Riptide: Fast End-to-End Binarized Neural Networks

Josh Fromm, Meghan Cowan, Matthai Philipose, Luis Ceze, and Shwetak Patel

<table>
<thead>
<tr>
<th>Device / Platform</th>
<th>GFLOPS</th>
<th>GFLOPS/W</th>
<th>Sec/Frame</th>
<th>WS/Frame</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Raspberry Pi-zero</td>
<td>0.32</td>
<td>0.24</td>
<td>56.42</td>
<td>75.01</td>
<td>$22</td>
</tr>
<tr>
<td>2 Raspberry Pi3</td>
<td>3.62</td>
<td>0.81</td>
<td>4.97</td>
<td>22.14</td>
<td>$37</td>
</tr>
<tr>
<td>3 Snapdragon 835</td>
<td>11.5</td>
<td>1.44</td>
<td>1.56</td>
<td>12.50</td>
<td>$13 (bulk)</td>
</tr>
<tr>
<td>4 Haswell i7-4790</td>
<td>181</td>
<td>1.68</td>
<td>0.10</td>
<td>10.71</td>
<td>$380</td>
</tr>
<tr>
<td>5 Titan V</td>
<td>15000</td>
<td>60</td>
<td>0.0012</td>
<td>0.3</td>
<td>$3000</td>
</tr>
</tbody>
</table>

The diagram shows a scatter plot comparing the top-1 accuracy and operations (G-Ops) for various deep neural network models, including Inception-v3, Inception-v4, ResNet-50, ResNet-101, ResNet-152, VGG-16, and VGG-19. The size of the circles represents the parameter size of each model.
1-bit Matrix Operations

- Quantize floats to +/-1
- $1.122 \times -3.112 \Rightarrow 1 \times -1$
- Notice:
  - $1 \times 1 = 1$
  - $1 \times -1 = -1$
  - $-1 \times 1 = -1$
  - $-1 \times -1 = 1$
- Replacing -1 with 0, this is just XNOR
- Retrain model to convergence

1-bit Matrix Operations: Cost/Benefit

```c
float x[], y[], w[];
...
for i in 1...N:
    y[j] += x[i] * w[i];
    2N ops

unsigned long x[], y[], w[];
...
for i in 1...N/64:
    y[j] += 64 - 2*popc(not(x_b[i] xor w_b[i]));
    3N/64 ops
```

Typically, lose ~10% accuracy

1-bit Matrix Operations: Cost/Benefit

float x[], y[], w[];
...
for i in 1...N:
    y[j] += x[i] * w[i];

unsigned long x[], y[], w[];
...
for i in 1...N/64:
    y[j] += 64 - 2*popc(not(x_b[i] xor w_b[i]));

Typically, lose ~10% accuracy

1-bit Matrix Operations: Cost/Benefit

~40x faster
1-bit Matrix Operations: Cost/Benefit

Runtime

1904 ms
380 ms

Full Precision Baseline
Unoptimized Binary Network
Implementation Challenges

No optimized linear algebra libraries like BLAS to leverage

- Need to implement optimizations from scratch
- Optimizations tuned for specific CPU
- CPUs have no native support for low bit data types
- Need to work on packed data

Baselines incredibly well optimized

- Optimized linear algebra libraries
- Hardware support for conventional deep learning
Are Binary Networks Actually Fast?

Majority of work in binarization is simulated

• Which binarization techniques can be implemented efficiently?

• What are the runtime bottlenecks in a binary model?

• How do I deploy a fast binary model on my platform?

To address these questions we introduce Riptide.
A one-stop solution to training and deploying fast binary networks on a variety of hardware platforms.

- Addresses implementation issues in mixed polarity quantization
- Introduces the Fused Glue operation, removing all floating-point arithmetic from binary models.
- Provides high-performance bitserial operators through TVM.
- Yields 4-12X speedups across various models and bitwidths while maintaining state-of-the-art accuracy.
- Available open-source today at github.com/jwfromm/Riptide
Implementing Binary Layers

\[ \text{features: float array} \otimes \text{kernels: float array} = \text{activations: float array} \]

- Multiply
- Accumulate
Implementing Binary Layers

features: float array

features: int array

kernels: float array

Multiply

Accumulate

= activations: float array
Implementing Binary Layers

features: float array
QH

kernels: float array

... ...

features: int array

kernels: int array

activations: float array

Multiply Accumulate

=
Implementing Binary Layers

features: float array

kernels: float array

activations: int array

features: int array

kernels: int array

Bitserial Accumulate

=
Quantization Polarity

Quantization Function: $\bar{x} = \text{sign}(x)$

- Bipolar Quantization
  - Implemented with bitwise-xnor and popcount
  - Well-suited for weights, which represent correlation (1) or inverse-correlation (-1)

- Unipolar Quantization
  - Implemented with bitwise-and and popcount
  - Well-suited for activations, which represent pattern-match (1) or no pattern-match (0)
Quantization Polarity

- XnorNet (all bipolar) -> 44.2% accuracy
- DorefaNet (bipolar weights unipolar activations) -> 50.0% accuracy

A (unipolar) \( \times \) 1 1 0 1 0 1 ...
W (bipolar) = 0 1 0 0 1 0 ...
Expected = 0 1 0 -1 0 -1 ...

Multiple meanings of 0 bits causes mixed polarity to unimplementable

Mixed Polarity Operation

\[ a \cdot w = \sum_{n=0}^{N-1} 2^N ( \text{popc}(a_n \land w) - \text{popc}(a_n \land \neg w)) \]

- Enables mixed polarity binary networks
- Doubles amount of inner loop compute but does not require additional memory operations
- Mixed polarity may offer compelling points on speedup to accuracy versus pure bipolar
Multibit Quantization

Quantization Function: $\tilde{x} = \text{linear}(x)$

- Translates naturally to integer representation
- Does not necessarily fit distribution

Multibit Quantization

Quantization Function: $\tilde{x} = HWGQ(x)$

- Better fit for Gaussian distribution
- Not implementable

Cai et al., “Deep Learning with Low Precision by Half-wave Gaussian Quantization.” 2017
Multibit Quantization

Value is based on unique bit pair rather than combination of bits, \((01 + 10 \neq 11)\)

Cai et al., "Deep Learning with Low Precision by Half-wave Gaussian Quantization." 2017
Implementing Binary Layers

- **features**: float array
- **kernels**: float array
- **activations**: int array

- **features**: int array
- **kernels**: int array
- **Bitwise Accumulate**

1-bit bipolar quantization

N-bit linear bipolar or unipolar quantization

Xnor-popcount / mixed polarity-popcount
Implementing Binary Models

- **Conv**
- QConv
- QConv
- QConv
- QConv
- QDense

Legend:
- Full Precision
- Binary
Implementing Binary Models

Computational Complexity: $\frac{NKKFHWNC}{43}$
Estimated Impact of Glue Layers

- Impact of glue layers is too high
- We must derive binarized glue for decent end-to-end speedups
Weight Scaling

\[ \alpha_k = mean(|W_k|) \]

\[ q(a) = \alpha_k a \]

- Introduced in XnorNet
- Allows scale of weights to be preserved
- Brought accuracy from 27% to 44%
- Now used ubiquitously

Quantized Weight Scaling

\[ q(a) = \alpha_k a \]

- Use approximate power of 2 (AP2)
- Replaces multiply with bitwise shift
- Constant at inference time
- Requires only a single instruction

\[ AP2(x) = 2^{\text{round}(\log_2(|x|))} \]
\[ wb = -\log_2(AP2(\alpha_k)) \]
\[ q(a) = (a + (1 \ll (wb - 1))) \gg wb \]

BatchNormalization

• Centers and scales output activations

• Essential for quantization, used in all binary techniques

• Must derive quantized versions of both centering and scaling

\[
\mu_k = \frac{1}{m} \sum_{i=1}^{m} (a_i) \\
\sigma_k^2 = \frac{1}{m} \sum_{i=1}^{m} (a_i - \mu_k)^2 \\
\hat{a}_i = \frac{a_i - \mu_k}{\sqrt{\sigma_k^2 + \epsilon}}
\]

Binary Scaling

• We can simply compute the AP2 of standard deviation

\[ AP2(x) = 2^{\text{round}(\log_2(|x|))} \]

\[ wb = -\log_2(AP2(\alpha_k)) \]

\[ sb = \log_2(AP2(\sqrt{\sigma_k^2 + \epsilon})) \]

\[ q(a) = (a + (1 \ll (wb - 1))) \gg (wb + sb) \]
Binary Center

To add a constant to a binarized tensor, we must use Fixed Point Quantization (FPQ) with the same bits and scale.

**Algorithm 9: Fixed point quantization (FPQ) function.**

*Input:* a tensor $X$ to quantize to $B$ bits with scale $S$

*Output:* $B$-bit quantized tensor $Y$

1. $\hat{X} = \text{clip}(X, -S, S)$
2. $g = \frac{S}{2^{b-1}}$ // Compute granularity
3. $Y = \text{round}(\frac{\hat{X}}{g})$

$$\hat{\mu} = \text{FPQ}(\mu, B, S)$$

$N$-bit input to next layer
$wb$ fractional bits

$$B = N + wb$$

$$S = 1 + \sum_{i=1}^{wb} \frac{1}{2^{2N-1}} \left(\frac{1}{2}\right)^i$$

$$= 1 + \frac{1}{2^{2N-1}} \left(1 - \frac{1}{2^{wb}}\right)$$
Fused Glue Operation

\[ AP2(x) = 2^{\text{round}(\log_2(|x|))} \]
\[ wb = -\log_2(AP2(\alpha_k)) \]
\[ sb = \log_2(AP2(\sqrt{\sigma_k^2 + \epsilon})) \]
\[ q(a) = (a + (1 \ll (wb - 1))) \gg (wb + sb) \]

This is the fused glue operation

All terms are constant at runtime except \( a \)

Only requires two integer operations

\[ B = N + wb \]
\[ S = 1 + \frac{1}{2^N - 1} \left( 1 - \frac{1}{2^{wb}} \right) \]
\[ \hat{\mu} = FPQ(\mu, B, S) \]
\[ cb = (1 \ll (wb - 1)) - \hat{\mu} \]
\[ q(a) = (a + cb) \gg (wb + sb) \]
Fully Binarized Network

Traditional Binary Network

Computational Complexity: $4HWF\ HWF\ 4HWF\ HWF\ 5HWF\ 3HWF\ \text{Total} = 18HWF$

- QConv
- Dequantize
- WeightScale
- BatchNorm
- Activation
- Quantize
- Bitpack
- QConv

Fully Binarized Network

Computational Complexity: $2HWF\ HWF\ 3HWF\ \text{Total} = 6HWF$

- QConv
- Fused Glue
- Clip
- Bitpack
- QConv

- 3X fewer glue operations
- No floating-point data
- No multiplication or division
## FBN Accuracy

<table>
<thead>
<tr>
<th>Model</th>
<th>Name</th>
<th>1-bit</th>
<th>2-bit</th>
<th>3-bit</th>
<th>full precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AlexNet</td>
<td>44.2%</td>
<td>—</td>
<td>—</td>
<td>56.6%</td>
</tr>
<tr>
<td>2</td>
<td>AlexNet</td>
<td>27.9%</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>3</td>
<td>AlexNet</td>
<td>43.6%</td>
<td>49.8%</td>
<td>48.4%</td>
<td>55.9%</td>
</tr>
<tr>
<td>4</td>
<td>AlexNet</td>
<td>43.3%</td>
<td>51.0%</td>
<td>—</td>
<td>56.6%</td>
</tr>
<tr>
<td>5</td>
<td>AlexNet</td>
<td>52.7%</td>
<td>—</td>
<td>—</td>
<td>58.5%</td>
</tr>
<tr>
<td>6</td>
<td>VGGNet</td>
<td>64.1%</td>
<td>—</td>
<td>—</td>
<td>69.8%</td>
</tr>
<tr>
<td>7</td>
<td>AlexNet</td>
<td>44.5%</td>
<td>52.5%</td>
<td>53.6%</td>
<td>56.5%</td>
</tr>
<tr>
<td>8</td>
<td>AlexNet</td>
<td>42.8%</td>
<td>50.4%</td>
<td>52.4%</td>
<td>56.5%</td>
</tr>
<tr>
<td>9</td>
<td>VGGNet</td>
<td>64.2%</td>
<td>67.1%</td>
<td>72.7%</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>VGGNet</td>
<td>54.4%</td>
<td>61.5%</td>
<td>65.2%</td>
<td>72.7%</td>
</tr>
<tr>
<td>11</td>
<td>ResNet18</td>
<td>47.9%</td>
<td>58.4%</td>
<td>61.8%</td>
<td>70.9%</td>
</tr>
</tbody>
</table>

- Our system is comparable to state-of-the-art techniques
- Unipolar quantization yields higher accuracies as expected
- Effective across various models
Measurement Platform

Raspberry Pi
ARM Cortex-A53

- Widely available and inexpensive
- Representative of IoT devices
  - Qualcomm Snapdragons
  - Azure Sphere
- Resource constrained / in need of acceleration
Separates compute and implementation into a declaration and schedule.
Schedules contain knobs that are attuned for the backend.

Tensor Expression Language
Schedule Optimization Space
AutoTVM: Optimize Tensor Operators

Chen et al., “TVM: An Automated End-to-End Optimizing Compiler for Deep Learning.” 2018
TVM Schedule Intrinsics

- **Tiling**: Break computation into chunks for better locality

- **Vectorization**: Use hardware SIMD instructions for more efficient operation execution.

- **Parallelization**: Leverage MIMD facilities such as multiple cores.

- **Loop Unrolling**: Replicate the body of loops to reduce overhead.
Fast Popcount

LLVM 8.0

| x8 | vmovl.8 | q0, d0 |
| vmovl.8 | q2, d1 |
| vand | q0, q0, q2 |
| vcnt.8 | q0, q0 |
| vpaddl.8 | q0, q0 |
| vadd.16 | q1, q0, q0 |
| vst1.16 | q1, addr |

Total: 49

Synthesized

| x8 | vand.8 | d0, d0, d1 |
| vcnt.8 | d0, d0 |
| vadd.8 | d0, d0, d1 |
| vadd.8 | d0, d0, d1 |
| vadd.8 | d0, d0, d1 |
| vadd.8 | d0, d0, d1 |
| vpadal.8 | q1, {d0,d1} |
| vst1.16 | q1, addr |

Total: 24
**Int-N Bit Packed Activations**

$x_0$ \ $x_1$ \ $x_2$ \ $x_3$ \ $x_4$ \ $x_5$ \ $x_6$ \ $x_7$ \ $x_8$ \ $x_9$ \ \ \ \ \ $x_N$

\[\frac{NHWC}{8}\] Bytes

---

**Int-16 Popcount Accumulation**

$c_0$ \ $c_1$ \ $c_2$ \ $c_3$ \ \ \ \ \ $c_N$

2\[NHWC\] Bytes

---

**Int-16 Quantized Prepacked Bits**

$q_0$ \ $q_1$ \ $q_2$ \ $q_3$ \ \ \ \ \ $q_N$

2\[NHWC\] Bytes

---

**Int-N Bit Packed Outputs**

$y_0$ \ $y_1$ \ $y_2$ \ $y_3$ \ $y_4$ \ $y_5$ \ $y_6$ \ $y_7$ \ $y_8$ \ $y_9$ \ \ \ \ \ $y_N$

\[\frac{NHWC}{8}\] Bytes

---

**BinaryConv**

**Fused Shift/Scale**

**Bit Pack**

**Bitpack Fusion**
Impact of Optimizations

- Combination of TVM optimizations gives 12X Speedup over baseline
- Each optimization has a significant impact
- Speedups from bitpack fusion are due to fewer memory operations
- With a high-quality implementation, we can study our design choices
Optimization Ablation Study

- Removing any optimization has a significant impact on performance.
- Using fused glue gives nearly a 2X speedup, as predicted by opcount estimates.
Glue Layer Impact

• Glue consistently takes a similar amount of time as core compute layers

• Our fused glue operation almost completely removes this cost
Impact of Polarity

- Baseline is near optimal
- Quantized layers have much more memory overhead
- Although unipolar quantization has twice as many operations, it is only marginally slower than bipolar quantization
Cumulative Speedup

Polarity = Unipolar

Relative Speedup

Squeezenet | Vggnet Model | Alexnet
---|---|---
10 | 8 | 6

Polarity = Bipolar

Relative Speedup

Squeezenet | Vggnet Model | Alexnet
---|---|---
12 | 10 | 8

Bitwidth
- 1
- 2
- 3
Layerwise Speedup

- Speedup is not consistent across layers
- May be possible to design a network of binarizable layers
Thank You!

Code:

Paper:
Backup Slides
<table>
<thead>
<tr>
<th>Model</th>
<th>Name</th>
<th>1-bit</th>
<th>2-bit</th>
<th>3-bit</th>
<th>full precision</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>ImageNet top-1 accuracy / Runtime (ms)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>AlexNet Xnor-Net [48]</td>
<td>44.2%</td>
<td>—</td>
<td>—</td>
<td>56.6%</td>
</tr>
<tr>
<td>2</td>
<td>AlexNet BNN [12]</td>
<td>27.9%</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>3</td>
<td>AlexNet DoReFaNet [63]</td>
<td>43.6%</td>
<td>49.8%</td>
<td>48.4%</td>
<td>55.9%</td>
</tr>
<tr>
<td>4</td>
<td>AlexNet QNN [27]</td>
<td>43.3%</td>
<td>51.0%</td>
<td>—</td>
<td>56.6%</td>
</tr>
<tr>
<td>5</td>
<td>AlexNet HWGQ [4]</td>
<td>—</td>
<td>52.7%</td>
<td>—</td>
<td>58.5%</td>
</tr>
<tr>
<td>6</td>
<td>VGGNet HWGQ [4]</td>
<td>—</td>
<td>64.1%</td>
<td>—</td>
<td>69.8%</td>
</tr>
<tr>
<td>7</td>
<td>AlexNet Riptide-unipolar (ours)</td>
<td>44.5%</td>
<td>52.5%</td>
<td>53.6%</td>
<td>56.5%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>150.4</td>
<td>196.8</td>
<td>282.8</td>
<td>1260.0</td>
</tr>
<tr>
<td>8</td>
<td>AlexNet Riptide-bipolar (ours)</td>
<td>42.8%</td>
<td>50.4%</td>
<td>52.4%</td>
<td>56.5%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>122.7</td>
<td>154.6</td>
<td>207.0</td>
<td>1260.0</td>
</tr>
<tr>
<td>9</td>
<td>VGGNet Riptide-unipolar (ours)</td>
<td>56.8%</td>
<td>64.2%</td>
<td>67.1%</td>
<td>72.7%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>243.8</td>
<td>387.2</td>
<td>610.0</td>
<td>2420.0</td>
</tr>
<tr>
<td>10</td>
<td>VGGNet Riptide-bipolar (ours)</td>
<td>54.4%</td>
<td>61.5%</td>
<td>65.2%</td>
<td>72.7%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>184.1</td>
<td>271.4</td>
<td>423.5</td>
<td>2420.0</td>
</tr>
<tr>
<td>11</td>
<td>ResNet18 Riptide-unipolar (ours)</td>
<td>47.9%</td>
<td>58.4%</td>
<td>61.8%</td>
<td>70.9%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>76.2</td>
<td>112.0</td>
<td>152.3</td>
<td>380.8</td>
</tr>
</tbody>
</table>