Mixed Precision Training

计算平台事业部PAI团队
Overview

• What is mixed-precision & Why mixed-precision
• How mixed-precision
• Mixed-precision tools on PAI-tensorflow
• Experimental results
What is mixed-precision

• mixed-precision
  • FP32 and FP16
  • *More precision format in the future*

• TensorCore
  • Matrix-multiply and accumulate units
    • FP16 storage/inputs
    • FP32/Fp16 accumulator

• Such as:
  • Conv
  • MatMul

Figure 2: Volta GV100 Tensor Core operation.
Why mixed-precision

- Two key points which matter in training/inference:
  - Computation
    - Tensorcore 8X higher throughput in MP than FP32 (15Tflops v.s. 120Tflops)
  - Memory access
    - Inputs is FP16
    - Memory access is reduced by 2X
Overview

• What is mixed-precision & Why mixed-precision
• How mixed-precision
• Mixed-precision tools on PAI-tensorflow
• Experimental results
How mixed-precision

• Key Strategies in mixed-precision training
  • Issues using FP16 for training and the solutions
    • Less bits in fraction: → Precision gap in sum
    • Less bits in exponent: → Gradients underflow
  • Arithmetic precision design
    • Considering both efficiency and performance
Issues using FP16 for training

• Less bits in fraction: Precision gap in sum
  • $A + B$, if $A/B > 2^{10}$, $B$ will degrade to zero.
    • For FP32, the ratio can be up to $2^{23}$
  • Common in weight update:
    • $W \leftarrow W + lr \times dW$
• Less bits in exponent: Gradients underflow
  • Gradients smaller that $2^{-24}$ will become zero
Precision gap in sum

• variables v.s. gradients
  • weight update: $W \leftarrow W + lr \times dW$ (lr normally in $[10^{-1}, 10^{-4}]$)

Variables: $2^{-16}$ to $2^{-4}$

gradients: $2^{-30}$ to $2^{-5}$

**Fig.** Variables and gradients histogram in Faster RCNN

• **Solution**: Variables stored in FP32, and optimizer computation in FP32
Gradients underflow in FP16

• Gradients of variables

Fig. Histogram for gradients of variables in Faster RCNN, respectively training in mixed-precision and FP32
Gradients underflow in FP16

- Gradients of activations

Fig. Histogram for gradients of variables in Faster RCNN, respectively training in mixed-precision and FP32
Gradients underflow in FP16

Solution: gradients shift using loss scaling
Gradients underflow in FP16

• Constant loss scaling
  • Scale the loss by a factor $S$
  • Backprop to compute the $dW$
  • Unscale $dW$ by $1/S$

• Automatic loss scaling
  • Start with a large scaling factor $S$
  • For each training iteration:
    • Scale the loss by $S$
    • Backprop to compute the $dW$
    • Unscale $dW$ by $1/S$
    • If $dW$ contains Inf/NaN, decrease $S$ by a step factor $S/\text{step}$
    • Otherwise, update $dW$ to $W$
      • If there is no Inf/NaN for $N$ updates, increase $S$ by a step factor $S*\text{step}$
How mixed-precision

• Key Strategies in mixed-precision training
  • Issues using FP16 for training and the solutions
    • Less bits in fraction: → Precision gap in sum
      • Solution: Variables stored in FP32, and optimizer computation in FP32
    • Less bit in exponent: → Gradients underflow
      • Solution: loss scaling
  • Arithmetic precision design
    • Considering both efficiency and performance
How mixed-precision

• Key Strategies in mixed-precision training
  • Issues using FP16 for training and the solutions
    • Less bits in fraction: → Precision gap in sum
      • Solution: Variables stored in FP32, and optimizer computation in FP32
    • Less bit in exponent: → Gradients underflow
      • Solution: loss scaling
  • Arithmetic precision design
    • Considering both efficiency and performance
Arithmetic precision design

- Arithmetic can be categorized into:
  1. Compute-bound
     - Convolution, Matmul
  2. Memory-bound
     ① Reductions
        - Batch-norm/layer-norm/group-norm
        - Softmax / Average pooling
     ② Element-wise operation
        - Add/mul, etc

Take advantage of Tensorcore:
- Inputs: FP16
- Accumulator: FP32
- Outputs: FP32

- Inputs/Outputs in FP16
- Computation in FP32

- Inputs/Outputs in FP16
  - Computation in FP32
Arithmetic precision design

• Compute-bound operations:
  • Inputs in FP16
  • Computation using Tensorcore
  • Outputs in FP32

• Memory-bound operations:
  • Inputs/outputs in FP16
  • Computation in FP32
How mixed-precision training

**FP32 training baseline**

- **Optimizer related** → should be in **FP32**
- **Computation: forward and backward** → Can be in **MP**
MP training (var in FP32):

- Convert the computation part to MP
- Remain the optimizer part in FP32
MP training (var in FP32):

- Loss Scaling strategy (*constant scaling*)
MP training (var in FP32):

- Auto Loss Scaling strategy
MP training (var in FP32):

- Auto Loss Scaling strategy

```text
Apply_gradients → FP32 Vars → FP16 Vars → Forward (in MP) → Loss

is_overflow

Yes: 1. skip update 2. downscale scaling factor → update

No: *inv(scaling) → FP32 grads

FP32 grads → FP16 grads → Backward (in MP)
```

*scaling
MP training (var in FP32):

- Auto Loss Scaling strategy
MP training (var in FP32):

- **Auto Loss Scaling strategy**

![Diagram]

- **Apply_gradients** → **FP32 Vars** → **FP16 Vars** → **Forward (in MP)** → **Loss**

  - **is_overflow**
    - **Yes** → **1. skip update 2. downscale scaling factor** → **Loss**
    - **No** → **FP32 grads** → **is_stable for certain iters**
      - **Yes** → **upscale scaling factor** → **FP32 Vars**
      - **No** → **FP16 grads** → **Backward (in MP)** → **scaling**

[Diagram]
Overview

• What is mixed-precision & Why mixed-precision
• How mixed-precision
• Mixed-precision tools on PAI-tensorflow
• Experimental results
• Graph Optimization + Loss Scaling Training Strategy
  • Graph Optimization: AutoMixedPrecision Graph Optimization Pass
  • MP Training Strategy: MP optimizer wrapper
    • Wrap the standard optimizers to automatically adopt the constant/automatic loss scaling strategy
      • \texttt{opt = tf.contrib.mixed_precision.MixedPrecisionOptimizer(\texttt{opt})}
    • Both constant/automatic loss scaling supported
Experimental results

- ResNet50 on ImageNet
## Experimental results

- Faster RCNN (VGG backbone) on PASCAL VOC 07

<table>
<thead>
<tr>
<th></th>
<th>mAP</th>
<th>mAP Diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP32 (baseline)</td>
<td>69.48</td>
<td>-</td>
</tr>
<tr>
<td>FP16 (no loss scaling)</td>
<td>67.83</td>
<td>-1.65</td>
</tr>
<tr>
<td>FP16 (constant scaling=8)</td>
<td>69.71</td>
<td>+0.23</td>
</tr>
<tr>
<td>FP16 (constant scaling=64)</td>
<td>69.96</td>
<td>+0.48</td>
</tr>
<tr>
<td>FP16 (constant scaling=256)</td>
<td>69.78</td>
<td>+0.30</td>
</tr>
<tr>
<td>FP16 (constant scaling=4096)</td>
<td>69.74</td>
<td>+0.26</td>
</tr>
<tr>
<td>FP16 (auto scaling)</td>
<td>69.68</td>
<td>+0.20</td>
</tr>
</tbody>
</table>
Experimental results

• SSD (VGG backbone) on PASCAL VOC 07+12

<table>
<thead>
<tr>
<th>训练精度</th>
<th>loss scaling策略</th>
<th>mAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP32</td>
<td>NO</td>
<td>77.46（baseline）</td>
</tr>
<tr>
<td>Mixed precision</td>
<td>NO</td>
<td>diverge</td>
</tr>
<tr>
<td></td>
<td>constant (=64)</td>
<td>77.63</td>
</tr>
<tr>
<td></td>
<td>auto scaling</td>
<td>77.42</td>
</tr>
</tbody>
</table>
Experimental results

- Small NMT on **WMT German-English**
  - Encoder: 2 layers
  - Decoder: 2 layers with attention

<table>
<thead>
<tr>
<th>Training Precision</th>
<th>Scaling Strategy</th>
<th>BLEU</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP32</td>
<td>NO</td>
<td>23.68</td>
</tr>
<tr>
<td>Mixed-precision</td>
<td>Auto scaling</td>
<td>23.79</td>
</tr>
</tbody>
</table>
• PGAN (Progressive growth of GAN)

PGAN

• G loss

PGAN

- Generation results (cifar10 dataset)

<table>
<thead>
<tr>
<th>Exp.</th>
<th>fp32</th>
<th>mp-auto-scaling</th>
<th>mp-no-scaling</th>
</tr>
</thead>
<tbody>
<tr>
<td>sliced_wasserstein</td>
<td>9.3764</td>
<td>9.1662</td>
<td>7.9601</td>
</tr>
</tbody>
</table>
Font Generation

Pyramid Embedded Generative Adversarial Network for Automated Font Generation
Font Generation

• G loss
Font Generation

- Generation results (金陵刻经体)

<table>
<thead>
<tr>
<th>fp32</th>
<th>mp-no-scaling</th>
<th>mp-auto-scaling</th>
</tr>
</thead>
<tbody>
<tr>
<td>培幕</td>
<td>培幕</td>
<td>培幕</td>
</tr>
<tr>
<td>慢族</td>
<td>慢族</td>
<td>慢族</td>
</tr>
<tr>
<td>漬痞</td>
<td>漬痞</td>
<td>漬痞</td>
</tr>
<tr>
<td>签脉</td>
<td>签脉</td>
<td>签脉</td>
</tr>
<tr>
<td>菩蕞</td>
<td>菩蕞</td>
<td>菩蕞</td>
</tr>
<tr>
<td>遵酌</td>
<td>遵酌</td>
<td>遵酌</td>
</tr>
</tbody>
</table>
Wide & Deep Learning

- Predict the probability that the individual has an annual income of over 50,000 dollars

Figure 1: The spectrum of Wide & Deep models.
Wide & Deep Learning

• Loss

<table>
<thead>
<tr>
<th>Exp</th>
<th>fp32</th>
<th>mp-no-scaling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>84.31%</td>
<td>84.27%</td>
</tr>
</tbody>
</table>

Wide & Deep Learning for Recommender Systems
More try: small inputs (for normalization layers)

- Underflow in FP16 gradients
  - Design the model to be more adaptive to FP16 representation
  - Move the gradient itself into the FP16 representable range
    - Especially the activation gradients

- Batch normalization

\[
\begin{align*}
\text{Input: } & \text{ Values of } x \text{ over a mini-batch: } B = \{x_1, \ldots, x_m\}; \\
& \text{Parameters to be learned: } \gamma, \beta \\
\text{Output: } & \{y_i = BN_{\gamma, \beta}(x_i)\}
\end{align*}
\]

\[
\begin{align*}
\mu_B &= \frac{1}{m} \sum_{i=1}^{m} x_i & \text{// mini-batch mean} \\
\sigma_B^2 &= \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_B)^2 & \text{// mini-batch variance} \\
\hat{x}_i &= \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} & \text{// normalize} \\
y_i &= \gamma \hat{x}_i + \beta \equiv BN_{\gamma, \beta}(x_i) & \text{// scale and shift}
\end{align*}
\]

- \textbf{BN} will stand for Batch Norm.
- \(f\) represents a layer upwards of the BN one.
- \(y\) is the linear transformation which scales \(x\) by \(\gamma\) and adds \(\beta\).
- \(\hat{x}\) is the normalized inputs.
- \(\mu\) is the batch mean.
- \(\sigma^2\) is the batch variance.
Small input

• Derivatives of BN layer

\[
\frac{\partial f}{\partial x_i} = \frac{m \frac{\partial f}{\partial \hat{x}_i} - \sum_{j=1}^{m} \frac{\partial f}{\partial \hat{x}_j} - \hat{x}_i \sum_{j=1}^{m} \frac{\partial f}{\partial \hat{x}_j} \cdot \hat{x}_j}{m \sqrt{\sigma^2 + \epsilon}}
\]

• Reduce the magnitude of the inputs
  • Reduce magnitude of the forward activations, so as to reduce the overflow in forward propagation when using FP16
  • Improve the magnitude of the derivatives

• Tips for Network with BN:
  • Normalize the layer to have std to be 1/S rather than 1.0
Small inputs

- ResNet32+CIFAR10
  - Activations and the gradients

activations

activation gradients
Small inputs

• ResNet32+CIFAR10
  • All without loss scaling

<table>
<thead>
<tr>
<th>精度</th>
<th>Input scale</th>
<th>准确度</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP32</td>
<td>1.0</td>
<td>93.58</td>
</tr>
<tr>
<td>Mixed-precision</td>
<td>1.0</td>
<td>93.41</td>
</tr>
<tr>
<td>Mixed-precision</td>
<td>1/8.0</td>
<td>93.43</td>
</tr>
</tbody>
</table>
Small inputs

- SSD on PASCAL VOC 07+12
- Activations and the gradients
Small inputs

- SSD on PASCAL VOC 07+12
  - Activations and the gradients

<table>
<thead>
<tr>
<th>训练精度</th>
<th>loss scaling策略</th>
<th>mAP (standard input)</th>
<th>mAP (small input, scaled 1/64)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP32</td>
<td>NO</td>
<td>77.46 (baseline)</td>
<td>77.25</td>
</tr>
<tr>
<td>Mixed precision</td>
<td>NO</td>
<td>diverge</td>
<td>77.75</td>
</tr>
<tr>
<td></td>
<td>constant (=64)</td>
<td>77.63</td>
<td>77.56</td>
</tr>
<tr>
<td></td>
<td>auto scaling</td>
<td>77.42</td>
<td>77.17</td>
</tr>
</tbody>
</table>
Conclusion

• Mixed-precision tools have been supported on PAI-tensorflow
• More effort is still conducted to explore more in mixed-precision
  • More precision supported
  • More training strategy
Thank you