In-Place Activated BatchNorm for Memory-Optimized Training of DNNs

Samuel Rota Bulò, Lorenzo Porzi, Peter Kontschieder
Mapillary Research

Code: https://github.com/mapillary/inplace_abn

CSC2548, 2018 Winter
Harris Chan
Jan 31, 2018
Overview

• Motivation for Efficient Memory management
• Related Works
  • Reducing precision
  • Checkpointing
  • Reversible Networks [9] (Gomez et al., 2017)
• In-Place Activated Batch Normalization
  • Review: Batch Normalization
  • In-place Activated Batch Normalization
• Experiments
• Future Directions
Overview

- Motivation for Efficient Memory management
- Related Works
  - Reducing precision
  - Checkpointing
  - Reversible Networks [9] (Gomez et al., 2017)
- In-Place Activated Batch Normalization
  - Review: Batch Normalization
  - In-place Activated Batch Normalization
- Experiments
- Future Directions
Why Reduce Memory Usage?

- Modern computer vision recognition models use deep neural networks to extract features
- Depth/width of networks ~ GPU memory requirements
  - Semantic segmentation: may even only do just a single crop per GPU during training due to suboptimal memory management
- More efficient memory usage during training lets you:
  - Train larger models
  - Use bigger batch size / image resolutions
- This paper focuses on increasing memory efficiency of the training process of deep network architectures at the expense of small additional computation time
Approaches to Reducing Memory

Reduce Memory by...

- Increasing Computation Time
- Reducing Precision (& Accuracy)
Overview

• Motivation for Efficient Memory management
• Related Works
  • Reducing precision
  • Checkpointing
  • Reversible Networks [9] (Gomez et al., 2017)
• In-Place Activated Batch Normalization
  • Review: Batch Normalization
  • In-place Activated Batch Normalization
• Experiments
• Future Directions
## Related Works: Reducing Precision

<table>
<thead>
<tr>
<th>Work</th>
<th>Weight</th>
<th>Activation</th>
<th>Gradients</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BinaryConnect</strong> (M. Courbariaux et al. 2015)</td>
<td>Binary</td>
<td>Full Precision</td>
<td>Full Precision</td>
</tr>
<tr>
<td><strong>Binarized neural networks</strong> (I. Hubara et al. 2016)</td>
<td>Binary</td>
<td>Binary</td>
<td>Full Precision</td>
</tr>
<tr>
<td><strong>Quantized neural networks</strong> (I. Hubara et al)</td>
<td>Quantized 2,4,6 bits</td>
<td>Quantized 2,4,6 bits</td>
<td>Full Precision</td>
</tr>
<tr>
<td><strong>Mixed precision training</strong> (P. Micikevicius et al. 2017)</td>
<td>Half Precision (fwd/bw) &amp; Full Precision (master weights)</td>
<td>Half Precision</td>
<td>Half Precision</td>
</tr>
</tbody>
</table>
Related Works: Reducing Precision

- **Idea:** During training, lower the precision (up to binary) of the weights / activations / gradients

<table>
<thead>
<tr>
<th>Strength</th>
<th>Weakness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reduce memory requirement and size of the model</td>
<td>Often decrease in accuracy performance (newer work attempts to address this)</td>
</tr>
<tr>
<td>Less power: efficient forward pass</td>
<td></td>
</tr>
<tr>
<td>Faster: 1-bit XNOR-count vs. 32-bit floating point multiply</td>
<td></td>
</tr>
</tbody>
</table>
Related Works: Computation Time

- **Checkpointing**: trade off memory with computation time

- **Idea**: During backpropagation, store a subset of activations (“checkpoints”) and recompute the remaining activations as needed

- Depending on the architecture, we can use different strategies to figure out which subsets of activations to store
Related Works: Computation Time

• Let $L$ be the number of identical feed-forward layers:

<table>
<thead>
<tr>
<th>Work</th>
<th>Spatial Complexity</th>
<th>Computation Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive</td>
<td>$O(L)$</td>
<td>$O(L)$</td>
</tr>
<tr>
<td>Checkpointing (Martens and Sutskever, 2012)</td>
<td>$O(\sqrt{L})$</td>
<td>$O(L)$</td>
</tr>
<tr>
<td>Recursive Checkpointing (T. Chen et al., 2016)</td>
<td>$O(\log L)$</td>
<td>$O(L \log L)$</td>
</tr>
<tr>
<td>Reversible Networks (Gomez et al., 2017)</td>
<td>$O(1)$</td>
<td>$O(L)$</td>
</tr>
</tbody>
</table>

Related Works: Computation Time
Reversible ResNet (Gomez et al., 2017)

Basic Residual Function

Idea: Reversible Residual module allows the current layer’s activation to be reconstructed exactly from the next layer’s. No need to store any activations for backpropagation!

Related Works: Computation Time
Reversible ResNet (Gomez et al., 2017)

• No noticeable loss in performance
• Gains in network depth: ~600 vs ~100
• 4x increase in batch size (128 vs 32)

<table>
<thead>
<tr>
<th>Advantage</th>
<th>Disadvantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Runtime cost: 1.5x of normal training (sometimes less in practice)</td>
<td></td>
</tr>
<tr>
<td>• Restrict reversible blocks to have a stride of 1 to not discard information (i.e. no bottleneck layer)</td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Classification error on CIFAR

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ResNet</td>
<td>RevNet</td>
</tr>
<tr>
<td>32 (38)</td>
<td>7.14%</td>
<td>7.24%</td>
</tr>
<tr>
<td>110</td>
<td>5.74%</td>
<td>5.76%</td>
</tr>
<tr>
<td>164</td>
<td>5.24%</td>
<td>5.17%</td>
</tr>
</tbody>
</table>

Table 4: Top-1 classification error on ImageNet (single crop)

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Error Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-101</td>
<td>23.01%</td>
</tr>
<tr>
<td>RevNet-104</td>
<td>23.10%</td>
</tr>
</tbody>
</table>

Overview

- Motivation for Efficient Memory management
- Related Works
  - Reducing precision
  - Checkpointing
  - Reversible Networks [9] (Gomez et al., 2017)
- In-Place Activated Batch Normalization
  - Review: Batch Normalization
  - In-place Activated Batch Normalization
- Experiments
- Future Directions
Review: Batch Normalization (BN)

• Apply BN on current features ($x_i$) across the mini-batch
• Helps reduce internal covariate shift & accelerate training process
• Less sensitive to initialization

---

**Algorithm 1:** Batch Normalizing Transform, applied to activation $x$ over a mini-batch.

Memory Optimization Strategies

- Let’s compare the various strategies for BN+Act:
  1. Standard
  2. Checkpointing (baseline)
  3. Checkpointing (proposed)
  4. In-Place Activated Batch Normalization I
  5. In-Place Activated Batch Normalization II
1: Standard BN Implementation

(a) Standard building block (memory-inefficient)
Gradients for Batch Normalization

\[
\frac{\partial l}{\partial \hat{x}_i} = \frac{\partial l}{\partial y_i} \cdot \gamma
\]

\[
\frac{\partial l}{\partial \sigma_B^2} = \sum_{i=1}^{m} \frac{\partial l}{\partial \hat{x}_i} \cdot (x_i - \mu_B) \cdot \frac{-1}{2} (\sigma_B^2 + \epsilon)^{-3/2}
\]

\[
\frac{\partial l}{\partial \mu_B} = \sum_{i=1}^{m} \frac{\partial l}{\partial \hat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_B^2 + \epsilon}}
\]

\[
\frac{\partial l}{\partial x_i} = \frac{\partial l}{\partial \hat{x}_i} \cdot \frac{1}{\sqrt{\sigma_B^2 + \epsilon}} + \frac{\partial l}{\partial \sigma_B^2} \cdot \frac{2(x_i - \mu_B)}{m} + \frac{\partial l}{\partial \mu_B} \cdot \frac{1}{m}
\]

\[
\frac{\partial l}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial l}{\partial y_i} \cdot \hat{x}_i
\]

\[
\frac{\partial l}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial l}{\partial y_i}
\]

2: Checkpointing (baseline)

(b) Checkpointing [4, 21]
3: Checkpointing (Proposed)

\[
\frac{\partial L}{\partial x_i} = \left\{ \frac{\partial L}{\partial y_i} - \frac{1}{m} \frac{\partial L}{\partial \gamma} \hat{x}_i - \frac{1}{m} \frac{\partial L}{\partial \beta} \right\} \frac{\gamma}{\sqrt{\sigma^2_B + \epsilon}}
\]

\[
\frac{\partial L}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial L}{\partial y_i} \hat{x}_i, \quad \frac{\partial L}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial L}{\partial y_i}
\]
In-Place ABN

- Fuse batch norm and activation layer to enable in-place computation, using only a single memory buffer to store results.
- Encapsulation makes it easy to implement and deploy
- Implemented INPLACE ABN-I layer in PyTorch as a new module
4: In-Place ABN I (Proposed)

Invertible Activation Function

(d) In-Place Activated Batch Normalization I (proposed method)
Leaky ReLU is Invertible

Figure 3. LEAKY RELU with slope $a$ (left) and its inverse (right).
5: In-Place ABN II (Proposed)

(e) In-Place Activated Batch Normalization II (proposed method)

\[
\frac{\partial L}{\partial x_i} = \left[ \frac{\partial L}{\partial y_i} - \frac{1}{\gamma m} \frac{\partial L}{\partial \gamma} y_i \right] - \frac{1}{m} \left( \frac{\partial L}{\partial \beta} + \frac{\beta}{\gamma} \frac{\partial L}{\partial \gamma} \right) \frac{\gamma}{\sqrt{\sigma_\beta^2 + \epsilon}} \\
\frac{\partial L}{\partial \gamma} = \frac{1}{\gamma} \left[ \sum_{j=1}^{m} \frac{\partial L}{\partial y_j} y_j - \beta \frac{\partial L}{\partial \beta} \right]
\]
## Strategies Comparisons

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Store</th>
<th>Computation Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard</td>
<td>$x, z, \sigma_B, \mu_B$</td>
<td>-</td>
</tr>
<tr>
<td>Checkpointing</td>
<td>$x, \sigma_B, \mu_B$</td>
<td>$BN_{\gamma,\beta}, \phi$</td>
</tr>
<tr>
<td>Checkpointing (proposed)</td>
<td>$x, \sigma_B$</td>
<td>$\pi_{\gamma,\beta}, \phi$</td>
</tr>
<tr>
<td>In-Place ABN I (proposed)</td>
<td>$z, \sigma_B$</td>
<td>$\phi^{-1}, \pi^{-1}_{\gamma,\beta}$</td>
</tr>
<tr>
<td>In-Place ABN II (proposed)</td>
<td>$z, \sigma_B$</td>
<td>$\phi^{-1}$</td>
</tr>
</tbody>
</table>
In-Place ABN (Proposed)

Algorithm 1 INPLACE-ABN Forward

Require: $x, \gamma, \beta$
1: $y, \sigma_B \leftarrow \text{BN}_{\gamma, \beta}(x)$
2: $z \leftarrow \phi(y)$
3: save for backward $z, \sigma_B$
4: return $z$

Algorithm 2 INPLACE-ABN Backward

Require: $\frac{\partial L}{\partial z}, \gamma, \beta$
1: $z, \sigma_B \leftarrow$ saved tensors during forward
2: $\frac{\partial L}{\partial y} \leftarrow \phi_{\text{backward}}(z, \frac{\partial L}{\partial z})$
3: $y \leftarrow \phi^{-1}(z)$
4: if INPLACE-ABN I (see Fig. 2(d)) then
5: $\hat{x} \leftarrow \pi_{\gamma, \beta}^{-1}(y)$
6: $\frac{\partial L}{\partial x}, \frac{\partial L}{\partial \gamma}, \frac{\partial L}{\partial \beta} \leftarrow \text{BN}^*_\gamma, \beta(\hat{x}, \frac{\partial L}{\partial y}, \sigma_B)$
7: else if INPLACE-ABN II (see Fig. 2(e)) then
8: $\frac{\partial L}{\partial x}, \frac{\partial L}{\partial \gamma}, \frac{\partial L}{\partial \beta} \leftarrow \text{BN}^\dagger_{\gamma, \beta}(y, \frac{\partial L}{\partial y}, \sigma_B)$
9: return $\frac{\partial L}{\partial x}, \frac{\partial L}{\partial \gamma}, \frac{\partial L}{\partial \beta}$
## In-Place ABN (Proposed)

<table>
<thead>
<tr>
<th>Strength</th>
<th>Weakness</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Reduce memory</strong> requirement by half compared to standard; same savings as check pointing</td>
<td>Requires invertible activation function</td>
</tr>
<tr>
<td>Empirically <strong>faster than naïve checkpointing</strong></td>
<td>...but still slower than standard (memory hungry) implementation.</td>
</tr>
<tr>
<td>Encapsulating BN &amp; Activation together makes it easy to implement and deploy (plug &amp; play)</td>
<td></td>
</tr>
</tbody>
</table>
Overview

• Motivation for Efficient Memory management
• Related Works
  • Reducing precision
  • Checkpointing
  • Reversible Networks [9] (Gomez et al., 2017)
• In-Place Activated Batch Normalization
  • Review: Batch Normalization
  • In-place Activated Batch Normalization

• Experiments
• Future Directions
Experiments: Overview

• 3 Major types:
  • Performance on: (1) Image Classification, (2) Semantic Segmentation
  • (3) Timing Analysis compared to standard / checkpointing

• Experiment Setup:
  • NVIDIA Titan Xp (12 GB RAM/GPU)
  • PyTorch
  • Leaky ReLU activation
# Experiments: Image Classification

<table>
<thead>
<tr>
<th></th>
<th>ResNeXt-101/ResNeXt-152</th>
<th>WideResNet-38</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset</td>
<td>ImageNet-1k</td>
<td>ImageNet-1k</td>
</tr>
<tr>
<td>Description</td>
<td>Bottleneck residual units are replaced with a multi-branch version = “cardinality” of 64</td>
<td>More feature channels but shallower</td>
</tr>
<tr>
<td>Data Augmentation</td>
<td>Scale smallest side = 256 pixels then randomly crop 224 × 224, per-channel mean and variance normalization</td>
<td>(Same as ResNeXt-101/152)</td>
</tr>
</tbody>
</table>
| Optimizer      | • SGD with Nesterov Updates  
• Initial learning rate=0.1  
• weight decay=10^{-4}  
• momentum=0.9  
• 90 Epoch, reduce by factor of 10 per 30 epoch | • (Same as ResNeXt)  
• 90 Epoch, linearly decreasing from 0.1 to 10^{-6} |
Experiments: Leaky ReLU impact

<table>
<thead>
<tr>
<th>Network</th>
<th>activation</th>
<th>224² center</th>
<th>224² 10-crops</th>
<th>320² center</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>training validation</td>
<td>top-1</td>
<td>top-5</td>
<td>top-1</td>
</tr>
<tr>
<td>ResNeXt-101</td>
<td>ReLU</td>
<td>77.74</td>
<td>93.86</td>
<td>79.21</td>
</tr>
<tr>
<td>ResNeXt-101</td>
<td>ReLU</td>
<td>76.88</td>
<td>93.42</td>
<td>78.74</td>
</tr>
<tr>
<td>ResNeXt-101</td>
<td>Leaky ReLU</td>
<td>77.04</td>
<td>93.50</td>
<td>78.72</td>
</tr>
<tr>
<td>ResNeXt-101</td>
<td>Leaky ReLU</td>
<td>76.81</td>
<td>93.53</td>
<td>78.46</td>
</tr>
</tbody>
</table>

Table 1. Imagenet validation set results using ResNeXt-101 and ReLU/Leaky ReLU exchanged activation functions during training and validation.

- Using Leaky ReLU performs slightly worse than with ReLU
- Within ~1%, except for 320² center crop—authors argued it was due to non-deterministic training behaviour
- Weaknesses
  - Showing an average + standard deviation can be more convincing of the improvements.
Experiments: Exploiting Memory Saving

1) Larger Batch Size
2) Deeper Network
3) Larger Network
4) Sync BN

Baseline

<table>
<thead>
<tr>
<th>Network</th>
<th>Batch Size</th>
<th>$224^2$ center</th>
<th>$224^2$ 10-crops</th>
<th>$320^2$ center</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNeXt-101, STD-BN</td>
<td>256</td>
<td>77.04</td>
<td>93.50</td>
<td>77.92</td>
</tr>
<tr>
<td>ResNeXt-101, INPLACE-ABN</td>
<td>512</td>
<td>78.08</td>
<td>93.79</td>
<td>79.38</td>
</tr>
<tr>
<td>ResNeXt-152, INPLACE-ABN</td>
<td>256</td>
<td>78.28</td>
<td>94.04</td>
<td>79.56</td>
</tr>
<tr>
<td>WideResNet-38, INPLACE-ABN</td>
<td>256</td>
<td>79.72</td>
<td>94.78</td>
<td>80.69</td>
</tr>
<tr>
<td>ResNeXt-101, INPLACE-ABN sync</td>
<td>256</td>
<td>77.70</td>
<td>93.78</td>
<td>78.98</td>
</tr>
</tbody>
</table>

Table 2. Imagenet validation set results using different architectures and training batch sizes.

- Performance increase for 1-3
- Similar performance with larger batch size vs deeper model (1 vs 2)
- Synchronized INPLACE-ABN did not increase the performance that much
Experiments: Semantic Segmentation

- **Semantic Segmentation**: Assign categorical labels to each pixel in an image

- **Datasets**
  - CityScapes
  - COCO-Stuff
  - Mapillary Vistas

Figure Credit: [https://www.cityscapes-dataset.com/examples/](https://www.cityscapes-dataset.com/examples/)
Experiments: Semantic Segmentation

• Architecture contains 2 parts that are jointly fine-tuned on segmentation data:
  • **Body**: Classification models pre-trained on ImageNet
  • **Head**: Segmentation specific architectures

• Authors used **DeepLabV3** as the head
  • Cascaded atrous (dilated) convolutions for capturing contextual info
  • Crop-level features encoding global context

• Maximize GPU Usage by:
  • **(FIXED CROP)** fixing the training crop size and therefore pushing the amount of crops per minibatch to the limit
  • **(FIXED BATCH)** fixing the number of crops per minibatch and maximizing the training crop resolutions

Experiments: Semantic Segmentation

- More training data (**FIXED CROP**) helps a little bit
- Higher input resolution (**FIXED BATCH**) helps even more than adding more crops
- No qualitative result: probably visually similar to DeepLabV3

### Table 3

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Std-BN + Leaky ReLU</td>
<td>16 × 512²</td>
<td>74.42</td>
<td>16 × 480²</td>
<td>20.30</td>
</tr>
<tr>
<td>InPlace-ABN, FIXED CROP</td>
<td>28 × 512²</td>
<td>75.80</td>
<td>24 × 480²</td>
<td>22.63</td>
</tr>
<tr>
<td>InPlace-ABN, FIXED BATCH</td>
<td>16 × 672²</td>
<td>77.04</td>
<td>16 × 600²</td>
<td>23.35</td>
</tr>
<tr>
<td>InPlace-ABN-sync, FIXED BATCH</td>
<td>16 × 672²</td>
<td><strong>77.58</strong></td>
<td>16 × 600²</td>
<td><strong>24.91</strong></td>
</tr>
</tbody>
</table>

Validation data results (single scale test) for semantic segmentation experiments on Cityscapes and COCO-Stuff, using ResNeXt-101 and WideResNet-38 network bodies and different batch normalization settings (see text). All result numbers in [%].
Experiments: Semantic Segmentation Fine-Tuned on CityScapes and Mapillary Vistas

<table>
<thead>
<tr>
<th></th>
<th>Cityscapes</th>
<th>Mapillary Vistas</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ResNeXt-152</td>
<td>WideResNet-38</td>
</tr>
<tr>
<td>INPLACE-ABNsync</td>
<td>12 × 680²</td>
<td>78.49</td>
</tr>
<tr>
<td>INPLACE-ABN</td>
<td>–</td>
<td>16 × 712² 78.45</td>
</tr>
<tr>
<td>INPLACE-ABNsync</td>
<td>–</td>
<td>16 × 712² 79.02</td>
</tr>
<tr>
<td>INPLACE-ABNsync</td>
<td>–</td>
<td>12 × 872² 79.16</td>
</tr>
<tr>
<td>INPLACE-ABNsync + CLASS-UNIFORM SAMPLING</td>
<td>–</td>
<td>12 × 872² 79.40</td>
</tr>
<tr>
<td>Mapillary Vistas</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSUN 2017 winner [35] (based on PSPNet)</td>
<td>ResNet-101</td>
<td></td>
</tr>
<tr>
<td>PSPNet + auxiliary loss</td>
<td>16 × 713²</td>
<td>49.76</td>
</tr>
<tr>
<td>+ Hybrid dilated convolutions [29]</td>
<td>16 × 713²</td>
<td>50.28</td>
</tr>
<tr>
<td>+ Inverse frequency label reweighting</td>
<td>16 × 713²</td>
<td>51.50</td>
</tr>
<tr>
<td>+ Cityscapes pretraining</td>
<td>16 × 713²</td>
<td>51.59</td>
</tr>
</tbody>
</table>

Table 4. Validation data results (single scale test, no horizontal flipping) for semantic segmentation experiments on Cityscapes and Vistas, using ResNeXt-152 and WideResNet-38 bodies with different settings for #crops per minibatch and crop sizes. All results in [%].

- Combination of INPLACE-ABN sync with larger crop sizes improves by ≈ 0.9% over the best performing setting in Table 3
- **Class- Uniform sampling:** Class-uniformly sampled from eligible image candidates, making sure to take training crops from areas containing the class of interest.
Experiments: Semantic Segmentation

- Currently state of the art for CityScapes for IoU class and iIoU (instance) Class
  - **iIoU**: Weighting the contribution of each pixel by the ratio of the class’ average instance size to the size of the respective ground truth instance.
Experiments: Timing Analyses

• They isolated a single BN+ACT+CONV block & evaluate the computational times required for a forward and backward pass

• **Result:** Narrowed the gap between standard vs checkpointing by half

• Ensured fair comparison by re-implementing checkpointing in PyTorch
Overview

• Motivation for Efficient Memory management

• Related Works
  • Reducing precision
  • Checkpointing
  • Reversible Networks [9] (Gomez et al., 2017)

• In-Place Activated Batch Normalization
  • Review: Batch Normalization
  • In-place Activated Batch Normalization

• Experiments

• Future Directions
Future Directions:

• Apply INPLACE-ABN in other...
  • **Architectures**: DenseNet, Squeeze-Excitation Networks, Deformable Convolutional Networks
  • **Problem Domains**: Object detection, instance-specific segmentation, 3D data learning

• Combine INPLACE-ABN with other memory reduction techniques, ex: Mixed precision training

• Apply same InPlace idea on ’newer’ Batch Norm, ex: Batch Renormalization*

---

Links and References

• Official Github code (PyTorch): https://github.com/mapillary/inplace_abn
• CityScapes Dataset: https://www.cityscapes-dataset.com/benchmarks/#scene-labeling-task

• Reduced Precision:
  • BinaryConnect: https://arxiv.org/abs/1511.00363
  • Binarized Networks: https://arxiv.org/abs/1602.02830
  • Mixed Precision Training: https://arxiv.org/abs/1710.03740

• Trade off with Computation Time
  • Recursive Checkpointing: https://arxiv.org/abs/1604.06174
  • Reversible Networks: https://arxiv.org/abs/1707.04585