MIXED PRECISION TRAINING FOR CONVOLUTIONAL TENSOR-TRAIN LSTM

Wonmin Byeon, August 23, 2020
APPLICATIONS

Spatio-Temporal Learning

Early Activity Recognition

Video Prediction

Table 2: Early activity recognition on the Something-Something V2 dataset using 41 categories as \[17]. (*) indicates the result by \[17].
CONVOLUTIONAL LSTM

$C(t-1)$

$H(t-1)$

$X(t)$

$P$

$W$

$LSTM$

$tanh$

Addition

Convolution

Element-wise product

Sigmoid

Hyperbolic tangent
CONVOLUTIONAL TENSOR-TRAIN LSTM
OPTIMIZATION TRICKS

• Speed-up tricks to train Convolutional LSTM/Convolutional Tensor-Train LSTM
  
  #1. Enabling mixed precision training using NVIDIA AMP

  #2. GPU Optimization: Fused Adam, Fused kernels in LSTM cell, Thread affinity binding

  #3. LSTM Activation Checkpointing

• Fast convergence without performance change
• Model parallelism using multi-streams
#1. ENABLING AMP

```python
from apex import amp

model, optimizer = amp.initialize(model, optimizer, opt_level="O1")

with amp.scale_loss(loss, optimizer) as scaled_loss:
    scaled_loss.backward()
if gradient_clipping:
    grad_norm = torch.nn.utils.clip_grad_norm_(amp.master_params(optimizer), clipping_threshold)

# save the checkpoint
checkpoint_info['amp'] = amp.state_dict()

# load the checkpoint
amp.load_state_dict(checkpoint['amp'])
```
#2. GPU Optimization

- **Fused Adam**: Adam optimizer

  ```python
  from apex.optimizers import FusedAdam
  optimizer = FusedAdam(model.parameters(), lr = learning_rate, eps = eps)
  ```
#2. GPU Optimization

- Fused optimizer: Adam optimizer

```python
from apex.optimizers import FusedAdam

optimizer = FusedAdam(model.parameters(), lr=learning_rate, eps=eps)
```

- Fused kernel with JIT: LSTM cell

```python
@torch.jit.script
def fuse_mul_add_mul(f, cell_states, i, g):
    return f * cell_states + i * g

self.cell_states = f * self.cell_states + i * g
self.cell_states = fuse_mul_add_mul(f, self.cell_states, i, g)
```
#2. GPU Optimization

- **Fused optimizer: Adam optimizer**
  ```python
  from apex.optimizers import FusedAdam
  optimizer = FusedAdam(model.parameters(), lr=learning_rate, eps=eps)
  ```

- **Fused kernel with JIT: LSTM cell**
  ```python
  @torch.jit.script
def fuse_mul_add_mul(f, cell_states, i, g):
      return f * cell_states + i * g
  
  self.cell_states = f * self.cell_states + i * g
  ```

- **Thread affinity binding: distributed computing**
  ```python
  from utils.gpu_affinity import set_affinity
  set_affinity(args.local_rank)
  ```
Application: video prediction
Machine: V100 x 8, 16GB
Batch Size: 16 videos
12 Conv. LSTM layers
Input/output image resolution: 128x128

TRAINING

Convolutional LSTM

No performance change!

Convergence time

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>+Optimization</th>
<th>+AMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hours</td>
<td>172</td>
<td>94</td>
<td>26</td>
</tr>
</tbody>
</table>

6.6x faster

GPU Memory

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>+Optimization</th>
<th>+AMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>GB</td>
<td>15</td>
<td>15</td>
<td>10</td>
</tr>
</tbody>
</table>

1.5x less
Application: video prediction
Machine: V100 x 8, 16GB
Batch Size: 16 videos
12 Conv. LSTM layers
Input/output image resolution: 128x128

TRAINING

Convolutional Tensor-Train LSTM

No performance change!

Convergence time

- Baseline: 184 hours
- +Opt.&AMP: 33 hours
- 5.5x faster

GPU Memory

- Baseline: 18 GB
- +Opt.&AMP: 12 GB
- 1.5x less
#3. ACTIVATION CHECKPOINTING

```python
from torch.utils.checkpoint import checkpoint

def chkpt_blk(cc_i, cc_f, cc_o, cc_g, cell_states):
    i = torch.sigmoid(cc_i)
    f = torch.sigmoid(cc_f)
    o = torch.sigmoid(cc_o)
    g = torch.tanh(cc_g)

    cell_states = fuse_mul_add_mul(f, cell_states, i, g)
    outputs = o * torch.tanh(cell_states)

    return outputs, cell_states

outputs, self.cell_states = checkpoint(chkpt_blk, cc_i, cc_f, cc_o, cc_g, self.cell_states)
```
TRAINING

Convolutional LSTM

No performance change!

Convergence time

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>+Optimization</th>
<th>+AMP</th>
<th>+Checkpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hours</td>
<td>172</td>
<td>94</td>
<td>26</td>
<td>22</td>
</tr>
</tbody>
</table>

7.8x faster

GPU Memory

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>+Optimization</th>
<th>+AMP</th>
<th>+Checkpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>GB</td>
<td>15</td>
<td>15</td>
<td>10</td>
<td>8</td>
</tr>
</tbody>
</table>

1.8x less memory usage
Application: video prediction
Machine: V100 x 8, 16GB
Batch Size: 16 videos
12 Conv. LSTM layers
Input/output image resolution: 128x128

TRAINING
Conv. Tensor-Train LSTM

No performance change!

Convergence time

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>+Opt.&amp;AMP</th>
<th>+Checkpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hours</td>
<td>184</td>
<td>33</td>
<td>50</td>
</tr>
</tbody>
</table>

5.5x faster

GPU Memory

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>+Opt.&amp;AMP</th>
<th>+Checkpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>GB</td>
<td>18</td>
<td>12</td>
<td>8</td>
</tr>
</tbody>
</table>

2.3x less memory usage
MODEL PARALLEL: MULTI-STREAMS

- Use multiple cuda-streams
- Execute multiple kernels that do not have data dependency in parallel

example model:
MODEL PARALLEL: MULTI-STREAMS

- Use multiple cuda-streams
- Execute multiple kernels that do not have data dependency in parallel

example model:

Operations of layers at different branches are overlapped.
MODEL PARALLEL: MULTI-STREAMS

Computation time comparison

![Time comparison chart showing 2.8x faster with multi-stream compared to single-stream]
SUMMARY

Speed-up summary to train Convolutional Tensor-Train LSTM

- Enabling AMP
- GPU Optimization
- Activation Checkpointing
  - ConvLSTM: \(7.8x\) speed-up, \(1.8x\) less memory usage
  - Conv. Tensor-Train LSTM: \(5.5x\) speed-up, \(2.3x\) less memory usage.
- Fast convergence without performance change

Model parallelism using multi-streams
CONCLUSION


- Project page: https://sites.google.com/nvidia.com/conv-tt-lstm/home

- Code Optimization: Sangkug Lym (NVIDIA)

- The code with the optimization tricks will be available soon: https://github.com/NVlabs/conv-tt-lstm