Hardware Architectures for Deep Neural Networks

ISCA Tutorial

June 22, 2019

Website: http://eyeriss.mit.edu/tutorial.html
Speakers and Contributors

Joel Emer  
Senior Distinguished Research Scientist  
NVIDIA

Vivienne Sze  
Professor  
MIT

Yu-Hsin Chen  
PhD Graduate  
MIT

Tien-Ju Yang  
PhD Candidate  
MIT
Outline (9AM-5PM)

• Overview of Deep Neural Networks
• DNN Kernel Computation
• DNN Accelerators
• DNN Model and Hardware Co-Design
• DNN Processing In/Near Memory
• Sparse DNN Accelerators
Participant Takeaways

• Understand the key design considerations for DNNs

• Be able to evaluate different implementations of DNN with benchmarks and comparison metrics

• Understand the tradeoffs between various architectures and platforms

• Assess the utility of various optimization approaches

• Understand recent implementation trends and opportunities
Growth of Machine Learning Workloads

Adopted from [J. Dean et al., IEEE Micro 2018]

Tutorial aims to cover **key concepts**, provide **insights** and **highlight trends** rather than be a comprehensive survey all work
Resources

• Eyeriss Project: [http://eyeriss.mit.edu](http://eyeriss.mit.edu)
  – Tutorial Slides
  – Energy modeling


**Synthesis Lecture Book coming soon!** *(Estimate end of summer)*

– Mailing List for updates
Background of Deep Neural Networks
Artificial Intelligence

“The science and engineering of creating intelligent machines”
- John McCarthy, 1956
“Field of study that gives computers the ability to learn without being explicitly programmed”

– Arthur Samuel, 1959
An algorithm that takes its basic functionality from our understanding of how the brain operates.
How Does the Brain Work?

- The basic computational unit of the brain is a neuron → 86B neurons in the brain
- Neurons are connected with nearly $10^{14} – 10^{15}$ synapses
- Neurons receive input signal from dendrites and produce output signal along axon, which interact with the dendrites of other neurons via synaptic weights
- Synaptic weights – learnable & control influence strength
Spiking-based Machine Learning

Artificial Intelligence

Machine Learning

Brain-Inspired

Spiking
Spiking Architecture

- Brain-inspired
- Integrate and fire
- Example: IBM TrueNorth

[Merolla et al., Science 2014; Esser et al., PNAS 2016]
Machine Learning with Neural Networks

Artificial Intelligence

Machine Learning

Brain-Inspired

Spiking

Neural Networks
Neural Networks: Weighted Sum
Many Weighted Sums

Input Layer

Hidden Layer

Output Layer

Weights
Deep Learning

Artificial Intelligence

Machine Learning

Brain-Inspired

Spiking

Neural Networks

Deep Learning
What is Deep Learning?

Image Source: [Lee et al., Comm. ACM 2011]
Overview of Deep Neural Networks
DNN Terminology 101

Input Layer

Hidden Layer

Output Layer

Neurons
DNN Terminology 101

Synapses

Input Layer
Hidden Layer
Output Layer
DNN Terminology 101

Each synapse has a **weight** for neuron **activation**

\[
Y_j = \text{activation} \left( \sum_{i=1}^{3} W_{ij} \times X_i \right)
\]

Input Layer

\( X_1 \)

\( X_2 \)

\( X_3 \)

Hidden Layer

\( Y_1 \)

\( Y_2 \)

\( Y_3 \)

\( Y_4 \)

Output Layer

\( W_{11} \)

\( W_{34} \)
Weight Sharing: multiple synapses use the same weight value

\[ Y_j = \text{activation}\left(\sum_{i=1}^{3} W_{ij} \times X_i\right) \]
DNN Terminology 101

L1 Neuron inputs
e.g. image pixels

Layer 1
L1 Neuron outputs
a.k.a. Activations

Input Layer

Hidden Layer

Output Layer
DNN Terminology 101

Input Layer

Hidden Layer

Output Layer

L2 Input
Activations

Layer 2

L2 Output
Activations

L2 Input
Activations
DNN Terminology 101

**Fully-Connected**: all i/p neurons connected to all o/p neurons

**Sparsely-Connected**

Input Layer -> Hidden Layer -> Output Layer
DNN Terminology 101

Feed Forward

Input Layer

Hidden Layer

Feedback

Output Layer
Popular Types of DNNs

• Fully-Connected NN
  – feed forward, a.k.a. multilayer perceptron (MLP)

• Convolutional NN (CNN)
  – feed forward, sparsely-connected w/ weight sharing

• Recurrent NN (RNN)
  – feedback

• Long Short-Term Memory (LSTM)
  – feedback + storage
Inference vs. Training

- **Training:** Determine weights
  - **Supervised:**
    - Training set has inputs and outputs, i.e., labeled
  - **Unsupervised / Self-Supervised:**
    - Training set is unlabeled
  - **Semi-supervised:**
    - Training set is partially labeled
  - **Reinforcement:**
    - Output assessed via rewards and punishments

- **Inference:** Apply weights to determine output
Deep Convolutional Neural Networks

Modern Deep CNN: 5 – 1000 Layers

\[ \text{CONV Layer} \rightarrow \text{Low-Level Features} \rightarrow \ldots \rightarrow \text{CONV Layer} \rightarrow \text{High-Level Features} \rightarrow \text{FC Layer} \rightarrow \text{Classes} \]

1 – 3 Layers
Deep Convolutional Neural Networks

CONV Layer → Low-Level Features → … → CONV Layer → High-Level Features → FC Layer → Classes

Convolution

Activation
Deep Convolutional Neural Networks

CONV Layer → Low-Level Features → ... → CONV Layer → High-Level Features → FC Layer → Classes

Fully Connected

Activation
Deep Convolutional Neural Networks

Optional layers in between CONV and/or FC layers

CONV Layer → NORM Layer → POOL Layer → CONV Layer → High-Level Features → FC Layer → Classes

Normalization

Pooling
Deep Convolutional Neural Networks

Convolutions account for more than 90% of overall computation, dominating runtime and energy consumption.
Convolution (CONV) Layer

a plane of input activations
a.k.a. **input feature map (fmap)**

filter (weights)
Convolution (CONV) Layer

Element-wise Multiplication

input fmap

filter (weights)
Convolution (CONV) Layer

- **Filter (weights)**
- **Element-wise Multiplication**
- **Partial Sum (psum)**
- **Accumulation**
- **Input fmap**
- **Output fmap**
- **An output activation**
Convolution (CONV) Layer

Sliding Window Processing

filter (weights)

input fmap

output fmap

an output activation

MIT
Convolution (CONV) Layer

Many Input Channels (C)
Convolution (CONV) Layer

- Many filters (M)
- Input fmap
- Output fmap
- Many output channels (M)
Convolution (CONV) Layer

Many Input fmaps (N)

Many Output fmaps (N)

filters
CNN Decoder Ring

- N – Number of input fmaps/output fmaps (batch size)
- C – Number of 2-D input fmaps/filters (channels)
- H – Height of input fmap (activations)
- W – Width of input fmap (activations)
- R – Height of 2-D filter (weights)
- S – Width of 2-D filter (weights)
- M – Number of 2-D output fmaps (channels)
- E – Height of output fmap (activations)
- F – Width of output fmap (activations)
CONV Layer Tensor Computation

\[
O[n][m][x][y] = \text{Activation}(B[m] + \sum_{i=0}^{R-1} \sum_{j=0}^{S-1} \sum_{k=0}^{C-1} I[n][k][Ux + i][Uy + j] \times W[m][k][i][j]),
\]

\[
0 \leq n < N, 0 \leq m < M, 0 \leq y < E, 0 \leq x < F,
\]

\[
E = (H - R + U)/U, F = (W - S + U)/U.
\]

<table>
<thead>
<tr>
<th>Shape</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(N)</td>
<td>fmap batch size</td>
<td></td>
</tr>
<tr>
<td>(M)</td>
<td># of filters / # of output fmap channels</td>
<td></td>
</tr>
<tr>
<td>(C)</td>
<td># of input fmap/filter channels</td>
<td></td>
</tr>
<tr>
<td>(H/W)</td>
<td>input fmap height/width</td>
<td></td>
</tr>
<tr>
<td>(R/S)</td>
<td>filter height/width</td>
<td></td>
</tr>
<tr>
<td>(E/F)</td>
<td>output fmap height/width</td>
<td></td>
</tr>
<tr>
<td>(U)</td>
<td>convolution stride</td>
<td></td>
</tr>
</tbody>
</table>
Naïve 7-layer for-loop implementation:

```plaintext
for (n=0; n<N; n++) {
    for (m=0; m<M; m++) {
        for (x=0; x<F; x++) {
            for (y=0; y<E; y++) {
                O[n][m][x][y] = B[m];
                for (i=0; i<R; i++) {
                    for (j=0; j<S; j++) {
                        O[n][m][x][y] += I[n][k][Ux+i][Uy+j] × W[m][k][i][j];
                    }
                }
                O[n][m][x][y] = Activation(O[n][m][x][y]);
            }
        }
    }
}
```
Traditional Activation Functions

**Sigmoid**

\[ y = \frac{1}{1+e^{-x}} \]

**Hyperbolic Tangent**

\[ y = \frac{e^x - e^{-x}}{e^x + e^{-x}} \]
Modern Activation Functions

Rectified Linear Unit (ReLU)

$y = \max(0, x)$

Leaky ReLU

$y = \max(\alpha x, x)$

Exponential LU

$y = \begin{cases} x, & x \geq \theta \\ \alpha (e^x - 1), & x < \theta \end{cases}$

$\alpha = \text{small const. (e.g. 0.1)}$

Image Source: Caffe Tutorial
FC Layer – from CONV Layer POV

filters

input fmaps

output fmaps
Fully-Connected (FC) Layer

- Height and width of output fmaps are 1 (E = F = 1)
- Filters as large as input fmaps (R = H, S = W)
- Implementation: **Matrix Multiplication**

\[
M_{CHW} \times N = M_{CHW}
\]

Filters

Input fmaps

Output fmaps
Pooling (POOL) Layer

- Reduce resolution of each channel independently
- Overlapping or non-overlapping → depending on stride

Increases translation-invariance and noise-resilience

Image Source: Caffe Tutorial
POOL Layer Implementation

Naïve 6-layer for-loop max-pooling implementation:

```c
for (n=0; n<N; n++) {
    for (m=0; m<M; m++) {
        for (x=0; x<F; x++) {
            for (y=0; y<E; y++) {
                max = -Inf;
                for (i=0; i<R; i++) {
                    for (j=0; j<S; j++) {
                        if (I[n][m][Ux+i][Uy+j] > max) {
                            max = I[n][m][Ux+i][Uy+j];
                        }
                    }
                }
                O[n][m][x][y] = max;
            }
        }
    }
}
```

for each pooled value

find the max with in a window
Normalization (NORM) Layer

- **Batch Normalization (BN)**
  - Normalize activations towards mean=0 and std. dev.=1 based on the statistics of the training dataset
  - put in between CONV/FC and Activation function

Believed to be key to getting high accuracy and faster training on very deep neural networks.

[Ioffe et al., ICML 2015]
BN Layer Implementation

- The normalized value is further scaled and shifted, the parameters of which are learned from training.

\[
y = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} \gamma + \beta
\]

- Data mean
- Learned scale factor
- Data std. dev.
- Learned shift factor
- Small const. to avoid numerical problems
Relevant Components for Tutorial

• Typical operations that we will discuss:
  – Convolution (CONV)
  – Fully-Connected (FC)
  – Max Pooling
  – ReLU