Model Compression

영남대학교
차세대 컴퓨터 시스템 연구실
석사과정생 이원호

Model compression and hardware acceleration for neural networks: A comprehensive survey.

Proceedings of the IEEE, 2020

차세대 컴퓨터 시스템 연구실

이원호

Basic Model Architecture

- MLP (Multi-Layer Perceptron)
  - Feedforward Fully Connected (FC)
  - Efficient processing 1-dimensional features

- CNN (Convolutional Neural Network)
  - Efficient processing 2-dimensional features

- RNN (Recurrent Neural Network)
  - Efficient processing sequence features
  - Handling temporal dynamics

Basic Model Architecture

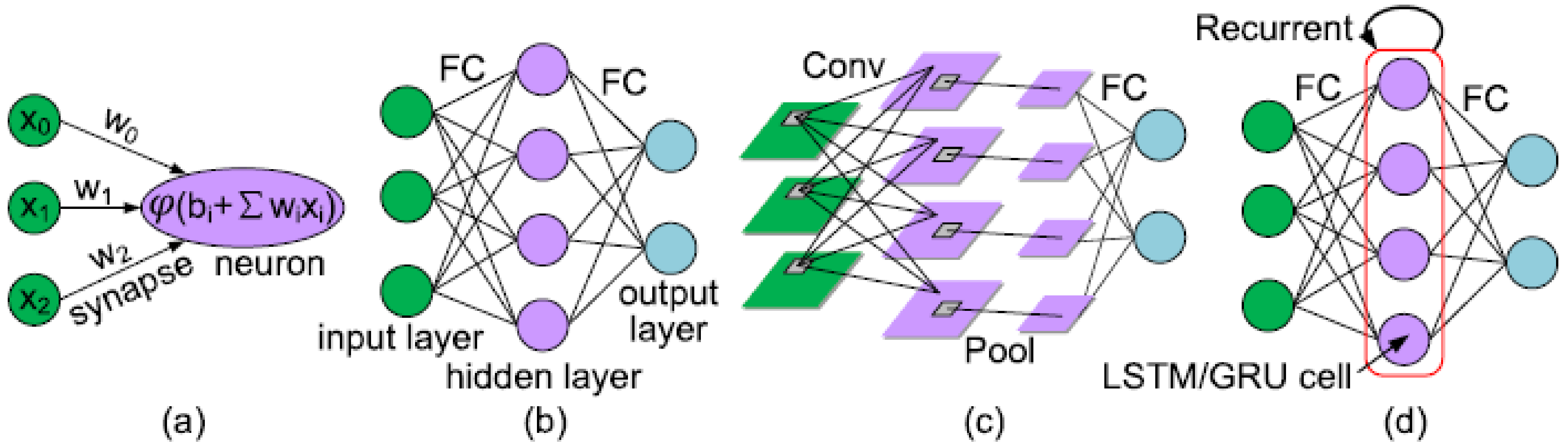


Fig. 2. *Neural network structures. (a) Single neuron. (b) MLP. (c) CNN. (d) RNN.*

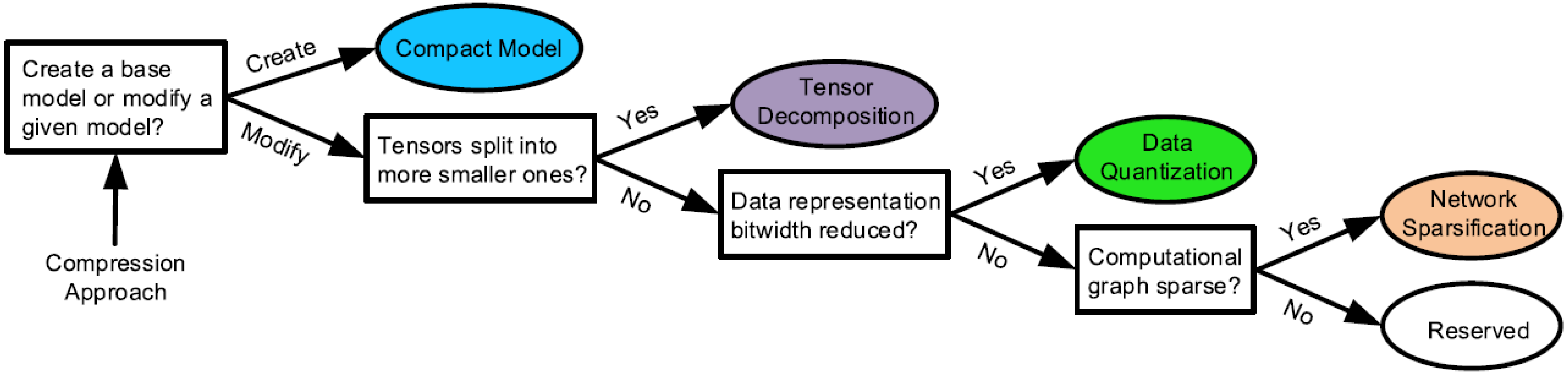
Compression Approaches

- Compact model
  - Design smaller base models that can still achieve acceptable application accuracy
- Tensor decomposition
  - Decompose the bloated parameters into a series of smaller matrices or tensors
- Data Quantization
  - Reduce the number of data bits
- Network sparsification
  - Reduce the connections/neurons to compress the original model

Compression Approaches

- **Compact model**
  - Design smaller base models that can still achieve acceptable application accuracy
- **Tensor decomposition**
  - Decompose the bloated parameters into a series of smaller matrices or tensors
- Data Quantization
  - Reduce the number of data bits
- Network sparsification
  - Reduce the connections/neurons to compress the original model

Taxonomy of neural network compression approcaches


Create a base model or modify a given model?
Compression Approach
Create
Compact Model
Modify
Tensors split into more smaller ones?
Yes
Tensor Decomposition
No
Data representation bitwidth reduced?
Yes
Data Quantization
No
Computational graph sparse?
Yes
Network Sparsification
No
Reserved

Compact Model

- Compact CNNs
- Compact RNNs
- Neural Architecture Search (NAS)

Compact CNNs

- CNN의 특징: 공유 가중치를 통한 지역 연결성 (Spartial correlation)

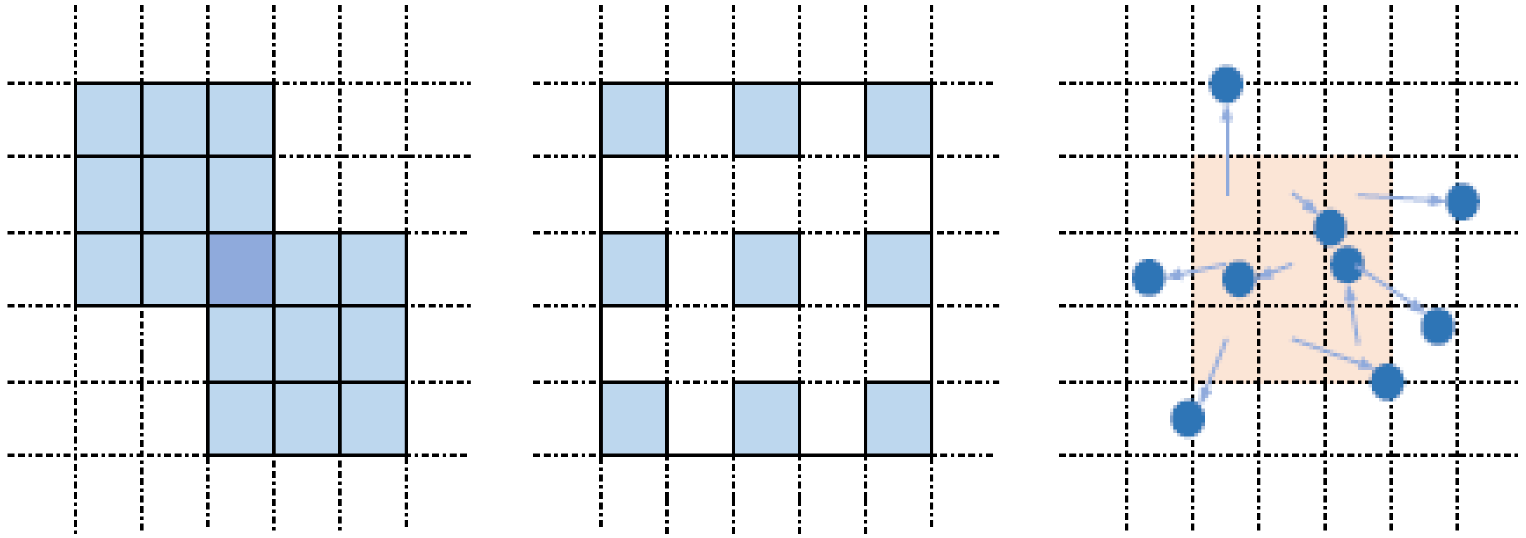


Fig. 5. *(a) Sampling locations in standard Conv (stacked 3x3 conv). (b) and (c) Sampling locations in atrous Conv (dilated 3 × 3 conv and deformable 3 × 3 conv, resp.).*

Compact CNNs

- Atrous Conv
  - Irregular kernels with holes dilations
- Deformable Conv
  - Generalization of atrous Conv
  - Add 2D offset factors

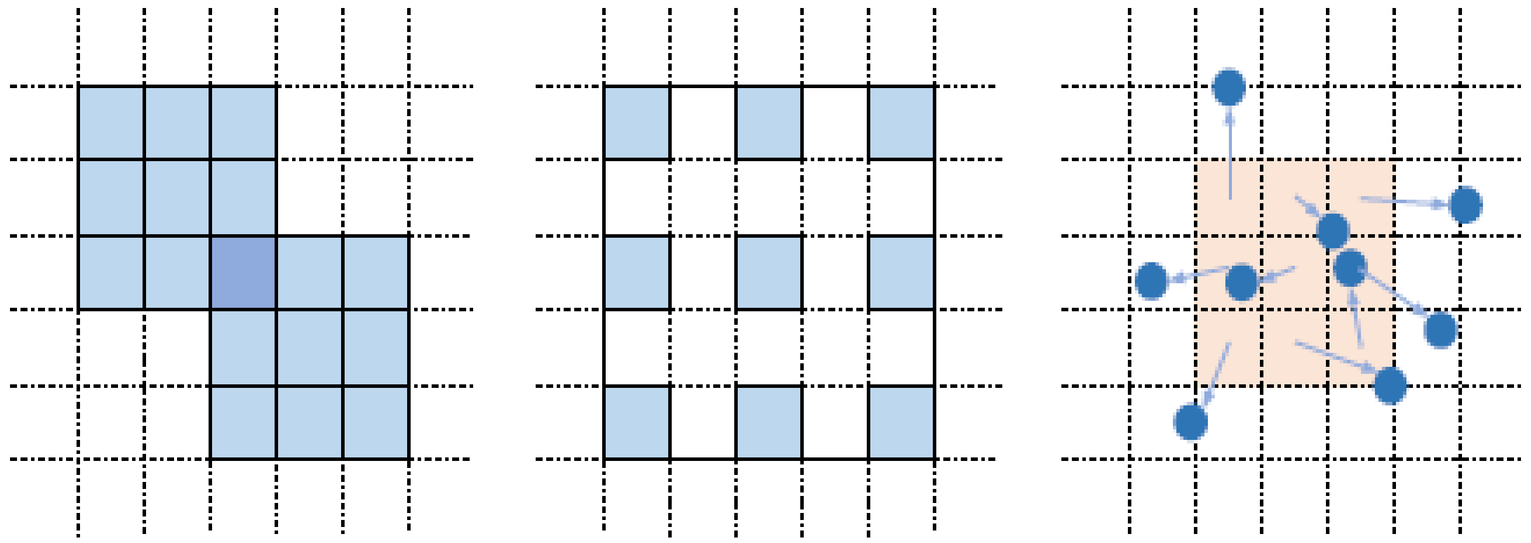


Fig. 5. *(a) Sampling locations in standard Conv (stacked 3x3 conv). (b) and (c) Sampling locations in atrous Conv (dilated 3 × 3 conv and deformable 3 × 3 conv, resp.).*

Compact CNNs

- 특징 맵이 다른 형태의 토폴로지로 연결될 때 생성 발생하는 관계
  (interlayer channel correlation)

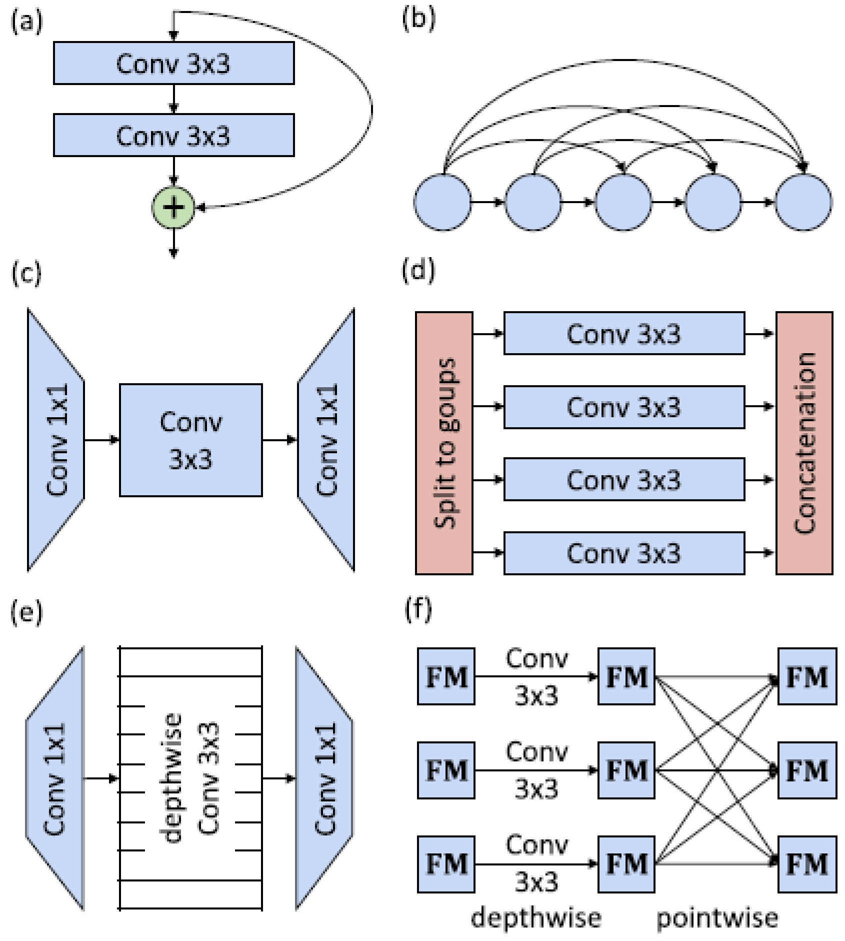


Fig. 6. *Typical interlayer and intralayer channel correlations. (a) Residual connection [30]. (b) Densely connected block [31]. (c) Bottleneck architecture [22]. (d) Group Conv [33]. (e) Reversed bottleneck architecture with depthwise Conv in between [3]. (f) Depthwise-separable Conv [34].*

Compact RNNs (Unit Level)

- LSTM
  - Vanilla RNN에 비해 장기 의존성 향상
- GRU
  - Forget gate + input gate → update gate
  - 불필요한 연결성 제거
- MGU
  - GRU에서 reset gate 와 update gate를 결합

Compact RNNs (Unit Level)

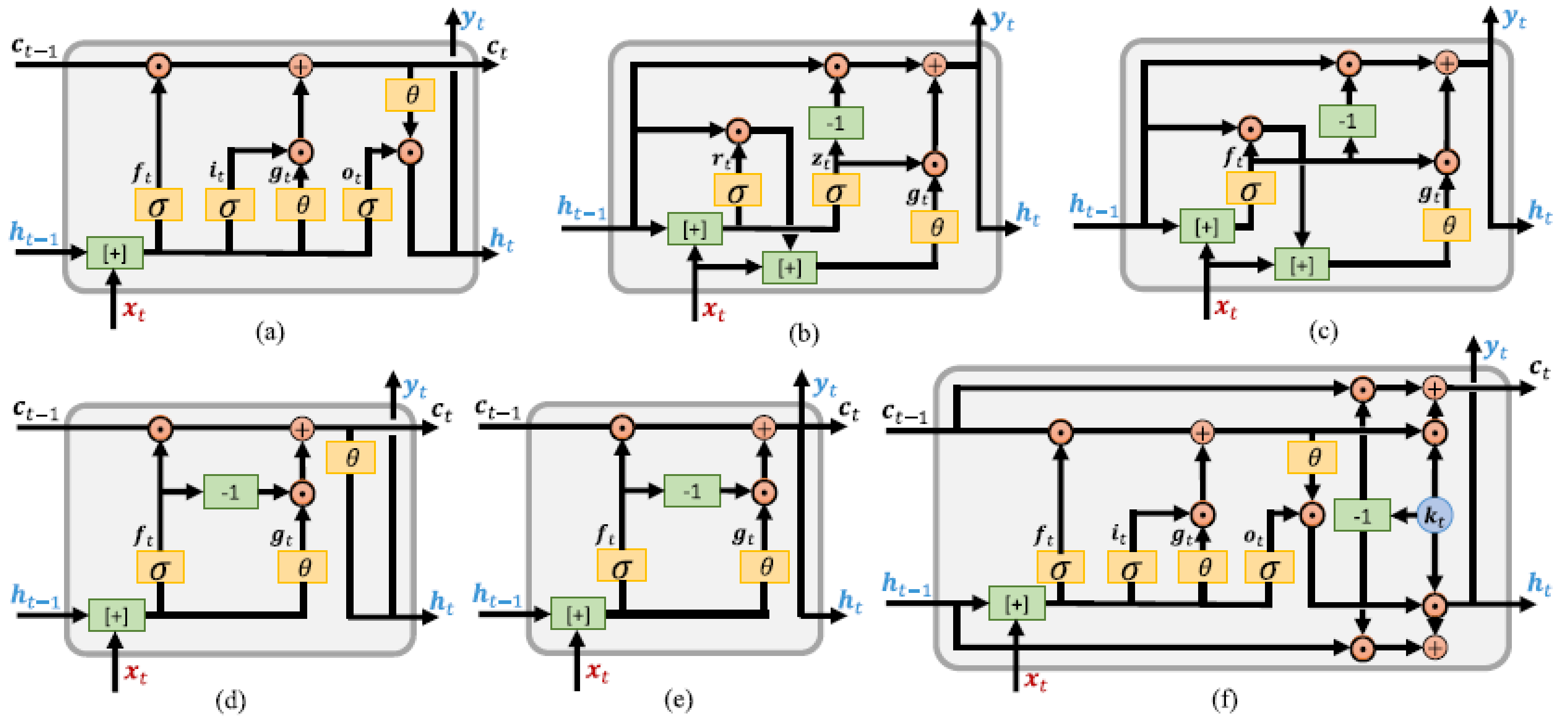


Fig. 7. *Examples of standard and compact RNN units. (a) LSTM [18]. (b) GRU [19]. (c) MGU [40]. (d) S-LSTM [41]. (e) JANET [42]. (f) Phased LSTM [43].*

Compact RNNs (Network Level)

- Linear recurrent projection
  - 차원의 hidden state 수를 줄임
- Factorized LSTM
  - LSTM의 weigh를 두 개의 작은 행렬로 축소시킴
- Residual connections
  - 여러 LSTM 간의 residual connection을 추가
- Skip-connection
  - 비연속적으로 연결된 RNN간에 직접적인 정보전달이 가능케함

Compact RNNs (Network Level)

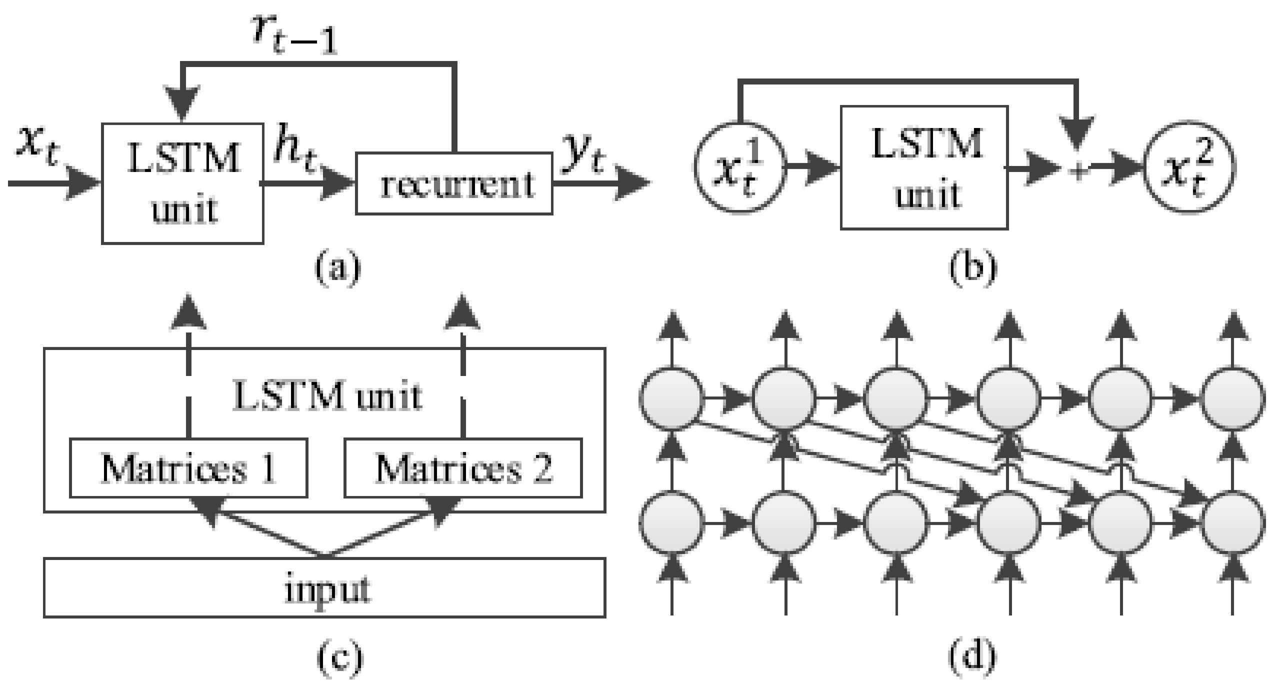


Fig. 8. *Examples of compact RNNs in network level. (a) Linear recurrent LSTM [52]. (b) Residual LSTM [54]. (c) Factorized LSTM [53]. (d) Skip-connected RNN [55].*

Neural Architecture Search

- 신경망의 종류가 다양해짐에 따라 적합한 구조를 찾는 작업의 비용이 증가함

- 이를 머신러닝 기술을 적용해 자동화하는 탐색 기술

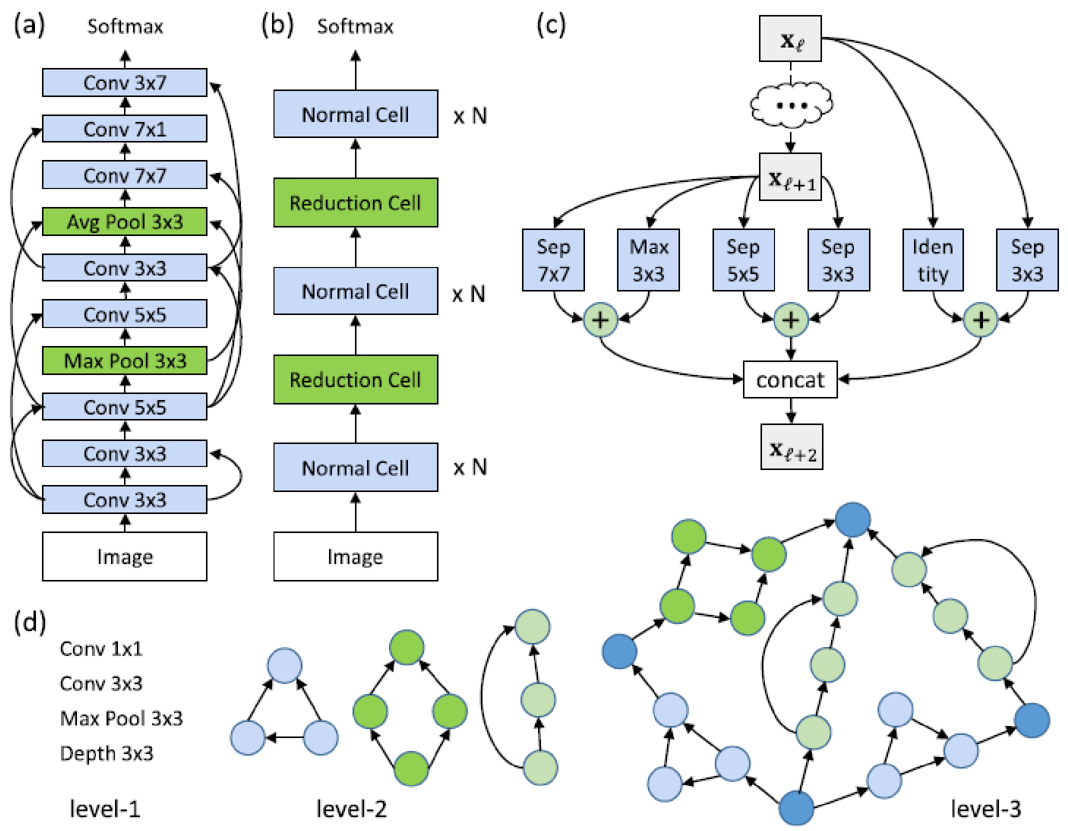


Fig. 9. *Illustration of NAS. (a) Full-network search [45]. (b) Cell-based search [58]. (c) Example of learned cell structures [66]. (d) Three-level hierarchical search [67].*

Tensor Decomposition

- Low-Rank Matrix Decomposition
- Tensorized Decomposition

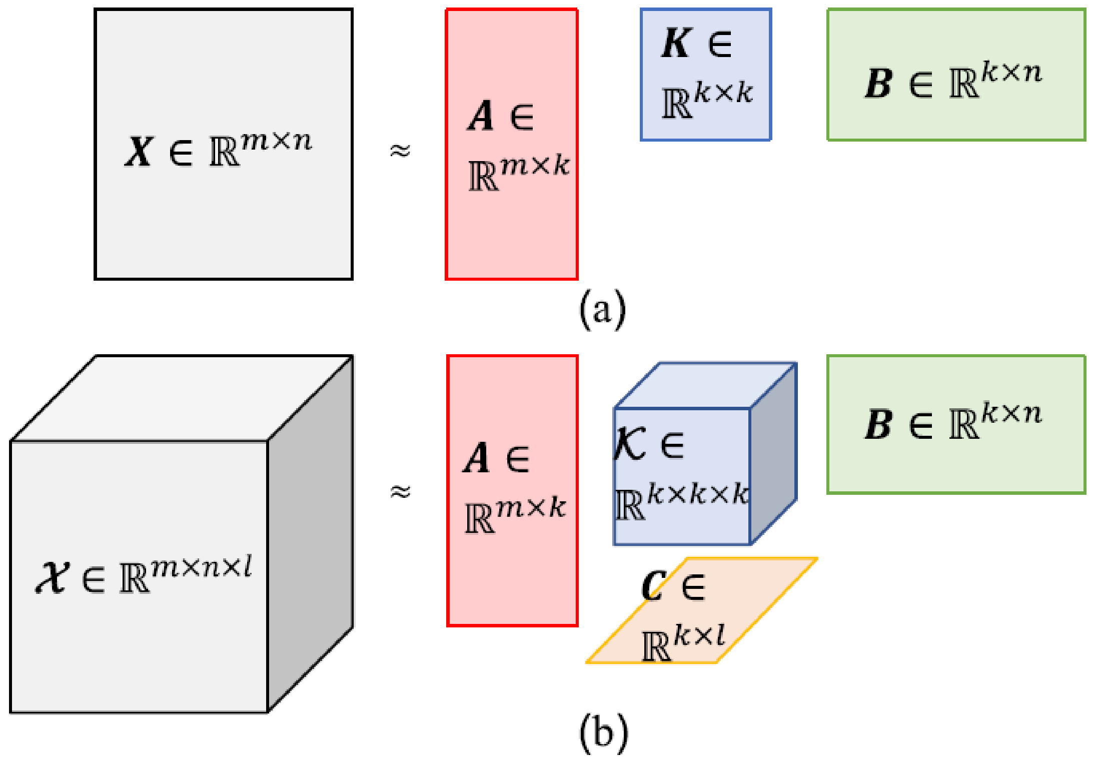

$\boldsymbol{X} \in \mathbb{R}^{m\times n}$
$\approx$
$\boldsymbol{A} \in \mathbb{R}^{m\times k}$
$\boldsymbol{K} \in \mathbb{R}^{k\times k}$
$\boldsymbol{B} \in \mathbb{R}^{k\times n}$
(a)
$\mathcal{X} \in \mathbb{R}^{m\times n\times l}$
$\approx$
$\boldsymbol{A} \in \mathbb{R}^{m\times k}$
$\mathcal{K} \in \mathbb{R}^{k\times k\times k}$
$\boldsymbol{C} \in \mathbb{R}^{k\times l}$
$\boldsymbol{B} \in \mathbb{R}^{k\times n}$
(b)

Fig. 11. *Analogy between low-rank matrix decomposition and tensor decomposition. (a) Low-rank matrix decomposition with k $\times$ k kernel. (b) Low-rank tensor decomposition with k $\times$ k $\times$ k kernel.*

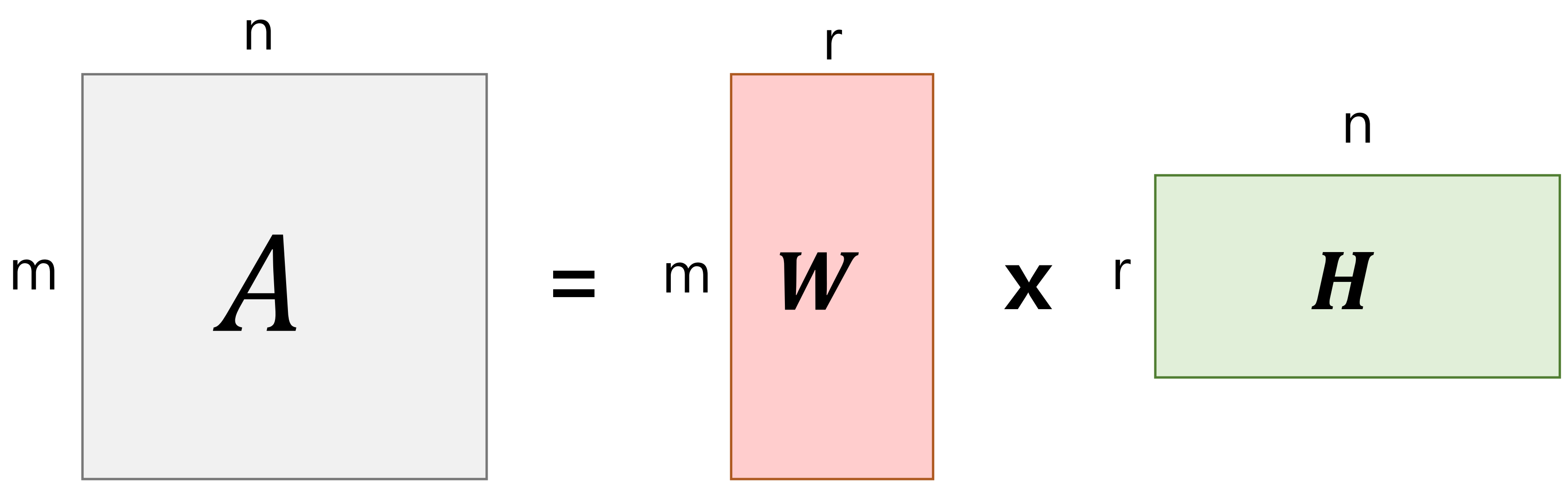
Low-Rank Matrix Decomposition

- Full-rank decomposition

- **Singluar value decomposition (SVD)**

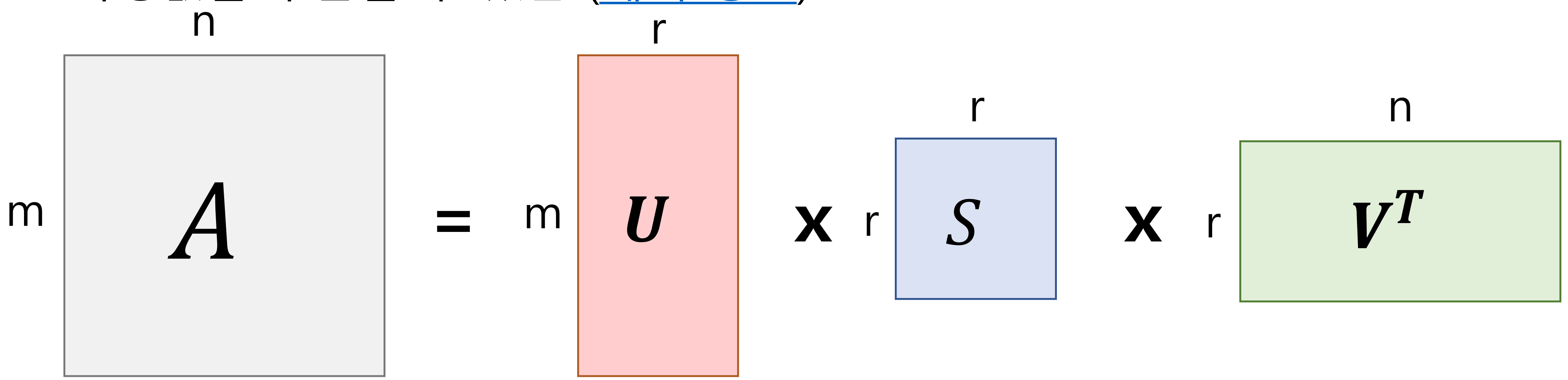
Full-rank decomposition

- 기존 A 행렬을 W와 H 행렬의 곱으로 분해
- Spartial complexity : O(mn) → O(r(m+n))



Singular value decomposition (SVD)

- 기존 A 행렬을 직교 행렬 U,V와 대각행렬 S로 분할
- Spartial complexity : O(mn) → O(r(m+n+1))
- 복원을 위해 r보다 작은 특정 값을 통해 기존 벡터가 가지는 특징값을 추출할 수 있음 (예시 링크)



Tensorized decomposition

- Tucker and classical prolongation (CP)
- Tensorizing and curse of dimensionality
- HT(Hierarchical tucker) and dimension tree

Tucker and classical prolongation (CP)

- 행렬에서의 SVD를 텐서 차원으로의 단순 확장

- 같은 크기의 d차원 텐서
- $\chi \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$, $K \in \mathbb{R}^{r_1 \times r_2 \times \cdots \times r_d}$, $F^{(i)} \in \mathbb{R}^{r_i \times n_i}$
- $\chi \approx K \times F^{(1)} \times F^{(2)} \times \cdots \times F^{(d)}$로 변환
- Spartial complexity : $O(n^d) \rightarrow O(nr + r^d)$

Tensorizing and curse of dimensionality

- 대규모 행렬 $W \in \mathbb{R}^{M \times N}, M = m_1 \times m_2 \times \cdots \times m_d \, , N = n_1 \times n_2 \times \cdots \times n_d$을 텐서$\mathcal{W} \in \mathbb{R}^{m_1 n_1 \times m_2 n_2 \times \cdots \times m_d n_d}$로 변환 (tensorizing)
- 해당 텐서에 Tucker decomposition 적용시 공간 복잡도는 다음과 같다.
  - $O((mn)^d) \to O(dmnr + r^d)$ 여전히 지수차원의 공간 복잡도 존재

- 신경망에서 tensor의 표현은 행렬들의 연결 또는 저차원 텐서의 결합으로 표현되기 때문에, 이런 관계를 분할해 차원의 저주 해결 가능

HT(Hierarchical tucker) and dimension tree

- 차원의 총 개수 d를 임의의 값 k를 따라 (1~d)를 (1~k),(1~d-k) 차원의 두 개 텐서로 반복적으로 나눠 계산하는 방식을 HT라 함
  - 이 때 k =1을 기준으로 한다면 이를 TT(Tensor Train)이라 한다.

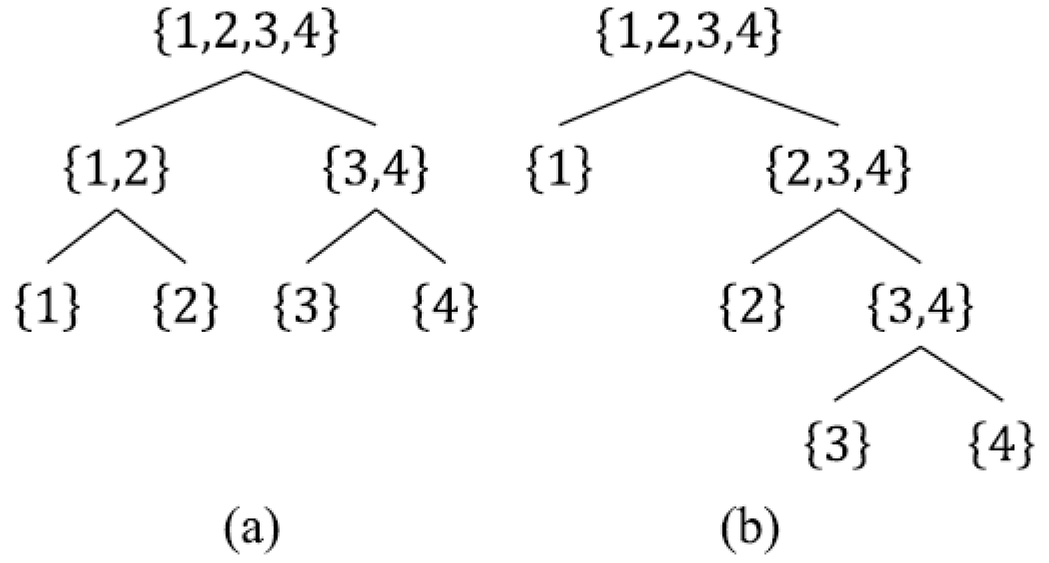


Fig. 12. ***Different dimension trees of tensor $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3 \times n_4}$. (a) Canonical dimension tree for the HT format. (b) Degenerate dimension tree for the TT format.***

Reference

- Model compression and hardware acceleration for neural networks: A comprehensive survey. Proceedings of the IEEE, 2020

- 공돌이의 수학 정리 노트

Model Compression

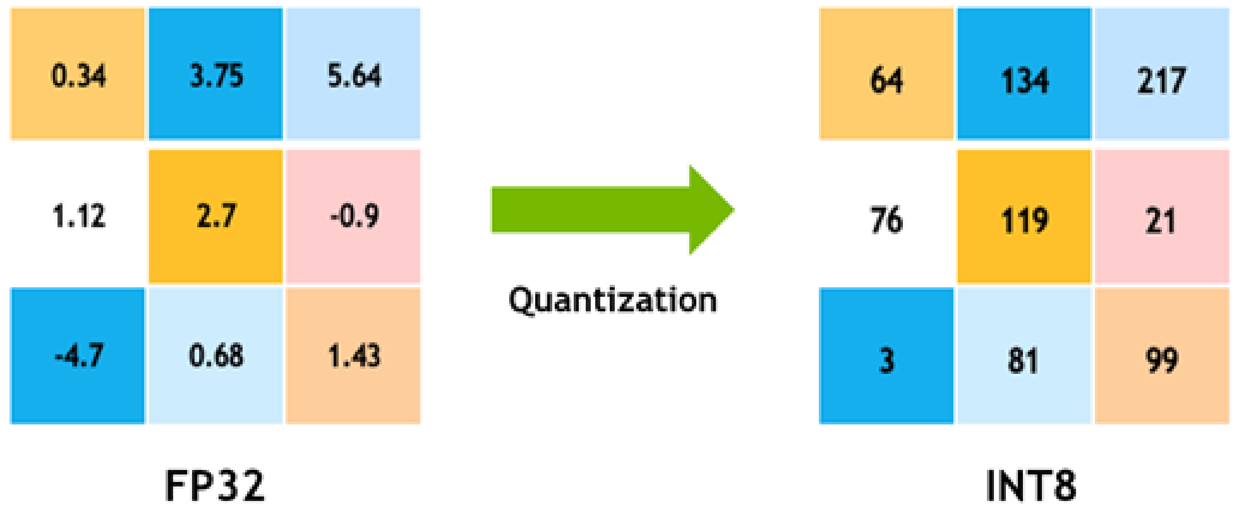
영남대학교
차세대 컴퓨터 시스템 연구실
석사과정생 이원호

Compression Approaches

- Compact Model
  - Design smaller base models that can still achieve acceptable application accuracy
- Tensor Decomposition
  - Decompose the bloated parameters into a series of smaller matrices or tensors
- **Data Quantization**
  - Reduce the number of data bits
- **Network Sparsification**
  - Reduce the connections/neurons to compress the original model

Data Quantization

- Quantization Data Object
  - **What** to be quantized
- Quantization Considerations
  - **How** to quantize



Reduce bit-width → save the memory capacity

Quantization Data Object

- Three types of quantization target
  - **Parameter (Weight)**
  - Propagation data (Activation, Error)
  - Gradient and Update (Gradient, Weight Update)
- In recent studies, Quantization to BN layers is being discussed

Quantization Consideration

- Data Object
- Problem formulation
  - Heuristic / Optimization
- Distribution of discrete levels
  - Uniform / Non-uniform
- Level projection
  - Deterministic / Stochastic

Quantization Consideration

- Data Object
- **Problem formulation**
  - Heuristic / Optimization
- Distribution of discrete levels
  - Uniform / Non-uniform
- Level projection
  - Deterministic / Stochastic

- Heuristic
  - Intuitively project data to discrete level

$$Q(x) = \Delta \cdot round(\frac{x}{\Delta})$$

- Optimization
  - Analytically approximation

$$min_Q ||X - Q(X)||_2^2 \quad Q_i \in X_Q \; for \; all \; i$$

Quantization Consideration

- Data Object
- Problem formulation
  - Heuristic / Optimization
- **Distribution of discrete levels**
  - Uniform / Non-uniform
- Level projection
  - Deterministic / Stochastic

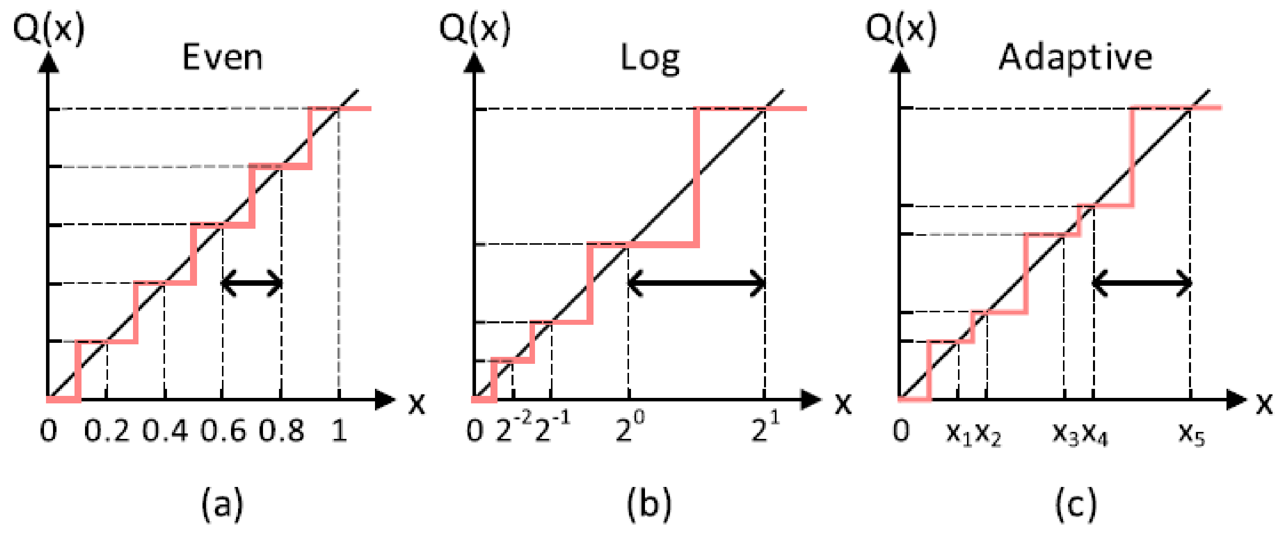


Fig. 15. *Distribution of discrete levels. (a) Uniform distribution with even step length. Nonuniform distribution with (b) logarithmic or (c) adaptive step length.*

Quantization Consideration

- Data Object

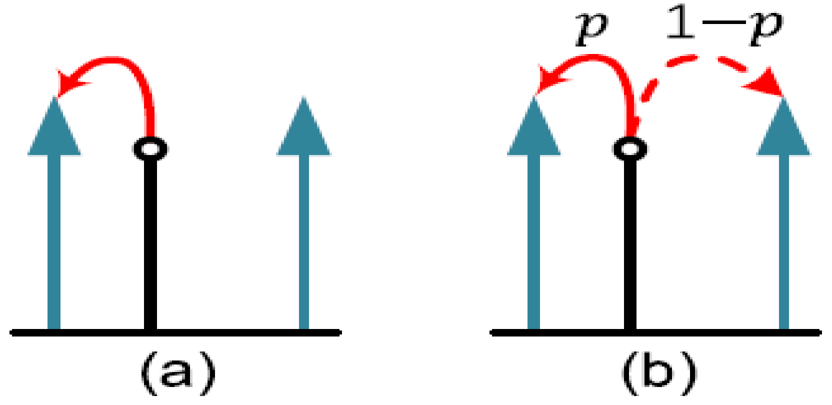
- Problem formulation
  - Heuristic / Optimization

- Distribution of discrete levels
  - Uniform / Non-uniform

- **Level projection**
  - Deterministic / Stochastic

- Deterministic way
  - Project data to nearest discrete level

- Stochastic way
  - Provides probabilities for two adjacent levels of data



Simple Quantization (single bit quantization)

- Binarization

$$\begin{cases} \text{det. :} & \begin{cases} Q(x) = 1, & x \geq 0 \\ Q(x) = -1, & \text{otherwise} \end{cases} \\ \text{sto. :} & \begin{cases} P(Q(x) = 1) = \text{clip}\left(\frac{1+x}{2}, 0, 1\right) \\ P(Q(x) = -1) = 1 - P(Q(x) = 1) \end{cases} \end{cases}$$

$$\min_{\alpha \in R^{+},\ Q^{B}} \|\boldsymbol{X} - \alpha Q^{B}(\boldsymbol{X})\|_2^2, \quad \text{s.t. } Q_i^B \in \{-1,\ 1\}$$

- Ternarization

$$\begin{cases} \text{det. :} & \begin{cases} Q(x) = 1, & x > 0.5 \\ Q(x) = -1, & x < -0.5 \\ Q(x) = 0, & \text{otherwise} \end{cases} \\ \text{sto. :} & \begin{cases} \begin{cases} P(Q'(x) = 1) = \text{clip}(|x|, 0, 1) \\ P(Q'(x) = 0) = 1 - P(Q'(x) = 1) \end{cases} \\ Q(x) = \text{sign}(x) \cdot Q'(x) \end{cases} \end{cases}$$

$$\min_{\alpha,\ Q^{T}(\boldsymbol{W})} \|\boldsymbol{W} - \alpha Q^{T}(\boldsymbol{W})\|_2^2, \quad \text{s.t. } Q_i^T \in \{-1,\ 0,\ 1\}.$$

Advanced Quantization (k- bit quantization)

- Low-bit quantization cannot always produce acceptable accuracy
- For accuracy improvement, increase the bit-width
- Multibit quantization

$$Q^k(w) = 2Q^k\left(\frac{\tanh(w)}{2\max(|\tanh(\boldsymbol{W})|)} + \frac{1}{2}\right) - 1$$

$$\min_{\{\alpha_j,\, Q^{Bj}\}_{j=1}^{k}} \left\| \boldsymbol{X} - \sum_{j=1}^{k} \alpha_j Q^{Bj} \right\|_2^2, \quad \text{s.t. } Q_i^{Bj} \in \{-1,\, 1\}$$

Influence of Quantization

- Quantization to weight, activation and gradient is easy
- Quantization to error, update, batch normalization result in performance degradation

- Impact to model
  - Gradient $\leq$ Weight < Activation < Error < Update
  - Quantization to update lower convergence

- Reduce the communication bandwidth in distributed training systems
- Gaussian distribution $\rightarrow$ bound range of quantization loss

Network Sparsification

- Sparsification Data Object
  - **What** to be sparsified
- Sparsification Consideration
  - **How** to sparsify

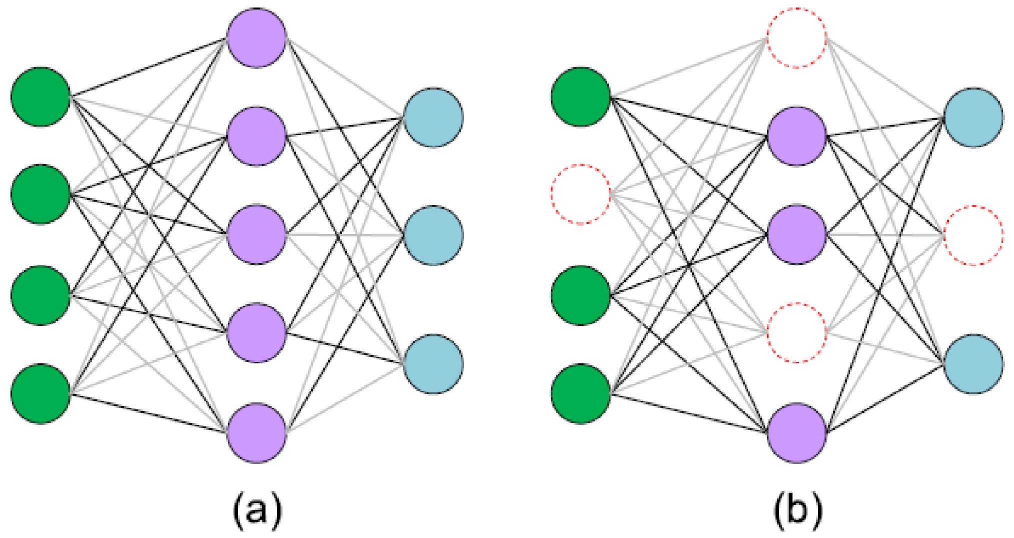


Fig. 17. *Sparsification data objects. (a) Weight pruning. (b) Neuron pruning.*

Reduce #operands → reduce #memory_access, #operations

Sparsification Data Object

- Two types of quantization target
  - Correspond to weight parameters (Weight , Gradient)

    → Weight pruning (reduce #edge)

  - Indicate neuron (Activation, Error)

    → Neuron pruning (reduce #node)

  - Sparse pattern of Update is coupled with other data objects (weight, activation, error)

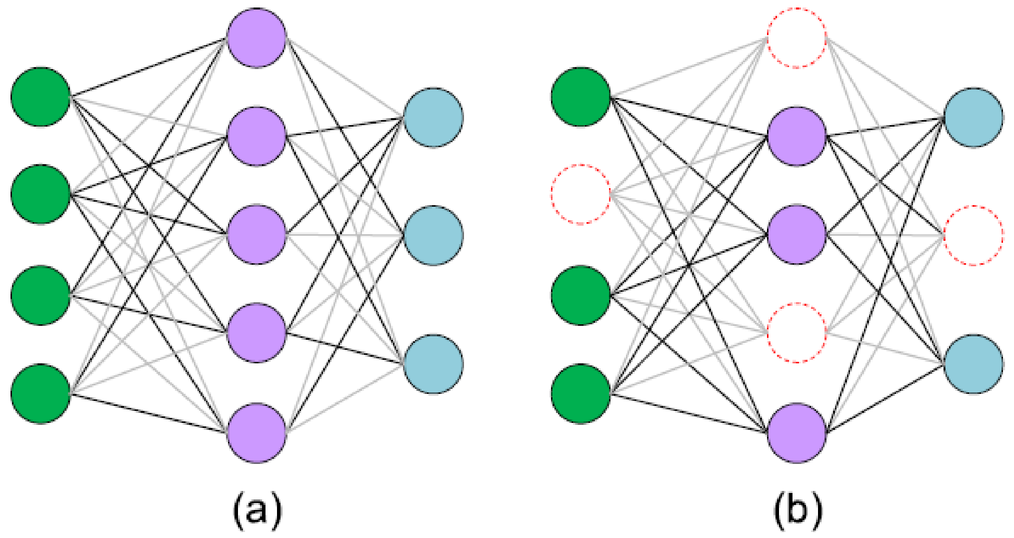


Fig. 17. *Sparsification data objects. (a) Weight pruning. (b) Neuron pruning.*

Sparsification Consideration

- Data Object
- Bit-width
- Sparsification grain/structures
- Different pass (forward/backward)

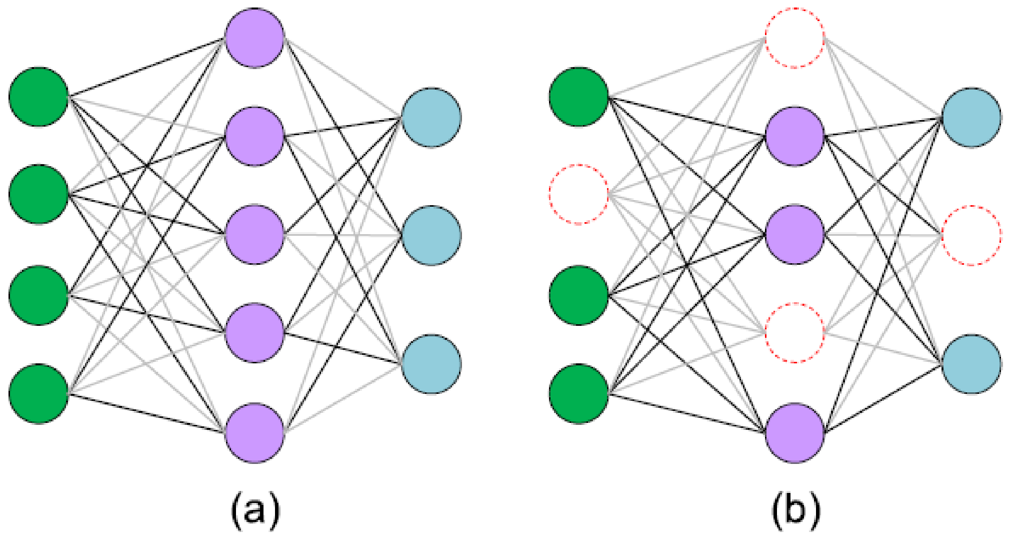


Fig. 17. *Sparsification data objects. (a) Weight pruning. (b) Neuron pruning.*

Sparsification grain/structures

- Element-wise → weight pruning
  - extra index overhead
- Vector-wise → weight pruning
- Block-wise → neuron pruning
  - Iteratively reordering the element-wise sparse weights

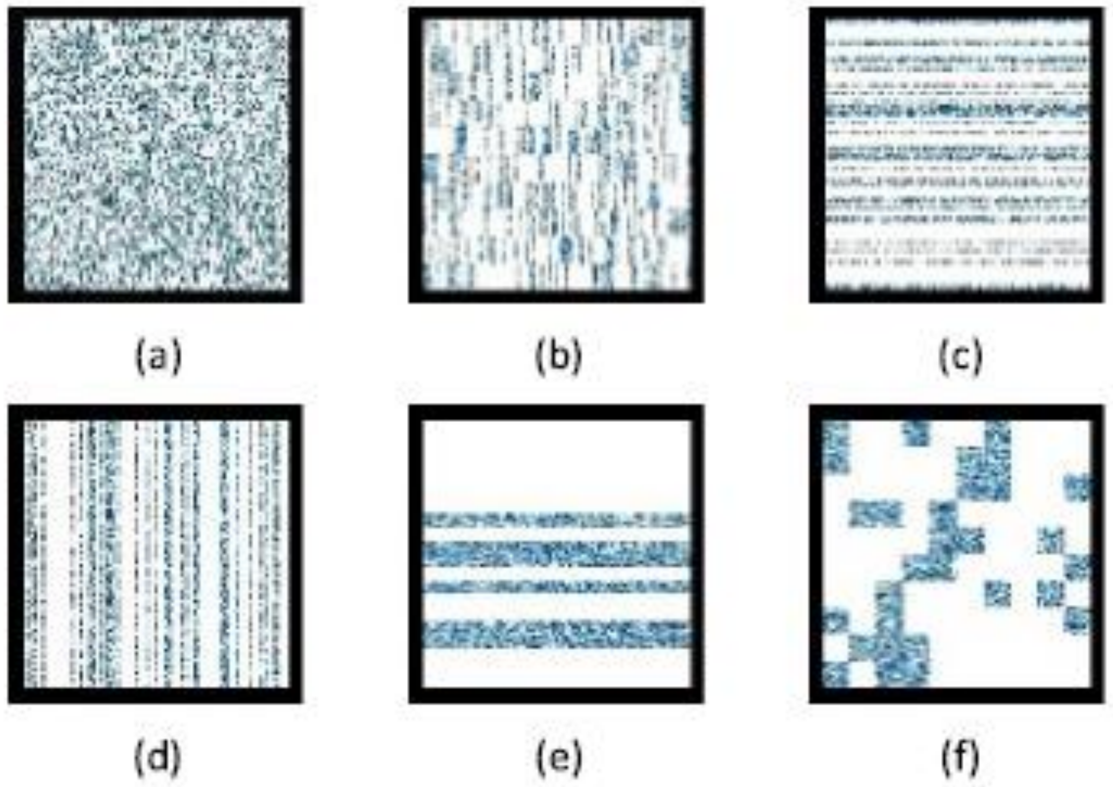
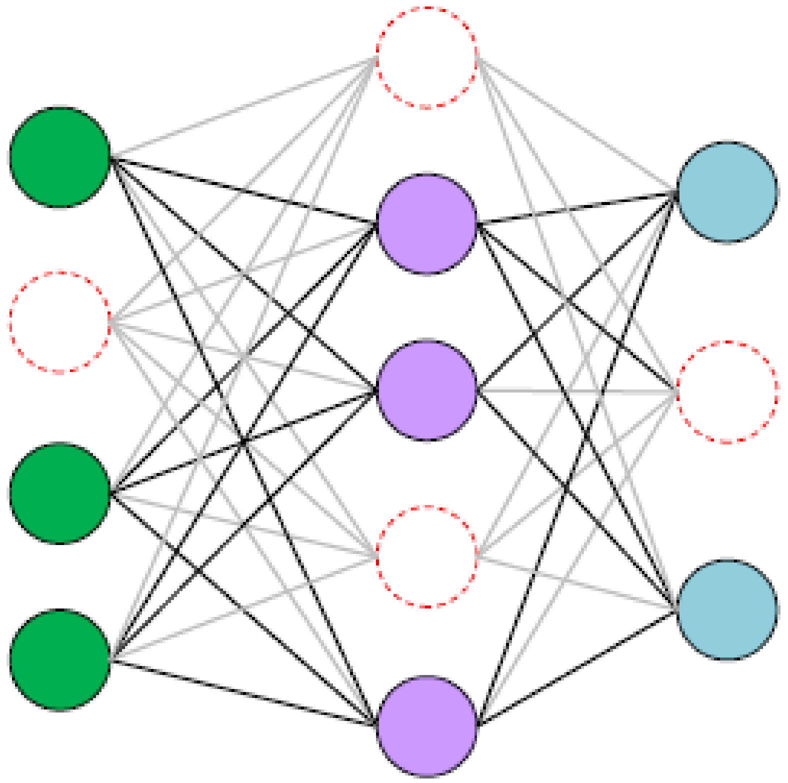


Fig. 18. *Sparsification structure. (a) Element-wise. (b)–(d) Vector-wise. (e) and (f) Block-wise.*

Constraints of Neuron pruning

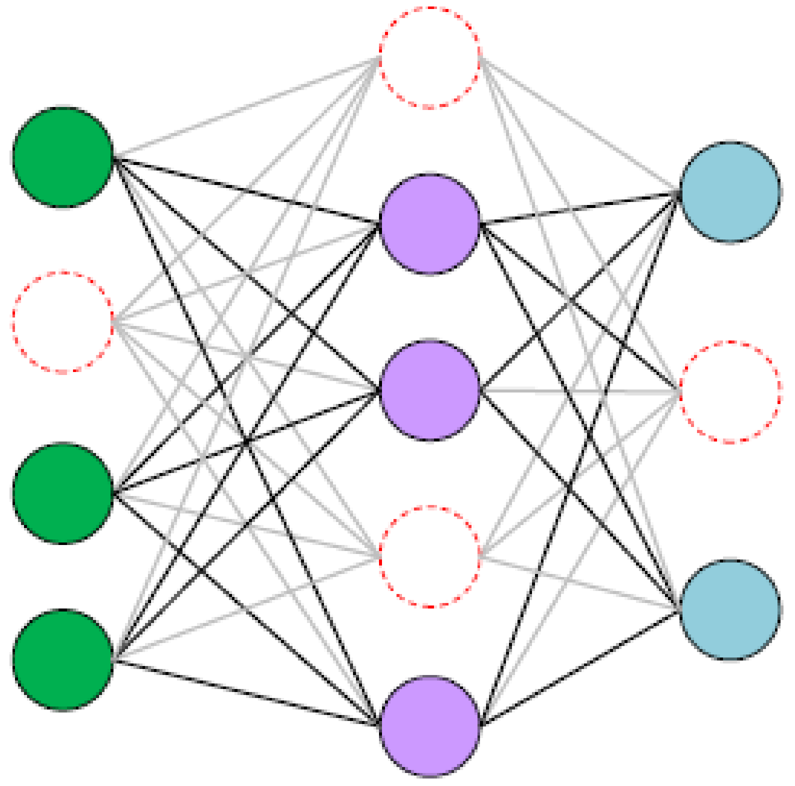
- Target data object
  - Activation (output)
  - Error
- After pruning one input FM, these can be removed
  - Corresponding weight filter in the current layer
  - Weight channel in the next layer
- Require complete calculation of output activations before pruning



Before-calculation Neuron pruning

- Require complete calculation of output activations before pruning

→ Preselection of unimportant neurons
  - Hashing search
  - Negative-activation prediction
  - Low-precision estimation
  - Prechecking neighboring pixels
  - Dimension-reduction estimation ….



Joint-way compression

- Compact Model and Others
- Tensor Decomposition and Network Sparsification
- Data Quantization and Network Sparsification

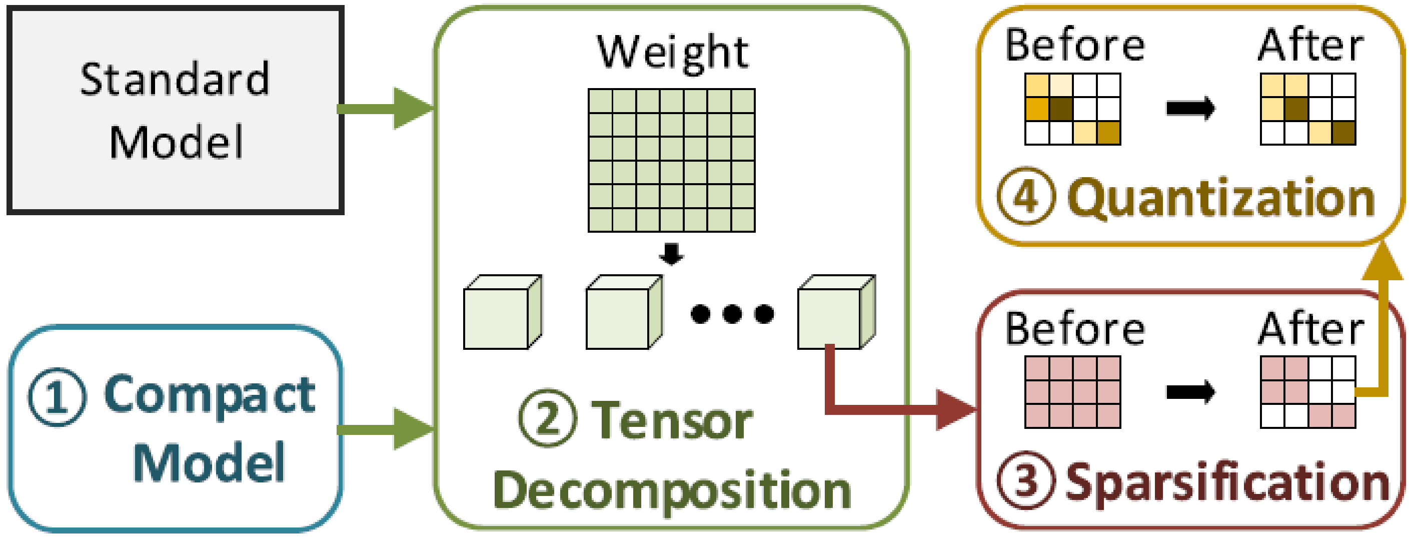


Fig. 19. *Joint-way compression.*

Joint-way compression

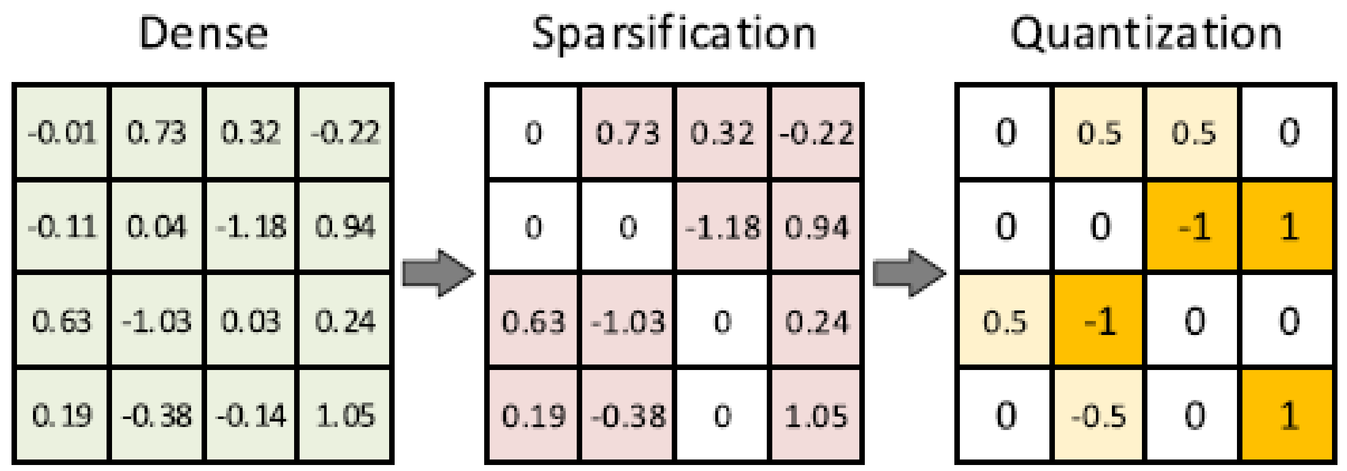
- Compact Model and Others
  - Compact model approach design the base network itself
  - Others compress a given base model

  → design of compact model is **orthogonal** to other compression approaches

- Tensor Decomposition and Network Sparsification
  - Sparsity by post-pruning < direct pruning of undecomposed model

Joint-way compression

- Data Quantization and Network Sparsification
    - Quantization improve sparsity



-0.01	0.73	0.32	-0.22
-0.11	0.04	-1.18	0.94
0.63	-1.03	0.03	0.24
0.19	-0.38	-0.14	1.05

0	0.73	0.32	-0.22
0	0	-1.18	0.94
0.63	-1.03	0	0.24
0.19	-0.38	0	1.05

0	0.5	0.5	0
0	0	-1	1
0.5	-1	0	0
0	-0.5	0	1

Fig. 20. ***Increasing sparsity through quantiza***

Neural Network Acceleration (Hardware)

- Two typical workloads in running neural networks

- Conv operation
  - Huge data reusability (activation, weights)
- MVM operation
  - Data cannot be reused without batching technique

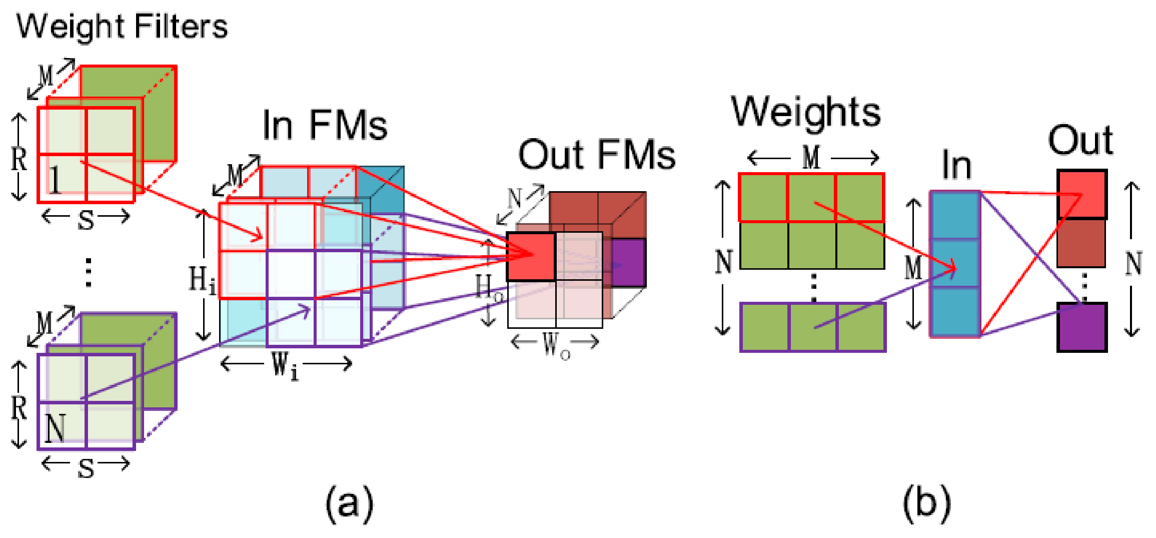


Fig. 21. *Computation pattern. (a) Conv layer. (b) FC layer. Adapted from [14].*

Domain-Specific Accelerator for Neural Network

- Rapid development of neural networks heavily relies on computing capability of the hardware
- Design goal is to minimize latency, energy, area
- On-GPU compression shows performance improvements below expectations
  - Redundant design of general-purpose processors for flexible programmability and general applicability

  → **motivation of specialized accelerator for neural network**
- Optimization targets
  - Processing architecture
  - Memory hierarchy
  - Dataflow mapping

Sole Hardware Optimization

- Parallel Compute Units and Orchestrated Memory Hierarchy

- Two aspect to accelerator design
  - Enhancing the compute parallelism
  - Optimizing the memory hierarchy

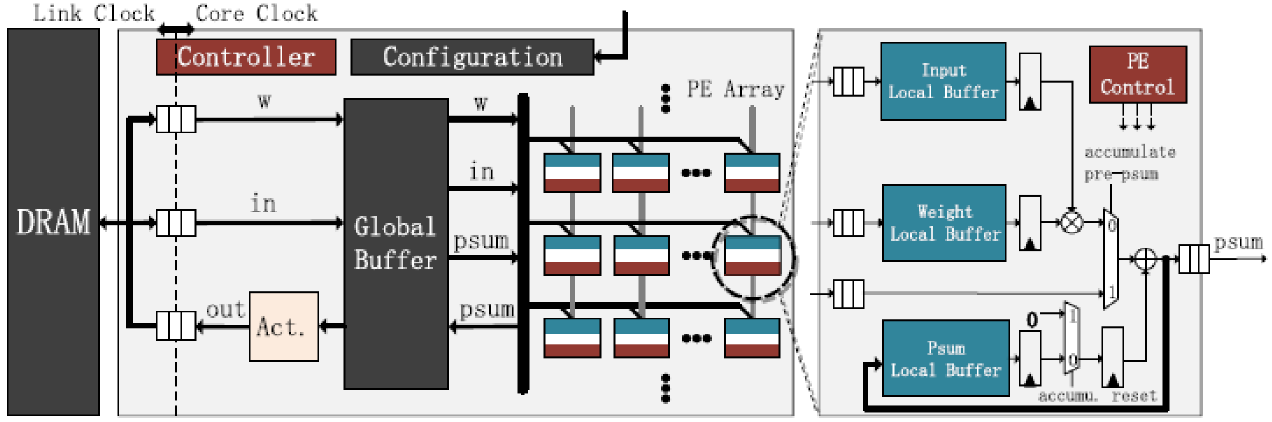


Fig. 22. ***Typical digital architecture of neural network accelerators. Adapted from [261].***

Dataflow

- Output stationary
- Weight stationary
- Input stationary
- Row stationary

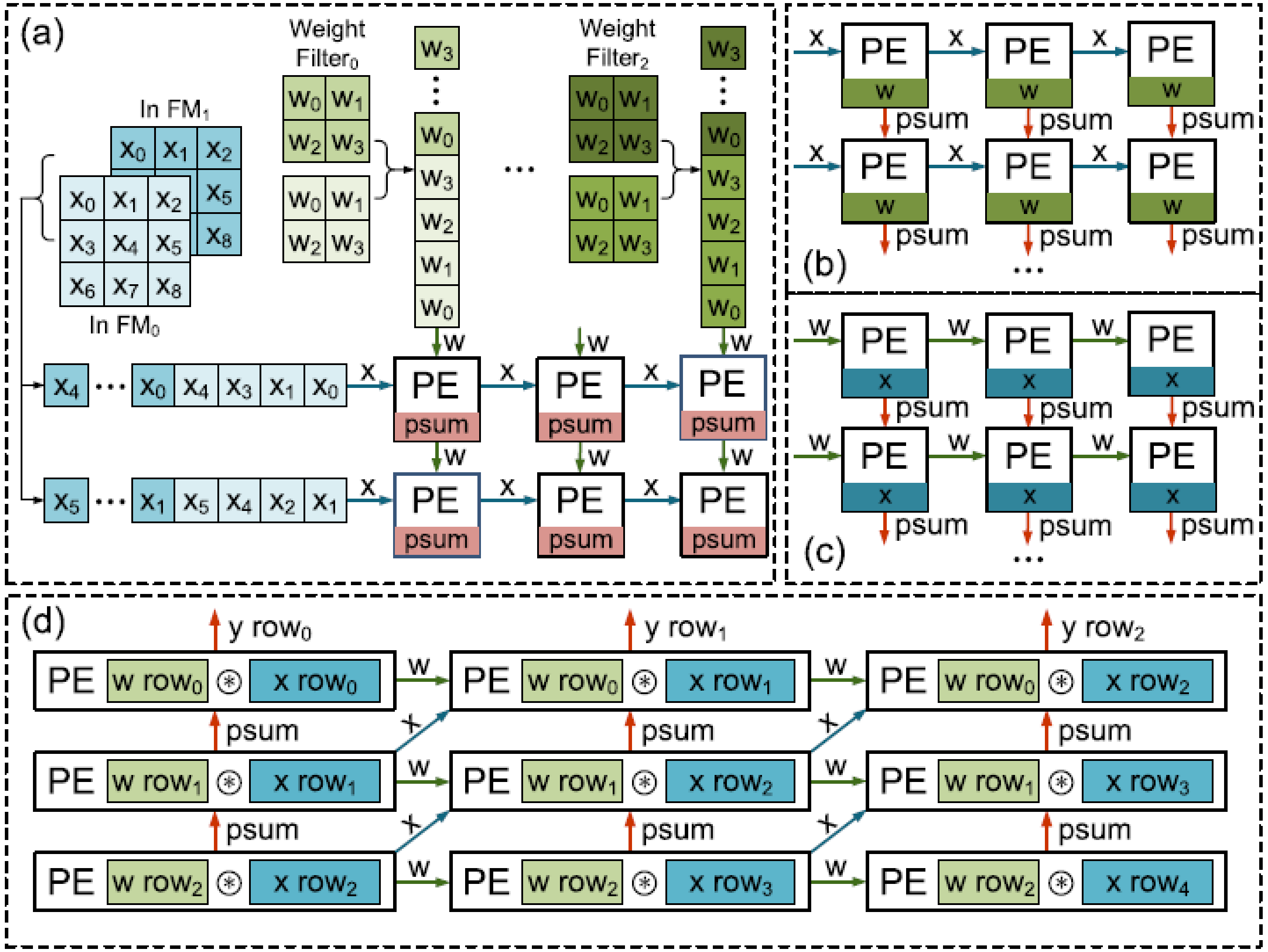


Fig. 23. ***Various dataflow designs. (a) Output stationary. (b) Weight stationary. (c) Input stationary. (d) Row stationary. Adapted from [14], [257], and [263].***

Algorithm and Hardware Codesign

- Sole hardware side optimization meet performance wall
  - Parallelism and data reuse are exhausted

  → both algorithm and hardware side optimization

- Four Types of Codesign space
  - Transformation into Compact Model
  - Tensorized Processing Engine
  - Quantized Architecture
  - Sparse Architecture

Transformation into Compact Model

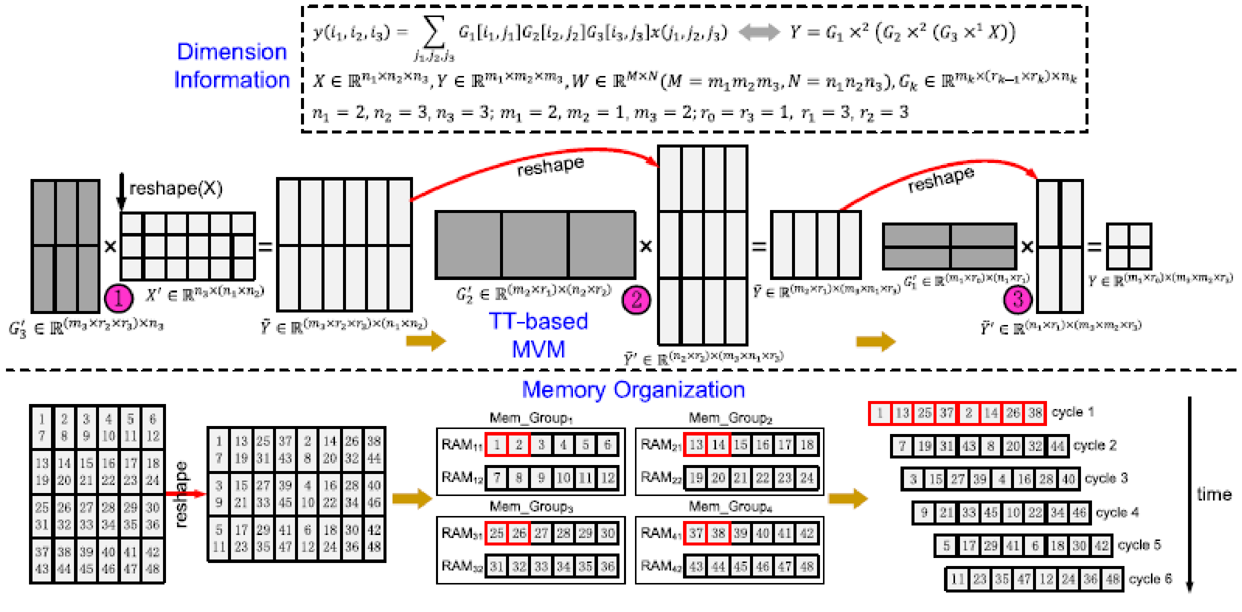
- Compact model reduce the model size without changing dense execution pattern

- Compact connections/operations and variable data path degrade the hardware utilization
- Two solutions
  - Considering flexible tiling/mapping schemes and an efficient NoC infrastructure (high bandwidth/data reuse)
  - Transform compressed network into compact models

Tensorized Processing Engine

- Decomposed network = dense matrix operation with smaller size

- Need additional support when it meet some abnormal operations included (e.g. reshaping)



Dimension Information
$y(i_1,i_2,i_3) = \sum_{j_1,j_2,j_3} G_1[i_1,j_1]G_2[i_2,j_2]G_3[i_3,j_3]x(j_1,j_2,j_3) \Leftrightarrow Y = G_1 \times^2 (G_2 \times^2 (G_3 \times^1 X))$
$X \in \mathbb{R}^{n_1 \times n_2 \times n_3}, Y \in \mathbb{R}^{m_1 \times m_2 \times m_3}, W \in \mathbb{R}^{M \times N} (M = m_1 m_2 m_3, N = n_1 n_2 n_3), G_k \in \mathbb{R}^{m_k \times (r_{k-1} \times r_k) \times n_k}$
$n_1 = 2, n_2 = 3, n_3 = 3; m_1 = 2, m_2 = 1, m_3 = 2; r_0 = r_3 = 1, r_1 = 3, r_2 = 3$
reshape(X)
reshape
reshape
1
2
3
$G'_3 \in \mathbb{R}^{(m_3 \times r_2 \times r_3) \times n_3}$
$X' \in \mathbb{R}^{n_3 \times (n_1 \times n_2)}$
$\tilde{Y} \in \mathbb{R}^{(m_3 \times r_2 \times r_3) \times (n_1 \times n_2)}$
$G'_2 \in \mathbb{R}^{(m_2 \times r_1) \times (n_2 \times r_2)}$
TT-based MVM
$\tilde{Y}' \in \mathbb{R}^{(n_2 \times r_2) \times (m_3 \times n_1 \times r_3)}$
$\tilde{Y} \in \mathbb{R}^{(m_2 \times r_1) \times (m_3 \times n_1 \times r_3)}$
$G'_1 \in \mathbb{R}^{(m_1 \times r_0) \times (n_1 \times r_1)}$
$\tilde{Y}' \in \mathbb{R}^{(n_1 \times r_1) \times (m_3 \times m_2 \times r_3)}$
$Y \in \mathbb{R}^{(m_1 \times r_0) \times (m_3 \times m_2 \times r_3)}$
Memory Organization
reshape
Mem_Group1
Mem_Group2
Mem_Group3
Mem_Group4
RAM11
RAM12
RAM21
RAM22
RAM31
RAM32
RAM41
RAM42
cycle 1
cycle 2
cycle 3
cycle 4
cycle 5
cycle 6
time

Fig. 25. *TT-based MVM and an example of its memory organization. Adapted from [11].*

Tensorized Processing Engine

- Accelerate TT decomposed neural network
- Reshaping
  - partitioning the working SRAM into multiple groups
  - Data selection mechanism

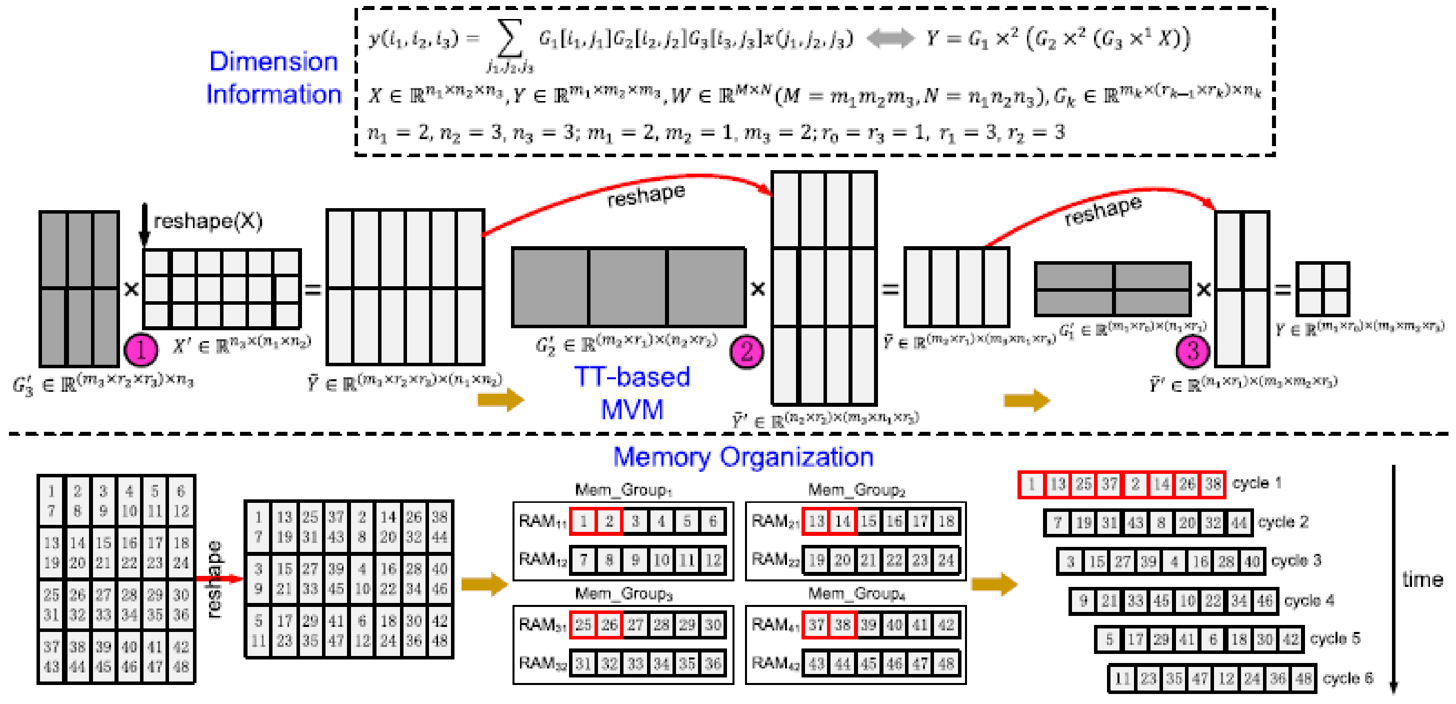

Dimension Information
$y(i_1,i_2,i_3) = \sum_{j_1,j_2,j_3} G_1[i_1,j_1]G_2[i_2,j_2]G_3[i_3,j_3]x(j_1,j_2,j_3) \Longleftrightarrow Y = G_1 \times^2 (G_2 \times^2 (G_3 \times^1 X))$
$X \in \mathbb{R}^{n_1 \times n_2 \times n_3}, Y \in \mathbb{R}^{m_1 \times m_2 \times m_3}, W \in \mathbb{R}^{M \times N} (M = m_1 m_2 m_3, N = n_1 n_2 n_3), G_k \in \mathbb{R}^{m_k \times (r_{k-1} \times r_k) \times n_k}$
$n_1 = 2, n_2 = 3, n_3 = 3; m_1 = 2, m_2 = 1, m_3 = 2; r_0 = r_3 = 1, r_1 = 3, r_2 = 3$
reshape(X)
reshape
reshape
TT-based MVM
Memory Organization
Mem_Group1
Mem_Group2
Mem_Group3
Mem_Group4
cycle 1
cycle 2
cycle 3
cycle 4
cycle 5
cycle 6
time

Fig. 25. *TT-based MVM and an example of its memory organization. Adapted from [11].*

Quantized Architecture

- Data-precision ≥ 8-bit fixed point for CNN inference

- Aggressive low-bit quantization
  - Ultrahigh execution performance + accuracy loss

- Two types of quantized architecture
  - Fixed bitwidth
  - Variable bitwidth

Quantized Architecture – Fixed bitwidth

- Only change high-precision multipliers to low-bit multipliers
  - Other overall architecture and dataflow remain same

- Extreme quantization with only binary/ternary data precision

- Weight and activations are binarized/ternarized

  → MAC operation can be implemented by simple XNOR and pop-count logic operations

Quantized Architecture – Variable bitwidth

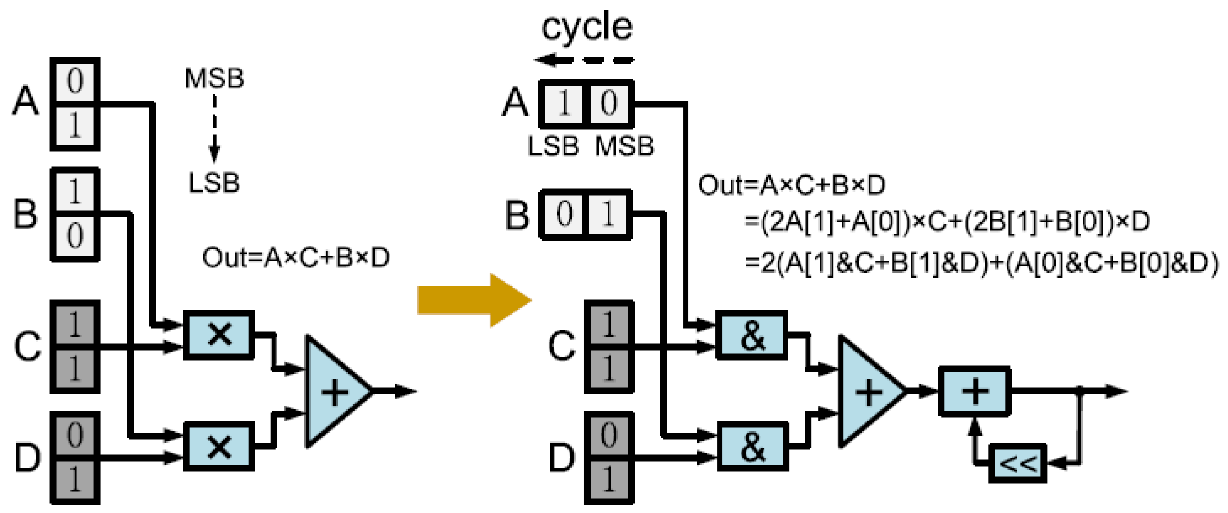
- Different layers/models require different bit-width

- Twi types of architecture
  - Bit-serial
  - Bit-decomposed

Quantized Architecture – Variable bit-width

Bit-serial architecture

- Serially feed one of the two operands bit by at each cycle
- MAC operation → AND logic operation, shifted accumulation
- Enhance the parallelism



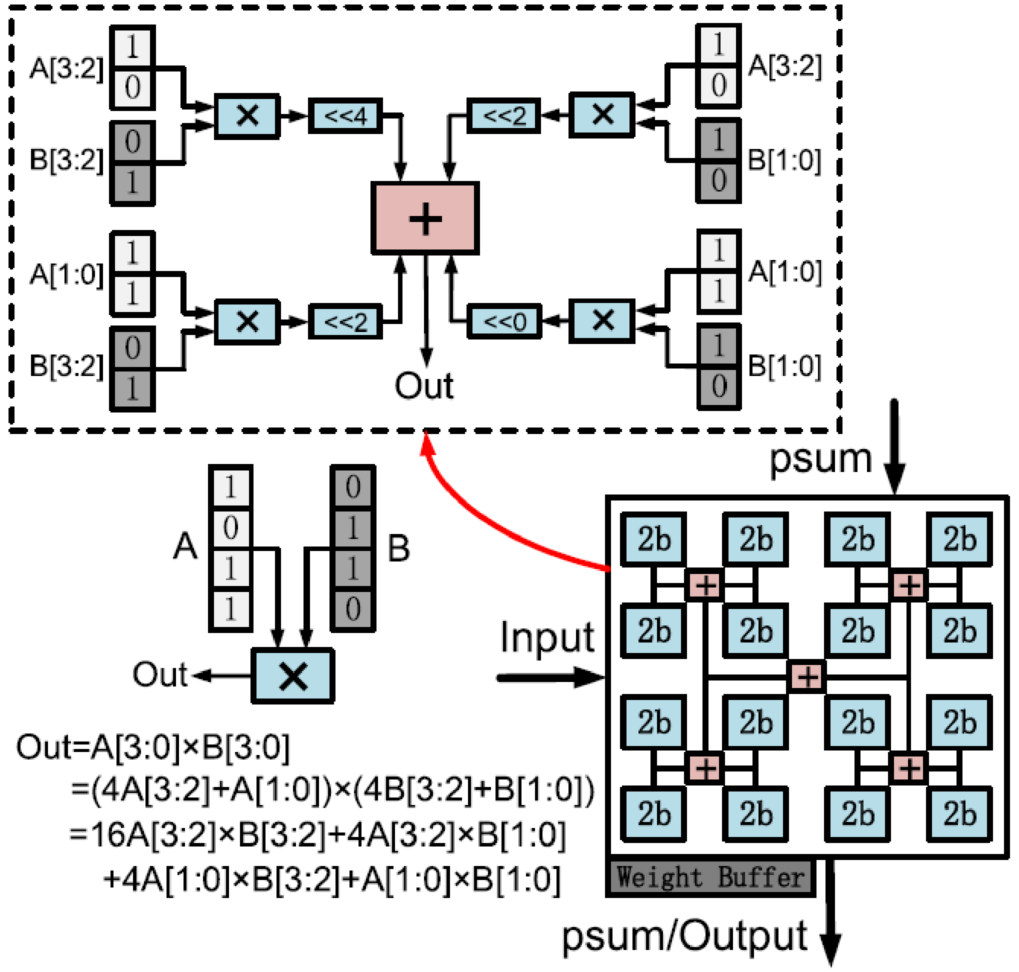
A
B
C
D
MSB
LSB
Out=A×C+B×D
cycle
LSB MSB
Out=A×C+B×D
=(2A[1]+A[0])×C+(2B[1]+B[0])×D
=2(A[1]&C+B[1]&D)+(A[0]&C+B[0]&D)

Fig. 27. *Bit-serial MAC processing. Adapted from [278].*

Quantized Architecture – Variable bitwidth

Bit-decomposed architecture

- Fuse multiple low-precision MACs to compute high-precision MAC
- Fusion group size is configurable to support flexible bitwidth



$$
\begin{aligned}
\text{Out} &= A[3:0] \times B[3:0] \\
&= (4A[3:2]+A[1:0]) \times (4B[3:2]+B[1:0]) \\
&= 16A[3:2] \times B[3:2] + 4A[3:2] \times B[1:0] \\
&\quad + 4A[1:0] \times B[3:2] + A[1:0] \times B[1:0]
\end{aligned}
$$

Fig. 28. *Bit fusion architecture based on bit-decomposed MAC processing. Adapted from [279].*

Spase Architecture

- Four exploitable sparse patterns
  - Weight sparsity
  - Input sparsity
  - Output sparsity
  - Combination of multiple patterns

Spase Architecture – Weight sparsity

- Naively skipping the MACs with zero weights

- Each PE access the required activation according to the weights index
  - Performs the residual MACs

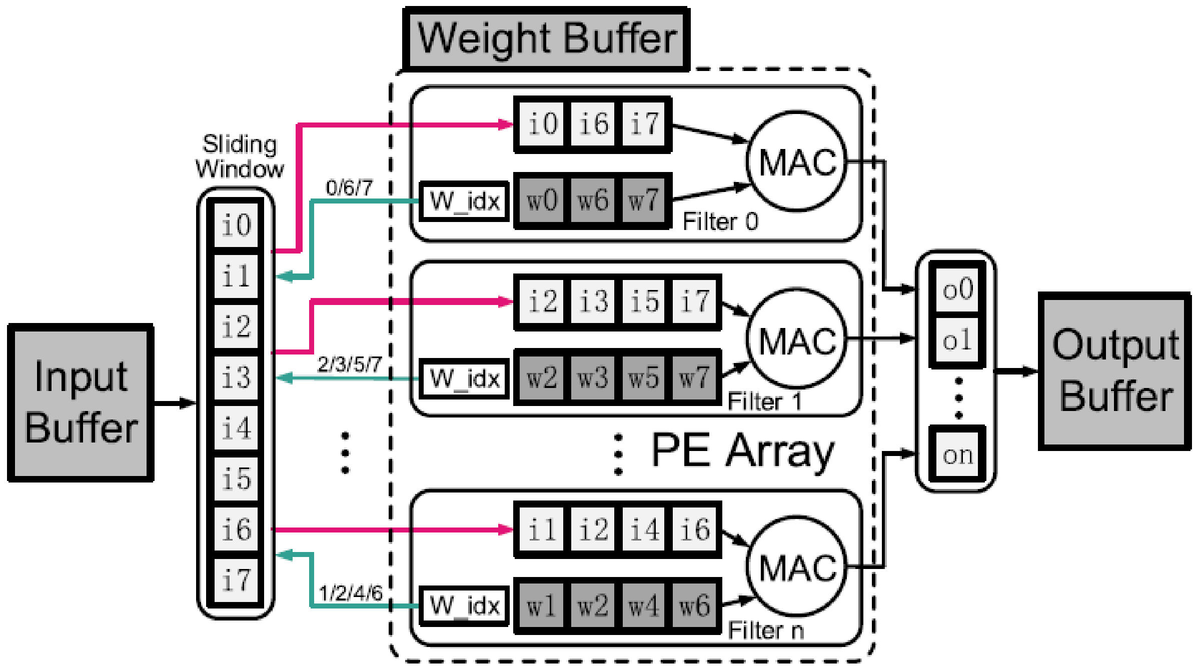


Fig. 29. *Example of accelerators with weight sparsity. Adapted from [281].*

Spase Architecture – Input sparsity

- To save bandwidth and energy
  - Run-length activation compression
  - Zero input detection
  - PE data gating

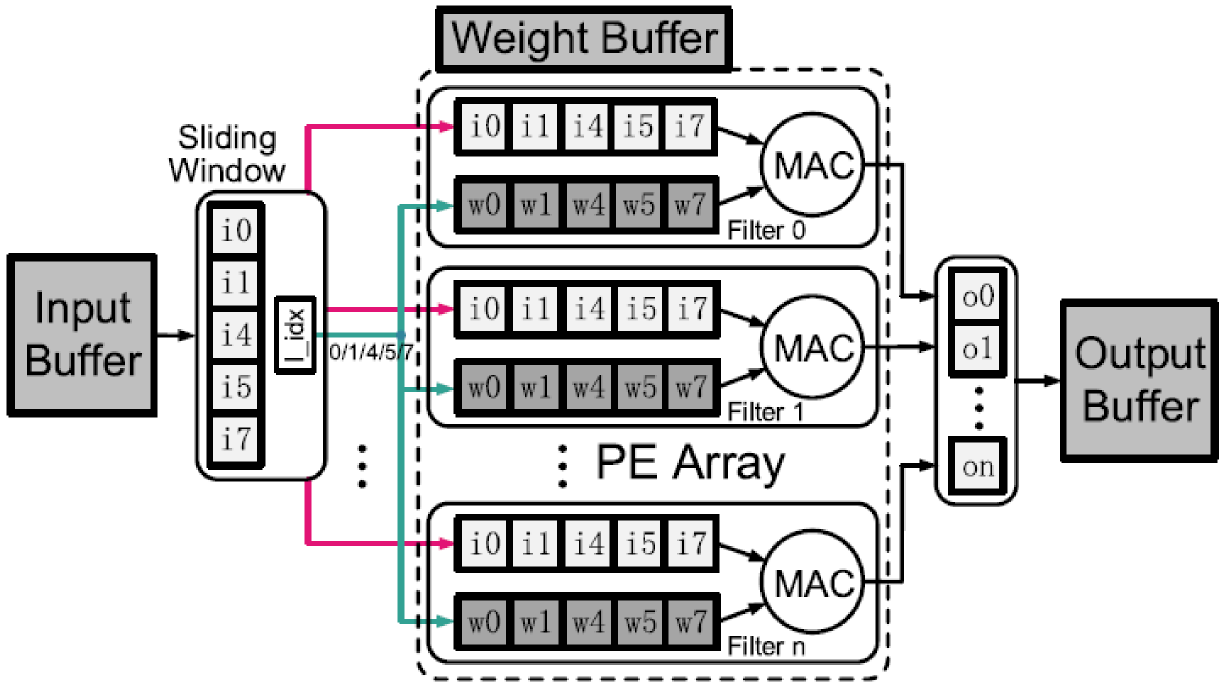


Fig. 31. *Example of accelerators with input sparsity. Adapted from [283].*

Spase Architecture – Output sparsity

- Predict unimportant neurons
- In-place speculation
  - Monitor the sign of the accumulated activation in each PE
  - Terminate its computation once a negative values is detected
- Decoupled speculation and execution
  - Prediction separately
  - Execute the sparse computation with predicted output index

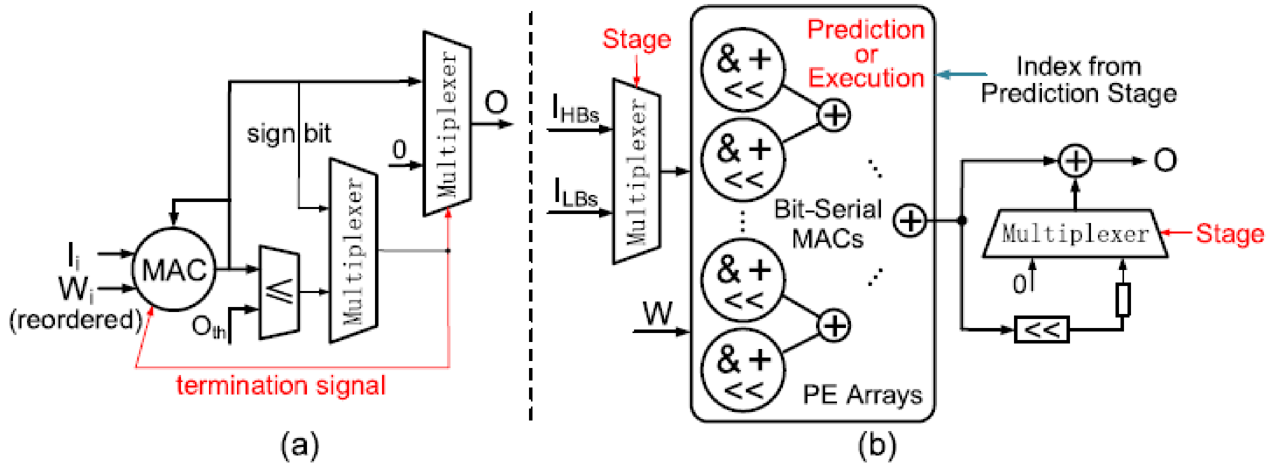


Fig. 32. *Examples of accelerators with output sparsity through (a) in-place speculation or (b) decoupled speculation and execution. Adapted from [242] and [244].*

Summary

- 모델 압축이 모델의 성능 저하를 일으키는 경향이 있지만, 무시할 정도의 수준으로 낮출 수 있음.
- 단일 하드웨어 최적화 측면에서 병렬화와 데이터 재사용에 대한 한계점들이 있었음
- 하드웨어와 압축 알고리즘을 함께 설계하는 방안이 증가하고 있음

Reference

- Model compression and hardware acceleration for neural networks: A comprehensive survey. Proceedings of the IEEE, 2020

- Nvidia Technical Blog

내용 정리

- notions