Mixed Precision Training

By- Sharan Narang , Gregory Diamos, Erich Elsen, Paulius Micikevicius, Jonah Alben, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, Hao Wu

- Published as a conference paper at ICLR 2018

1. Introduction

With the advancement of technology in the Deep Learning community, better and complex models are being introduced to solve everyday tasks like **Image recognition, Language Modelling, Machine Translation, Speech Recognition,** etc. These models usually require more computational and memory resources to train. This paper attempts to reduce these hardware requirements by using lower precision representation and arithmetic. The technique proposed in this paper addresses two of the three limiters of any program, Memory Bandwidth, Arithmetic Bandwidth, and Latency.

- **Memory Bandwidth** is lowered by using a fewer number of bits to store the same values.
- **Arithmetic Bandwidth** is lowered by using half precision math. It is observed that half-precision math increases throughput 2 to 8 times in recent GPUs.

They use IEEE half-precision format (FP16) instead of the traditional single-precision (FP32) format for reducing precision. Then, they propose three methods to prevent model accuracy loss while training with reduced precision:

- Maintaining a master copy of weights in FP32
- Loss-scaling that minimizes gradient values becoming zeros
- FP16 arithmetic with accumulation in FP32

2. Related Work

There were many papers before this proposing reduced precision. But, none of them used reduced precision during forward and backward passes. This is most crucial,

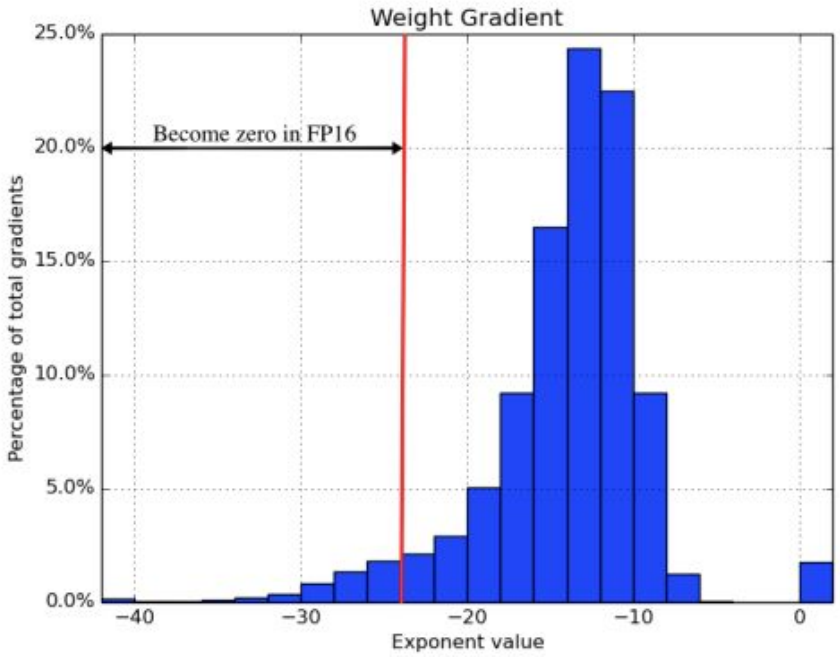
because **forward** and **backward passes** use most of the computational and memory resources. Also, none of the **Hyperparameters** are adjusted. Also, models trained with these techniques, even **state-of-the-art** models, **do not incur accuracy loss**.

3. Implementation

3.1. FP32 Master copy of weights

One may ask, why do we need to store a FP32 Master copy of weights at all? Why not do everything in the FP16 format? There are two reasons for this:

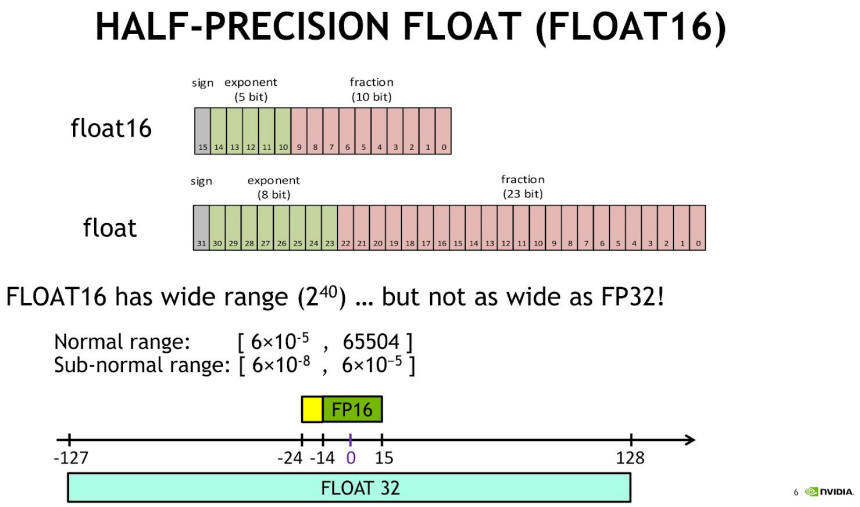
- **Small weight updates** are sometimes too small (smaller than 2^(-24), to be precise) to be represented in FP16 and are taken as zero by the format. The following figure illustrates that almost 5% of the weight updates are smaller than 2^(-24). This would adversely affect the model accuracy.



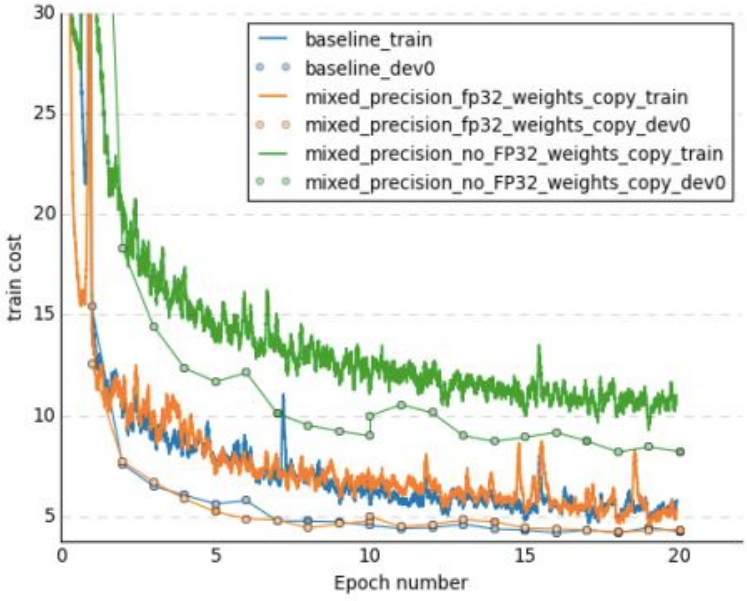
(b) Gradient histogram for Mandarin training run

- **Arithmetic** in the GPU can make the weight update zero when right shifting bits to align the binary point with the weight. FP16 has 10 bits of mantissa, so in order for the GPU to make the weight update zero, the

implicit bit has to be shifted by 11 or more bits. Thus, the weight must be at least 2^(11) (~2048) times bigger than the weight update.



These curves show a relative accuracy loss of 80% when a FP32 master-copy of weights is not made:



(a) Training and validation (dev0) curves for Mandarin speech recognition model

Even though maintaining a master copy for the weights by 50%, this is compensated by the savings in memory requirements during training, due to activations also being stored in Half-precision format.

The **main algorithm** is as follows:

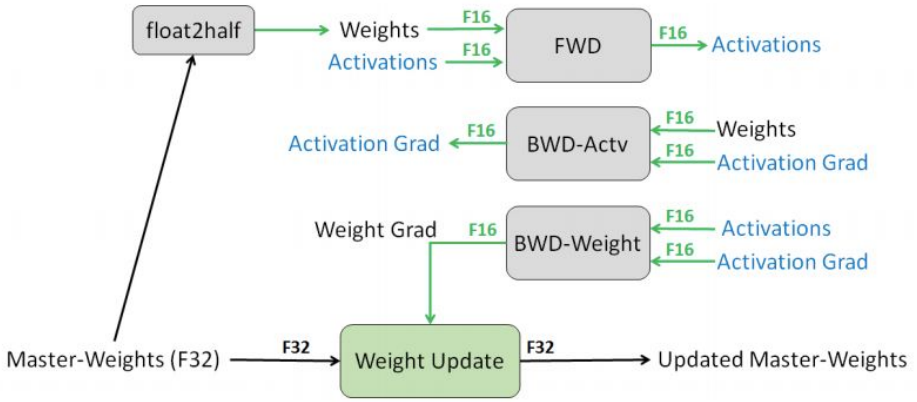
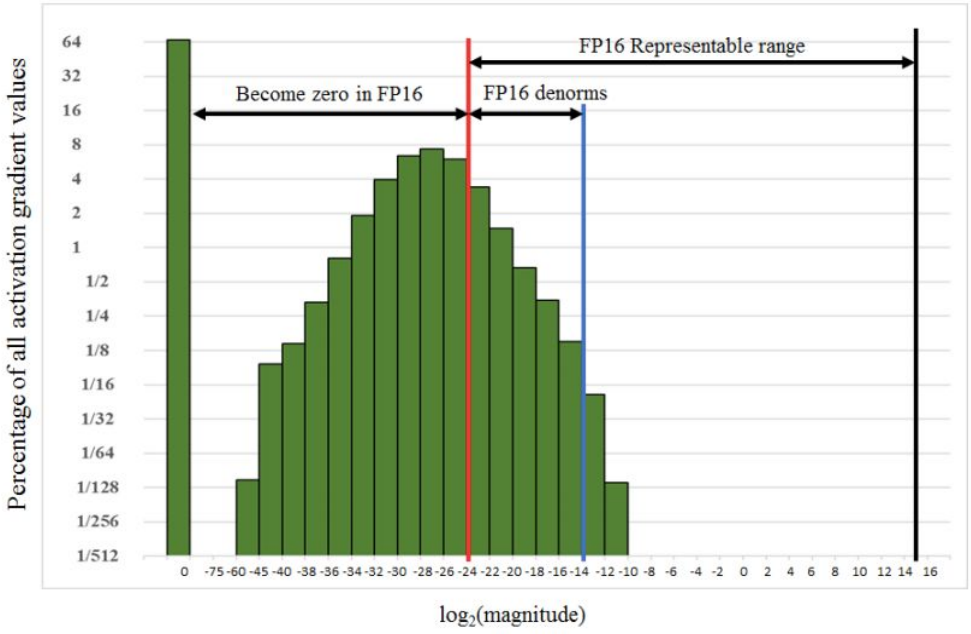


Figure 1: Mixed precision training iteration for a layer.

1. F32 weights are converted to F16.
2. Forward and backward passes are done, all in F16.
3. Weight updates obtained in F16, are converted to F32.
4. Weight updates are done in F32, and the F32 master copy of weight is updated.
5. Step 1 for the next epoch.

3.2. Loss Scaling

In FP16, the lowest exponent can be -24. Above it, until the exponent is -14, all numbers represented in FP16 will suffer **Underflow**. This range is called the **FP16 denormalisation range** (FP16 denorms, for short). This is called exponent bias of FP16. Now, gradient values during NN Training tend to be dominated by small magnitudes. This can be seen from the histogram below. Most of the FP16 representable range was left unused, while many values were below the shift them to occupy more of the representable range and preserve values that are otherwise lost to zeros. Scaling up the gradients will shift them to occupy more of the representable range and preserve values that are otherwise lost to zeros.



To scale the gradients, what we do is, we scale the loss value computed in the forward pass, prior to starting back-propagation. By chain rule, back-propagation ensures that all the gradient values are scaled by the same amount. This requires no extra operations during back-propagation and keeps the relevant gradient values from becoming zeros. Weight gradients must be unscaled before weight update to maintain the update magnitudes as in FP32 training. It is simplest to perform this unscaling right after the backward pass but before gradient clipping or any other gradient-related computations, ensuring that no hyper-parameters (such as gradient clipping threshold, weight decay, etc.) have to be adjusted.

Choosing a Scaling Factor: A Scaling Factor can be chosen empirically, as long as the scaled gradients don't cause overflow during backward pass or any other computational disturbances.

3.3. Arithmetic Precision

There are mainly three categories of NN Arithmetic: **Vector dot-products, Reductions, and Pointwise operations**. To maintain model accuracy, these observations were given for carrying out Arithmetic with Mixed Precision:

- **Vector dot-products** must accumulate the partial products into an FP32 value, which should be converted to FP16 before writing to memory. Without this accumulation in FP32, some FP16 models did not match the accuracy of the baseline models.
- **Reductions** should be carried out in FP32. The values for these reductions are read and written in FP16, but the arithmetic is performed in FP32.
- **Pointwise** operations, such as non-linearities and element-wise matrix products, are memory bandwidth limited. Since arithmetic precision does not impact the speed of these operations, either FP16 or FP32 math can be used.

4. Results

4.1. CNNs for ILSVRC Classification Task

Models: Alexnet, VGG-D, GoogLeNet, Inception v2, Inception v3, and pre-activation Resnet-50
Dataset: ILSVRC
GPU(Baseline FP32): NVIDIA's Maxwell or Pascal GPU
GPU(MP FP16): Volta V100
Framework: Caffe framework modified to use Volta TensorOps, except for Resnet50 which used PyTorch.
Loss Scaling: Not required
Results: In all of these cases MP was able to match the top-1 accuracy of baseline FP32 training session using identical hyper-parameters.

Model	Baseline	Mixed Precision	Reference
AlexNet	56.77%	56.93%	(Krizhevsky et al., 2012)
VGG-D	65.40%	65.43%	(Simonyan and Zisserman, 2014)
GoogLeNet (Inception v1)	68.33%	68.43%	(Szegedy et al., 2015)
Inception v2	70.03%	70.02%	(Ioffe and Szegedy, 2015)
Inception v3	73.85%	74.13%	(Szegedy et al., 2016)
Resnet50	75.92%	76.04%	(He et al., 2016b)

4.2. Detection CNNs

Models: Faster-RCNN, Multibox-SSD
Dataset: VOC-2007, VOC-2007+VOC-2012 respectively
GPU(Baseline FP32): NVIDIA's Maxwell or Pascal GPU
GPU(MP FP16): Volta V100.

Loss Scaling: Not required, Required by a factor of 8
Results: SSD detector failed to train in FP16 without loss-scaling. By losing small gradient values to zeros, poor weights are learned and training diverges. But, when loss scaling was implemented, both the models were able to match their baseline performances.

Model	Baseline	MP without loss-scale	MP with loss-scale
Faster R-CNN	69.1%	68.6%	69.7%
Multibox SSD	76.9%	diverges	77.1%

4.3. Speech Recognition

Models: DeepSpeech 2 for both English and Mandarin datasets
Dataset: Internal Datasets containing 6000 hours of English speech, and 2600 hours of Mandarin Speech.
GPU(Baseline FP32): NVIDIA's Maxwell or Pascal GPU
GPU(MP FP16): Maxwell GPUs using FP16 storage only
Results: Similar to classification and detection networks, mixed precision training works well for recurrent neural networks trained on large scale speech datasets. These speech models are the largest models trained using this technique. Also, the number of time-steps involved in training a speech model are unusually large compared to other applications using recurrent layers. As shown in table 3, Pseudo FP16 results are roughly 5 to 10% better than the baseline. This suggests that the half-precision storage format may act as a regularizer during training.

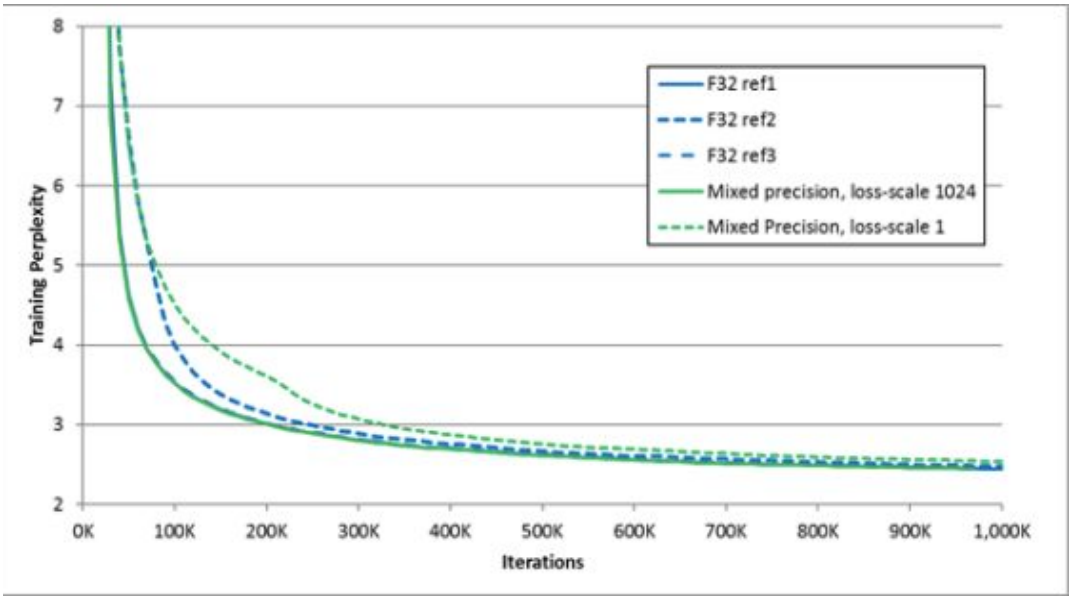
Model/Dataset	Baseline	Mixed Precision
English	2.20	1.99
Mandarin	15.82	15.01

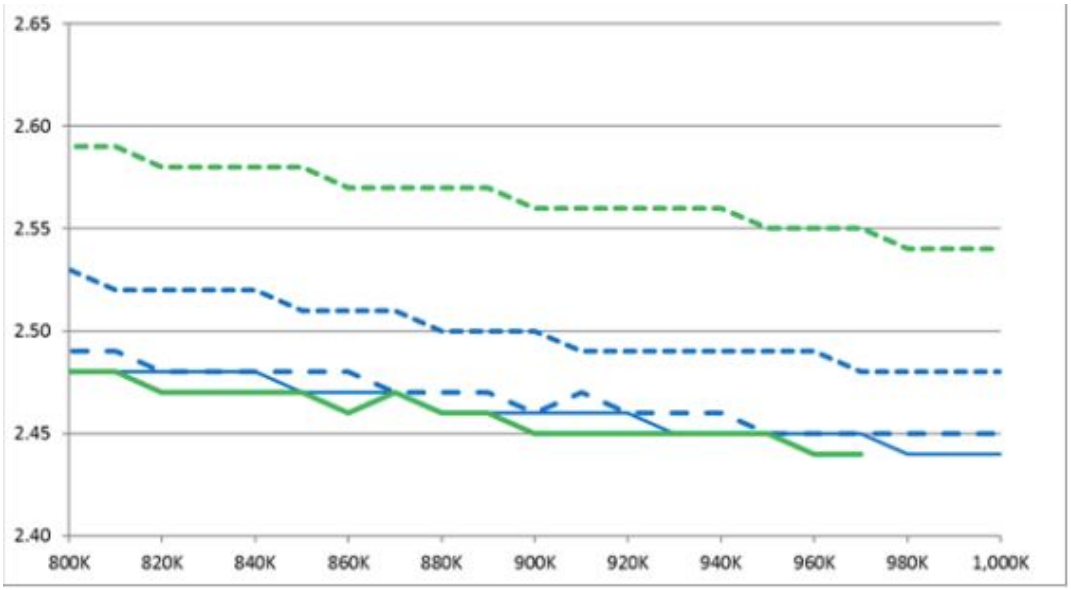
Character Error Rate for both the testings. English results are reported on the WSJ '92 test set. Mandarin results are reported on an internal test set.

4.4. Machine Translation

Models: 3x1024 LSTM
Dataset: WMT15
GPU(Baseline FP32): NVIDIA's Maxwell or Pascal GPU
GPU(MP FP16): Volta V100

Results: Mixed-precision with loss-scaling matched the FP32 results, while no loss-scaling resulted in a slight degradation in the results. The 5-layer model exhibited the same training behavior.



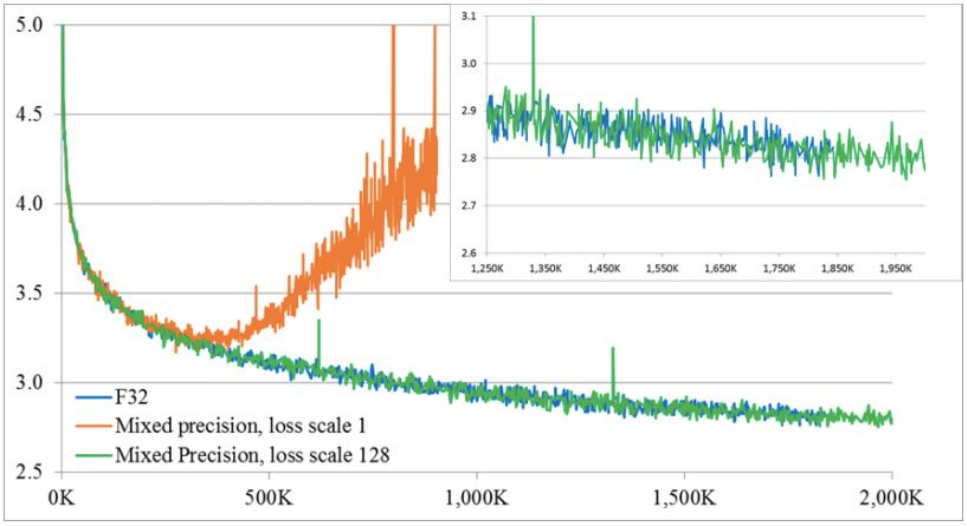


English to French translation network training perplexity, 3x1024 LSTM model with attention. Ref1, ref2 and ref3 represent three different FP32 training runs.

4.5. Language Modelling

Models: bigLSTM

GPU(Baseline FP32): NVIDIA's Maxwell or Pascal GPU
GPU(MP FP16): Volta V100
Results: To match FP32 perplexity training this network with FP16 requires loss-scaling. Without loss scaling the training perplexity curve for FP16 training diverges, compared with the FP32 training, after 300K iterations. Scaling factor of 128 recovers all the relevant gradient values and the accuracy of FP16 training matches the baseline run.



4.6. DCGAN Results

Models: DCGAN
Dataset: CelebFaces dataset
GPU(Baseline FP32): NVIDIA's Maxwell or Pascal GPU
GPU(MP FP16): Volta V100
Results: Qualitatively the outputs of FP32 and mixed-precision training appear comparable. This network did not require loss-scaling to match FP32 results.

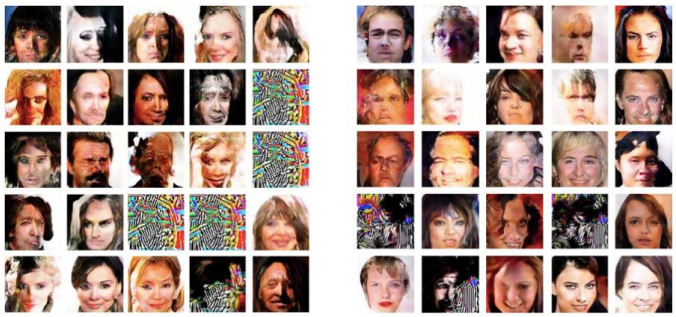


Figure 6: An uncurated set of face images generated by DCGAN. FP32 training (left) and mixed-precision training (right).

5. Conclusion

Mixed Precision is an important and a very useful technique that allows us to reduce the memory consumption as well as time spent in memory and arithmetic operations of deep neural networks. The results show that this technique can be used to train a variety of models without loss in accuracy, and without any hyper-parameter tuning. For some models, loss scaling has to be incorporated as well in order to match the baseline model accuracy, but then, the computational tasks decrease manifold. However, there are a few drawbacks that can be improved upon in the future:

- **Batch Normalization** hates FP16 representation.
- **Latency:** Latency is the time delay between carrying out a task, and receiving the instructions to do so. When programs are latency-limited, MP doesn't help much.