Accelerator Overlays: Spec What You Want, Build What You Need

Ephrem Wu
FPL 2019
FPGA Overlay Accelerator

Accelerator Overlay

FPGA Silicon

© Copyright 2019 Xilinx
FPGA Overlay Accelerator

Application Software

Accelerator Overlay

FPGA Silicon
Vision Neural Networks
Image Classification

cat
traffic light
table
airplane
ImageNet Challenge Accuracy Progress 2010—2017

ImageNet Challenge Winner Top-5 Error Rates

<table>
<thead>
<tr>
<th>Year</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010</td>
<td>NEC-UIUC</td>
</tr>
<tr>
<td>2011</td>
<td>XRCE</td>
</tr>
<tr>
<td>2012</td>
<td>SuperVision (a.k.a. AlexNet)</td>
</tr>
<tr>
<td>2013</td>
<td>Clarifai</td>
</tr>
<tr>
<td>2014</td>
<td>GoogLeNet</td>
</tr>
<tr>
<td>2015</td>
<td>MSRA</td>
</tr>
<tr>
<td>2016</td>
<td>Trims-Soushen</td>
</tr>
<tr>
<td>2017</td>
<td>SENet</td>
</tr>
</tbody>
</table>

© Copyright 2019 Xilinx
2015—2019: Faster, Smaller, More Accurate

Graphs based on M. Tan and Q. V. Le, “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks,” ICML 2019
Single crop, single model

Graphs based on M. Tan and Q. V. Le, “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks,” ICML 2019

Single crop, single model

Graphs based on M. Tan and Q. V. Le, “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks,” ICML 2019

Single crop, single model

Graphs based on M. Tan and Q. V. Le, “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks,” ICML 2019

Single crop, single model

Network architecture search with depth/width/resolution scaling

Network architecture search

Hand-designed

Graphs based on M. Tan and Q. V. Le, “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks,” ICML 2019
Single crop, single model

© Copyright 2019 Xilinx
High-Throughput Linear Layers

> High clock rates
High-Throughput Linear Layers

> High clock rates

>> Build fast (peak DSP clock rate) matrix multipliers on FPGAs
High-Throughput Linear Layers

- High clock rates
  - Build fast (peak DSP clock rate) matrix multipliers on FPGAs
  - Keep interconnects between memory and compute simple
High-Throughput Linear Layers

High clock rates

- Build fast (peak DSP clock rate) matrix multipliers on FPGAs
- Keep interconnects between memory and compute simple
- Can still get 90% peak clock rate in overlay processor
- Similar principle to RISC

High-Throughput Linear Layers

> High clock rates
  >> Build fast (peak DSP clock rate) matrix multipliers on FPGAs
  >> Keep interconnects between memory and compute simple
  >> Can still get 90% peak clock rate in overlay processor
  >> Similar principle to RISC

> High compute efficiency
High-Throughput Linear Layers

> High clock rates
  >> Build fast (peak DSP clock rate) matrix multipliers on FPGAs
  >> Keep interconnects between memory and compute simple
  >> Can still get 90% peak clock rate in overlay processor
  >> Similar principle to RISC

> High compute efficiency
  >> Tensors reshaped into matrices
  >> Matrices blocked for FPGA matrix multipliers

© Copyright 2019 Xilinx
High-Throughput Linear Layers

> High clock rates
  >> Build fast (peak DSP clock rate) matrix multipliers on FPGAs
  >> Keep interconnects between memory and compute simple
  >> Can still get 90% peak clock rate in overlay processor
  >> Similar principle to RISC

> High compute efficiency
  >> Tensors reshaped into matrices
  >> Matrices blocked for FPGA matrix multipliers
  >> Memory allocation, address generation, and non-linear layers key to success
A Domain-Specific Architecture for Neural Networks


© Copyright 2019 Xilinx
A Domain-Specific Architecture for Neural Networks

Squint and every architecture looks the same?
A Domain-Specific Architecture for Neural Networks

Keep these networks simple

- Tensor Buffer
- Neural-Net Program
- Auxiliary Processing
- Accumulators
- Matrix-Vector Multiplier (MxV)

© Copyright 2019 Xilinx
A Domain-Specific Architecture for Neural Networks

CISC instructions are back. Thank you, dataflow graphs.
\[ X = \max(AB + c1^T, 0) \]

\[ X = \text{relu}(\text{add}(	ext{matmul}(A, B), \text{broadcast}(c))) \]

No explicit nested loops.
Dimensions in tensors.
One tensor memory roundtrip.
A Domain-Specific Architecture for Neural Networks

Use small PEs and DSP cascades. Go for near-Fmax!
MxV: Don’t Leave Performance on the Table

- Achieved Operating Clock Rate (MHz)
- Max. Datasheet DSP Clock Rate (MHz)
- Realized Performance (%)

References from FPL17
MxV: Don’t Leave Performance on the Table

- Achieved Operating Clock Rate (MHz)
- Max. Datasheet DSP Clock Rate (MHz)
- Realized Performance (%)

References from FPL17

© Copyright 2019 Xilinx
Compute-Efficient Neural Processor Overlay [FPGA 2019]

- Platform: VCU1525 board with VU9P-2 FPGA
- Case Study: GoogLeNet v1 Inference
  - 3 parallel GoogLeNets with independent weights
  - Each network runs with batch size 1
  - Aggregate 3046 images/sec, 3.3 ms latency
- Compute
  - DSP supertile arrays running at 720 MHz
  - 56% DSP48 tiles consumed, DSP cycles 95% utilized
  - Per-tensor block floating-point, 8-/16-bit significands
- Memory
  - No external DRAM on accelerator card used
  - All tensors stored in UltraRAM & BRAM
  - 1/2 DSP clock rate to simplify timing convergence

© Copyright 2019 Xilinx
A Domain-Specific Architecture for Neural Networks

Use low-precision here.
Uniform Quantization

\[ x \]

Original dist. (e.g. in fp32)
Uniform Quantization

\[ x \]

Threshold scalar to be learned

\[ \frac{x}{s} \]

Original dist.

Scale


© Copyright 2019 Xilinx
Uniform Quantization

\[ \frac{x}{s} \]

\[ \frac{x}{s} \]

Scaled FP32 numbers rounded to integers

Original dist.

Scale

Round
Uniform Quantization

\[ x \]

\[ \frac{x}{s} \]

\[ \left\lfloor \frac{x}{s} \right\rfloor \]

\[ \text{clip}\left(\left\lfloor \frac{x}{s} \right\rfloor, -Z, Q - Z - 1\right) \]

Original dist.

Scale

Round

Clip

© Copyright 2019 Xilinx
Uniform Quantization

\[ x \]

\[ \frac{x}{s} \]

\[ \left\lfloor \frac{x}{s} \right\rfloor \]

\[ \text{clip}\left(\left\lfloor \frac{x}{s} \right\rfloor, -Z, Q - Z - 1 \right) \]

Original dist.

Scale

Round

Clip

Zero point

Exact zero
Uniform Quantization

\[ x \]

\[ \frac{x}{s} \]

\[ \left\lfloor \frac{x}{s} \right\rfloor \]

\[ \text{clip}\left(\left\lfloor \frac{x}{s} \right\rfloor, -Z, Q - Z - 1\right) \]

Zero point

Bins

Original dist.

Scale

Round

Clip

Exact zero
Uniform Quantization

\[ \frac{x}{s} \]

\[ \left\lfloor \frac{x}{s} \right\rfloor \]

Range can be asymmetric. This example favors precision over range.
\[ q = \text{clip}\left(\left\lfloor \frac{x}{s} \right\rfloor, -Z, Q - Z - 1\right) \]
INT-to-Float

\[
\hat{x} = sq
\]

Floating-point numbers with quantization noise for simulation using floating-point hardware

\[
q = \text{clip}\left(\left\lfloor \frac{x}{s} \right\rfloor, -Z, Q - Z - 1\right)
\]

Numbers that enter integer matrix multiplier hardware
<table>
<thead>
<tr>
<th>Example</th>
<th>Zero Point $Z$</th>
<th>Bins $Q$</th>
<th>Integer Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>int8</td>
<td>128</td>
<td>256</td>
<td>$[-128, 127]$</td>
</tr>
</tbody>
</table>

### Which int8?

<table>
<thead>
<tr>
<th>Example</th>
<th>Zero Point $Z$</th>
<th>Bins $Q$</th>
<th>Integer Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>int8</td>
<td>128</td>
<td>256</td>
<td>$[-128, 127]$</td>
</tr>
<tr>
<td>Symmetric int8</td>
<td>127</td>
<td>255</td>
<td>$[-127, 127]$</td>
</tr>
</tbody>
</table>

### Which int8?

<table>
<thead>
<tr>
<th>Example</th>
<th>Zero Point Z</th>
<th>Bins $Q$</th>
<th>Integer Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>int8</td>
<td>128</td>
<td>256</td>
<td>$[-128, 127]$</td>
</tr>
<tr>
<td>Symmetric int8</td>
<td>127</td>
<td>255</td>
<td>$[-127, 127]$</td>
</tr>
<tr>
<td>uint8</td>
<td>0</td>
<td>256</td>
<td>$[0, 255]$</td>
</tr>
</tbody>
</table>

In general...

<table>
<thead>
<tr>
<th></th>
<th>Zero Point $Z$</th>
<th>Bins $Q$</th>
<th>Integer Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^b$ levels with zero point $Z$</td>
<td>$Z \in [0,2^b - 1]$</td>
<td>$2^b$</td>
<td>$[-Z, 2^b - 1 - Z]$</td>
</tr>
</tbody>
</table>

A Domain-Specific Architecture for Neural Networks

Use in-package memory
SRAM: URAM + BRAM
DRAM: HBM
UltraScale™ Memory

Distributed RAM
(bits to kilobits)

Block RAM
(10s of megabits)

External DDR DRAM
(10s of gigabits)
UltraScale+™ Memory

- Distributed RAM (bits to kilobits)
- Block RAM (10s of megabits)
- UltraRAM (100s of megabits)
- HBM (10s of gigabits)

New in UltraScale+™

External DDR DRAM (10s of gigabits)

Bandwidth

Capacity

© Copyright 2019 Xilinx
Use FPGA URAM for Capacity and Bandwidth

- DDR4 4x64
  - Bandwidth: 0.0768 TB/s
- VU3P 320 URAMs
  - 10MB + ECC
  - Bandwidth: 3.072 TB/s
- VU9P 960 URAMs
  - 30MB + ECC
  - Bandwidth: 9.216 TB/s
- VU13P 1280 URAMs
  - 40MB + ECC
  - Bandwidth: 12.288 TB/s

Holds three copies of GoogLeNet 8-bit weights and activations
A Domain-Specific Architecture for Neural Networks

- Reduction in DSP mult-add cascade
- Broadcast in fabric pipeline
- Been there since Day One of DSP blocks.
A Domain-Specific Architecture for Neural Networks

Key to compute efficiency: Tensor-to-matrix memory allocation and address generation
A Domain-Specific Architecture for Neural Networks

Map channels to MxV lanes to simplify address generation.
2D Convolution: Tensor View

Input Tensor $\mathbf{X} \in \mathbb{R}^{H_1 \times W_1 \times C_{in}}$

Input Channel 0
Input Channel 1
Input Channel 2
2D Convolution: Tensor View

Input Tensor $\mathcal{X} \in \mathbb{R}^{H_1 \times W_1 \times C_{in}}$

Output Tensor $\mathcal{Y} \in \mathbb{R}^{H_2 \times W_2 \times C_{out}}$

Output channel 0, Output channel 1, Output channel 2, Output channel 3
Although input and output tensors have three axes, convolution is 2D, not 3D.

There’s a 2D convolution filter mask for every input-output channel pair (3 × 4 = 12 in this example).
2D Convolution: Tensor View

Input Tensor $\mathbf{X} \in \mathbb{R}^{H_1 \times W_1 \times C_{in}}$

Output Tensor $\mathbf{Y} \in \mathbb{R}^{H_2 \times W_2 \times C_{out}}$

Filter Weight Tensor $\mathbf{W} \in \mathbb{R}^{F_y \times F_x \times C_{in} \times C_{out}}$
2D Convolution: Tensor View

Input Tensor $X \in \mathbb{R}^{H_1 \times W_1 \times C_{in}}$

Output Tensor $Y \in \mathbb{R}^{H_2 \times W_2 \times C_{out}}$

Filter Weight Tensor $W \in \mathbb{R}^{F_y \times F_x \times C_{in} \times C_{out}}$

© Copyright 2019 Xilinx
2D Convolution: From Tensor to Flat View

Parallel convolution sliding windows, one per input channel

Input Image

Input Channels
2D Convolution: Flat View

Parallel convolution sliding windows, one per input channel
2D Convolution: Flat View

Parallel convolution sliding windows, one per input channel
2D Convolution: Flat Silicon View

- Input channel broadcast
- Per-channel element-wise multiplication with filter weights
2D Convolution: Flat Silicon View

Input Image

Input Channels

Reduction per output channel

Accumulates $3 \times 3 \times 3 = 27$ products

© Copyright 2019 Xilinx
Interpreting Weights (Sort of)

Input Image

Looking for edges

Looking for green patches

© Copyright 2019 Xilinx
Reshaping Tensors to Matrices

3-Mode Output Tensor

\[ \mathbf{Y} \in \mathbb{R}^{H \times W \times C_{out}} \]

Elements per Output Channel

\[ N = 28 \times 28 = 784 \]

Output Matrix

\[ \mathbf{Y} \in \mathbb{R}^{C_{out} \times H \times W} \]

\[ C_{out} = 192 \]
Reshaping Tensors to Matrices

3-Mode Output Tensor
\[ Y \in \mathbb{R}^{H_2 \times W_2 \times C_{out}} \]

Elements per Output Channel
\[ N = 28 \times 28 = 784 \]

Output channels
\[ M = 192 \]

Output Matrix
\[ Y \in \mathbb{R}^{C_{out} \times H_2 \times W_2} \]
Reshaping Tensors to Matrices

3-Mode Output Tensor
\[ Y \in \mathbb{R}^{H_2 \times W_2 \times C_{out}} \]

Elements per Output Channel
\[ N = 28 \times 28 = 784 \]

Output Matrix
\[ Y \in \mathbb{R}^{C_{out} \times H_2 W_2} \]

Output channels
\[ M = 192 \]

4-Mode Weight Tensor
\[ W \in \mathbb{R}^{F_y \times F_x \times C_{in} \times C_{out}} \]

Input Channels × Weights per Filter
\[ K = 3 \times 3 \times 128 = 1152 \]

Weight Matrix
\[ W \in \mathbb{R}^{C_{out} \times F_y F_x C_{in}} \]

© Copyright 2019 Xilinx
Reshaping Tensors to Matrices

3-Mode Output Tensor
\[ \mathbf{Y} \in \mathbb{R}^{H_2 \times W_2 \times C_{out}} \]

- Elements per Output Channel
  \[ N = 28 \times 28 = 784 \]

- Output Matrix
  \[ \mathbf{Y} \in \mathbb{R}^{C_{out} \times H_2 \times W_2} \]

- Output channels
  \[ M = 192 \]

4-Mode Weight Tensor
\[ \mathbf{W} \in \mathbb{R}^{F_y \times F_x \times C_{in} \times C_{out}} \]

- Input Channels x Weights per Filter
  \[ K = 3 \times 3 \times 128 = 1152 \]

- Weight Matrix
  \[ \mathbf{W} \in \mathbb{R}^{C_{out} \times F_y \times F_x \times C_{in}} \]

© Copyright 2019 Xilinx
Reshaping Tensors to Matrices for Matrix Multipliers

3-Mode Output Tensor
\[ Y \in \mathbb{R}^{H_2 \times W_2 \times C_{out}} \]

Output Matrix
\[ Y \in \mathbb{R}^{C_{out} \times H_2 \times W_2} \]

Elements per Output Channel
\[ N = 28 \times 28 = 784 \]

Output channels
\[ M = 192 \]

\[ C_{out} = 192 \]

4-Mode Weight Tensor
\[ W \in \mathbb{R}^{F_y \times F_x \times C_{in} \times C_{out}} \]

Weight Matrix
\[ W \in \mathbb{R}^{C_{out} \times F_y \times F_x \times C_{in}} \]

Input Channels × Weights per Filter
\[ K = 3 \times 3 \times 128 = 1152 \]

\[ C_{in} = 128 \]

3-Mode Input Tensor
\[ X \in \mathbb{R}^{H_1 \times W_1 \times C_{in}} \]

Input Matrix
\[ X \in \mathbb{R}^{F_y \times F_x \times C_{in} \times H_2 \times W_2} \]

\[ N = 28 \times 28 = 784 \]

\[ K = 1152 \]
Reshaping Tensors to Matrices for Matrix Multipliers

3-Mode Output Tensor
\( \mathcal{Y} \in \mathbb{R}^{H_2 \times W_2 \times C_{out}} \)

Output channels
\( N = 28 \times 28 = 784 \)

Output Matrix
\( Y \in \mathbb{R}^{C_{out} \times H_2 \times W_2} \)

3×3
\( \cdot \cdot \cdot \)

3×3
\( \cdot \cdot \cdot \)

3×3
\( \cdot \cdot \cdot \)

\( C_{out} = 192 \)

4-Mode Weight Tensor
\( \mathcal{W} \in \mathbb{R}^{F_y \times F_x \times C_{in} \times C_{out}} \)

Input Channels × Weights per Filter
\( K = 3 \times 3 \times 128 = 1152 \)

Weight Matrix
\( W \in \mathbb{R}^{C_{out} \times F_y \times F_x \times C_{in}} \)

3×3
\( \cdot \cdot \cdot \)

3×3
\( \cdot \cdot \cdot \)

3×3
\( \cdot \cdot \cdot \)

\( C_{in} = 128 \)

3-Mode Input Tensor
\( \mathcal{X} \in \mathbb{R}^{H_1 \times W_1 \times C_{in}} \)

Input Matrix
\( X \in \mathbb{R}^{F_y \times F_x \times C_{in} \times H_2 \times W_2} \)

\( N = 28 \times 28 = 784 \)

\( C_{in} = 128 \)
Applies to Any-Dimensional Convolution

Conv1D: Speech, biomedical data classification, anomaly detection
Applies to Any-Dimensional Convolution

Conv1D: Speech, biomedical data classification, anomaly detection
Conv2D: Image classification, object detection
Applies to Any-Dimensional Convolution

Conv1D: Speech, biomedical data classification, anomaly detection
Conv2D: Image classification, object detection
Conv3D: Medical imaging, video analytics
Applies to Any-Dimensional Convolution

Conv1D: Speech, biomedical data classification, anomaly detection
Conv2D: Image classification, object detection
Conv3D: Medical imaging, video analytics
Conv4D: Light-field imaging for material recognition
Applies to Any-Dimensional Convolution

Conv1D: Speech, biomedical data classification, anomaly detection
Conv2D: Image classification, object detection
Conv3D: Medical imaging, video analytics
Conv4D: Light-field imaging for material recognition

\[ \mathbf{y} = \mathbf{w} \mathbf{x} \]
Applies to Any-Dimensional Convolution

Conv1D: Speech, biomedical data classification, anomaly detection
Conv2D: Image classification, object detection
Conv3D: Medical imaging, video analytics
Conv4D: Light-field imaging for material recognition

\[ Y = WX \]

Output Matrix
\[ Y \in \mathbb{R}^{C_{out} \times (O_1 \cdots O_N)} \]

Weight Matrix
\[ W \in \mathbb{R}^{C_{out} \times (F_1 \cdots F_N)C_{in}} \]

Input Matrix
\[ X \in \mathbb{R}^{(F_1 \cdots F_N)C_{in} \times (I_1 \cdots I_N)} \]

\[ Y_{\text{N-mode output tensor}} \in \mathbb{R}^{O_1 \times \cdots \times O_N \times C_{out}} \]

\[(N+1)\text{-Mode Weight Tensor} \quad W_{\in \mathbb{R}^{F_1 \times \cdots \times F_N \times C_{in} \times C_{out}}} \]

\[ X_{\text{N-mode input tensor}} \in \mathbb{R}^{I_1 \times \cdots \times I_N \times C_{in}} \]
Applies to Any-Dimensional Convolution

Conv1D: Speech, biomedical data classification, anomaly detection
Conv2D: Image classification, object detection
Conv3D: Medical imaging, video analytics
Conv4D: Light-field imaging for material recognition

\( \mathbf{Y} = \mathbf{W} \mathbf{X} \)

Output Matrix
\( \mathbf{Y} \in \mathbb{R}^{C_{out} \times (O_1 \cdots O_N)} \)

Weight Matrix
\( \mathbf{W} \in \mathbb{R}^{C_{out} \times (F_1 \cdots F_N)C_{in}} \)

Input Matrix
\( \mathbf{X} \in \mathbb{R}^{(F_1 \cdots F_N)C_{in} \times (I_1 \cdots I_N)} \)

Basically, parenthesize axes to make both hardware and software happy
Decide What You Need to Optimize in Hardware

> Convolution parameters and flavors
  >> Padding, strides, number of axes
  >> Dilated, transposed, depthwise-separable,…
  >> See Session M2, FPL 19.

> Winograd algorithm: patch size vs. speed-up vs. numerical stability

> Balance ease of use vs. energy efficiency using mixed data types

> Hardware-software co-design
  >> Good data movement instructions keep compute fed and software stable
  >> FPGAs provides adaptability, especially for non-linear layers
Natural-Language Processing
Tesla theorized that the application of electricity to the brain enhanced intelligence. In 1912, he crafted "a plan to make dull students bright by saturating them unconsciously with electricity," wiring the walls of a schoolroom and, "saturating [the schoolroom] with infinitesimal electric waves vibrating at high frequency. The whole room will thus, Mr. Tesla claims, be converted into a health-giving and stimulating electromagnetic field or 'bath.'" The plan was, at least provisionally approved by then superintendent of New York City schools, William H. Maxwell.
Stanford Question and Answering Dataset (SQuAD) 1.1 Quarterly High Scores

Chart compiled from SQuAD submissions as of 8/21/2019

1½ years to surpass human exact match

© Copyright 2019 Xilinx
Stanford Question and Answering Dataset (SQuAD) 1.1

SQuAD 1.1 Quarterly High Scores

• 1½ years to surpass human exact match
• ½ year more to surpass human F1

Chart compiled from SQuAD submissions as of 8/21/2019

© Copyright 2019 Xilinx
SQuAD 2.0 (with Unanswerable Questions)

SQuAD 2.0 Monthly High Scores

Score (%)

Max of Machine Exact Match
Max of Machine F1
Human Exact Match
Human F1

10 months to surpass human scores

Chart compiled from SQuAD submissions as of 8/21/2019
© Copyright 2019 Xilinx
Convolution vs. Attention

- Weights fixed after training
- Filter extent limited

- Weights computed on-the-fly
- Large extent (context)

You shall know a word by the company it keeps.

John R. Firth, 1957

Attention display created with Google Tensor2Tensor using the Transformer Base model.
# Attention Mechanism

## Associative Array or Content-Addressable Memory

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apple</td>
<td>Red</td>
</tr>
<tr>
<td>Grass</td>
<td>Green</td>
</tr>
<tr>
<td>Sky</td>
<td>Blue</td>
</tr>
</tbody>
</table>
# Attention Mechanism

What does this query match?

<table>
<thead>
<tr>
<th>Query</th>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grass</td>
<td>Apple</td>
<td>Red</td>
</tr>
<tr>
<td>Grass</td>
<td>Green</td>
<td></td>
</tr>
<tr>
<td>Sky</td>
<td></td>
<td>Blue</td>
</tr>
</tbody>
</table>
Attention Mechanism

Exact match between query and one of the keys

<table>
<thead>
<tr>
<th>Query</th>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grass</td>
<td>Apple</td>
<td>Red</td>
</tr>
<tr>
<td>Grass</td>
<td>Grass</td>
<td>Green</td>
</tr>
<tr>
<td>Sky</td>
<td></td>
<td>Blue</td>
</tr>
</tbody>
</table>
Attention Mechanism

Returns associated value. Each query is a one-hot vector.

<table>
<thead>
<tr>
<th>Query</th>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grass</td>
<td>Grass</td>
<td>Green</td>
</tr>
</tbody>
</table>
### Attention in Neural Networks

#### Attention: Soft Associative Array

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>[-0.2, 1.1, -0.9]</td>
<td>[235, 28, 35]</td>
</tr>
<tr>
<td>[0.4, 0.7, -1.1]</td>
<td>[30, 100, 14]</td>
</tr>
<tr>
<td>[-0.1, -1.2, 1.1]</td>
<td>[74, 126, 207]</td>
</tr>
</tbody>
</table>
## Attention in Neural Networks

What does this query match?

<table>
<thead>
<tr>
<th>Query</th>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>[-0.2, 1.1, -0.9]</td>
<td>[-0.2, 1.1, -0.9]</td>
<td>[235, 28, 35]</td>
</tr>
<tr>
<td>[0.4, 0.7, -1.1]</td>
<td>[0.4, 0.7, -1.1]</td>
<td>[30, 100, 14]</td>
</tr>
<tr>
<td>[-0.1, -1.2, 1.1]</td>
<td>[-0.1, -1.2, 1.1]</td>
<td>[74, 126, 207]</td>
</tr>
</tbody>
</table>
**Attention in Neural Networks**

Can we “soft-match?” Dot product measures similarity.

<table>
<thead>
<tr>
<th>Query</th>
<th>Key</th>
<th>Dot Product</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>[-0.2, 1.1, -0.9]</td>
<td>[-0.2, 1.1, -0.9]</td>
<td>2.1</td>
<td>[235, 28, 35]</td>
</tr>
<tr>
<td>[0.4, 0.7, -1.1]</td>
<td>[0.4, 0.7, -1.1]</td>
<td>1.7</td>
<td>[30, 100, 14]</td>
</tr>
<tr>
<td>[-0.1, -1.2, 1.1]</td>
<td>[-0.1, -1.2, 1.1]</td>
<td>-2.3</td>
<td>[74, 126, 207]</td>
</tr>
</tbody>
</table>
### Attention in Neural Networks

#### Exponentiate

<table>
<thead>
<tr>
<th>Query</th>
<th>Key</th>
<th>Exp</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>[-0.2, 1.1, -0.9]</td>
<td>[-0.2, 1.1, -0.9]</td>
<td>7.8</td>
<td>[235, 28, 35]</td>
</tr>
<tr>
<td>[0.4, 0.7, -1.1]</td>
<td>[0.4, 0.7, -1.1]</td>
<td>5.4</td>
<td>[30, 100, 14]</td>
</tr>
<tr>
<td>[-0.1, -1.2, 1.1]</td>
<td>[-0.1, -1.2, 1.1]</td>
<td>0.1</td>
<td>[74, 126, 207]</td>
</tr>
</tbody>
</table>
**Attention in Neural Networks**

Normalize. Softmax!

<table>
<thead>
<tr>
<th>Query</th>
<th>Key</th>
<th>Weight</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>[-0.2, 1.1, -0.9]</td>
<td>[-0.2, 1.1, -0.9]</td>
<td>58.9%</td>
<td>[235, 28, 35]</td>
</tr>
<tr>
<td>[0.4, 0.7, -1.1]</td>
<td>[0.4, 0.7, -1.1]</td>
<td>40.3%</td>
<td>[30, 100, 14]</td>
</tr>
<tr>
<td>[-0.1, -1.2, 1.1]</td>
<td>[-0.1, -1.2, 1.1]</td>
<td>0.76%</td>
<td>[74, 126, 207]</td>
</tr>
</tbody>
</table>
Attention in Neural Networks

Weighted sum of values:

<table>
<thead>
<tr>
<th>Query</th>
<th>Key</th>
<th>Weight</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>[-0.2, 1.1, -0.9]</td>
<td>[-0.2, 1.1, -0.9]</td>
<td>58.9%</td>
<td>[235, 28, 35]</td>
</tr>
<tr>
<td>[0.4, 0.7, -1.1]</td>
<td>[0.4, 0.7, -1.1]</td>
<td>40.3%</td>
<td>[30, 100, 14]</td>
</tr>
<tr>
<td>[-0.1, -1.2, 1.1]</td>
<td>[-0.1, -1.2, 1.1]</td>
<td>0.76%</td>
<td>[74, 126, 207]</td>
</tr>
</tbody>
</table>
Attention in Neural Networks

In matrix form for multiple queries in parallel hardware:

Attention \((Q, K, V) = \text{softmax} (QK^T)\)

Need more softmax than typical vision neural networks. Can add in ACAP/FPGA.
Word Embeddings

Words represented as vectors (>500 components)
“You shall know a word by the company it keeps.”

Word analogy
\[ \text{is} : \text{was} :: ? : \text{were} \]

© Copyright 2019 Xilinx
Word Embeddings

Word analogy

is : was :: ? : were

is - was

is

was

were


© Copyright 2019 Xilinx
Word analogy

is : was :: ? : were

is - was

is

is - was + were

was

were

Word Embeddings

Word analogy

is : was :: are : were

Word Embeddings

Word analogy

is : was :: are : were

Embedding analogy

is – was ≈ are – were

© Copyright 2019 Xilinx
Word Embeddings

Word analogy

is : was :: are : were

Embedding analogy

is – was \approx are – were


© Copyright 2019 Xilinx
Model Sizes

> Embedding storage in Translation Networks
  >> Transformer Base: ~32K × 512 fp32
  >> OpenAI GPT-2: ~50K × 1600 fp32
  >> Above examples considered small
  >> Can convert fp32 to 8-bit or 16-bit data types

> Model sizes (Number of parameters)
  >> Transformer Base: 61M
  >> OpenAI GPT-2: 1.6B

> Instruction set and tensor SRAM tuned for random & burst HBM

> Beam search

Transformer: Vaswani et al., “Attention Is All You Need,” NIPS 2017
OpenAI GPT-2: Radford et al., “Language Models are Unsupervised Multitask Learners,” OpenAI 2019
https://d4mucpksyw.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf
Combining Vision and Language
From Image Classification to Object Detection
Is the bowl to the right of the green apple?

What type of fruit in the image is round?
Era of Easy Scaling Over

- Moore’s Law
  Transistor improvement slows

- Amdahl’s Law
  Multicore not enough

- Dennard Scaling
  Power density rises

Source: John Hennessy and David Patterson, *Computer Architecture: A Quantitative Approach*, 6/e. 2018

© Copyright 2019 Xilinx
Built to Last

> Model commands after NN framework function calls
  >> Linear layers: matrix multiplication & convolution
  >> Attention is powerful and good for parallelism
  >> Beam search in NLP: Amdahl’s Law
  >> Activation functions, N-D pooling, layer normalization, NMS, LSTM, RNN, GRU, ...

> Compute is easy. Memory is hard and is key to a adaptable design.

> Neural networks do merge, e.g. captioning, visual reasoning

> ML moves fast. Plan instruction superset upfront for adaptability.

> Implement what you need today. Reconfigure HW to adapt to changes.
Thank you