Gist: Efficient Data Encoding for Deep Neural Network Training

July 2018
Space is limited

80 GB

Maximum GPU size
But Neural Networks keep demanding
Why do we care?

AI is going to be everywhere

- Education
- Health care
- Entertainment
- Transportation
- Communication
Why do we care?

AI is going to be everywhere

Education | Health care | Entertainment | Transportation | Communication
Why do we care?

AI is going to be everywhere

- Education
- Health care
- Entertainment
- Transportation
- Communication

CNNs
Why is this hard?
Why is this hard?

1. No previous analysis
Why is this hard?

1. No previous analysis
2. No previous solution
What do we do about space?
What do we do about space?

We buy more compute power!
What do we do about space?

We buy more compute power!
What do we do about space?

We buy more compute power!

What if we have limited machines?
What do we do about space?

We buy more compute power!

What if we have limited machines?

Compromise cost
What do we do about space?

We buy more compute power!

What if we have limited machines?

Compromise cost
Compromise Communicate time
What do we do about space?

We buy more compute power!

What if we have limited machines?

Compromise cost
Compromise Communicate time
What do we do about space?

We buy more compute power!

What if we have large models?

What if we have limited machines?

Compromise cost
Compromise Communicate time
What do we do about space?

We buy more compute power!

What if we have large models?

Compromise batch size

What if we have limited machines?

Compromise cost
Compromise Communicate time
What do we do about space?

We buy more compute power!

What if we have large models?

- Compromise batch size

What if we have limited machines?

- Compromise cost
- Compromise Communicate time

ZeRo

MP

DP
What do we do about space?

We buy more compute power!

What if we have large models?

What if we still need more space?

What if we have limited machines?

Compromise batch size

Compromise cost

Compromise Communicate time
What do we do about space?

We buy more compute power!

What if we have large models?
Compromise batch size.

What if we have limited machines?
Compromise cost; Compromise Communicate time.

What if we still need more space?
Then we optimize.
But How
What if
What if We Encode
What if

We Encode

Encoding is good
What if

We Encode

Encoding is good

- Compression
- Fast
What if We Encode

Encoding is good
- Compression
- Fast

What do we encode?
What is taking up all the space?
What is taking up all the space?

$x \xrightarrow{w_1} \cdot \xrightarrow{}$
What is taking up all the space?

\[ \mathcal{X} \xrightarrow{w_1} a_1 = xw_1 \]
What is taking up all the space?

\[ x \overset{w_1}{\rightarrow} a_1 = xw_1 \quad z_1 = ReLu(a) \quad \]
What is taking up all the space?

\[ x \xrightarrow{w_1} a_1 = xw_1 \xrightarrow{z_1 = \text{ReLU}(a)} w_2 \]
What is taking up all the space?

\[ \mathcal{X} \xrightarrow{w_1} a_1 = xw_1 \xrightarrow{z_1 = \text{ReLU}(a)} w_2 \xrightarrow{a_2 = w_2z_1} \]
What is taking up all the space?
What is taking up all the space?
What is taking up all the space?

Permanent Values
What is taking up all the space?

\[ a_1 = x w_1 \]

\[ z_1 = ReLu(a) \]

\[ a_2 = w_2 z_1 \]

\[ z_2 = ReLu(a_2) \]

\[ \hat{y} \]

Permanent Values

Imm. Consumed
What is taking up all the space?

Permanent Values

What about these?

Imm. Consumed

$X$ $\rightarrow$ $W_1$ $\rightarrow$ $a_1 = xw_1$ $\rightarrow$ $z_1 = ReLu(a)$ $\rightarrow$ $W_2$ $\rightarrow$ $a_2 = w_2z_1$ $\rightarrow$ $z_2 = ReLu(a_2)$ $\rightarrow$ $\hat{y}$
The backward pass

\[ x \xrightarrow{w_1} a_1 = xw_1 \xrightarrow{z_1 = \text{ReLU}(a)} \xrightarrow{w_2} a_2 = w_2z_1 \xrightarrow{z_2 = \text{ReLU}(a_2)} \hat{y} \]
The backward pass

\[ a_1 = x w_1 \quad z_1 = ReLu(a) \quad w_2 \quad a_2 = w_2 z_1 \quad z_2 = ReLu(a_2) \quad \hat{y} \]

\[ \frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \frac{\partial z_2}{\partial w_2} \]
The backward pass

\[ a_1 = x w_1 \]
\[ z_1 = \text{ReLU}(a) \]
\[ w_2 \]
\[ a_2 = w_2 z_1 \]
\[ z_2 = \text{ReLU}(a_2) \]
\[ \hat{y} \]

\[ \frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \frac{\partial z_2}{\partial w_2} \]
The backward pass

\[ \frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_2} \]

\[ \frac{\partial z_2}{\partial w_2} = z_2 > 0? \frac{\partial a_2}{\partial w_2} : 0 \]
The backward pass

\[ \frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_2} \]

\[ \frac{\partial z_2}{\partial w_2} = z_2 > 0 \text{?} \frac{\partial a_2}{\partial w_2} : 0 \]
Where does memory allocation go?
Temporal GAP

T = i  \quad T = j
Temporal GAP

$\tilde{z}_1$ is created

$T = i$ to $T = j$
Temporal GAP

\[ \tilde{\gamma}_1 \] is created forward pass

\[ T = i \quad \text{and} \quad T = j \]

Time
Temporal GAP

\[ t_1 \] is created

forward pass

backward pass

\[ T = i \]

\[ T = j \]

Time
Temporal GAP

\( T = i \) is created

forward pass

backward pass

\( T = j \) is reused

\( \sim_1 \) is created

\( \sim_1 \) is reused

Time
Temporal GAP

$\Delta T = j - i$

$T = i$ is created

$T = j$ is reused
Temporal GAP + Encoding
Solution 1: vDNN - Stash away the problem

Time $T = i$ to $T = j$
Solution 1: vDNN - Stash away the problem

CPU

GPU

$\sim 1$ is created

$T = i$ $T = j$

Time
Solution 1: vDNN - Stash away the problem

\[ \tilde{Z}_1 \text{ is created} \]

\[ \tilde{Z}_1 \text{ is transferred to cpu} \]

\[ T = i \quad T = j \]
Solution 1: vDNN - Stash away the problem

- **GPU**
  - $\sim 1$ is created
  - T = i
  - T = j
  - forward pass

- **CPU**
  - $\sim 1$ is transferred to cpu

Time
Solution 1: vDNN - Stash away the problem

- **CPU**
  - $\mathcal{Z}_1$ is transferred to CPU

- **GPU**
  - $\mathcal{Z}_1$ is created
  - Forward pass
  - Backward pass
  - $T = i$ to $T = j$

Time flow from $T = i$ to $T = j$.
Solution 1: vDNN - Stash away the problem

GPU

T = i

is created
forward pass
backward pass

T = j

CPU

is transferred to cpu

is transferred to cpu

Time
Solution 1: vDNN - Stash away the problem

CPU

1 is transferred to cpu

GPU

1 is created
forward pass
backward pass
1 is reused

T = i

T = j

Time
Solution 1: vDNN - Stash away the problem

Why is this bad?
Solution 1: vDNN - Stash away the problem

Why is this bad?

Communication is a bottle neck
Solution 2: Reduce precisions
Solution 2: Reduce precisions

\[ x \xrightarrow{w_1} a_1 = xw_1 \]

\[ z_1 = \text{ReLU}(a) \]

\[ z_1' = \text{Reduce}(z_1) \]
Solution 2: Reduce precisions

\[ a_1 = xw_1 \]
\[ z_1 = \text{ReLU}(a) \]
\[ z_1 = \text{Reduce}(z_1) \]
\[ w_2 \]

\( \mathcal{X} \)
\( w_1 \)
Solution 2: Reduce precisions

\[ a_1 = xw_1 \]

\[ z_1 = \text{ReLU}(a) \]

\[ z_1 = \text{Reduce}(z_1) \]

\[ a_2 = z_1^*w_2 \]
Solution 2: Reduce precisions

\[ a_1 = x w_1 \]
\[ z_1 = \text{ReLU}(a) \]
\[ z_1' = \text{Reduce}(z_1) \]

\[ a_2 = z_1' w_2 \]
\[ z_2 = \text{ReLU}(a_2) \]
Solution 2: Reduce precisions

\[ x \xrightarrow{w_1} a_1 = xw_1 \quad z_1 = ReLu(a) \quad z_1^\ast = Reduce(z_1) \quad \xrightarrow{w_2} a_2 = z_1^\ast w_2 \quad z_2 = ReLu(a_2) \quad z_2^\ast = Reduce(z_2) \quad \xrightarrow{\hat{y}} \]
Solution 2: Reduce precisions

\[ x \xrightarrow{w_1} a_1 = xw_1 \xrightarrow{\text{Reduce}(z_1)} z_1 = \text{ReLU}(a_1) \xrightarrow{w_2} a_2 = z_1^*w_2 \xrightarrow{\text{Reduce}(z_2)} z_2 = \text{ReLU}(a_2) \xrightarrow{\hat{y}} \hat{y} \]

```python
>>> x = np.float32(262.223)
>>> y = np.float16(262.223)
>>> x, y
(262.223, 262.2)
```

Animesh Jain\textsuperscript{*}, Amar Phadke
Microsoft Research
animesh.jain@umich.edu, amar@microsoft.com

\textsuperscript{*}Equal contribution, in alphabetical order.
Solution 2: Reduce precisions
Solution 2: Reduce precisions

\[ a_1 = xw_1 \]
\[ z_1 = \text{ReLU}(a) \]
\[ z_1 = \text{Reduce}(z_1) \]
\[ w_2 \]
\[ a_2 = z_1^* w_2 \]
\[ z_2 = \text{ReLU}(a_2) \]
\[ z_2 = \text{Reduce}(z_2) \]
\[ \hat{y} \]
What do we want?
What do we want?

1. provide high memory footprint reduction
What do we want?

1. provide high memory footprint reduction

2. Low performance overhead
What do we want?

1. provide high memory footprint reduction
2. Low performance overhead
3. Minimal effect on performance
GIST ideas
GIST ideas

1. Binarize
GIST ideas

1. Binarize

2. Sparse Storage and Dense Compute
GIST ideas

1. Binarize

2. Sparse Storage and Dense Compute

3. Aggressive Lossy Encoding.
GIST ideas

1. Binarize
2. Sparse Storage and Dense Compute
3. Aggressive Lossy Encoding.
GIST ideas

1. Binarize

2. Sparse Storage and Dense Compute

3. Aggressive Lossy Encoding.

Layer specific
GIST ideas

1. Binarize
2. Sparse Storage and Dense Compute
3. Aggressive Lossy Encoding.

Layer specific
Lossless
What are the different layers?

[Diagram showing the feature map breakdown for different networks: AlexNet (256), NiN (256), Overfeat (256), VGG16 (64), Inception (64). The breakdown includes Relu→Pool, Relu/Pool→Conv, and Others.]
1. Binarize

\[ x \xrightarrow{w_1} a_1 = xw_1 \xrightarrow{z_1 = \text{ReLU}(a)} w_2 \xrightarrow{a_2 = w_2z_1} \hat{y} \]

\[
\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \frac{\partial z_2}{\partial w_2}
\]

\[
\frac{\partial z_2}{\partial w_2} = \begin{cases} 
z_2 > 0 & \frac{\partial a_2}{\partial w_2} \\
0 & \text{otherwise}
\end{cases}
\]
1. Binarize

\[ x \xrightarrow{w_1} a_1 = xw_1 \xrightarrow{z_1 = ReLu(a)} w_2 \rightarrow a_2 = w_2z_1 \xrightarrow{\hat{y} = ReLu(a_2)} \hat{y} \]

\[ \frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_2} \frac{\partial z_2}{\partial w_2} \]

\[ \frac{\partial z_2}{\partial w_2} = \begin{cases} z_2 > 0 & \frac{\partial a_2}{\partial w_2} : 0 \end{cases} \]
1. Binarize

\[ x \xrightarrow{w_1} a_1 = xw_1 \quad z_1 = ReLu(a) \xrightarrow{w_2} a_2 = w_2z_1 \xrightarrow{z_2} ReLu(a_2) \xrightarrow{\hat{y}} \]

\[ \frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \frac{\partial z_2}{\partial w_2} \]

\[ \frac{\partial z_2}{\partial w_2} \begin{cases} z_2 > 0 & \frac{\partial a_2}{\partial w_2} : 0 \\ z_2 \leq 0 & \end{cases} \]
1. Binarize

\[ x \xrightarrow{w_1} a_1 = xw_1 \xrightarrow{z_1 = \text{ReLU}(a)} w_2 \xrightarrow{a_2 = w_2z_1} \hat{y} \]

\[ \frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \frac{\partial z_2}{\partial w_2} \]

\[ \frac{\partial z_2}{\partial w_2} = \begin{cases} z_2 > 0 \? \frac{\partial a_2}{\partial w_2} : 0 \end{cases} \]

\[ z_2 > 0 \]

Binary information
1. Binarize

\[
\begin{align*}
\mathcal{X} & \xrightarrow{w_1} a_1 = xw_1 \\
& \xrightarrow{z_1 = \text{ReLU}(a)} w_2 \\
& \xrightarrow{a_2 = w_2z_1} z_2 = \text{ReLU}(a_2) \\
& \xrightarrow{\hat{y}} \\
\end{align*}
\]

\[
\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \frac{\partial z_2}{\partial w_2}
\]

\[
\frac{\partial z_2}{\partial w_2} = \begin{cases} 
1 & z_2 > 0 \\
0 & \text{otherwise}
\end{cases}
\]

Binary information: 1 bit
1. Binarize

<table>
<thead>
<tr>
<th></th>
<th>9</th>
<th>2</th>
<th>6</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>-8</td>
<td>-1</td>
<td>0</td>
<td>-6</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>-6</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
1. Binarize

<table>
<thead>
<tr>
<th>9</th>
<th>2</th>
<th>6</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>-8</td>
<td>-1</td>
<td>0</td>
<td>-6</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>-6</td>
<td>-2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

ReLu
1. Binarize

<table>
<thead>
<tr>
<th></th>
<th>9</th>
<th>2</th>
<th>6</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>-8</td>
<td>-1</td>
<td>0</td>
<td>-6</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>-6</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

ReLu

Old method
1. Binarize

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>2</td>
<td>6</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>-8</td>
<td>-1</td>
<td>0</td>
<td>-6</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>-6</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>2</td>
<td>6</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Old method

ReLU
1. Binarize

<table>
<thead>
<tr>
<th></th>
<th>9</th>
<th>2</th>
<th>6</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>-8</td>
<td>-1</td>
<td>0</td>
<td>-6</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>-6</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Old method

<table>
<thead>
<tr>
<th>9</th>
<th>2</th>
<th>6</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Size: $32 \times 16$ bits = ?

ReLU
1. Binarize

<table>
<thead>
<tr>
<th>9</th>
<th>2</th>
<th>6</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>-8</td>
<td>-1</td>
<td>0</td>
<td>-6</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>-6</td>
<td>-2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>9</th>
<th>2</th>
<th>6</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Size: \(32 \times 16\) bits = ?

Old method

ReLU

GIST method
1. **Binarize**

<table>
<thead>
<tr>
<th></th>
<th>9</th>
<th>2</th>
<th>6</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>-8</td>
<td>-1</td>
<td>0</td>
<td>-6</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>-6</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

- **ReLu**
  - Old method
    - 9 2 6 5
    - 0 0 0 0
    - 10 5 0 0
    - 3 2 1 0
  - GIST method
    - 1 1 1 1
    - 0 0 0 0
    - 1 1 1 1
    - 1 1 1 1

Size: 32 * 16 bits = ?
1. Binarize

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>2</td>
<td>6</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>-8</td>
<td>-1</td>
<td>0</td>
<td>-6</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>-6</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

ReLu

Old method

<table>
<thead>
<tr>
<th>9</th>
<th>2</th>
<th>6</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Size: 32 * 16 bits = ?

GIST method

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Size: 1 * 16 bits = ?
1. Binarize
1. Binarize

Not always possible
1. Binarize

Not always possible

eg. ResNet (Residual Network)
1. Binarize

Not always possible

eg. ResNet (Residual Network)
ReLu -> Conv
1. Binarize

Not always possible

eg. ResNet (Residual Network)

ReLu -> Conv

\[ dX = f(X, dY) \]
(d) Convolution layer
1. Binarize

Not always possible

eg. ResNet (Residual Network)
ReLu -> Conv

Works well in

eg. ReLu -> pool

\[ dX = f(X, dY) \]
(d) Convolution layer
2. Sparse storage and dense compute
2. Sparse storage and dense compute

ReLu outputs have high sparsity.
2. Sparse storage and dense compute

ReLu outputs have high sparsity.

Sometimes > 80%. VGG16
2. Sparse storage and dense compute

ReLu outputs have high sparsity.

Sometimes > 80%. VGG16

\[
M = \begin{bmatrix}
1 & 0 & 2 & 0 \\
0 & 0 & 0 & 0 \\
3 & 0 & 0 & 4 \\
0 & 0 & 0 & 0
\end{bmatrix}
\]

values = [1, 2, 3, 4]
row_indices = [1, 1, 3, 3]
col_indices = [1, 3, 1, 4]
2. Sparse storage and dense compute

ReLu outputs have high sparsity.

Sometimes > 80%. VGG16

\[
M = \begin{bmatrix}
1 & 0 & 2 & 0 \\
0 & 0 & 0 & 0 \\
3 & 0 & 0 & 4 \\
0 & 0 & 0 & 0
\end{bmatrix}
\]

values = [1, 2, 3, 4]
row_indices = [1, 1, 3, 3]
col_indices = [1, 3, 1, 4]

Effectiveness threshold: 50% - 20%
2. Sparse storage and dense compute

ReLu outputs have high sparsity.

Sometimes > 80%. VGG16

\[
M = \begin{bmatrix}
1 & 0 & 2 & 0 \\
0 & 0 & 0 & 0 \\
3 & 0 & 0 & 4 \\
0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

\[
\text{values} = [1, 2, 3, 4] \\
\text{row_indices} = [1, 1, 3, 3] \\
\text{col_indices} = [1, 3, 1, 4]
\]

Effectiveness threshold: 50% - 20%

Compressed Sparse Row (CSR)
## 2. Sparse storage and dense compute

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>6</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>-8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-6</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>0</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
2. Sparse storage and dense compute

\[
\begin{array}{cccc}
0 & 0 & 6 & 5 \\
-8 & 0 & 0 & -6 \\
0 & 5 & 0 & -2 \\
0 & 0 & 0 & 0 \\
\end{array}
\]
2. Sparse storage and dense compute

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>6</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>-8</td>
<td>0</td>
<td>0</td>
<td>-6</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>0</td>
<td>-2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>0</th>
<th>6</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>-8</td>
<td>0</td>
<td>0</td>
<td>-6</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>-6</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
2. Sparse storage and dense compute

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>6</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>-8</td>
<td>0</td>
<td>0</td>
<td>-6</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>0</td>
<td>-2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>-8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>-6</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Size: 32 * 16 bits = 512 bits

Normal
2. Sparse storage and dense compute

Normal

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>6</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-8</td>
<td>0</td>
<td>0</td>
<td>-6</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>0</td>
<td>-2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Encoding

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>6</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-8</td>
<td>0</td>
<td>0</td>
<td>-6</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>-6</td>
<td>-2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Size: 32 * 16 bits
= 512 bits
2. Sparse storage and dense compute

<table>
<thead>
<tr>
<th>Normal</th>
<th>Encoding</th>
<th>Non-zero Vals:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 6 5</td>
<td>-8 0 0 -6</td>
<td>6 5 -8 -6 5 -2</td>
</tr>
<tr>
<td>0 5 0 -2</td>
<td>0 5 -6 -2</td>
<td></td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>0 0 1 0</td>
<td></td>
</tr>
</tbody>
</table>

Size: 32 * 16 bits = 512 bits
2. Sparse storage and dense compute

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>6</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>-8</td>
<td>0</td>
<td>0</td>
<td>-6</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>0</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Non-zero Vals:
6
5
-8
-6
5
-2

Cols:
2
3
0
3
1
3

Size: 32 * 16 bits
= 512 bits

0
0
6
5

-8
0
0
-6

0
5
-6
-2

0
0
1
0

Normal

Encoding
2. Sparse storage and dense compute

<table>
<thead>
<tr>
<th>Normal</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 6 5</td>
<td>6 5 -8 -6 5 -2</td>
</tr>
<tr>
<td>-8 0 0 -6</td>
<td>2 3 0 3 1 3</td>
</tr>
<tr>
<td>0 5 0 -2</td>
<td>0 0 1 1 2 2</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td></td>
</tr>
</tbody>
</table>

Size: 32 * 16 bits = 512 bits
2. Sparse storage and dense compute

<table>
<thead>
<tr>
<th>Normal</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 6 5</td>
<td>0 0 6 5</td>
</tr>
<tr>
<td>-8 0 0 -6</td>
<td>-8 0 0 -6</td>
</tr>
<tr>
<td>0 5 0 -2</td>
<td>0 5 -6 -2</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>0 0 1 0</td>
</tr>
</tbody>
</table>

Non-zero vals: 6 5 -8 -6 5 -2
Cols: 2 3 0 3 1 3
Rows: 0 0 1 1 2 2

Size: 32 * 16 bits = 512 bits
Size: 32 * 6 + 4 * 6 * 2
= 240 bits
2. Sparse storage and dense compute

2X

Normal

Encoding

Non-zero Vals:

| 6 | 5 | -8 | -6 | 5 | -2 |

Cols:

| 2 | 3 | 0 | 3 | 1 | 3 |

Rows:

| 0 | 0 | 1 | 1 | 2 | 2 |

Size: \(32 \times 6 + 4 \times 6 \times 2\) = 240 bits

Size: \(32 \times 16\) bits = 512 bits
When does the encoding happen?

- $T = i$
- $T = j$

```
GPU
```

```
Time
```
When does the encoding happen?

$\sim_{1} \quad$ A result from a ReLu Layer fed into a Convolution layer
When does the encoding happen?

$\sim_1 \rightarrow$ A result from a ReLu Layer fed into a Convolution layer
When does the encoding happen?

A result from a ReLu Layer fed into a Convolution layer

\[ \mathcal{S}_1 \rightarrow \text{Sparse format} \]

\[ \mathcal{S}_1 \text{ is created} \]

\[ T = i \rightarrow T = j \]
When does the encoding happen?

\( \approx_1 \rightarrow \) A result from a ReLu Layer fed into a Convolution layer.

- \( \approx_1 \) in Sparse format
- Compressed
- GPU
  - \( \approx_1 \) is created
  - Forward pass
  - \( T = i \) to \( T = j \)

Time
When does the encoding happen?

$\tau_1$ A result from a ReLu Layer fed into a Convolution layer

$\tau_1$ Sparse format

Compressed

GPU

$\tau_1$ is created

forward pass

backward pass

$T = i$

$T = j$

Time
When does the encoding happen?

\[ \sim_1 \rightarrow \text{A result from a ReLu Layer fed into a Convolution layer} \]

![Diagram showing the process of encoding](image)

- **T = i** is created
- Forward pass
- Backward pass
- **T = j**

**Sparse format**

**Compressed**

**Decompressed**
When does the encoding happen?

A result from a ReLu Layer fed into a Convolution layer

$\approx_1$ is created

forward pass

backward pass

$\approx_1$ is reused

$T = i$

$T = j$

$T = j$ is reused

GPU

Sparse format

Compressed

Decompressed

Time
When does the encoding happen?

\( \tilde{z}_1 \rightarrow \) A result from a ReLu Layer fed into a Convolution layer

Why is this not bad?

\( \tilde{z}_1 \) Sparse format

\( \tilde{z}_1 \) Sparse format

Compressed

Decompressed

GPU

\( \tilde{z}_1 \) is created

forward pass

backward pass

\( \tilde{z}_1 \) is reused

\( T = i \)

\( T = j \)

Time
When does the encoding happen?

A result from a ReLu Layer fed into a Convolution layer

Why is this not bad?

Sparse format

Compressed

Decompressed

Overheads, but not bottlenecks
SSDC at work

(a) Baseline

Encoded X (2 MB)

(b) After applying SSDC encoding on X

Static memory allocation

10 MB

10

1

6 MB

5 MB

8 MB

2

10

10 MB

Stashed feature-map

Immediately consumed
3. Aggressive Lossy Encoding: Delayed Precision Reduction

\[ a_1 = x w_1 \]
\[ z_1 = \text{ReLU}(a) \]
\[ z_1^* = \text{Reduce}(z_1) \]
\[ w_2 \]
\[ a_2 = z_1^* w_2 \]
\[ z_2 = \text{ReLU}(a_2) \]
\[ z_2^* = \text{Reduce}(z_2) \]
\[ \hat{y} \]
3. Aggressive Lossy Encoding: Delayed Precision Reduction

\[ x \xrightarrow{w_1} a_1 = xw_1 \xrightarrow{z_1 = \text{ReLU}(a)} w_2 \xrightarrow{a_2 = z_1^*w_2} \hat{y} \]
3. Aggressive Lossy Encoding: Delayed Precision Reduction

\[ \mathbf{a}_1 = x \mathbf{w}_1, \quad \mathbf{z}_1 = ReLu(a) \]

\[ \mathbf{w}_2 \]

\[ \mathbf{a}_2 = \mathbf{w}_2 \mathbf{z}_1, \quad \mathbf{z}_2 = ReLu(a_2) \]

\[ \mathbf{z}_2^* = Reduce(z_2) \]

\[ \hat{y} \]
3. Aggressive Lossy Encoding: Delayed Precision Reduction

\[ a_1 = xw_1 \quad z_1 = ReLu(a) \quad w_2 \quad a_2 = w_2z_1 \quad z_2 = ReLu(a_2) \quad \hat{y} \]
3. Aggressive Lossy Encoding: Delayed Precision Reduction
3. Aggressive Lossy Encoding: Delayed Precision Reduction

Unaddressed: ResNet
4. In-place computation

\[ a_1 = x w_1 \]
\[ z_1 = \text{ReLU}(a) \]
\[ a_2 = w_2 z_1 \]
\[ z_2 = \text{ReLU}(a_2) \]
\[ \hat{y} \]
4. In-place computation

\[ a_1 = xw_1 \]

\[ z_1 = \text{ReLU}(a) \]

\[ a_2 = w_2z_1 \]

\[ z_2 = \text{ReLU}(a_2) \]

\[ \hat{y} \]
4. In-place computation

Training Deep Nets with Sublinear Memory Cost
GIST Solution

Original computation graph
Directed graph for DNN execution

Gist Schedule Builder

Modified computation graph
Gist encodings
Runtime execution

Lifetime of data structures

CNTK static memory allocator
CNTK

Microsoft Cognitive Toolkit
Commercial-grade distributed deep learning
CNTK

Commercial-grade distributed deep learning

Python API
CNTK

Commercial-grade distributed deep learning

Python API

Static memory analyzer
Evaluation setup
Evaluation setup

- Nvidia Maxwell GTX Titan X [26] card with 12 GB of GDDR5 memory using cuDNN v6.0 and CUDA 8.0
Evaluation setup

- Nvidia Maxwell GTX Titan X [26] card with 12 GB of GDDR5 memory using cuDNN v6.0 and CUDA 8.0

- AlexNet, NiN, Overfeat, VGG16, Inception, Resnet
Evaluation setup

- Nvidia Maxwell GTX Titan X [26] card with 12 GB of GDDR5 memory using cuDNN v6.0 and CUDA 8.0

- AlexNet, NiN, Overfeat, VGG16, Inception, Resnet

- ImageNet training dataset
Evaluation setup

- Nvidia Maxwell GTX Titan X [26] card with 12 GB of GDDR5 memory using cuDNN v6.0 and CUDA 8.0

- AlexNet, NiN, Overfeat, VGG16, Inception, Resnet

- ImageNet training dataset

- Baselines
  - CNTK baseline
  - Investigation baseline
Evaluation setup

- Nvidia Maxwell GTX Titan X [26] card with 12 GB of GDDR5 memory using cuDNN v6.0 and CUDA 8.0

- AlexNet, NiN, Overfeat, VGG16, Inception, Resnet

- ImageNet training dataset

- Baselines
  - CNTK baseline
  - Investigation baseline

- Comparison Metric \[ MFR = \frac{\text{Memory Footprint of Baseline}}{\text{Memory Footprint after encoding}} \]
Results
Performance Overhead
Performance Overhead
Memory footprint: Lossless Optimization

The diagram shows the contribution to memory footprint for different models (AlexNet, NiN, Overfeat, VGG16, Inception) under various conditions (Base, S, B, S+B, S+B+I).

- Other FM
- Relu/Pool→Conv
- Relu→Pool
- Imm. consumed

The percentage contributions are visualized in bars, with each bar segment representing a different operation or condition.
Memory footprint: Lossless Optimization

The chart shows the contribution to the memory footprint for different optimization techniques applied to various neural networks: AlexNet, NiN, Overfeat, VGG16, and Inception. The techniques include:

- **Other FM**
- **Relu/Pool→Conv**
- **Relu→Pool**
- **Imm. consumed**

The y-axis represents the contribution to the memory footprint in percentage, ranging from 0% to 100%. The x-axis contains the models and their variations, such as Base, S, B, S+B, and S+B+I.

The chart indicates the effective optimization by comparing the memory footprint of the models before and after applying the techniques.
Memory footprint: Lossless Optimization

![Graph showing memory footprint contributions for different models and layers.](image)

- **AlexNet**: Contributions from Various Blocks
  - **Base**: 1.06x
  - **S**: 1.26x
  - **B**: 1.35x
  - **S+B**: 1.56x
  - **S+B+I**: 1.14x

- **NiN**: Contributions from Various Blocks
  - **Base**: 1.14x
  - **S**: 1.24x
  - **B**: 1.47x
  - **S+B**: 1.47x
  - **S+B+I**: 1.1x

- **Overfeat**: Contributions from Various Blocks
  - **Base**: 1.16x
  - **S**: 1.34x
  - **B**: 1.55x
  - **S+B**: 1.65x
  - **S+B+I**: 1.17x

- **VGG16**: Contributions from Various Blocks
  - **Base**: 1.04x
  - **S**: 1.09x
  - **B**: 1.14x
  - **S+B**: 1.14x
  - **S+B+I**: 1.17x

Legend:
- Other FM
- Relu/Pool→Conv
- Relu→Pool
- Imm. consumed

© Harvard SEAS

**DASlab**
Memory footprint: Lossless Optimization

[Bar chart showing contribution to memory footprint for different models and configurations.]

- **AlexNet**: Base (1.06x), S (1.26x), B (1.35x), S+B (1.56x)
- **NiN**: Base (1.14x), S (1.24x), B (1.47x), S+B (1.55x)
- **Overfeat**: Base (1.1x), S (1.3x), B (1.46x), S+B (1.55x)
- **VGG16**: Base (1.16x), S (1.34x), B (1.65x), S+B (1.65x)
- **Inception**: Base (1.04x), S (1.09x), B (1.14x), S+B (1.17x)
Memory footprint: Lossless Optimization

![Graph showing memory footprint contributions for different models and layers](image-url)
Memory footprint: Lossless Optimization
Memory footprint: Lossless Optimization

The diagram shows the memory footprint ratio against the CNTK baseline for different models and optimization techniques. The models compared are AlexNet, NiN, Overfeat, VGG16, and Inception. The optimization techniques include SDCC, Binarization, and Inplace. The memory footprint ratios range from 1x to 16x, with Binarization showing the highest reduction across all models.
Memory footprint: Lossy Optimization

![Bar chart showing contribution to memory footprint for different models and optimization levels.](chart.png)

- **Feature-maps** and **Imm. Consumed**
- Models: AlexNet, NiN, Overfeat, VGG16, Inception
- Optimization levels: Base-FP32, DPR-FP16, DPR-FP8
- Contribution percentages are indicated for each model and optimization level.
Accuracy: Lossy Optimization

Graphs showing the training accuracy loss for different models and configurations. The models include AlexNet, Overfeat, VGG16, and Inception. The configurations are FP32, All-FP16, and Gist-FP10, Gist-FP8.
Batch size increase

![Graph showing speedup against the largest minibatch that fits on baseline for different ResNet models: ResNet-509, ResNet-851, ResNet-1202, and ResNet-509. The y-axis represents speedup as a multiple of 0.9 to 1.3, and the x-axis represents different ResNet models. The bars indicate the speedup for baseline and Gist configurations. The trend shows that as networks get deeper, speedup increases.]
Next steps
Next steps

- CNTK alternatives
Next steps

- CNTK alternatives

- Related reduction to other generalized neural networks
Next steps

- CNTK alternatives

- Related reduction to other generalized neural networks

- FP8 and FP10