Neural Network Inference with NNEF

May 2018

Radhakrishna Giduthuri | Editor, OpenVX Working Group
radha.giduthuri@amd.com | @RadhaGiduthuri
Agenda

• Neural network ecosystem and inference
• OpenVX tensor and neural network function APIs
• Neural network models and API mapping
• NNEF-based inference workflow
• Available open-source tools
• Example: Tensorflow (VGG16 model) → NNEF → OpenVX → application
Neural network ecosystem and inference

- Problem definition
- Datasets
- Neural Network Architecture
  - Neural Net Training Frameworks
  - Trained Models
  - Desktop and Cloud Hardware
  - Vision and Neural Net Inferencing Runtime
  - Vision/AI Applications
  - Diverse Inferencing Acceleration Hardware
  - FPGA
  - DSP
  - CPU
  - GPU
  - Custom Hardware
OpenVX tensor object

- **vx_tensor** is a multi-dimensional array that supports at least 4 dimensions

1-D tensor
e.g., 6 element vector

2-D tensor
e.g., 6 by 6 matrix

3-D tensor
e.g., dimensions [6, 6, 4]

4-D tensor
e.g., dimensions [6, 6, 4, 3]

**Data type of tensor data elements**

- **VX_TYPE_UINT8**
  
<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
  
  signed integer (8 bits)

- **VX_TYPE_INT8**
  
<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
  
  signed integer (8 bits)

- **VX_TYPE_INT16**
  
<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
  
  signed integer (8 bits) | fraction (8 bits)

- **VX_TYPE_FLOAT32**
  
  | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
  |-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
  | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
  
  sign exponent (8 bits) | fraction (23 bits)

[0.15625] = 0.15625

[1] core specification does not mandate float-point support
OpenVX tensor APIs

- **Tensor creation**
  
  ```
  vx_tensor vxCreateTensor(context, no_of_dims, dims, data_type, fixed_point_pos);
  vx_status vxReleaseTensor(&tensor);
  ```

- **Tensor objects for intermediate results**
  
  ```
  vx_tensor vxCreateVirtualTensor(graph, no_of_dims, dims, data_type, fixed_point_pos);
  ```

- **Tensor attributes**
  
  ```
  VX_TENSOR_NUMBER_OF_DIMS
  VX_TENSOR_DIMS
  VX_TENSOR_DATA_TYPE
  VX_TENSOR_FIXED_POINT_POSITION
  ```

Input: 4-D [224, 224, 3, batches]
Feature maps: 4-D [96, 96, 32, batches] - intermediate result
Output: 2-D [1024, batches]
OpenVX tensor APIs

• Copy a patch from/into a tensor object

```c
vx_status vxCopyTensorPatch(tensor,
   vx_size number_of_dims, // number of patch dimensions (i.e., tensor dim count)
   vx_size view_start[], // patch start point in each dimension
   vx_size view_end[], // patch end point in each dimension
   vx_size user_stride[], // user memory strides in each dimension
   void * user_ptr, // address of the memory location
   vx_enum usage, // access mode: VX_READ_ONLY or VX_WRITE_ONLY
   vx_enum user_memory_type // memory type: VX_MEMORY_TYPE_TYPE_HOST
);
```

3-D tensor: [6, 6, 4] INT16

view_start = [2, 2, 0, 0]
view_end = [6, 5, 3, 1]
user_stride = [2, 12, 72, 288]
OpenVX tensor operations

- Basic element-wise tensor operations
  - `vxTensorAddNode`
  - `vxTensorSubtractNode`
  - `vxTensorMultiplyNode`
  - `vxTensorConvertDepthNode`
  - `vxTensorTableLookupNode`

- Basic tensor-level operations
  - `vxTensorTransposeNode`
  - `vxTensorMatrixMultiplyNode`

- Tensor types supported: INT16(Q8.8), INT8(Q8.0), and UINT8(UQ8.0)
  - Other types may be supported by a vendor.
OpenVX neural network extension

- Eight neural network “layer” nodes

<table>
<thead>
<tr>
<th>vxActivationLayer</th>
<th>vxConvolutionLayer</th>
<th>vxDeconvolutionLayer</th>
</tr>
</thead>
<tbody>
<tr>
<td>vxFullyConnectedLayer</td>
<td>vxNormalizationLayer</td>
<td>vxPoolingLayer</td>
</tr>
<tr>
<td>vxSoftmaxLayer</td>
<td>vxROIPoolingLayer</td>
<td>...</td>
</tr>
</tbody>
</table>

- Tensor types supported: INT16(Q8.8), INT8(Q8.0), and UINT8(UQ8.0)
  - Other types may be supported by a vendor

- Example supported networks
  - Overfeat, Alexnet, GoogLeNet versions (Inception)
  - LSTM, RNN/BiRNN, Faster-RCNN, FCN

- Conformance tests will be up to some “tolerance” in precision
  - To allow for optimizations, e.g., weight compression
Example: residual building block in ResNet

Useful tensor operations:
- vxConvolutionLayer
- vxActivationLayer
- vxTensorAddNode
Example: Recurrent Neural Network: LSTM

A. Forward Pass

Let $x^t$ be the input vector at time $t$, $N$ be the number of LSTM blocks and $M$ the number of inputs. Then we get the following weights for an LSTM layer:

- Input weights: $W_z, W_i, W_f, W_o \in \mathbb{R}^{N \times M}$
- Recurrent weights: $R_z, R_i, R_f, R_o \in \mathbb{R}^{N \times N}$
- Peephole weights: $p_i, p_f, p_o \in \mathbb{R}^N$
- Bias weights: $b_z, b_i, b_f, b_o \in \mathbb{R}^N$

Then the vector formulas for a vanilla LSTM layer forward pass can be written as:

$$
\begin{align*}
\bar{z}^t &= W_z x^t + R_z y^{t-1} + b_z & \text{block input} \\
\bar{z}^t &= g(\bar{z}^t) \\
i^t &= W_i x^t + R_i y^{t-1} + p_i \odot c^{t-1} + b_i & \text{input gate} \\
i^t &= \sigma(i^t) \\
\bar{f}^t &= W_f x^t + R_f y^{t-1} + p_f \odot c^{t-1} + b_f & \text{forget gate} \\
f^t &= \sigma(\bar{f}^t) \\
c^t &= \bar{z}^t \odot i^t + c^{t-1} \odot f^t & \text{cell} \\
o^t &= W_o x^t + R_o y^{t-1} + p_o \odot c^t + b_o & \text{output gate} \\
o^t &= \sigma(o^t) \\
y^t &= h(c^t) \odot o^t & \text{block output}
\end{align*}
$$

How to handle manage $(c^t, c^{t-1})$ & $(y^t, y^{t-1})$ swap? - use vx_delay object with vxAgeDelay API
OpenVX delay object is useful for RNNs

```
vx_delay vxCreateDelay(
    vx_context context,
    vx_reference exemplar,
    vx_size count
);
```

Example:
```
vx_tensor exemplar = vxCreateTensor(context, ...);
vx_delay tensor_delay = vxCreateDelay(context, (vx_reference)exemplar, 2);
vxReleaseTensor(&exemplar);

...  
vx_tensor tensor_0 = (vx_tensor)vxGetReferenceFromDelay(tensor_delay, 0);
vx_tensor tensor_1 = (vx_tensor)vxGetReferenceFromDelay(tensor_delay, -1);

...  
vxAgeDelay(tensor_delay);
```
Example: LSTM with Delay object

A. Forward Pass

Let $x^t$ be the input vector at time $t$, $N$ be the number of LSTM blocks and $M$ the number of inputs. Then we get the following weights for an LSTM layer:

- Input weights: $W_z, W_i, W_f, W_o \in \mathbb{R}^{N \times M}$
- Recurrent weights: $R_z, R_i, R_f, R_o \in \mathbb{R}^{N \times N}$
- Peephole weights: $p_i, p_f, p_o \in \mathbb{R}^{N}$
- Bias weights: $b_z, b_i, b_f, b_o \in \mathbb{R}^{N}$

Then the vector formulas for a vanilla LSTM layer forward pass can be written as:

\[
\begin{align*}
\tilde{z}^t &= W_z x^t + R_z y^{t-1} + b_z \\
\tilde{z}^t &= g(\tilde{z}^t) \\
\tilde{i}^t &= W_i x^t + R_i y^{t-1} + p_i \odot c^{t-1} + b_i \\
\tilde{i}^t &= \sigma(\tilde{i}^t) \\
\tilde{f}^t &= W_f x^t + R_f y^{t-1} + p_f \odot c^{t-1} + b_f \\
\tilde{f}^t &= \sigma(\tilde{f}^t) \\
c^t &= \tilde{z}^t \odot \tilde{i}^t + c^{t-1} \odot f^t \\
\tilde{o}^t &= W_o x^t + R_o y^{t-1} + p_o \odot c^t + b_o \\
\tilde{o}^t &= \sigma(\tilde{o}^t) \\
y^t &= h(c^t) \odot o^t
\end{align*}
\]

Handle manage $(c^t, c^{t-1}) \& (y^t, y^{t-1})$ swap using \texttt{vx\_delay} object and \texttt{vxAgeDelay} API.
Example: adding LSTM to OpenVX graph

A. Forward Pass

Let \( x^t \) be the input vector at time \( t \), \( N \) be the number of LSTM blocks and \( M \) the number of inputs. Then we get the following weights for an LSTM layer:

- Input weights: \( W_z, W_i, W_f, W_o \in \mathbb{R}^{N \times M} \)
- Recurrent weights: \( R_z, R_i, R_f, R_o \in \mathbb{R}^{N \times N} \)
- Peephole weights: \( p_i, p_f, p_o \in \mathbb{R}^N \)
- Bias weights: \( b_z, b_i, b_f, b_o \in \mathbb{R}^N \)

Then the vector formulas for a vanilla LSTM layer forward pass can be written as:

\[
\begin{align*}
\hat{z}^t &= W_z x^t + R_z y^{t-1} + b_z \\
\hat{z}^t &= g(\hat{z}^t) \\
\hat{i}^t &= W_i x^t + R_i y^{t-1} + p_i \odot c^{t-1} + b_i \\
f^t &= \sigma(\hat{f}^t) \\
\hat{f}^t &= W_f x^t + R_f y^{t-1} + p_f \odot c^{t-1} + b_f \\
f^t &= \sigma(\hat{f}^t) \\
c^t &= \hat{z}^t \odot \hat{i}^t + c^{t-1} \odot f^t \\
\hat{o}^t &= W_o x^t + R_o y^{t-1} + p_o \odot c^t + b_o \\
o^t &= \sigma(\hat{o}^t) \\
y^t &= h(c^t) \odot o^t
\end{align*}
\]

Useful tensor operations:
- \texttt{vxTensorMatrixMultiplyNode} (8)
- \texttt{vxTensorMultiplyNode} (6)
- \texttt{vxTensorAddNode} (4)
- \texttt{vxActivationLayer} (5)

* most vendors prefer adding a separate LSTM layer for better performance tuning
Example: adding LSTM to OpenVX graph

A. Forward Pass

Let $x^t$ be the input vector at time $t$, $N$ be the number of LSTM blocks and $M$ the number of inputs. Then we get the following weights for an LSTM layer:

- Input weights: $W_z, W_i, W_f, W_o \in \mathbb{R}^{N \times M}$
- Recurrent weights: $R_z, R_i, R_f, R_o \in \mathbb{R}^{N \times N}$
- Peephole weights: $p_i, p_f, p_o \in \mathbb{R}^N$
- Bias weights: $b_z, b_i, b_f, b_o \in \mathbb{R}^N$

Then the vector formulas for a vanilla LSTM layer forward pass can be written as:

$$
\begin{align*}
\tilde{z}^t &= W_z x^t + R_z y^{t-1} + b_z \\
\tilde{z}^t &= g(\tilde{z}^t) \\
\tilde{i}^t &= W_i x^t + R_i y^{t-1} + p_i \odot c^{t-1} + b_i \\
\tilde{i}^t &= \sigma(\tilde{i}^t) \\
\tilde{f}^t &= W_f x^t + R_f y^{t-1} + p_f \odot c^{t-1} + b_f \\
\tilde{f}^t &= \sigma(\tilde{f}^t) \\
\tilde{o}^t &= W_o x^t + R_o y^{t-1} + p_o \odot c^t + b_o \\
\tilde{o}^t &= \sigma(\tilde{o}^t) \\
y^t &= h(c^t) \odot o^t
\end{align*}
$$
Example: VGG-16 Model

Useful tensor operations:
- vxConvolutionLayer
- vxActivationLayer
- vxPoolingLayer
- vxFullyConnectedLayer
- vxSoftmaxLayer
Complete inference workflow example

**Steps**
- Download VGG-16 pre-trained tensorflow model
- Convert tensorflow model into NNEF container
- Generate C code to create an OpenVX graph
- Build a test application to run inference
Open-source tools and example

• Convert tensorflow model into NNEF container
  - https://github.com/KhronosGroup/NNEF-Tools
    [converter/tensorflow]

• Generate C code to create OpenVX graph from NNEF container
    [utils/inference_generator]

• Examples
  - https://github.com/rgiduthuri/openvx_tutorial
Example walkthrough & wrap-up