Mixed precision training

With Apache MXNet

Apache MXNet Meetup
April 2018, Seattle

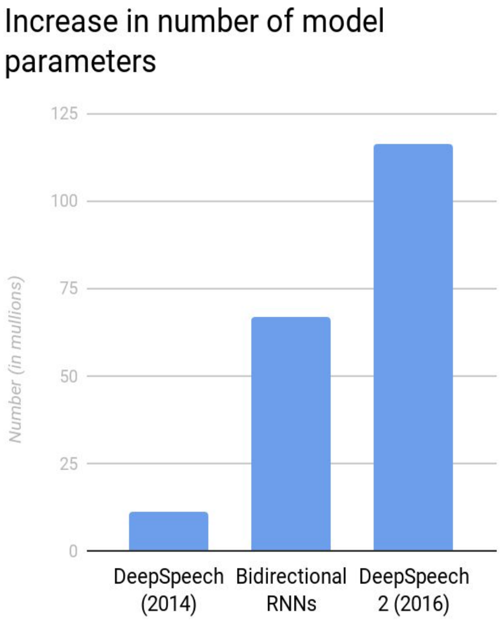
Rahul Huilgol (Amazon)
Dick Carter (Nvidia)

Outline

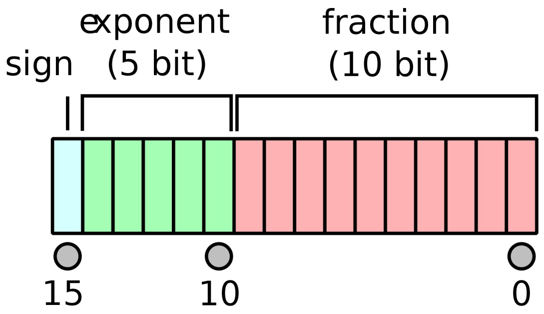
- Motivation
- Advantages
- Hardware support
- Mixed precision for deep learning
- Challenges
- Results

Motivation

- Trends in deep learning
  - Larger and more complex models
  - Larger training datasets
- Increased resource requirements
  - Compute
  - Memory



Reduced precision



- Using half precision floating point (float16)
- Advantages
  - Arithmetic speed
  - Memory bandwidth
  - Amount of memory used
- Using this in combination with single precision is **Mixed precision**

Hardware support

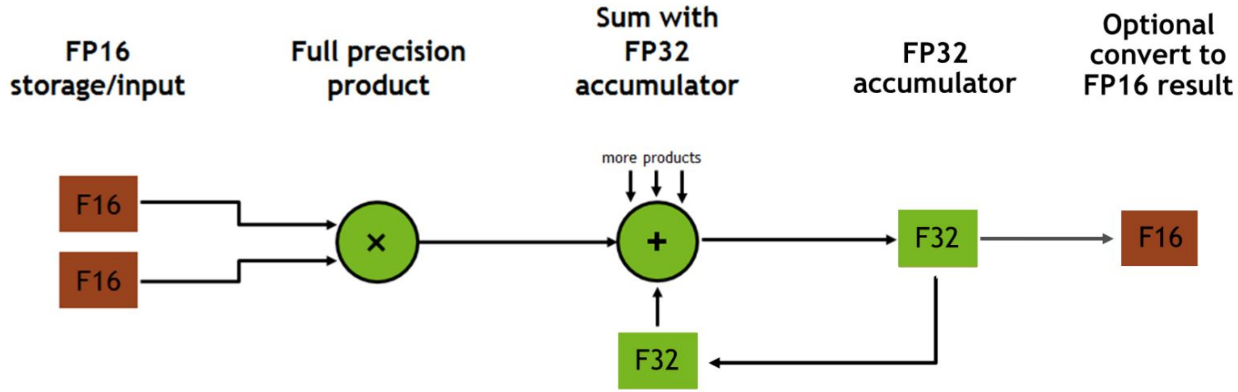
- Early support for float16 was only as a storage type
- Compute was slow
  - By casting to float32 and back
- Recent GPUs have specialized support for float16 arithmetic
  - Volta range of GPUs by Nvidia have **Tensor Cores**
  - Theoretically 2x-8x performance for matrix multiplication

Mixed precision for deep learning

- Potential to speed up training and inference
- Challenges
  - Maintaining precision of arithmetic
  - Imprecise weight updates
  - Gradient underflow
- We can retain same model accuracy as float32 by addressing the above

Maintaining precision

- Tensor Cores
    - Accumulate half precision products into single precision outputs



F16 accumulator is also available for inference

Source: Nvidia’s documentation about tensor cores

Maintaining precision

- Tensor Cores
  - Accumulate half precision products into single precision outputs
- Avoid large reductions in float16
  - Softmax
  - Batchnorm

Using Mixed precision with Apache MXNet

Symbolic

- Add a Cast layer to the network for layers to be computed in float16

```
data = mx.sym.Cast(data=data, dtype=np.float16)
```

- Cast back the output of network to float32 before softmax layer

Using Mixed precision with Apache MXNet

Gluon

- Cast block or network to float16

```
net = net.cast(np.float16)
```

- Cast data

```
data = data.astype(np.float16)
```

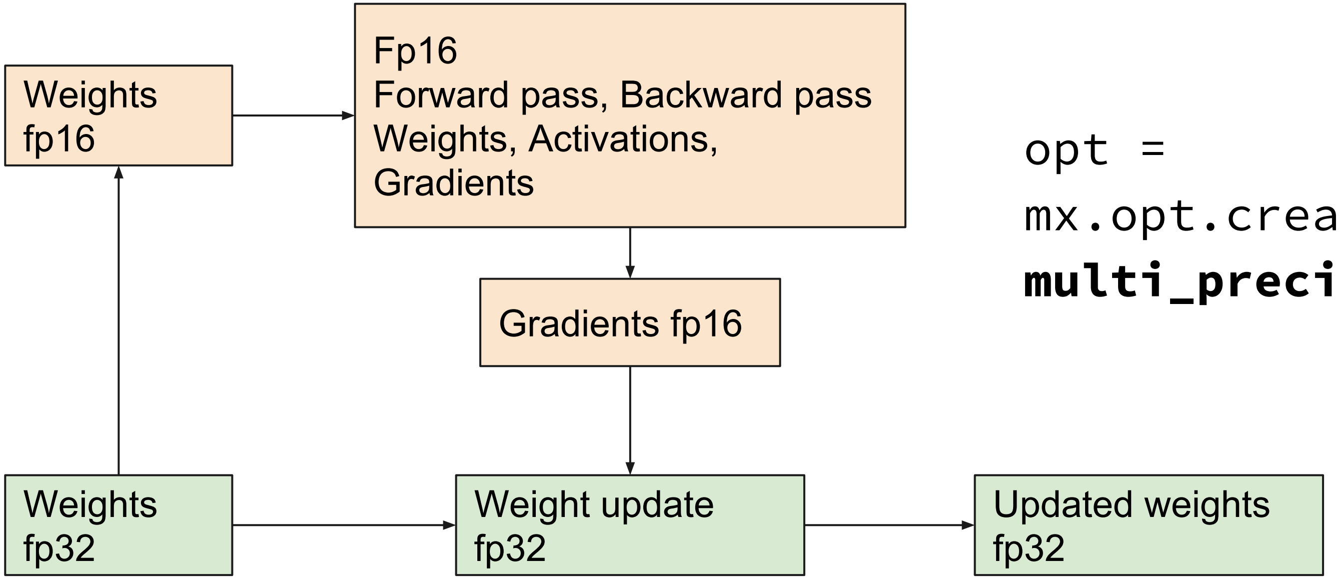
Imprecise weight updates

updates = (weight gradients) x (learning rate)
weight -= updates

- Updates may become too small for fp16 range
- Update may be too small compared to the weight (if >2048 times), then float16 addition causes update to become 0

- Solution: Maintain master copy of weights in float32

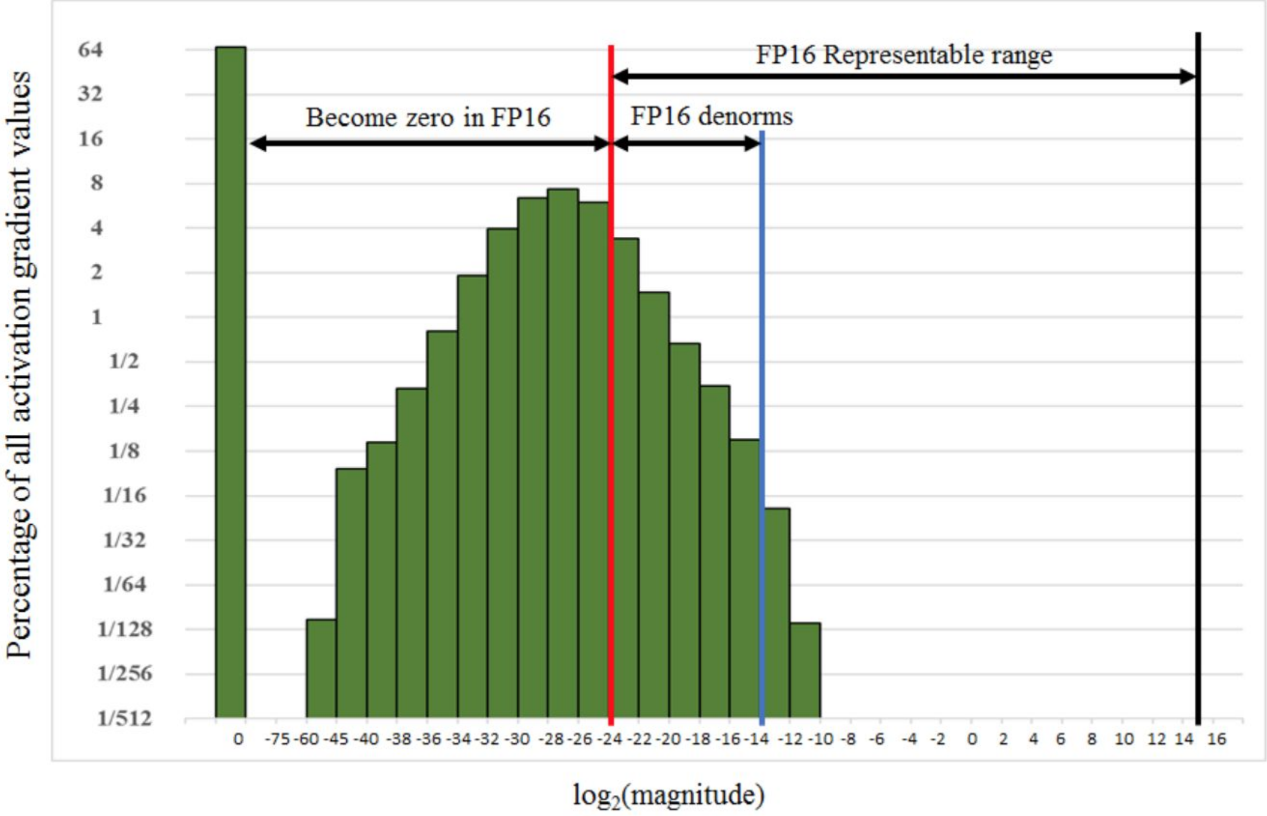
Imprecise weight updates




```
opt =
mx.opt.create('sgd',
multi_precision=True)
```

Gradient underflow

- Float16 exponents can range from -14 to 15
  - But gradients are usually small, i.e. negative exponents
- Small gradients when represented in float16 will become 0
- Can cause some networks to diverge
  - An example : Multibox SSD network



Histograms of gradients for Multibox SSD.
Source: Mixed Precision Training by Narang, et al. ICLR 2018

Loss scaling

Shift gradients to representable range

- Scale the loss computed after forward pass before backprop
- So all gradients are scaled, and don’t become zero
- Unscale the gradients before weight update, right after backward pass

Choosing scaling factor

- Pick a factor from 8, 32, 64, 128, etc as long as doesn’t cause overflow

Loss scaling with Apache MXNet

Gluon

```
loss = gluon.loss.SoftmaxCrossEntropyLoss(weight=128)
```

Symbolic

```
mx.sym.SoftmaxOutput(..., grad_scale=128.0)
```

Optimizers for both interfaces

```
opt = mx.opt.create(..., rescale_grad=1.0/128)
```

NVIDIA research's training results
With mixed precision

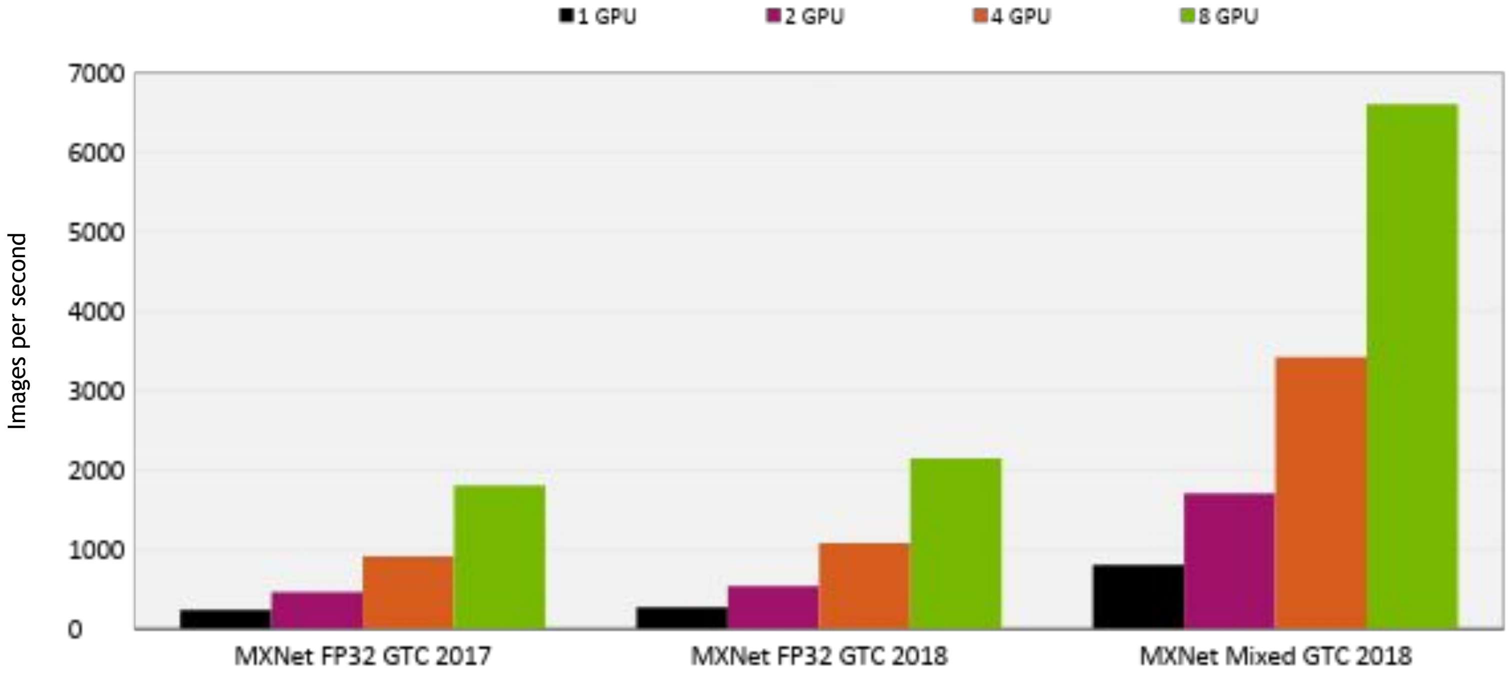
Successfully applied to many networks including :

- Imagenet CNNs
- Detection
- Language Translation
- Speech
- Text to Speech
- GAN
- Image enhancement
- Wavenet

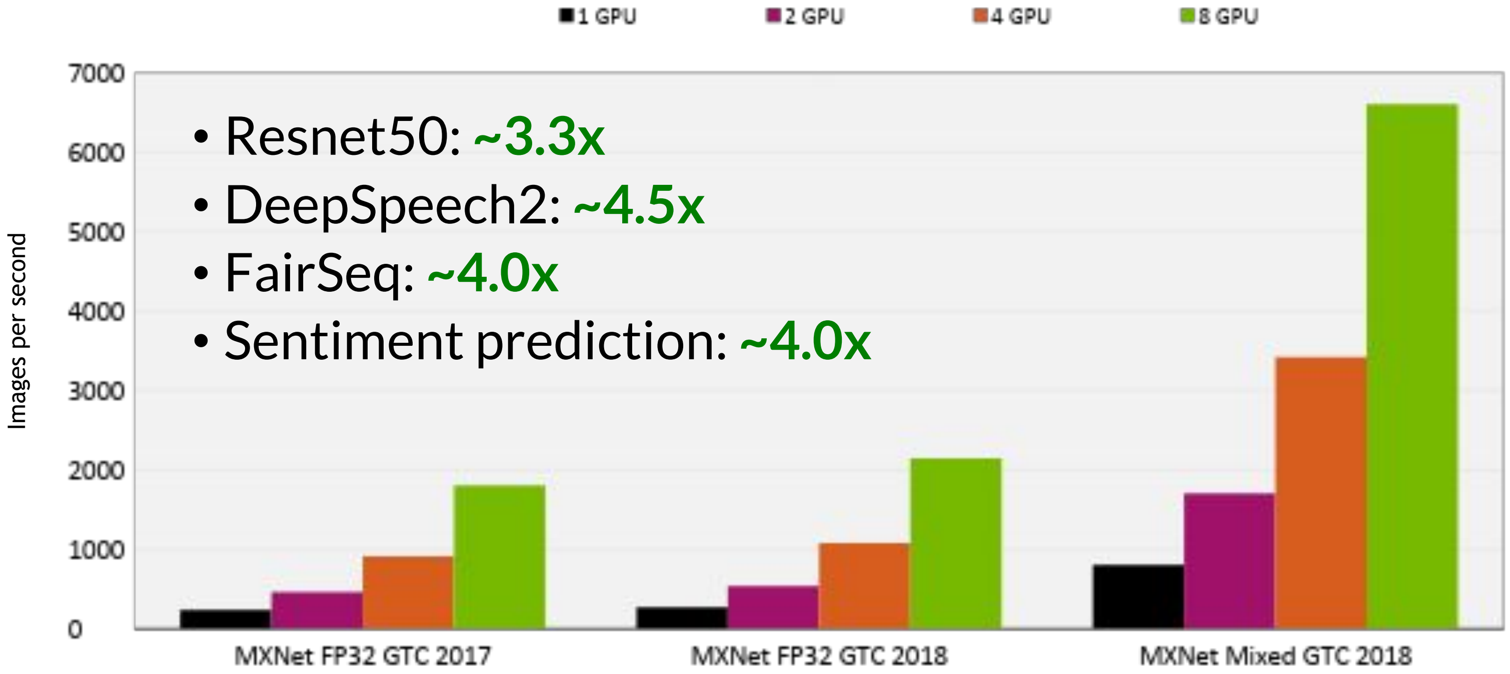
ILSVRC12 Networks, Top-1 Accuracy

Network	FP32 Baseline	Mixed precision
AlexNet	56.8%	56.9%
VGG-D	65.4%	65.4%
GoogLeNet	68.3%	68.4%
Inception v2	70.0%	70.0%
Inception v3	73.9%	74.1%
Resnet 50	75.9%	76.0%
ResNeXt 50	77.3%	77.5%

MXNet Resnet50: fp32 vs mixed-precision



MXNet Resnet50: fp32 vs mixed-precision



- Resnet50: **~3.3x**
- DeepSpeech2: **~4.5x**
- FairSeq: **~4.0x**
- Sentiment prediction: **~4.0x**

Conclusions

- **Mixed precision training benefits:**
  - Faster: Math and memory I/O speedups
  - Smaller: Can explore larger minibatches and inputs
- **Solutions developed to address potential issues**
  - FP32 accumulation via Tensor Cores to maintain accuracy
  - 32-bit master weights for precise weight updates
  - Loss scaling to handle gradient underflow
- **Mixed precision matches FP32 training accuracy for a variety of:**
  - Tasks: classification, regression, generation
  - Problem domains: images, language translation, language modeling, speech
  - Network architectures: feed forward, recurrent
  - Optimizers: SGD, Adagrad, Adam

Information Sources

Where to learn about mixed precision training

GTC 2018 Talks, available publicly soon:

S8923 - Training Neural Networks with Mixed Precision: Theory and Practice

S81012 - Training Neural Networks with Mixed Precision: Real Examples

Also on the web:

Mixed- Precision Training of Deep Neural Networks (NVIDIA Developer Blog)

Training with Mixed Precision (NVIDIA User Guide)

Thank you!

Questions?