ImageNet in 18 minutes

for the masses

Motivation

- training was fast in Google

- no technical reason it can't be fast outside of Google

- many things are easier procedurally

    share your jupyter servers, run any code, collaborate with anyone

Stanford Dawn Competition

Stanford: 13 days on g3 instance (Oct)

Yuxin Wu: 21 hours on Pascal DGX-1 (Dec)

diux: 14 hours on Volta DGX-1 (Jan)

Intel: 3 hours on 128 c5.18xlarge (April)

fast.ai: 2:57 on p3.16xlarge (April)

Google: 30 mins on TPU pod (April)

this result: 18 minutes on 16 p3.16xlarge

Overview

Part 1: How to train ImageNet in 18 minutes

Part 2: Democratization of large scale training

Part 3: What's next

Part 1: ImageNet in 18 minutes

How to train fast?

Step 1: Find a good single machine model

Step 2: Use 16 machines to train with 16x larger batch size

Step 3: Solve administrative/engineering challenges

Step 1: finding good model

Google's "High Performance Models" -- synthetic data only + only 1k im/sec

tf.Estimator? Nope

Google's "slim" repo -- used internally but external repo unmaintained

TensorPack. Worked + 2k im/sec. Original DAWN submission 14 hours

fast.ai PyTorch model with fp16: 5k-10k im/sec. 3 hours

Step 1: finding good model

Want model which:

a. Has high training throughput (5.5k images/second is good)

5x difference in throughput between best and "official" implementation

b. Has good statistical efficiency (32 epochs instead of 100)

2.5x difference in number of epochs between best tuned and typical

Step 1: finding good model: throughput tricks

ImageNet has a range of scales + convolutions don't care about image-size, so can train on smaller images first

2x smaller image = 4x faster

Throughput:

17k -> 5.8k -> 3.3k

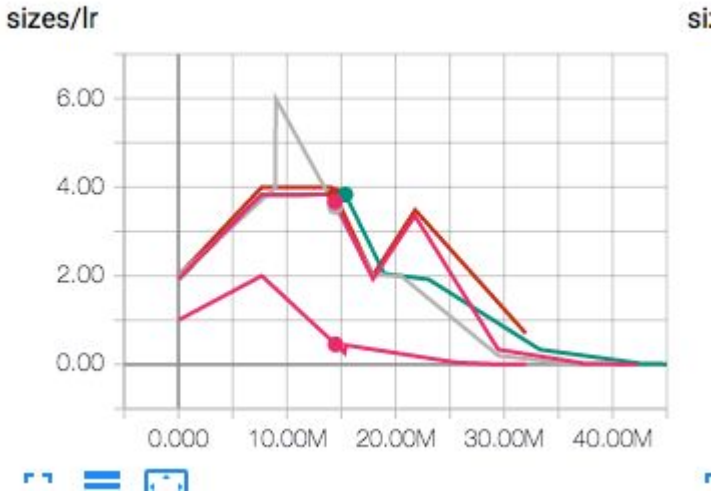
Result: 33 epochs < 2 hours on 1

Step 1: finding good model: statistical efficiency

Good SGD schedule = less epochs needed.

Best step length depends on:

1. Batch size
2. Image size
3. All the previous steps



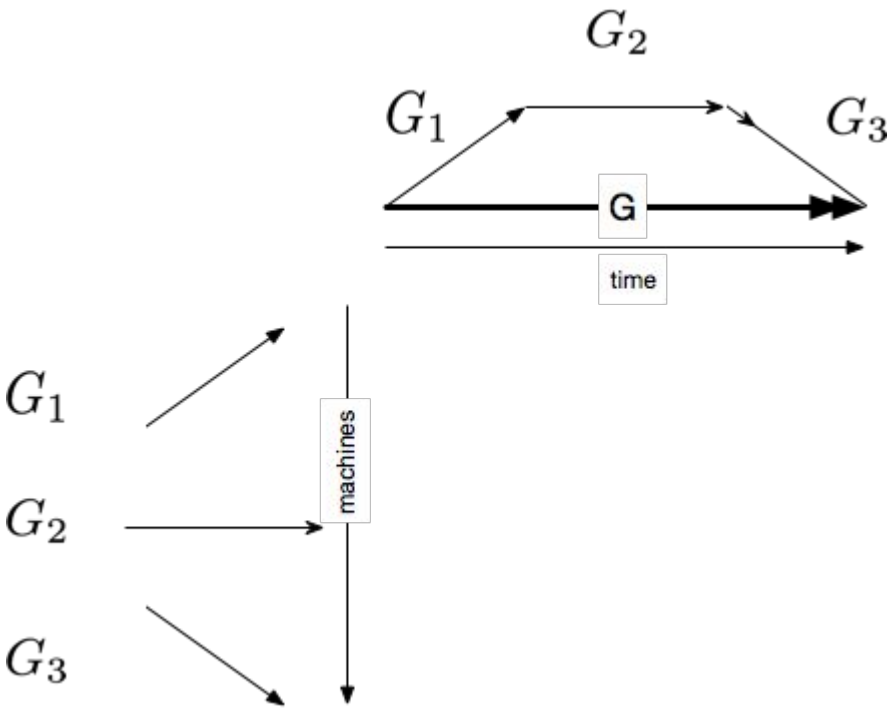
Step 2: split over machines

$$L = \sum_i l_i$$

$$G = \sum_i g_i$$

$$G = \underbrace{g_1 + g_2}_{G_1} \underbrace{+ g_3}_{G_2} + \underbrace{\cdots + g_n}_{G_3}$$

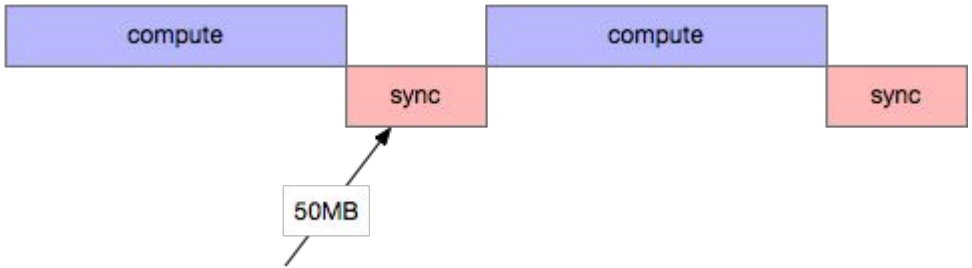
Step 2: split over machines



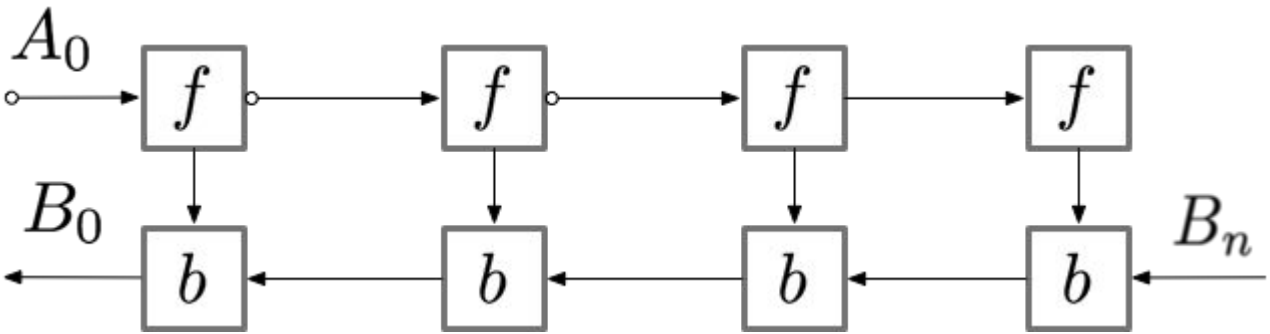
Linear scaling:

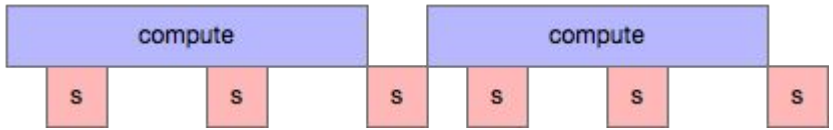
compute k times more gradients=
reduce number of steps k times

Synchronizing gradients



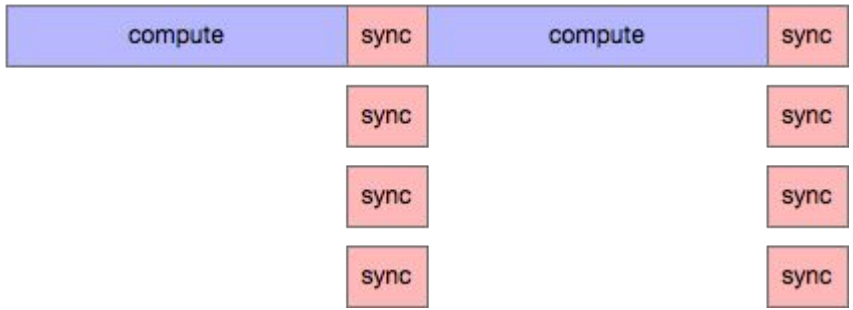
Synchronizing gradients: better way





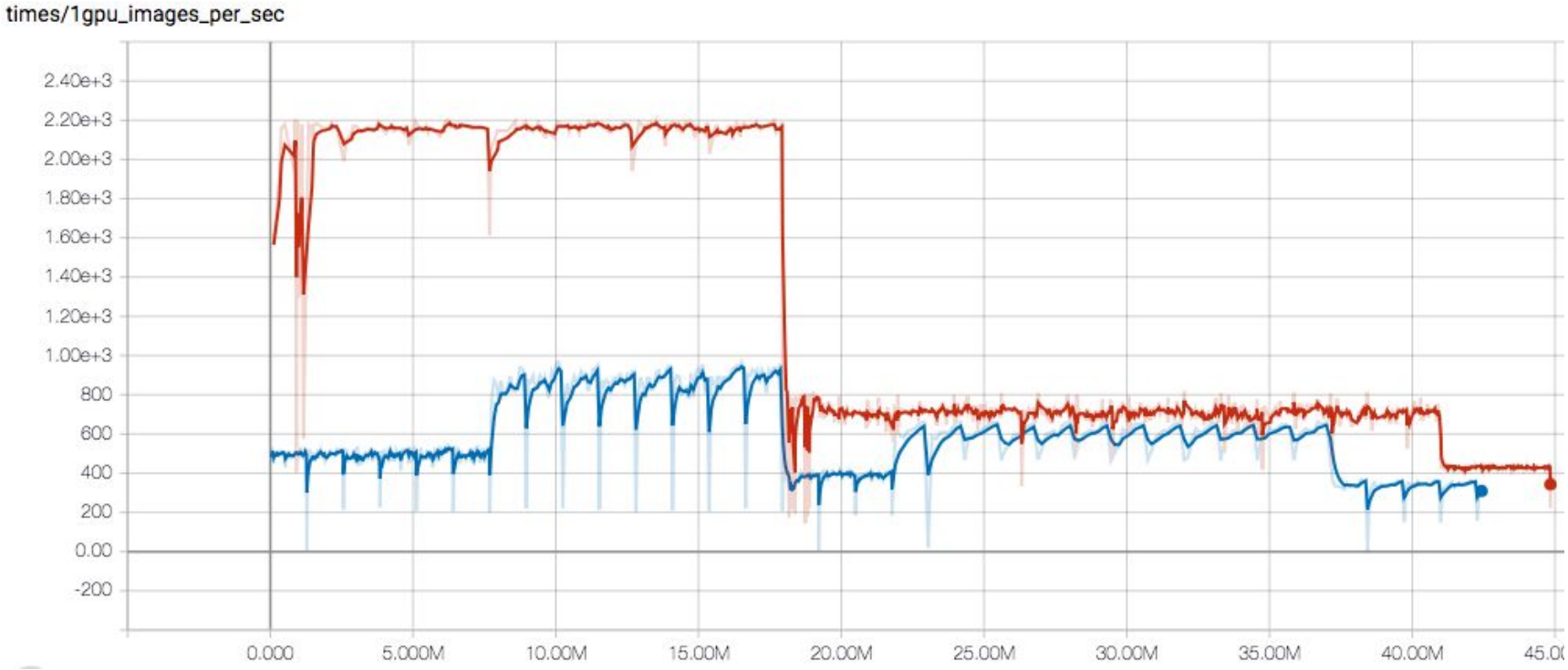
Synchronizing gradients: compromise

Use 4 NCCL rings instead of 1



Synchronizing gradients

16 machine vs 1 machine. In the end 85% efficiency (320 ms compute, 40 ms sync)



Step 3: challenges

Amazon Limits

Account had $3M

dedicated rep

weeks of calls/etc

Hello,

I'm sorry; but the service team were unable to process the limit increase you requested for EC2 instances at this time.

These limits are put in place to help you gradually ramp up activity and decrease the likelihood of large bills due to sudden, unexpected spikes. Once we have a broader window of usage on your account to review, we are happy to reassess any requests.

If you'd like to appeal this decision, please reply to this case with as much detail of your use case as possible to expedite review of your request.

I hope that helps, but please let me know if you have any questions.

Best regards,

Fabricio S.
Amazon Web Services

Amazon limits

New way

Yaroslav Bulatov <yaroslavvb@gmail.com>
to Chetan

No problem, here's the info
account number: 331439827203
ideally limits would match limits on old account number (316880547378), ie for p3.16xlarge

us-east-1: 32
us-east-2: 16
us-west-2: 16

•••

Kapoor, Chetan
to me

Done. New limits will be effective in about 20 mins.

Thanks,
-Chetan

how to handle data?

ImageNet is 150 GB, need to stream it at 500MB/s for each machine, how?

- EFS?

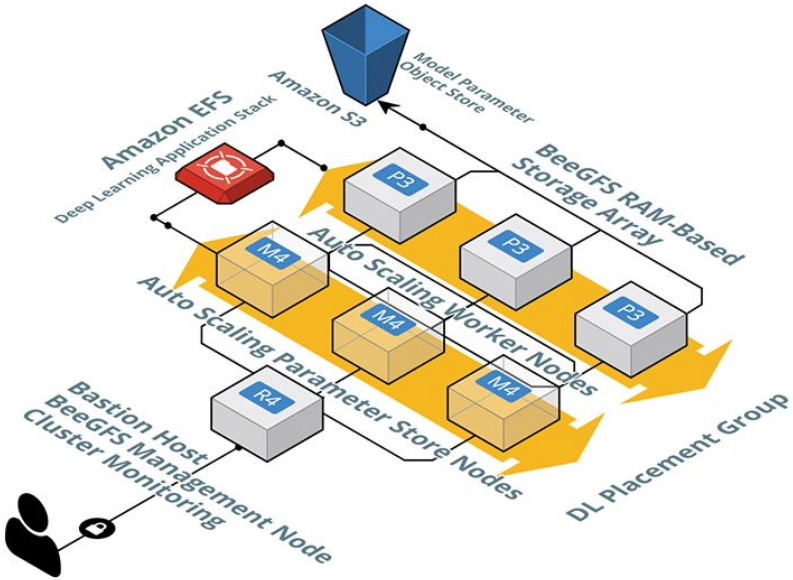
- S3?

- AMI?

- EBS?

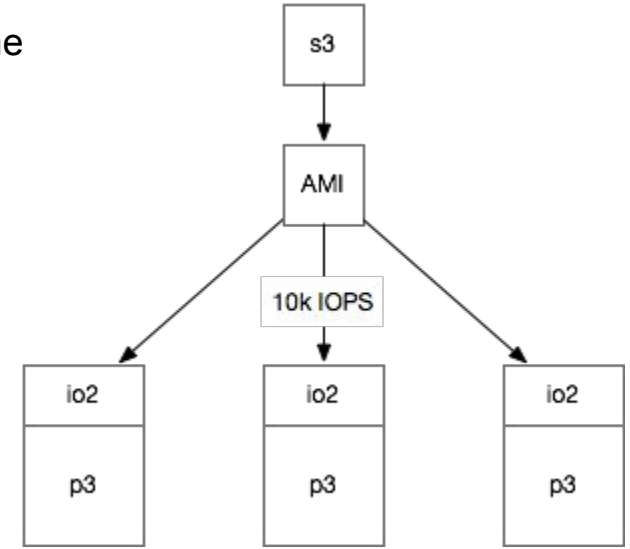
how to handle data?

ImageNet is 150 GB, need to stream it at 500MB, how?

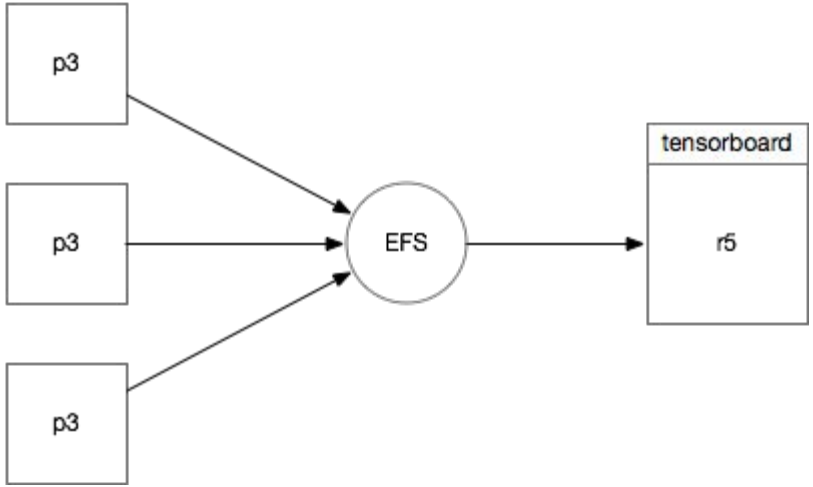


how to handle data?

Solution: bake into AMI, use high perf root volume
First pull adds 10 mins



How to keep track of results?



http://18.208.163.195:6006/

how to share with others?

1 machine: "git clone …; python train.py"

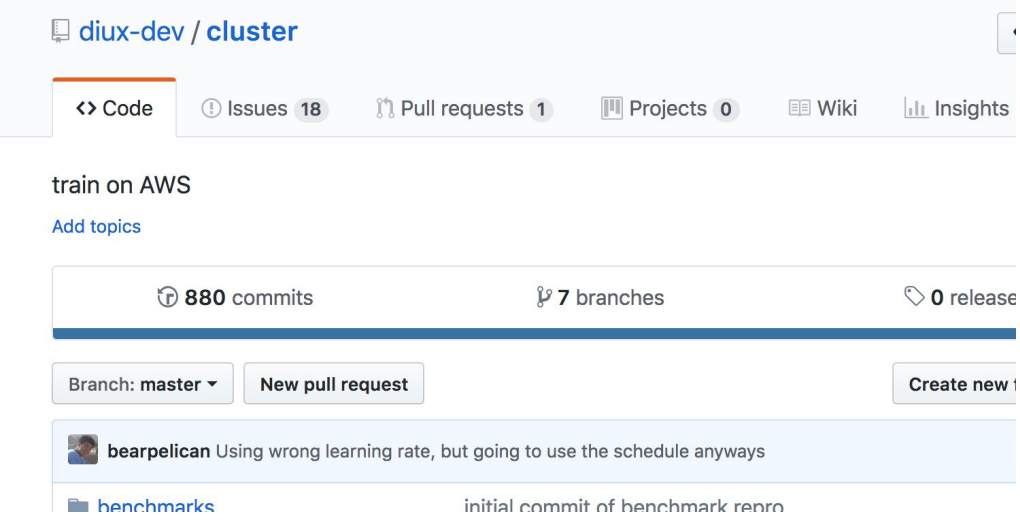
2 machines: ???

how to share with others?

1 machine: "git clone …; python train.py"

2 machines: ???

Setting up security groups/VPCs/subnets/EFS/mount points/placement groups



how to share with others?

Automate distributed parts into a library (ncluster)

```
import ncluster

task1 = ncluster.make_task(instance_type='p3.16xlarge')
task2 = ncluster.make_task(instance_type='p3.16xlarge')
task1.run('pip install pytorch')
task1.upload('script.py')
task1.run(f'python script.py --master={task2.ip}')
```

how to share with others?

Automate distributed parts into a library (ncluster)

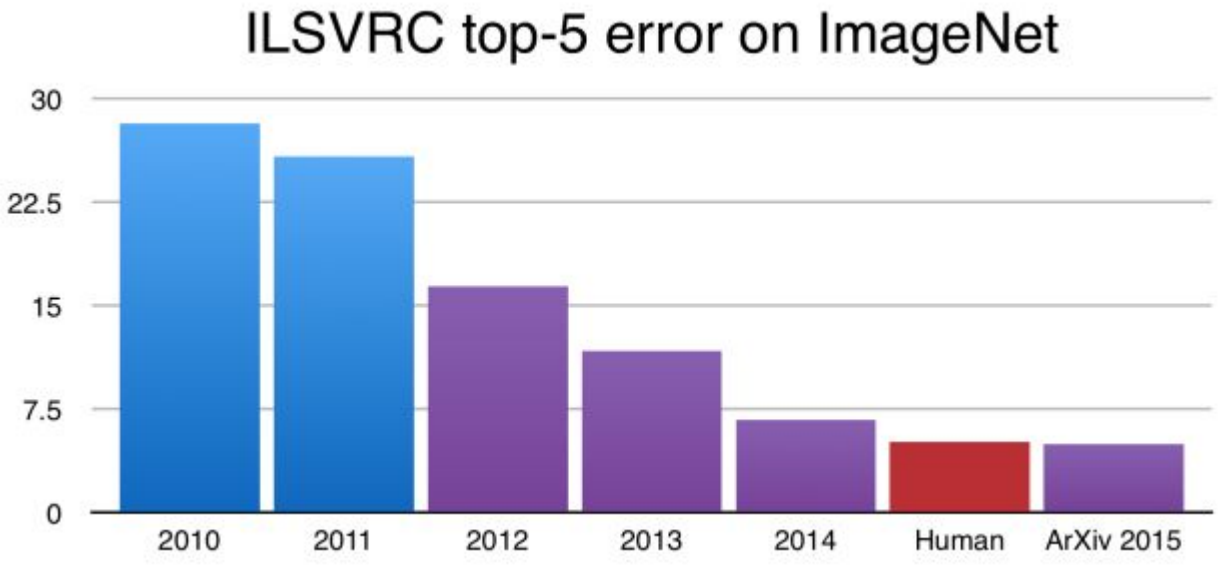
```
import ncluster

task1 = ncluster.make_task(instance_type='p3.16xlarge')
task2 = ncluster.make_task(instance_type='p3.16xlarge')
task1.run('pip install pytorch')
task1.upload('script.py')
task1.run(f'python script.py --master={task2.ip}')
```

https://github.com/diux-dev/imagenet18

pip install -r requirements.txt
aws configure
python train.py # pre-warming
python train.py

Part 2: democratizing large scale training

- Need to try many ideas fast



Part 2: democratizing large scale training

Ideas on MNIST-type datasets often don't transfer, need to scale up research.

- Academic datasets: dropout helps

- Industrial datasets: dropout hurts

Part 2: democratizing large scale training

Research in industrial labs introduces a bias

Google: make hard things easy

easy things impossible

10k CPUs vs Alex Krizhevsky 2 GPUs

async for everything

Part 2: democratizing large scale training

Linear scaling + per-second billing = get result faster for same cost

Training on 1 GPU for 1 week = train on 600 GPUs for 16 minutes

Spot instances = 66% cheaper

Part 2: democratizing large scale training

DGX-1 costs 150k, 10 DGX-1's cost 1.5M

You can use 1.5M worth of hardware for $1/minute

```
import ncluster

job = ncluster.make_job(num_tasks=10, instance_type="p3.16xlarge")

job.upload('myscript.py')

job.run('python myscript.py')
```

Part 3: what's next

Synchronous SGD: bad if any machine stops or fails to come up

Happens for 16, will be more frequent for more machines.

MPI comes from HPC, but need to specialize for the cloud.

Part 3: what's next

18 minutes too slow. Schedule specific to ImageNet

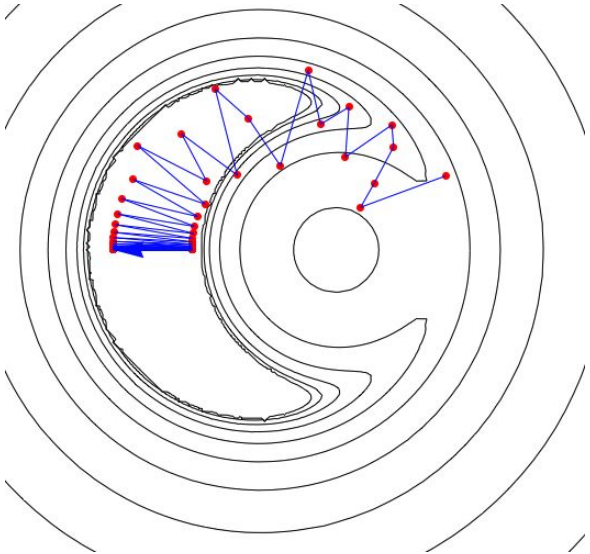
Should be: train any network in 5 minutes.

- Used batch size 24k, but 64k is possible (Tencent's ImageNet in 4 minutes)

- Only using 25% of available bandwidth

- Larger model = larger critical batch size (explored in https://medium.com/south-park-commons/otodscinos-the-root-cause-of-slow-neural-net-training-fec7295c364c)

Tuning is too hard. 1. SGD only gives direction, but not step length. Hence need schedule tuning



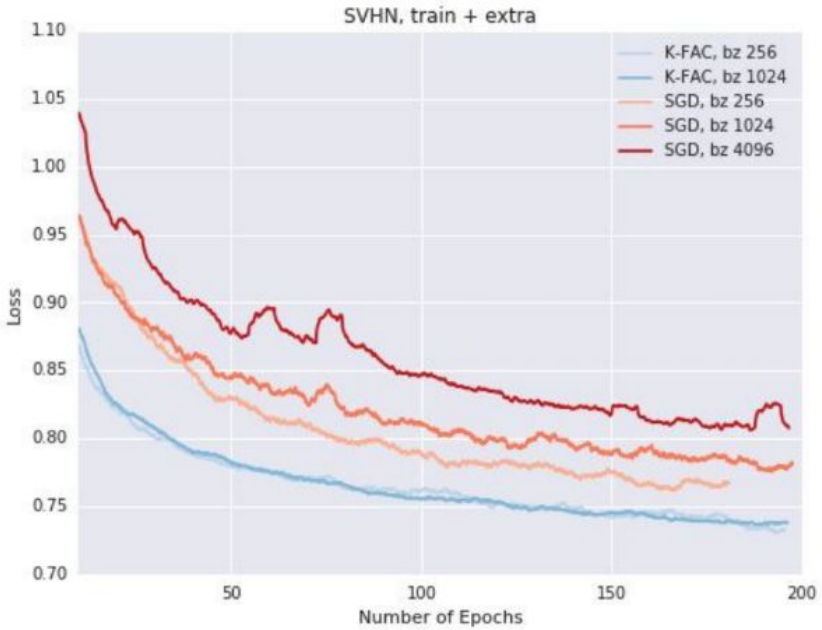
2. SGD not robust -- drop the D. Hence need other hyperparameter tuning

Machine Learning is new alchemy -- "round towards zero" to "round to even" error rate went from 25% to 99%

100k of AWS credits spent on "graduate student descent"

Part 3: what's next

SGD hits critical batch size too early



Part 3: what's next

Scalable second order methods in last 2 years:

Should address both robustness and schedule elements of SGD.

KFAC, Shampoo, scalable Gauss-Jordan, KKT, Curveball

Mostly tested on toy datasets. Need to try them out out and find which works on large scale.

Part 3: what's next

https://github.com/diux-dev/ncluster