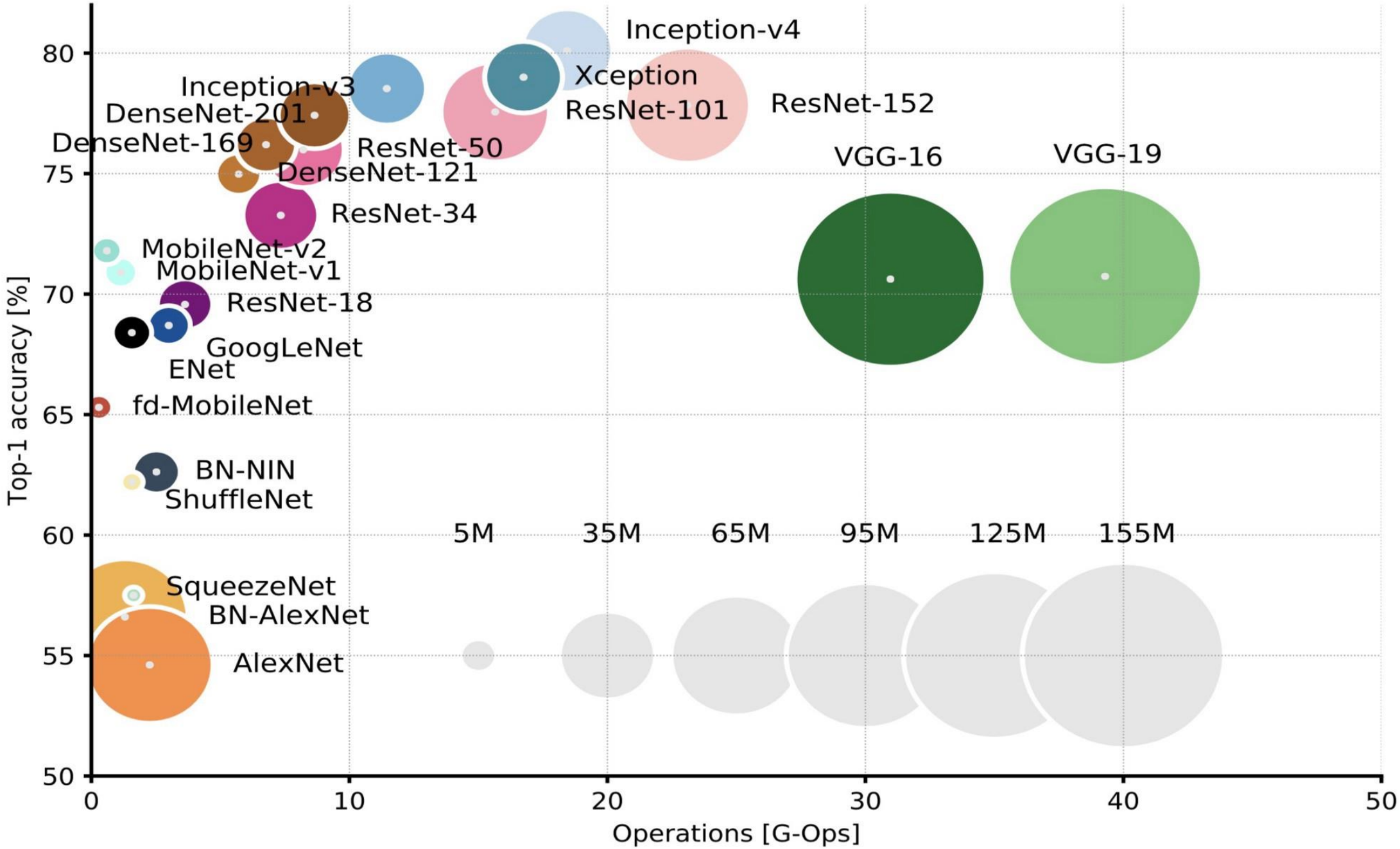
ResNet and Distributed Training

Xiaobo

Agenda

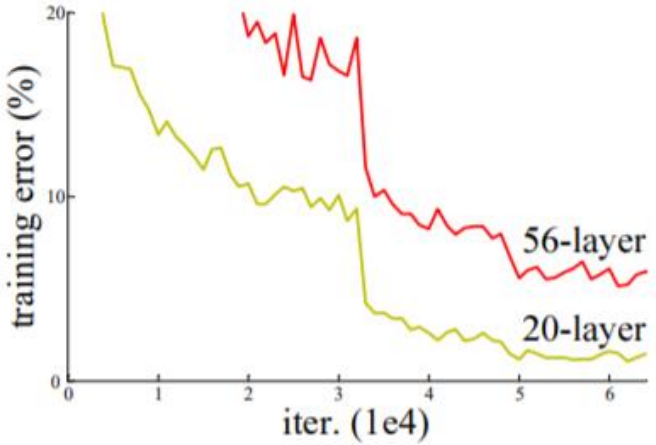
- ResNet
  - Motivation
  - Basic Block
  - Bottleneck Block
- Data loader for TinyImageNet

- Distributed Training on Blue Waters
  - Bash script for multiple nodes
  - Distributed Training Initialization
  - P2P and collective communication
  - Synchronous/Asynchronous SGD


Inception-v4
Xception
Inception-v3
ResNet-101
ResNet-152
DenseNet-201
DenseNet-169
ResNet-50
VGG-16
VGG-19
DenseNet-121
ResNet-34
MobileNet-v2
MobileNet-v1
ResNet-18
GoogLeNet
ENet
fd-MobileNet
BN-NIN
ShuffleNet
5M
35M
65M
95M
125M
155M
SqueezeNet
BN-AlexNet
AlexNet
Top-1 accuracy [%]
80
75
70
65
60
55
50
0
10
20
30
40
50
Operations [G-Ops]

ResNet

- Deep networks more challenging to train.
- Vanishing gradient problem.
- Introduce skip/shortcut connections to facilitate the “flow of information” from lower layers to higher layers.



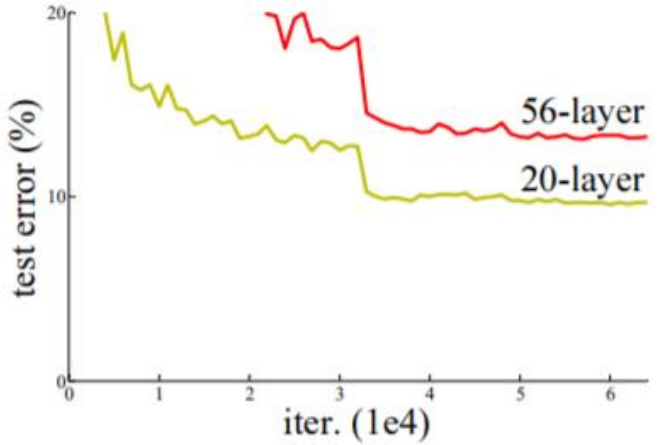


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer “plain” networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

Source: He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

ResNet Result

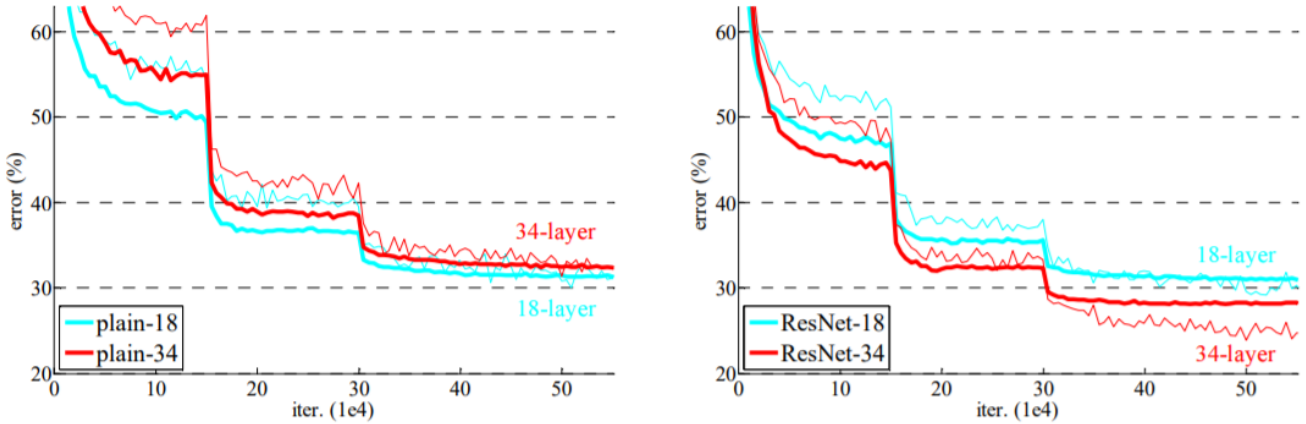


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	**25.03**

Table 2. Top-1 error (%, 10-crop testing) on ImageNet validation. Here the ResNets have no extra parameter compared to their plain counterparts. Fig. 4 shows the training procedures.

method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43†
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

Table 4. Error rates (%) of **single-model** results on the ImageNet validation set (except † reported on the test set).

Source: He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

Notation

- x: Single input with dimension $W \times H \times C_1$
- $[K, K]C_2$: Kernel with size D and C_2 channels, $\mathrm{K} \times K \times C_2$
- F: Residual function

Basic Block

```
class BasicBlock(nn.Module):
    expansion = 1
    def __init__(self, inplanes, planes, stride=1, downsample=None):
        super(BasicBlock, self).__init__()
        self.conv1 = conv3x3(inplanes, planes, stride)
        self.bn1 = nn.BatchNorm2d(planes)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = conv3x3(planes, planes)
        self.bn2 = nn.BatchNorm2d(planes)
        self.downsample = downsample
        self.stride = stride

    def forward(self, x):
        residual = x
        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)
        out = self.conv2(out)
        out = self.bn2(out)
        if self.downsample is not None:
            residual = self.downsample(x)
        out += residual
        out = self.relu(out)
        return out
```

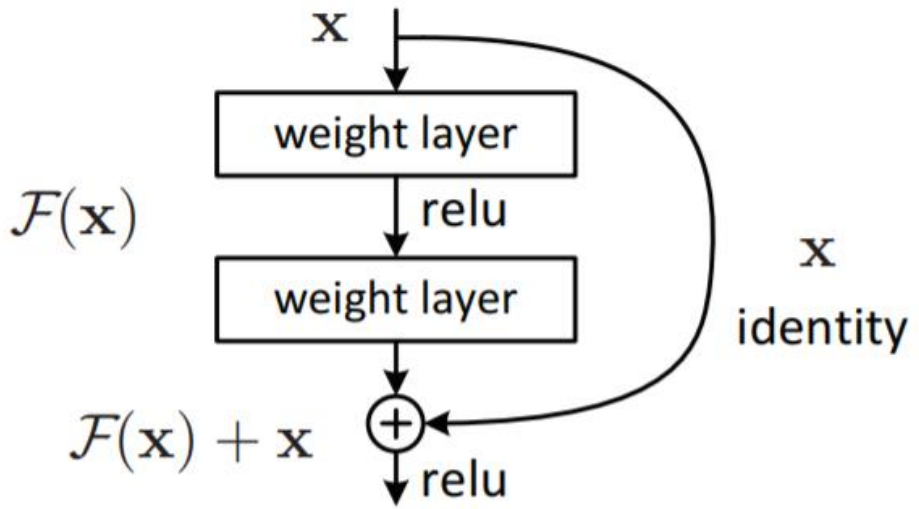
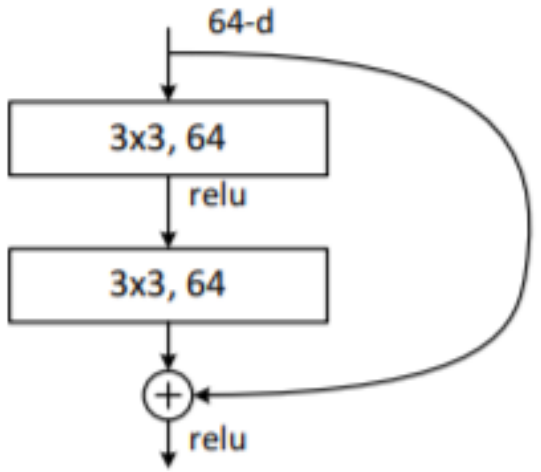


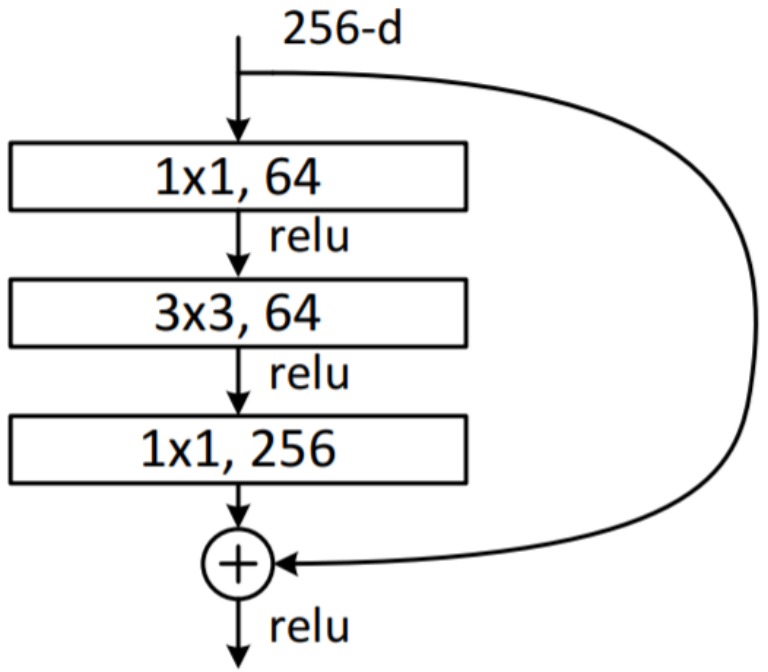
Figure 2. Residual learning: a building block.



$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}$$

Bottleneck Block

- The three layers are 1×1, 3×3, and 1×1 convolutions, where the 1×1 layers are responsible for reducing and then increasing (restoring) dimensions, leaving the 3×3 layer a bottleneck with smaller input/output dimensions.



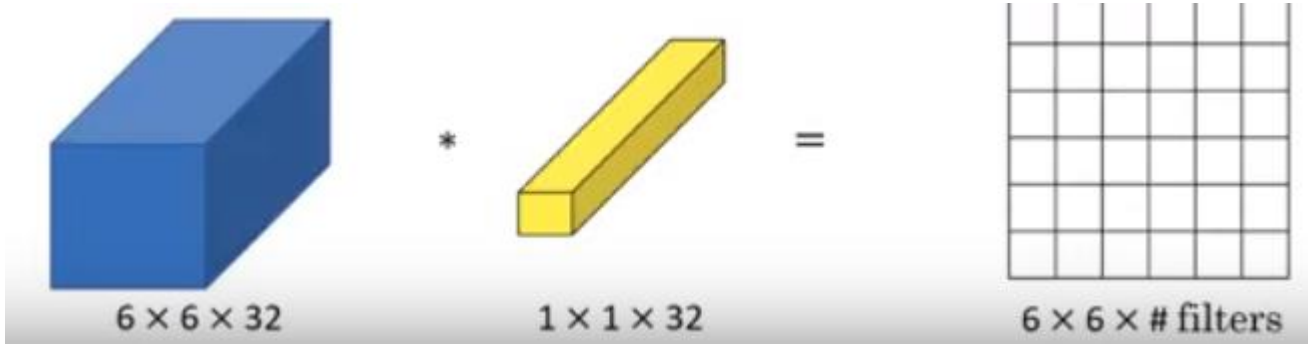
Dimension Mismatch

- Spatial mismatch
  - Use linear projection(not essential for addressing the degradation problem.)

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + W_s\mathbf{x}$$

  - Some operation with stride

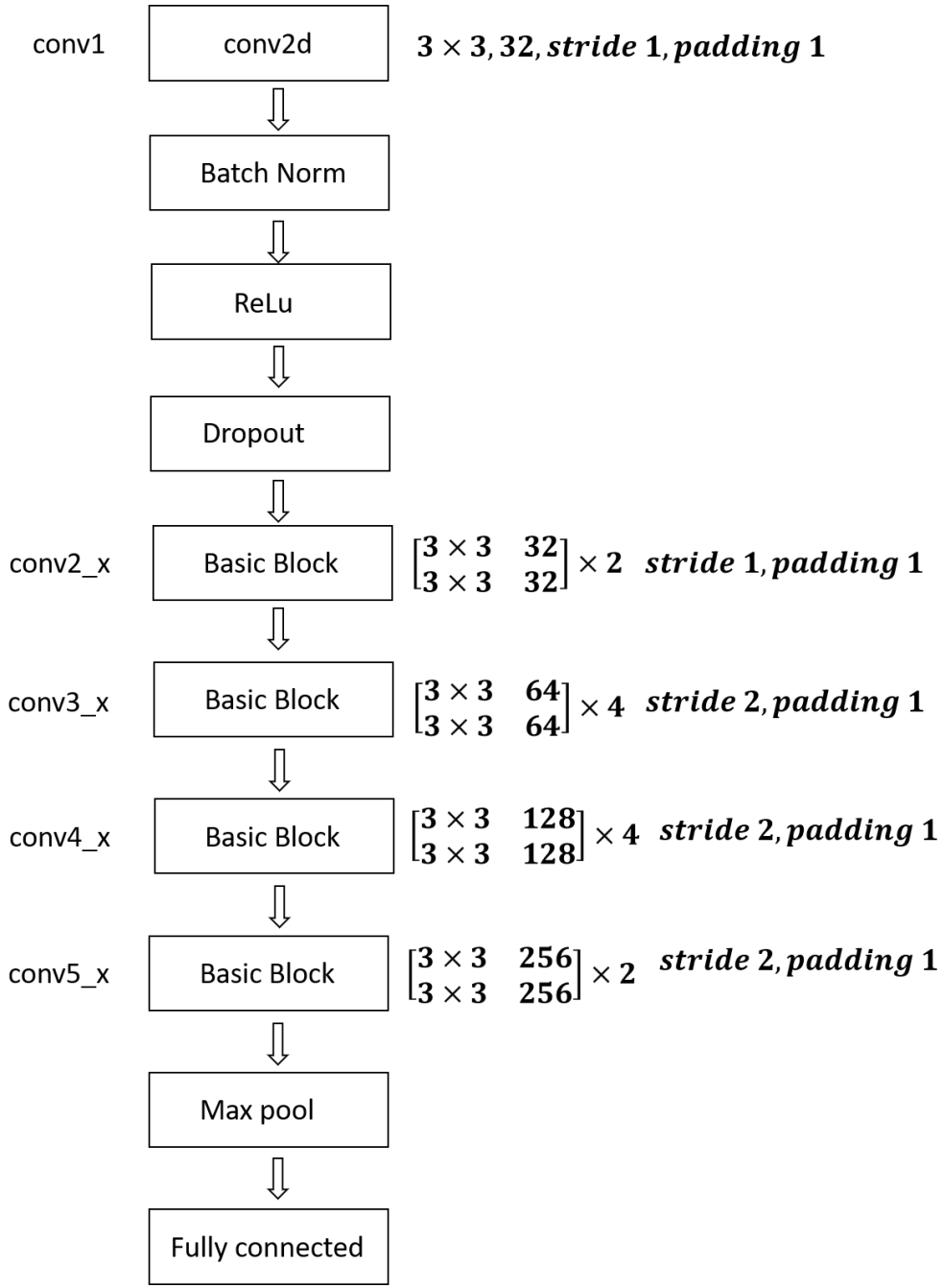
- Channel mismatch
  - 1x1 Convolution
  - Input: $W \times H \times C_1$
  - Kernel: $1\times 1 \times C_2$
  - Output: $W \times H \times C_2$




```
nn.Conv2d(channel_in,channel_out ,kernel_size=1, stride=1)
```

HW4 Structure

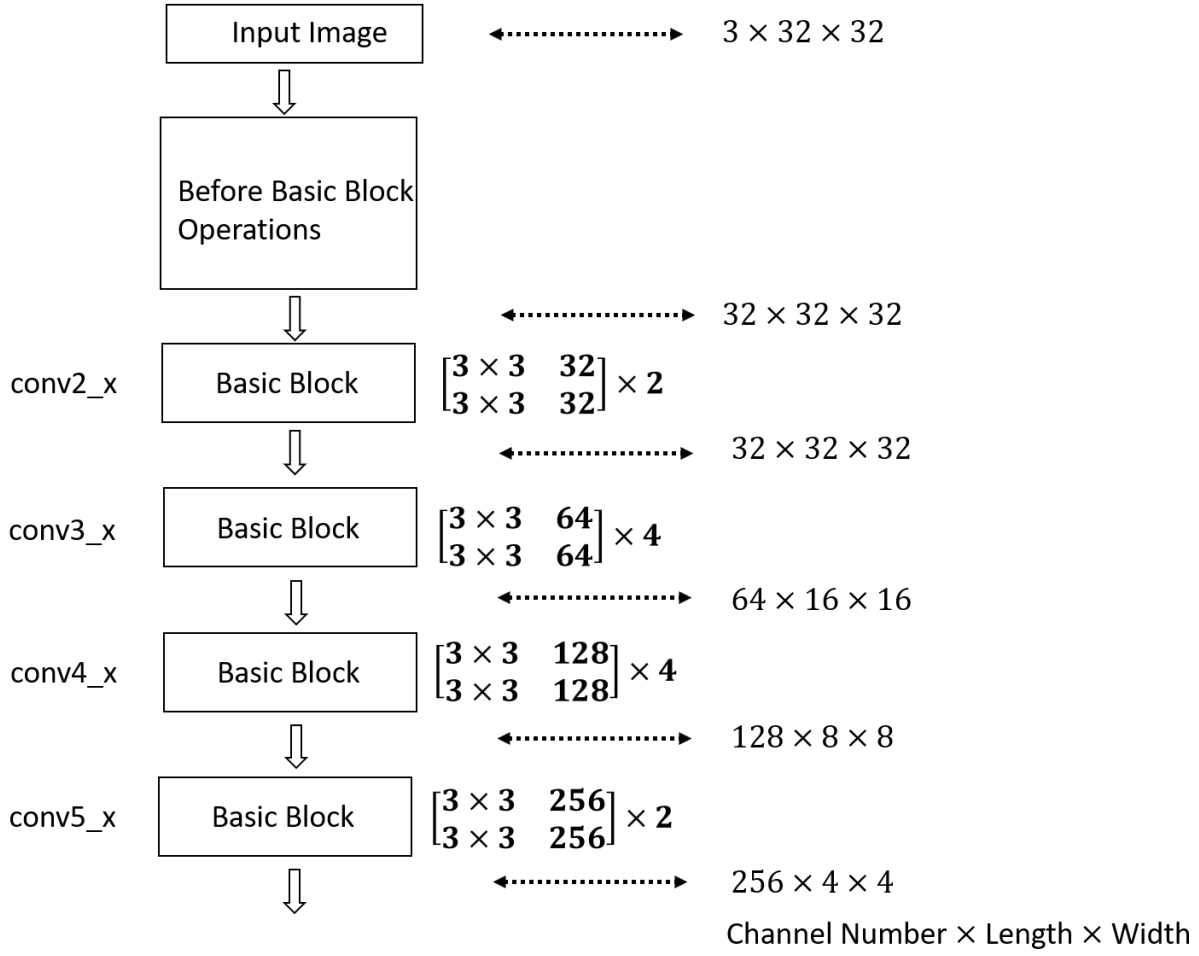
- Your implementation has to match the structure on the right.
- Note:
  - $3 \times 3 \;\; 32$ means the kernel size of the convolution is 3×3 with 32 output channels
  - $\begin{bmatrix} 3 \times 3 & 32 \\ 3 \times 3 & 32 \end{bmatrix}$ means a stack of two convolutions
  - $\begin{bmatrix} 3 \times 3 & 32 \\ 3 \times 3 & 32 \end{bmatrix} \times 2$ means 2 of the stacks
  - stride and padding are the corresponding arguments for convolution function.



Dimension of Resnet on CIFAR100

- For conv3_x, conv4_x, conv5_x, the stride is 2 just for the first module in each stacks of basic blocks, which corresponding to "image" size reduced by 2 in each stack of basic blocks.

- https://github.com/pytorch/vision/blob/master/torchvision/models/resnet.py



CIFAR and ImagNet datasets

- CIFAR10
  - The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.
- CIFAR100
  - it has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class. The 100 classes in the CIFAR-100 are grouped into 20 superclasses.
- ImageNet
  - ImageNet has 1000 image categories with more than 14 million 256x256 color images.
- TinyImageNet
  - TinyImagenet has 200 classes. Each class has 500 training images, 50 validation images, and 50 test images. The images are down-sampled to 64x64 pixels.

Dataset Location on Blue Waters

- CIFAR100 and TinyImageNet datasets are located in

  /projects/training/bayw/HW4_Dataset

- For CIFAR100, copy cifar-100-python to your own directory.

- For TinyImageNet, transfer tiny-imagenet-200.zip to your own directory and unzip it. Unzipping the file may take half an hour. Once you unzip it, it contains three folders train, val and test. We will use train and val for HW4.

Generate Data Loader with ImageFolder

- torchvision.datasets.ImageFolder

A generic data loader where the images are arranged in this way:

```
root/dog/xxx.png
root/dog/xxy.png
root/dog/xxz.png

root/cat/123.png
root/cat/nsdf3.png
root/cat/asd932_.png
```

Pre-processing on TinyImageNet Validation data

- Originally, all the validation images are in the same directory.
- ***create_val_folder()*** is to separate validation images into separate sub-folders

```
def create_val_folder(val_dir):
    """
    This method is responsible for separating validation
    images into separate sub folders
    """
    # path where validation data is present now
    path = os.path.join(val_dir, 'images')
    # file where image2class mapping is present
    filename = os.path.join(val_dir, 'val_annotations.txt')
    fp = open(filename, "r")  # open file in read mode
    data = fp.readlines()  # read line by line

    '''
    Create a dictionary with image names as key and
    corresponding classes as values
    '''
    val_img_dict = {}
    for line in data:
        words = line.split("\t")
        val_img_dict[words[0]] = words[1]
    fp.close()
    # Create folder if not present, and move image into proper folder
    for img, folder in val_img_dict.items():
        newpath = (os.path.join(path, folder))
        if not os.path.exists(newpath):  # check if folder exists
            os.makedirs(newpath)
        # Check if image exists in default directory
        if os.path.exists(os.path.join(path, img)):
            os.rename(os.path.join(path, img), os.path.join(newpath, img))
    return
```

Data Loader for TinyImageNet

- You can check the index for each class by print ***train_dataset.class_to_idx***

```
train_dir = '/u/training/YOUR_BW_ID/scratch/tiny-imagenet-200/train'
train_dataset = datasets.ImageFolder(train_dir,
         transform=transform_train)
#print(train_dataset.class_to_idx)
train_loader = torch.utils.data.DataLoader(train_dataset,
        batch_size=batch_size, shuffle=True, num_workers=8)

val_dir = '/u/training/instr030/scratch/tiny-imagenet-200/val/'

if 'val_' in os.listdir(val_dir+'images/')[0]:
    create_val_folder(val_dir)
    val_dir = val_dir+'images/'
else:
    val_dir = val_dir+'images/'

val_dataset = datasets.ImageFolder(val_dir,
        transform=transforms.ToTensor())
#print(val_dataset.class_to_idx)
val_loader = torch.utils.data.DataLoader(val_dataset,
        batch_size=batch_size, shuffle=False, num_workers=8)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
for images, labels in train_loader:
    images = images.to(device)
    labels = labels.to(device)

for images, labels in val_loader:
    images = images.to(device)
    labels = labels.to(device)
```

Distributed Training On Blue Waters

- Distributed Backend
  - GLOO
  - MPI
  - NCCL

- Blue Waters uses MPI
  - MPI supports CPU communication
  - CUDA tensor has to be sent to CPU before communication

Backend	`gloo`		`mpi`		`nccl`	
Device	CPU	GPU	CPU	GPU	CPU	GPU
send	✓	✘	✓	?	✘	✘
recv	✓	✘	✓	?	✘	✘
broadcast	✓	✓	✓	?	✘	✓
all_reduce	✓	✓	✓	?	✘	✓
reduce	✓	✘	✓	?	✘	✓
all_gather	✓	✘	✓	?	✘	✓
gather	✓	✘	✓	?	✘	✘
scatter	✓	✘	✓	?	✘	✘
barrier	✓	✘	✓	?	✘	✓

Bash Script

- In order to run 2 processes on 2 nodes, you have to put
  - ***nodes = 02:ppn***
  - ***aprun –n 2 –N 1 python main.py***
- In order to run Python MPI,
  - ***module load bwpy***
  - ***module load bwpy-mpi***
- bwpy contains pytorch 0.3.0 instead of 0.4.0.

```
#!/bin/bash
#PBS -l nodes=02:ppn=16:xk
#PBS -l walltime=06:00:00
#PBS -N sync_sgd_cifar100
#PBS -e $PBS_JOBID.err
#PBS -o $PBS_JOBID.out
#PBS -m bea
#PBS -M YOUR_EMAIL
cd #YOUR CODE DIRECTORY
. /opt/modules/default/init/bash # NEEDED to add module commands to shell
module load bwpy
module load bwpy-mpi
aprun -n 2 -N 1 python sync_sgd_cifar100.py
```

Initialization for Pytorch distributed training

- There are 3 ways to do initialization on distributed training in Pytorch.
  - TCP initialization
  - Shared file-system initialization
  - Environment variable initialization

- `MASTER_PORT` - required; has to be a free port on machine with rank 0
- `MASTER_ADDR` - required (except for rank 0); address of rank 0 node
- `WORLD_SIZE` - required; can be set either here, or in a call to init function
- `RANK` - required; can be set either here, or in a call to init function

```
import torch.distributed as dist
import os
import subprocess
from mpi4py import MPI
cmd = "/sbin/ifconfig"
out, err = subprocess.Popen(cmd, shell=True, stdout=subprocess.PIPE,
    stderr=subprocess.PIPE).communicate()
ip = str(out).split("inet addr:")[1].split()[0]
name = MPI.Get_processor_name()
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
num_nodes = int(comm.Get_size())
ip = comm.gather(ip)
if rank != 0:
  ip = None
ip = comm.bcast(ip, root=0)
os.environ['MASTER_ADDR'] = ip[0]
os.environ['MASTER_PORT'] = '2222'
backend = 'mpi'
dist.init_process_group(backend, rank=rank, world_size=num_nodes)
dtype = torch.FloatTensor
```

Alternative solution

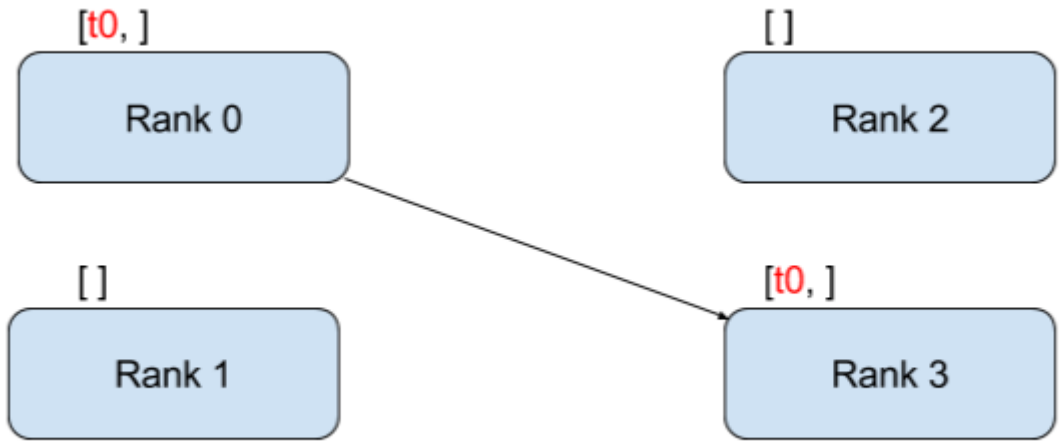
- Blue Waters provides an alternative solution which has a Python with Pytorch1.0.1 supports MPI as well.
- The sample pbs file is located in
  - /projects/training/bayw/docker_sample
- In order to run this Python, you need to do the following
  - Change the directory to your job directory in pbs file (.pbs file)
    - cd /u/training/trainXXX/YOUR_JOB_DIRECTORY
  - Change the file name to your own python file in bash file (.sh file)
    - python YOUR_PYTHON_FILE.py
- Either using this docker or old version of python(bwpy) is your option.

Communication between nodes

- Point-to-Point Communication
  - Synchronous
  - Asynchronous
- Collective Communication
  - Synchronous

Point-to-Point Communication

- A transfer of data from one process to another is called a point-to-point(P2P) communication.
- These are achieved through the ***send*** and ***recv*** functions or their immediate counter-parts, ***isend*** and ***irecv***.



Blocking vs Nonblocking Communications

- Blocking Communication:When a process calls send it specifies a destination to send to that destination. While the message is being sent, the sending process is blocked (i.e., suspended).
- Nonblocking Communication: If send is nonblocking, it returns control to the caller immediately, before the message is sent. The advantage of this scheme is that the sending process can continue computing in parallel with the message transmission, instead of having the CPU go idle.
- When using nonblocking we have to be careful about with our usage of the sent and received tensors. Since we do not know when the data will be communicated to the other process, we should not modify the sent tensor nor access the received tensor before req.wait() has completed.

Blocking vs Nonblocking Communications

Blocking P2P communication

```
"""Blocking point-to-point communication."""

def run(rank, size):
    tensor = torch.zeros(1)
    if rank == 0:
        tensor += 1
        # Send the tensor to process 1
        dist.send(tensor=tensor, dst=1)
    else:
        # Receive tensor from process 0
        dist.recv(tensor=tensor, src=0)
    print('Rank ', rank, ' has data ', tensor[0])
```

Nonblocking P2P communication

```
def run(rank, size):
    tensor = torch.zeros(1)
    req = None
    if rank == 0:
        tensor += 1
        # Send the tensor to process 1
        req = dist.isend(tensor=tensor, dst=1)
        print('Rank 0 started sending')
    else:
        # Receive tensor from process 0
        req = dist.irecv(tensor=tensor, src=0)
        print('Rank 1 started receiving')
    req.wait()
    print('Rank ', rank, ' has data ', tensor[0])
```

Collective Communication

- As opposed to point-to-point communication, collectives allow for communication patterns across all processes in a **group.**

- A group is a subset of all our processes.

- By default, collectives are executed on the all processes, also known as the **world**.

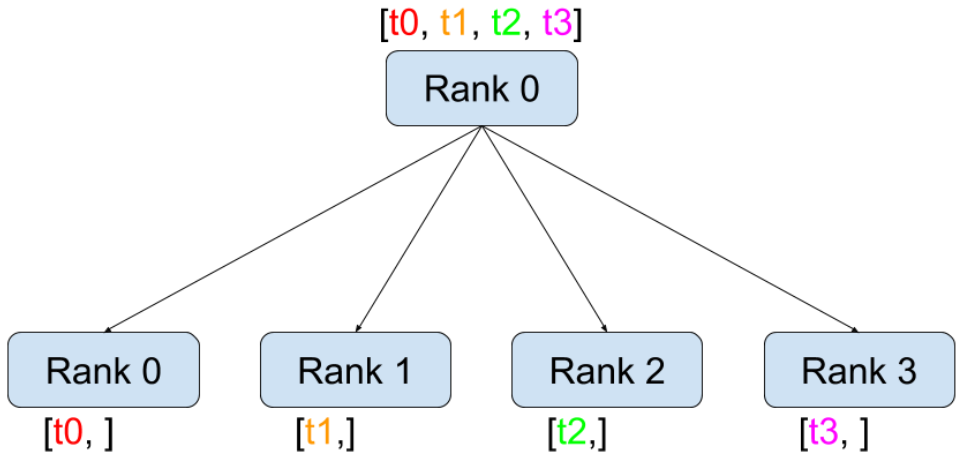
Collective Communication

- Scatter
- Gather
- Reduce
- All-Reduce
- Broadcast
- All-Gather

- One of the things to remember about collective communication is that it implies a *synchronization point* among processes. This means that all processes must reach a point in their code before they can all begin executing again.

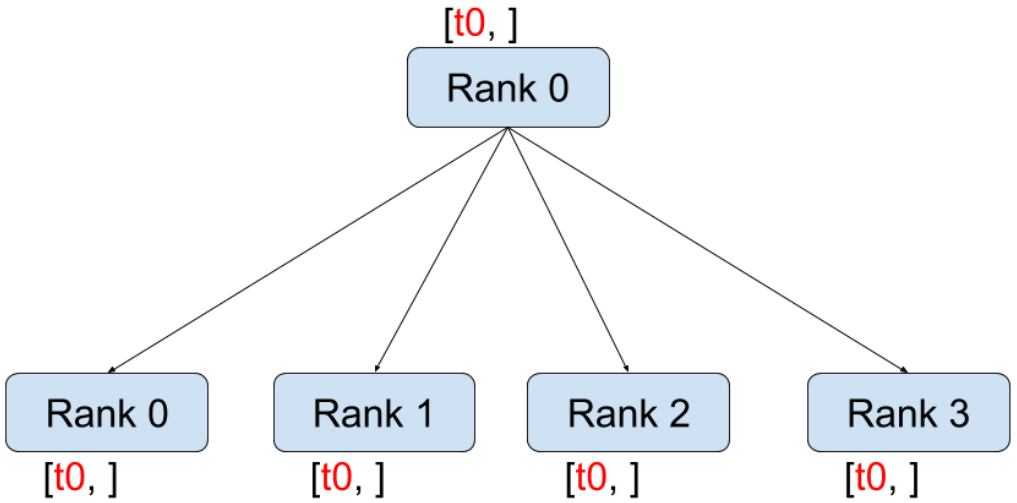
Scatter

- Scatter involves a designated root process sending data to all processes in a communicator.



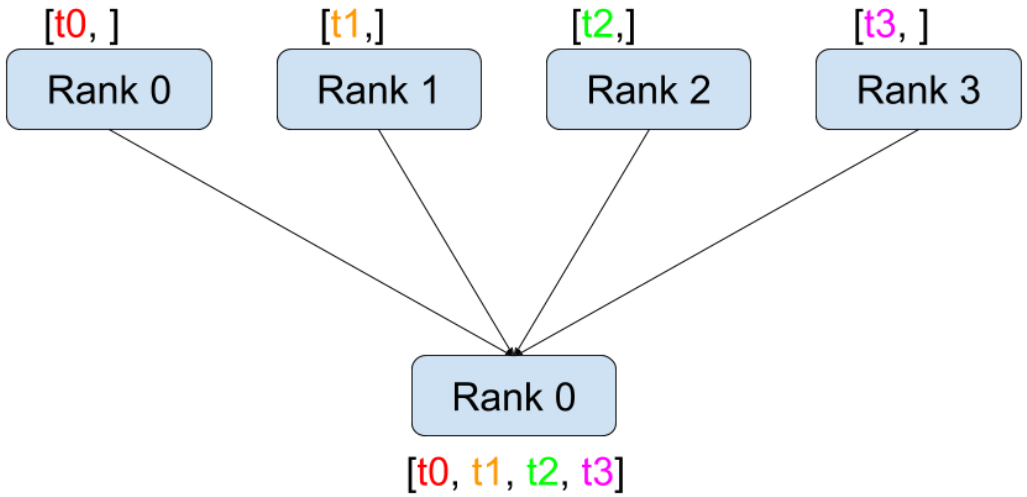
Broadcast

- During a broadcast, one process sends the same data to all processes in a communicator.



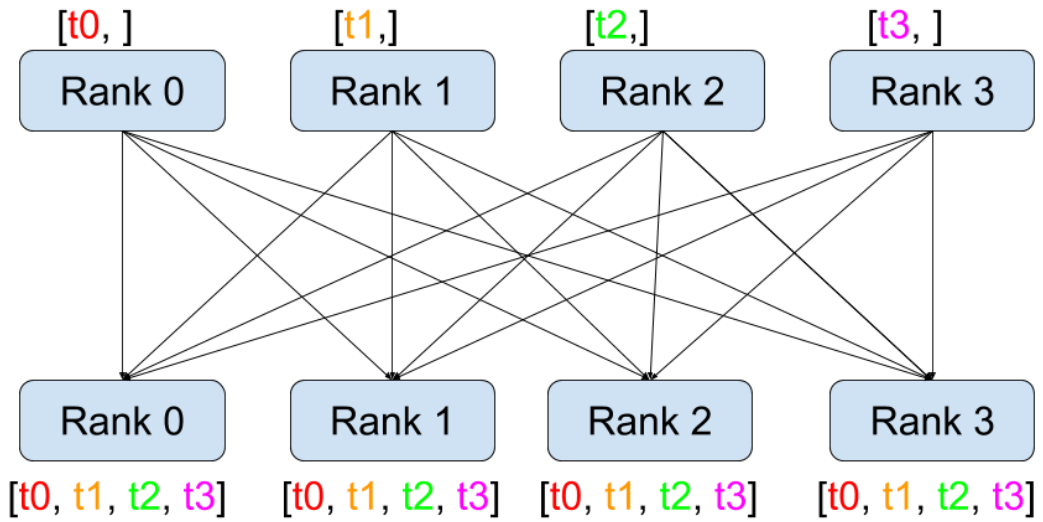
Gather

- Gather takes elements from many processes and gathers them to one single process



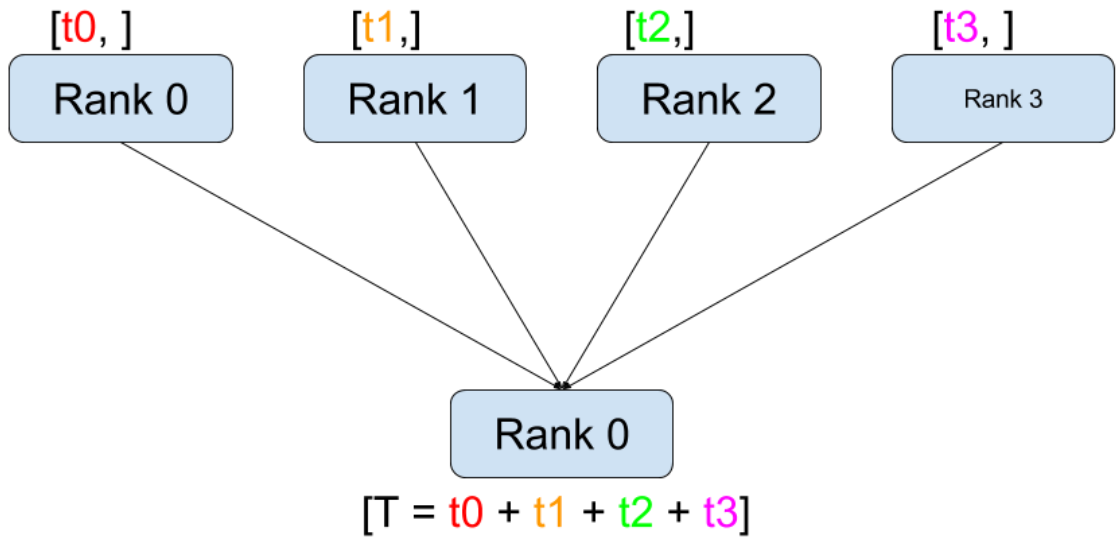
All-Gather

- Given a set of elements distributed across all processes, All-gather will gather all of the elements to all the processes.



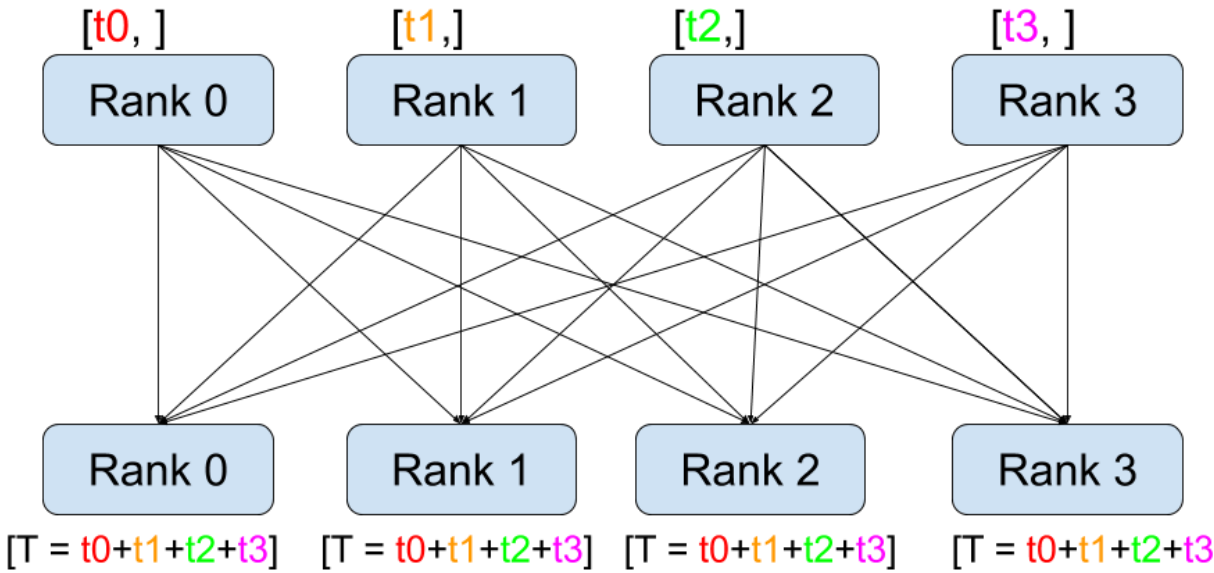
Reduce

- Similar to Gather, Reduce takes an array of input elements on each process and returns an array of output elements to the root process. The output elements contain the reduced result.



All-Reduce

- All-Reduce will reduce the values and distribute the results to all processes.



Synchronous SGD

Data: CIFAR100
Result: Classification Model on CIFAR100
Do Synchronous Initialization for All servers;
for *Server = 0,1,2,...K-1 (Do in Parallel)* **do**
  for *epoch = 0,1,2,...N-1* **do**
    for *each mini-batch data* **do**
      - Calculate gradient for each server
      - AllReduce to synchronous the gradient of all servers.
      - Update weight of the model.
    end
  end
end

Algorithm 1: Synchronous SGD

- Use ***all_reduce()*** from ***torch.distributed*** to transfer the tensors

```
for param in model.parameters():
    tensor0 = param.grad.data.cpu()
    dist.all_reduce(tensor0,op=dist.reduce_op.SUM)
    tensor0 /= float(num_nodes)
    param.grad.data = tensor0.cuda()
```

Asynchronous SGD

Data: CIFAR100
Result: Classification Model on CIFAR100
Do Synchronous Initialization for All servers;
Server 0 to be parameter server;
Server 1...K-1 to be worker servers;
for *Server = 1,2,...K-1 (Do in Parallel)* **do**
  for *epoch = 0,1,2,...N-1* **do**
    for *each mini-batch data* **do**
      - Calculate gradient for each server
      - Send the gradient to parameter server.
      - Get the updated parameter from parameter server and update worker's own parameter.

    end
  end
end
Note:

1. For parameter server, it should receive gradients from each worker server whenever they send the request.
2. Use ***dist.isend*** and ***dist.irecv*** to send and receive tensors.

Algorithm 2: Asynchronous SGD

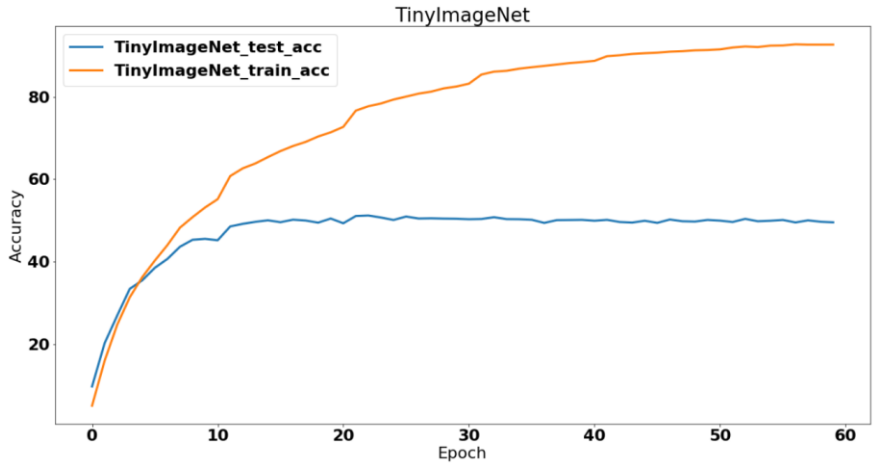
- Use ***isend()*** and ***irecv()*** from ***torch.distributed*** to transfer the tensors

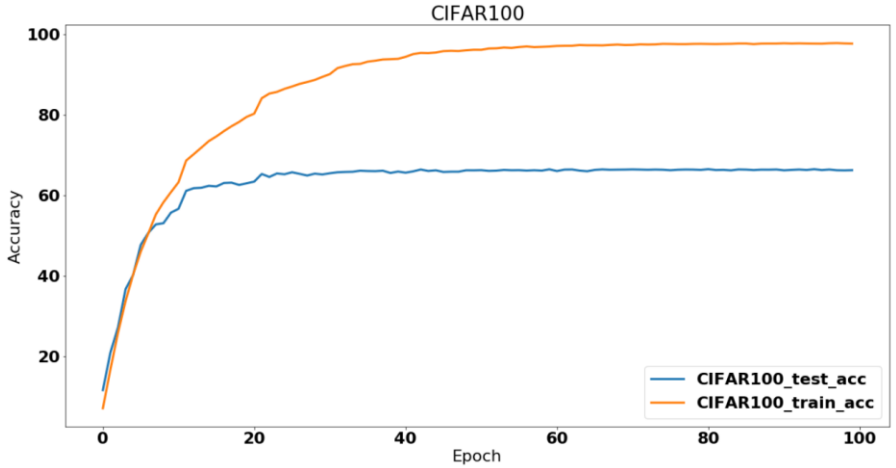
How to test your distributed training code

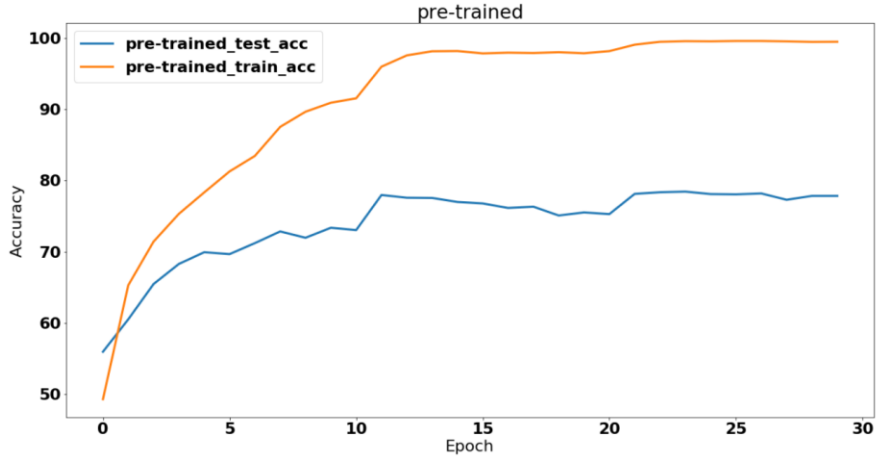
- Develop and test using two nodes with short wall time 10 mins
- Once code works, you can use 4 nodes with longer wall time.
- On Blue Waters, the queueing time is longer for multiple nodes job. Please start distributed training part as early as possible.

HW4 Tasks

- Write your own ResNet given by the structure we provide
  - Train on CIFAR100
  - Train on TinyImageNet
- Use pre-trained ResNet18
  - Train on CIFAR100
- Distributed Training on your own ResNet with
  - Synchronous SGD
  - *Asynchronous SGD (10 bonus points)







What to submit

- For each task you should submit an individual main file.

  1. `resnet_cifar100.py`
  2. `resnet_tinyimagenet.py`
  3. `pretrained_cifar100.py`
  4. `sync_sgd_cifar100.py`
  5. `async_sgd_cifar100.py`

  Note: You can have extra python files including some helper functions, but please make sure you have an individual main file for each tasks.

	Accuracy
ResNet CIFAR100	60%
ResNet TinyImageNet	50%
Pre-trained ResNet CIFAR100	70%
ResNet CIFAR100 with synchronous SGD	60%
*ResNet CIFAR100 with asynchronous SGD	60%

- For each task, you should report your achieved test accuracy and plot the learning curve. The x-axis of your plot is epoch numbers and y-axis of your plot is test accuracy.

- We provide partial code in the following directory, which basically covers the code mentioned above. It is not required to use these code, but you can use it as a reference.

```
/projects/training/bayw/HW4
```

Q&A

- Piazza Question:
  1. There is an accuracy difference between servers for synchronous SGD?
  2. …

- Potential Question:
  1. How to write distributed code?
  2. …