6.5930/1

Hardware Architecture for Deep Learning

Einsums + Transformers

February 13, 2026

Joel Emer and Vivienne Sze

Massachusetts Institute of Technology

Electrical Engineering and Computer Science

Extended Einsums

Acknowledgements

Based on collaborations with:

- Michael Gilbert (MIT)
- Nandeeka Nayak (UC Berkeley)
- Toluwa Odemuyiwa (UC Davis)
- Michael Pellauer (NVIDIA)

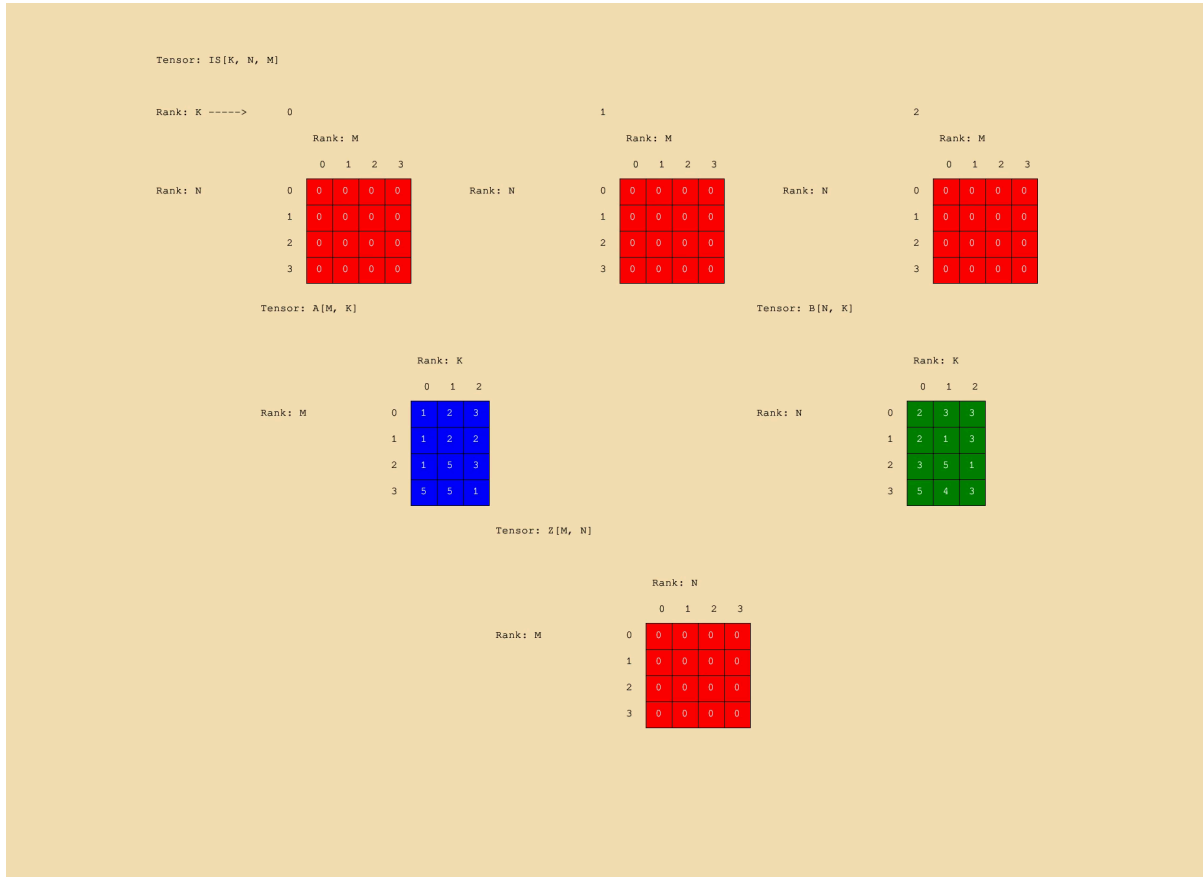
Example Einsum

$$Z_{m,n} = A_{m,k} \times B_{n,k}$$

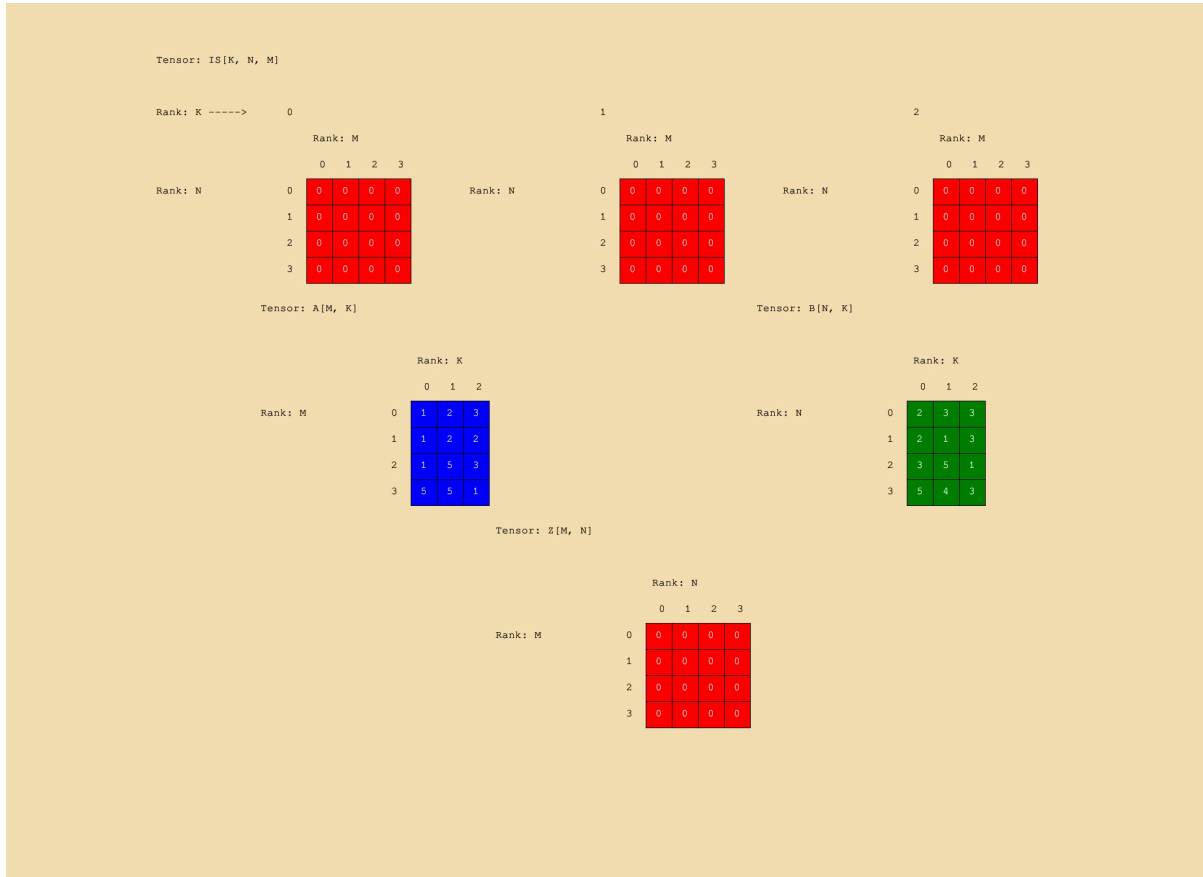
Operational Definition for Einsums (ODE):

- Traverse all points in space of all legal rank variable (index) values, i.e., the iteration space
- At each point in iteration space:
  - Calculate value on right hand at the specified rank variable values for each operand
  - Assign calculated value to operand at specified rank variable values on left hand side
  - Unless that operand is non-zero, then reduce value into it

Matrix multiply - simple traversal



Matrix multiply - complex traversal



Tensor References

$A_{k,m}$	• Rank variables: k, m • Rank names: "K", "M" • Rank shapes: K, M
$A^{K,M}_{k,m}$	• Rank variables: k, m • Rank names: "K", "M" • Rank shapes: K, M
$A^{K=X,M=X}_{k,m}$	• Rank variables: k, m • Rank names: "K", "M" • Rank shapes: X, X
$A^{RR=XX,SS=XX}_{k,m}$	• Rank variables: k, m • Rank names: "RR", "SS" • Rank shapes: XX, XX

Einsum Patterns

$$Z_{m,n} = A_{m,k} \times B_{k,n}$$ Matrix-matrix multiply

One rank (k) is contracted, and two ranks (m and n) are uncontracted

$$Z_{m,n} = A_{k,m} \times B_{k,n}$$ Matrix-matrix multiply

Again, one rank (k) is contracted, and two ranks (m and n) are uncontracted

$$Z_{n,m} = A_{k,m} \times B_{n,k}$$ Matrix-matrix multiply

Again, one rank (k) is contracted, and two ranks (m and n) are uncontracted

$$Z_{p,q} = A_{p,r} \times B_{q,r}$$ Matrix-matrix multiply

It doesn’t matter what the rank variables are.

More Einsum Patterns

$Z_m = A_{k,m} \times B_k$	Matrix-vector multiply
$Z_m = B_k \times A_{k,m}$	Matrix-vector multiply (its commutative)
$Z_{m,n} = A_m \times B_n$	Cartesian product
$Z_m = A_m \times B_m$	Element-wise multiply
$Z_m = A_m + B_m$	Element-wise addition (different operator)

Rank Variable Patterns

$$Z_i = A_i \times B_i$$ Element-wise matrix multiplication

$$Z_{i,j} = A_{i,j} \times B_{i,j}$$ Element-wise matrix multiplication

Repeated pairs of rank variables have no effect

$$Z_{(i,j)} = A_{(i,j)} \times B_{(i,j)}$$ Element-wise matrix multiplication

Repeated pairs of rank variables can be treated as one rank variable

$$Z_{ij} = A_{ij} \times B_{ij}$$ Element-wise matrix multiplication

So the tuple coordinates can be treated as a single coordinate, where

$$ij = i * J + j$$

Partitioned Einsum Patterns

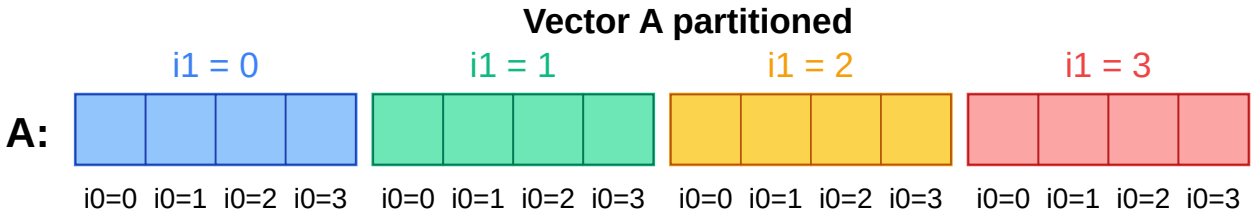
$$Z_i = A_i \times B_i$$ Element-wise matrix multiplication

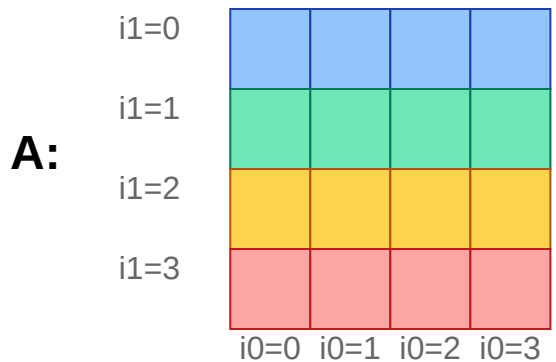
$$\text{If we assume } i = i1 \times I0 + i0$$

$$Z_{i1,i0} = A_{i1,i0} \times B_{i1,i0}$$ Element-wise matrix multiplication

Now we have a partitioned tensor {v-click}

Partitioned Tensor Visualization





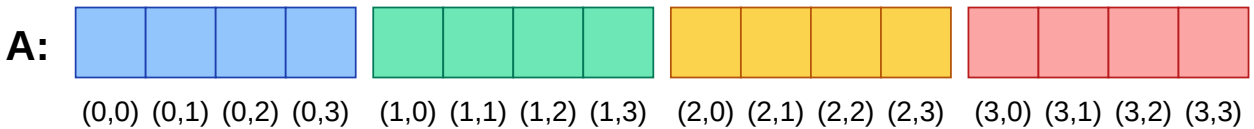
Partitioning always adds a rank {v-click}

Flattening

$$Z_{i1,i0} = A_{i1,i0} \times B_{i1,i0}$$

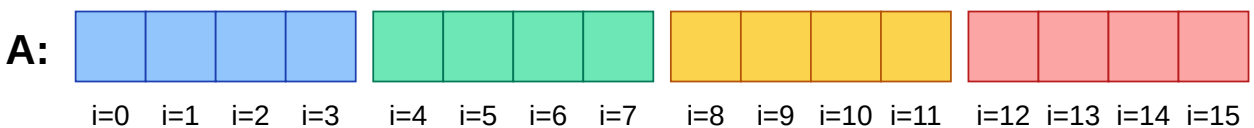
$$Z_{(i1,i0)} = A_{(i1,i0)} \times B_{(i1,i0)}$$

Vector A flattened



Since i = i1 * I0 + i0

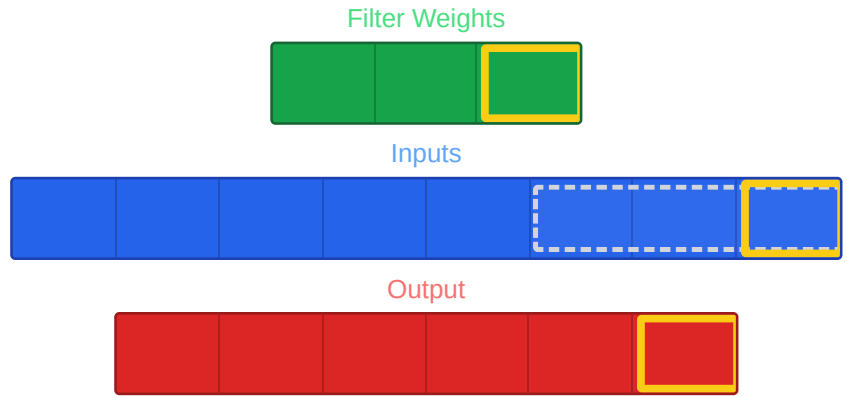
Vector A flattened and relabeled



Matrix-Matrix Multiplication Variants?

$Z_{m,n} = A_{k,m} \times B_{k,n}$		Matrix-matrix multiply
$Z_{m,p,n} = A_{k,m,p} \times B_{k,n}$	$Z_{mp,n} = A_{k,mp} \times B_{k,n}$	Matrix-matrix multiply
$Z_{m,n} = A_{k,l,m} \times B_{k,l,n}$	$Z_{m,n} = A_{kl,m} \times B_{kl,n}$	Matrix-matrix multiply
$Z_{m,p,n} = A_{k,l,m,p} \times B_{k,l,n}$	$Z_{mp,n} = A_{kl,mp} \times B_{kl,n}$	Matrix-matrix multiply

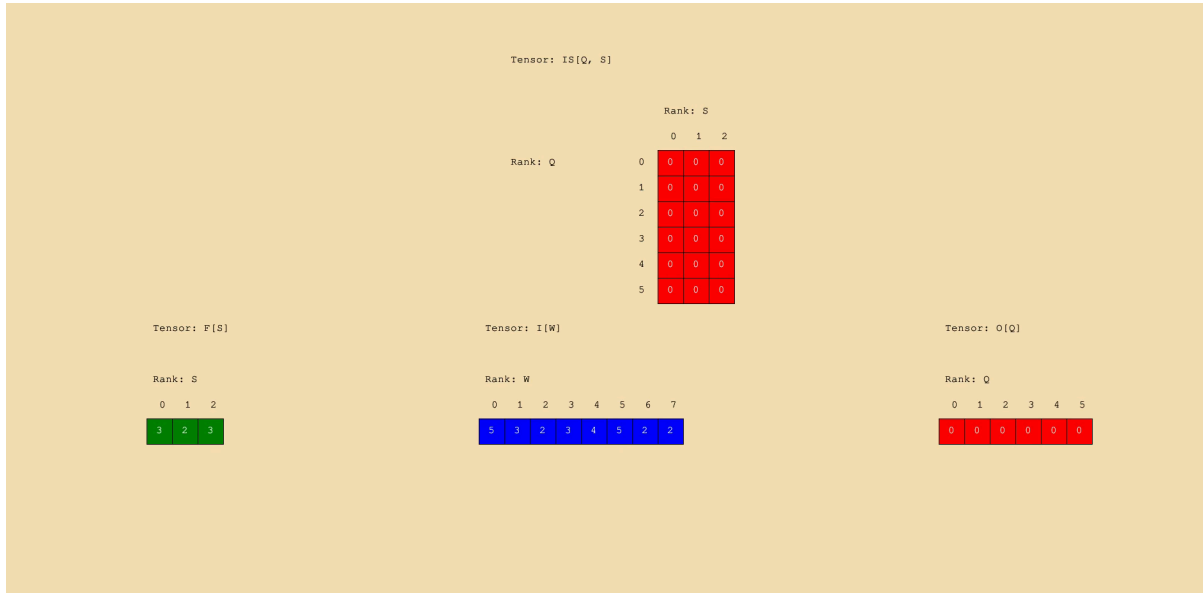
Convolution



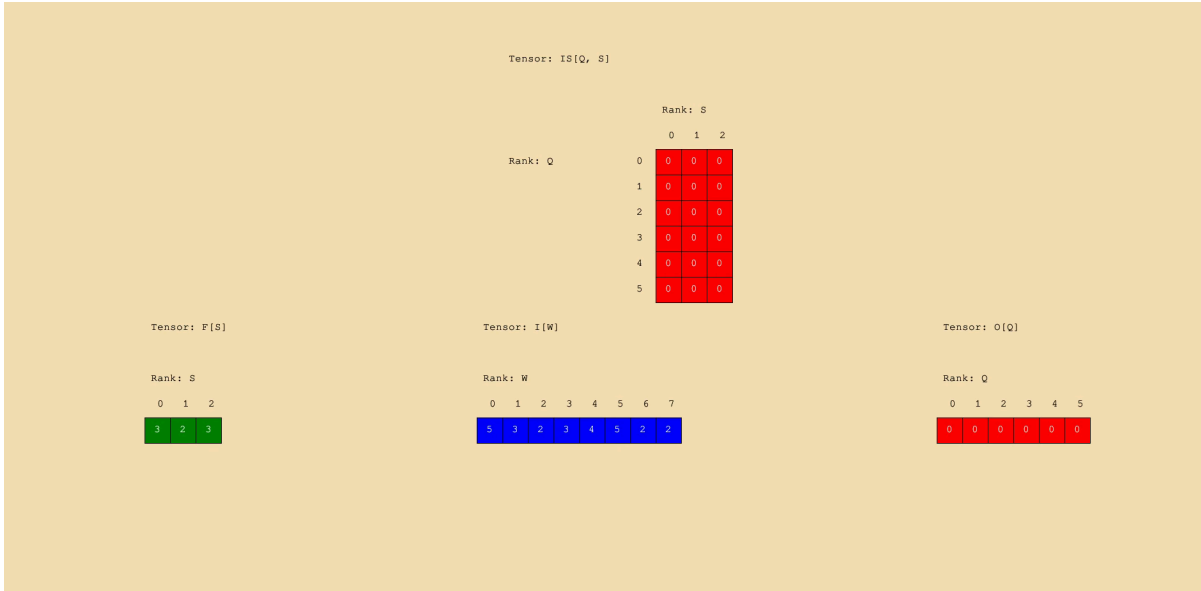
$$O_q = I_{q+s} \times F_s$$ One-dimensional convolution

$$O_q^Q = I_{q+s}^W \times F_s^S$$ One-dimensional convolution

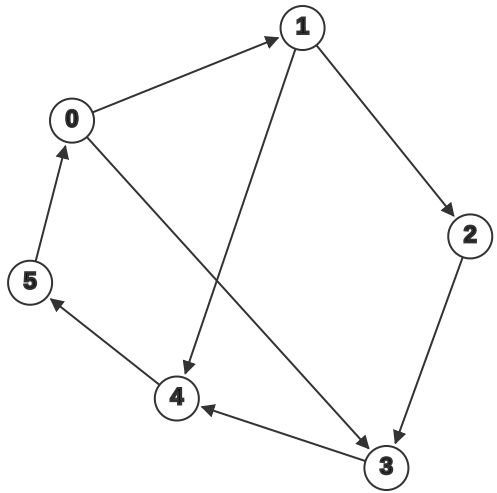
1D convolution - simple traversal

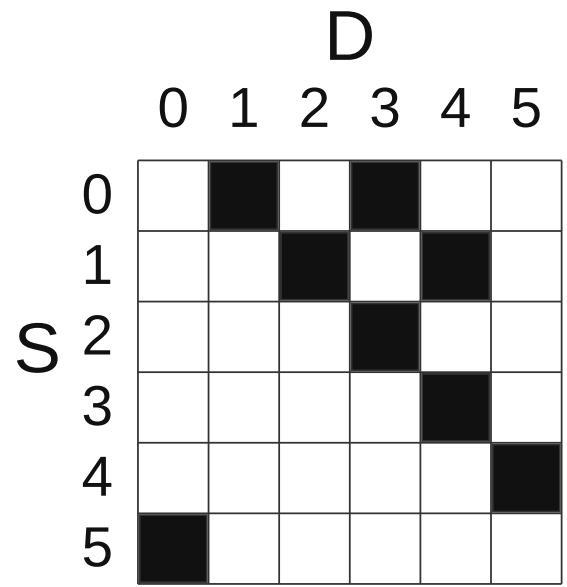


1D convolution - complex traversal



Rank Shapes





Adjaceny matrix declaration:

$$G^{S=6,D=6} \quad or \quad G^{S=V,D=V}$$

6.5930/1

Hardware Architectures for Deep Learning

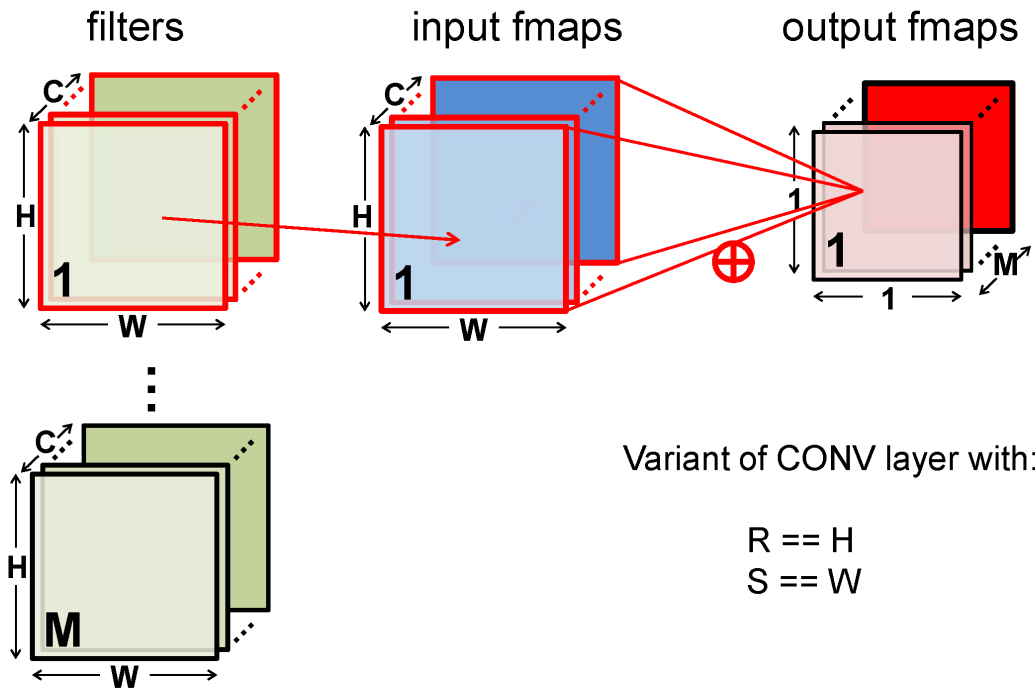
Kernel Computation

February 9, 2026

Joel Emer & Vivienne Sze

Massachusetts Institute of Technology
Electrical Engineering & Computer Science

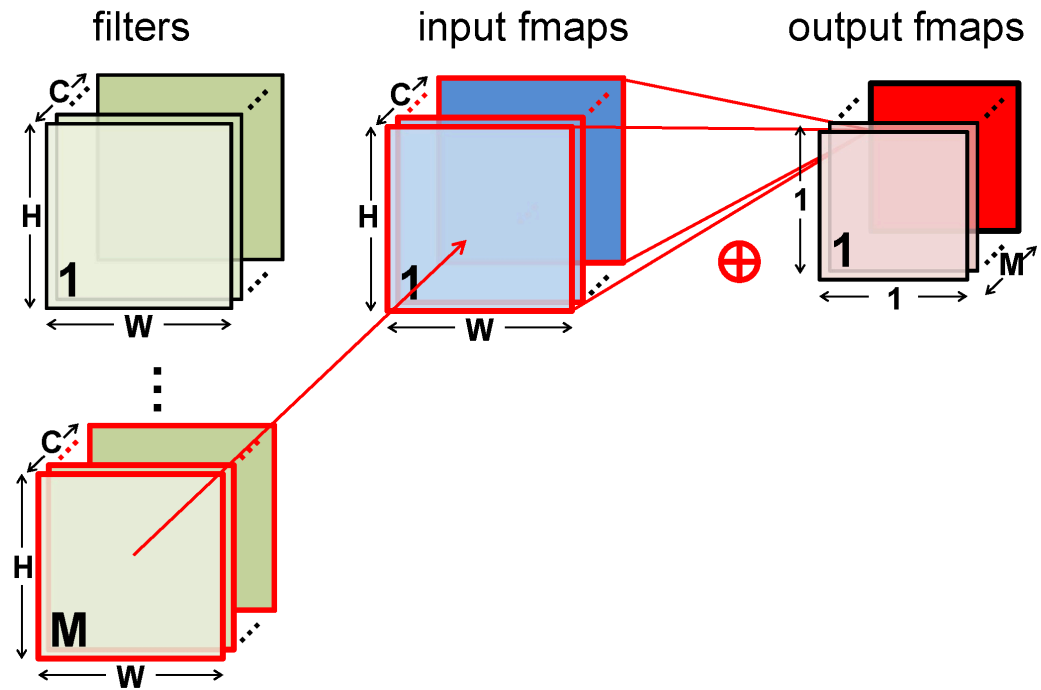
Fully Connected Computation



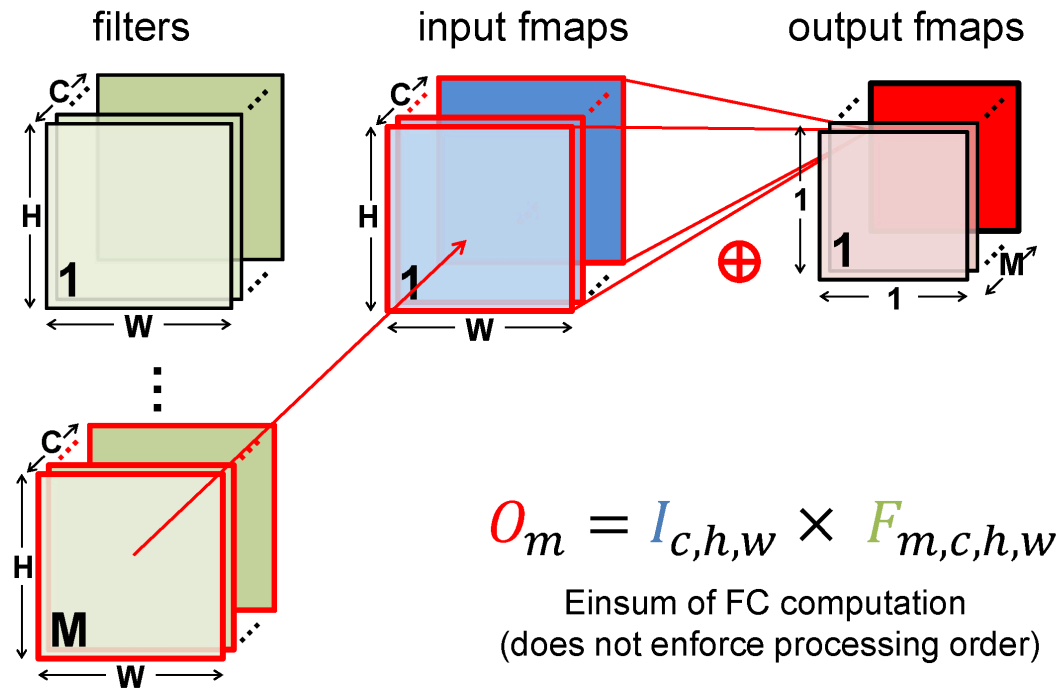
Variant of CONV layer with:

R == H
S == W

Fully Connected Computation



Fully Connected Computation



$$O_m = I_{c,h,w} \times F_{m,c,h,w}$$

Einsum of FC computation
(does not enforce processing order)

Fully Connected Computation

```
int i[C][H][W];      # Input activations
int f[M][C][H][W];   # Filter weights
int o[M];            # Output activations


for m in [0, M):
  o[m] = 0;
  for c in [0, C):
    for h in [0, H):
       for w in [0, W):
          o[m] += i[c][h][w]*f[m][c][h][w]
```

Loop nest of FC computation
(enforces some processing order)

Fully Connected Computation

```
int i[C][H][W];      # Input activations
int f[M][C][H][W];   # Filter weights
int o[M];            # Output activations


for m in [0, M):
  o[m] = 0;
  for c in [0, C):
    for h in [0, H):
       for w in [0, W):
          o[m] += i[c][h][w]*f[m][c][h][w]
```

Should be bias, which we will ignore for simplicity

Loop nest of FC computation
(enforces some processing order)

Convert FC Compute to Matrix-Vector Multiply

```
int i[C][H][W];      # Input activations
int f[M][C][H][W];   # Filter weights
int o[M];            # Output activations


for m in [0, M):
  o[m] = 0;
  for c in [0, C):
    for h in [0, H):
      for w in [0, W):
        o[m] += i[c][h][w]*f[m][c][h][w]
```

Flatten C, H, W ranks to CHW

```
int i[CHW];          # Input activations
int f[M][CHW];       # Filter weights
int o[M];            # Output activations


for m in [0, M):
  o[m] = 0;
    for chw in [0, CHW):
      o[m] += i[chw]*f[CHW*m + chw]
```

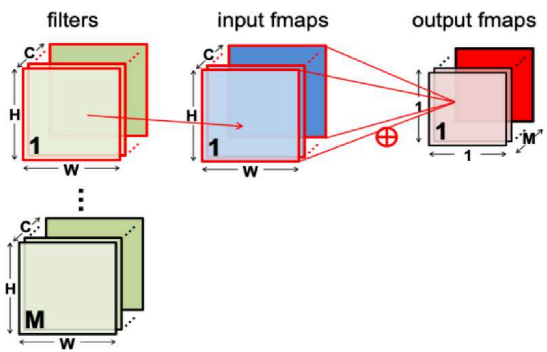
Convert FC Compute to Matrix-Vector Multiply

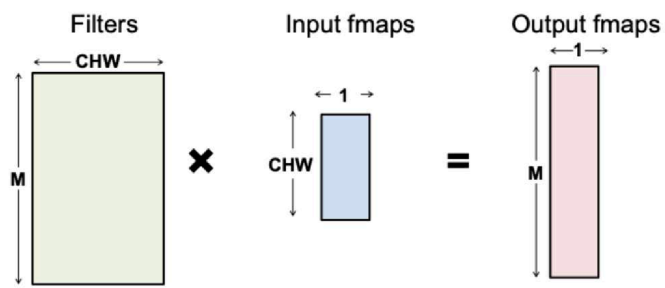
```
int i[C][H][W];      # Input activations
int f[M][C][H][W];   # Filter weights
int o[M];            # Output activations


for m in [0, M):
  o[m] = 0;
  for c in [0, C):
    for h in [0, H):
      for w in [0, W):
        o[m] += i[c][h][w]*f[m][c][h][w]
```

```
int i[CHW];          # Input activations
int f[M][CHW];       # Filter weights
int o[M];            # Output activations


for m in [0, M):
  o[m] = 0;
    for chw in [0, CHW):
      o[m] += i[chw]*f[m][chw]
```

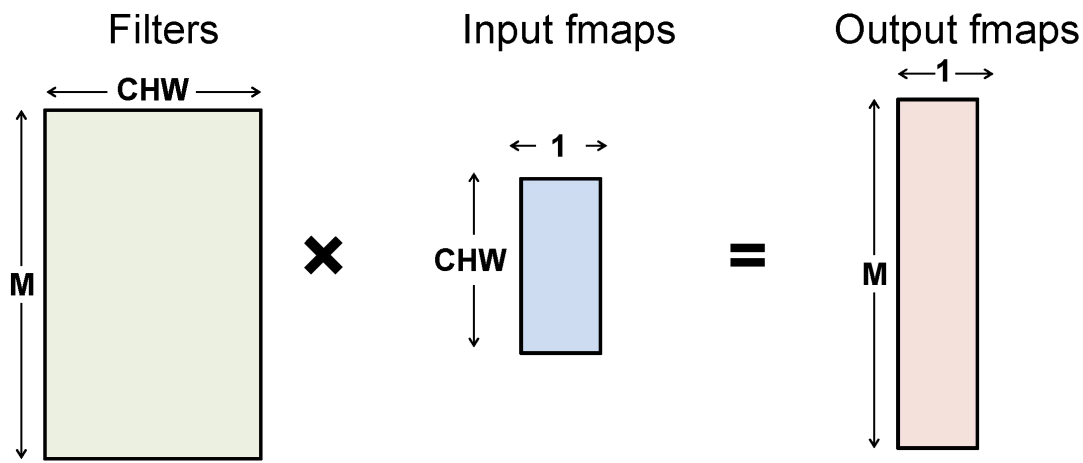




Sze and Emer

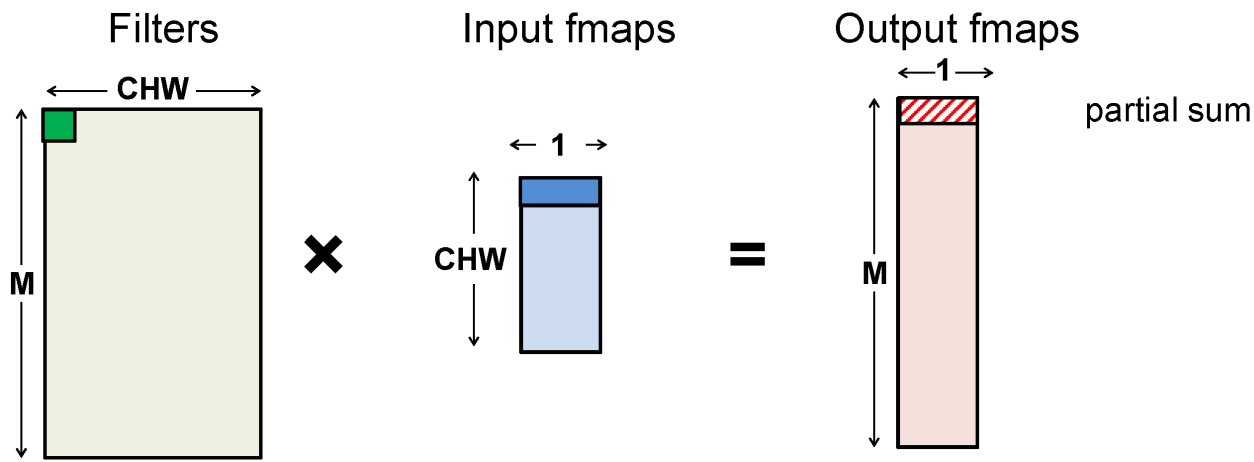
FC Compute as Matrix-Vector Multiply

Multiply all inputs in all channels by a weight and sum



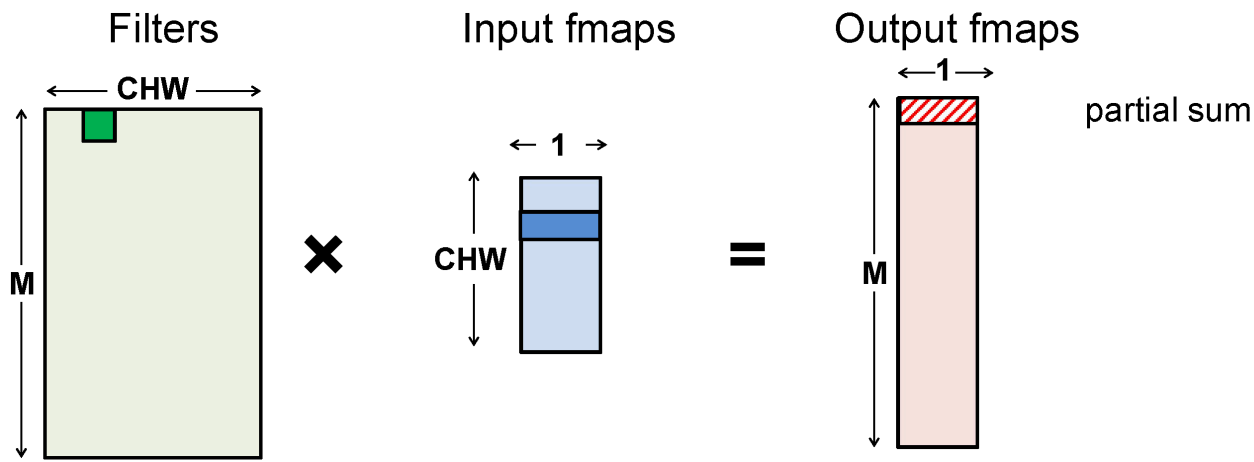
FC Compute as Matrix-Vector Multiply

Multiply all inputs in all channels by a weight and sum
(increment chw)



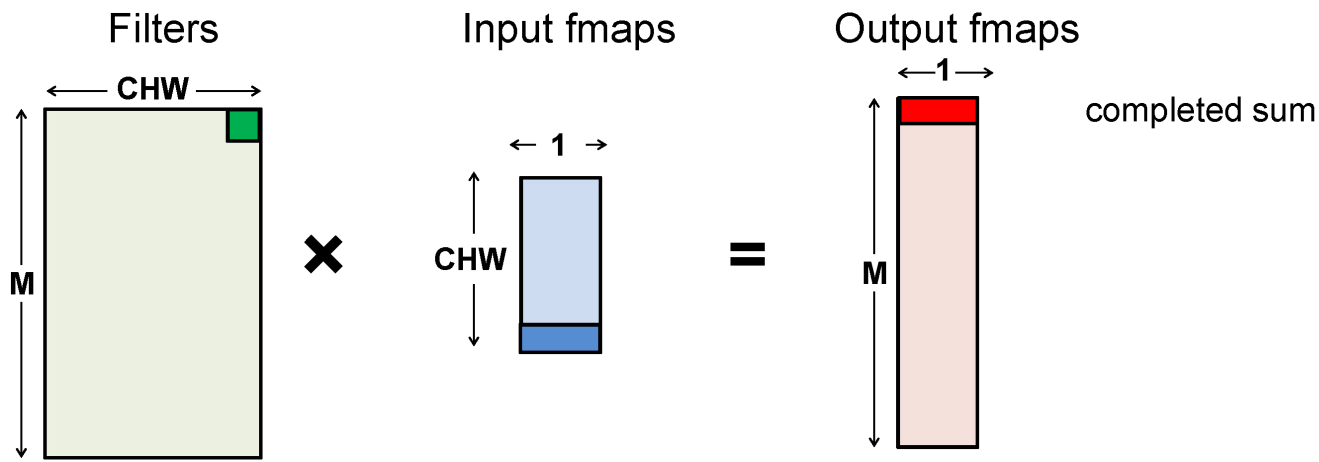
FC Compute as Matrix-Vector Multiply

Multiply all inputs in all channels by a weight and sum
(increment chw)



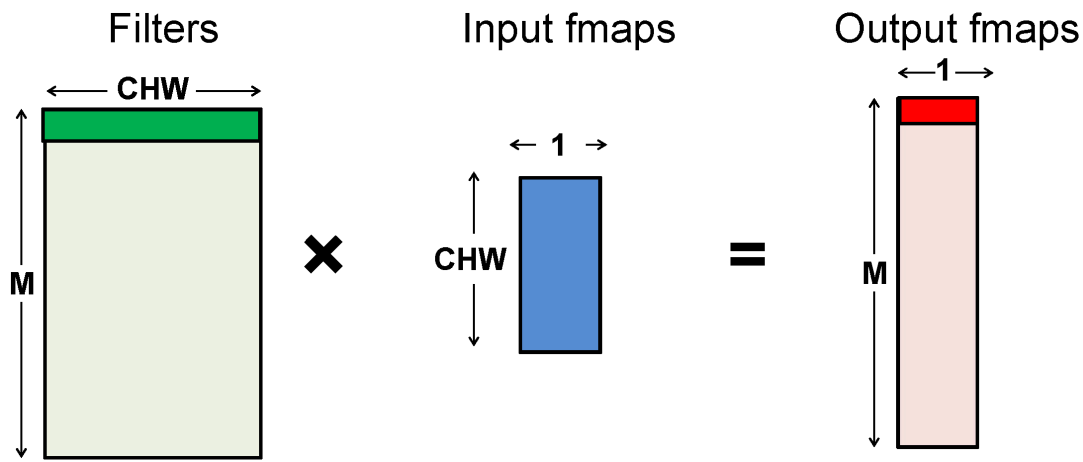
FC Compute as Matrix-Vector Multiply

Multiply all inputs in all channels by a weight and sum
(increment chw)



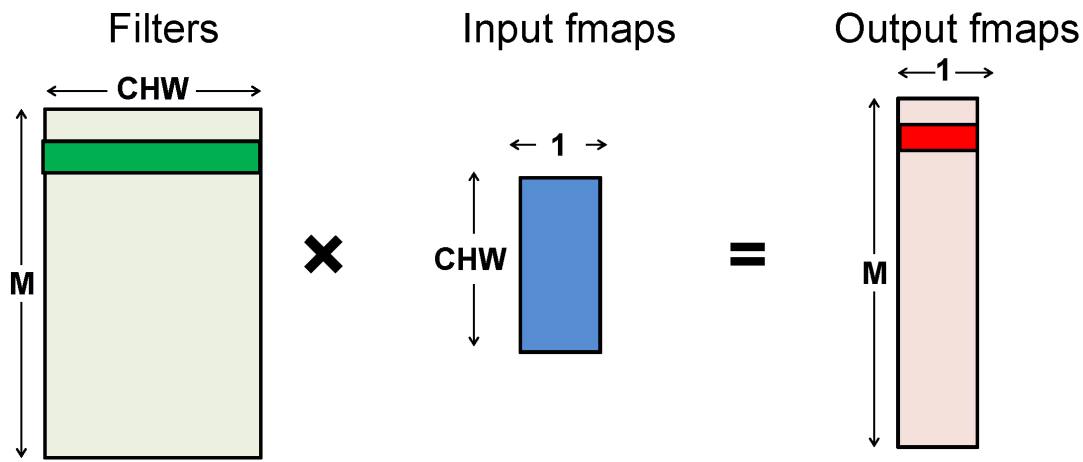
FC Compute as Matrix-Vector Multiply

Multiply all inputs in all channels by a weight and sum



FC Compute as Matrix-Vector Multiply

Multiply all inputs in all channels by a weight and sum
(increment m)



Einsum for Flattened FC

Original

Flattened

Einsum for Flattened FC

Original Flattened

$$I_{c,h,w} \rightarrow I_{H \times W \times c + W \times h + w} \rightarrow I_{chw}$$

Einsum for Flattened FC

Original Flattened

$$I_{c,h,w} \rightarrow I_{H\times W\times c+W\times h+w} \rightarrow I_{chw}$$

$$F_{m,c,h,w} \rightarrow F_{m,H\times W\times c+W\times h+w} \rightarrow F_{m,chw}$$

Einsum for Flattened FC

Original Flattened

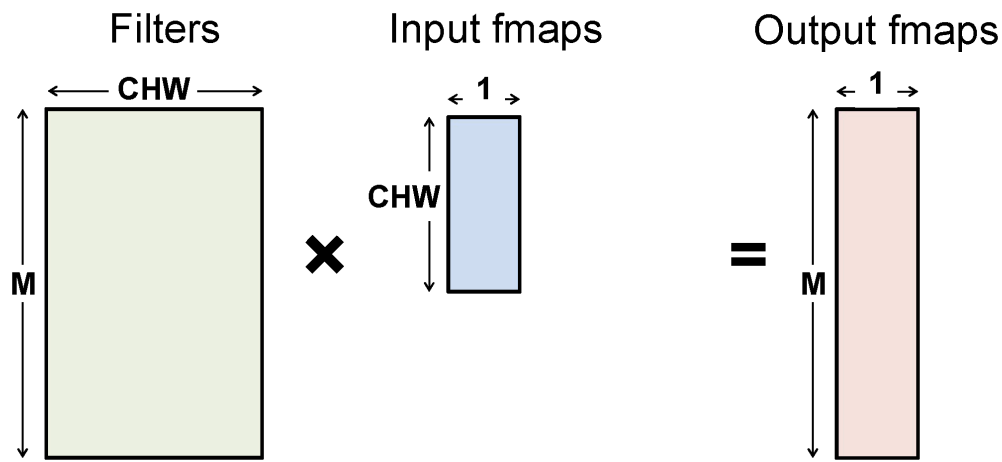
$$I_{c,h,w} \rightarrow I_{H\times W\times c+W\times h+w} \rightarrow I_{chw}$$

$$F_{m,c,h,w} \rightarrow F_{m,H\times W\times c+W\times h+w} \rightarrow F_{m,chw}$$

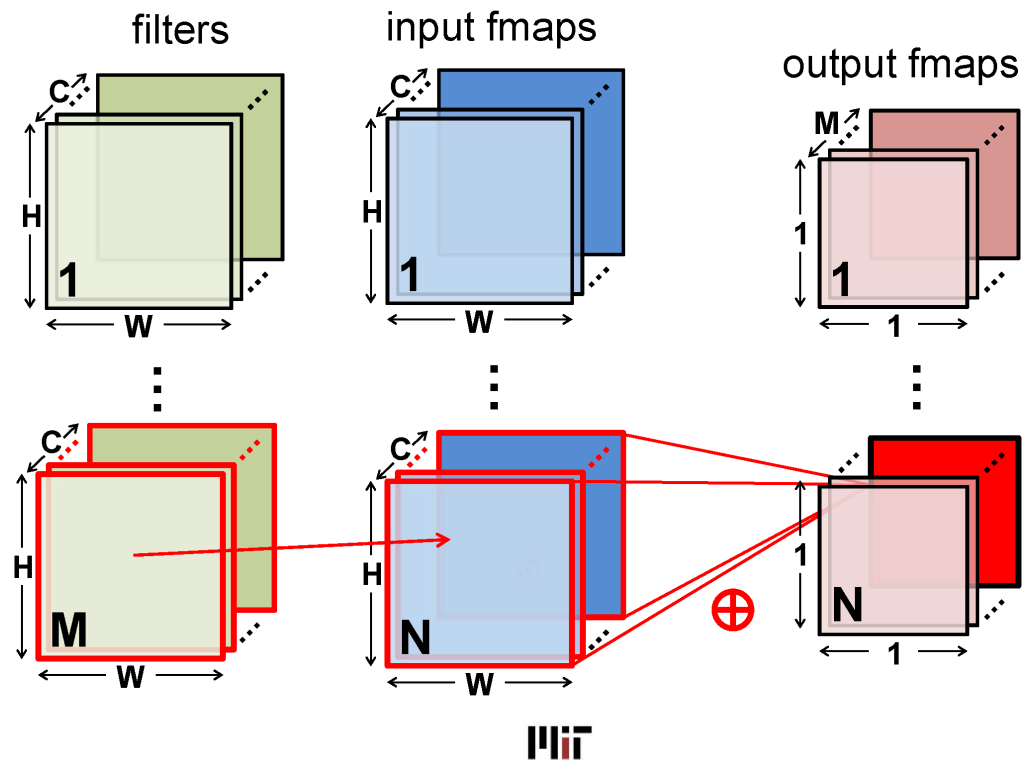
$$O_m = I_{c,h,w} \times F_{m,c,h,w} \rightarrow O_m = I_{chw} \times F_{m,chw}$$

Einsum for FC as Matrix Vector

$$O_m = I_{chw} \times F_{m,chw}$$

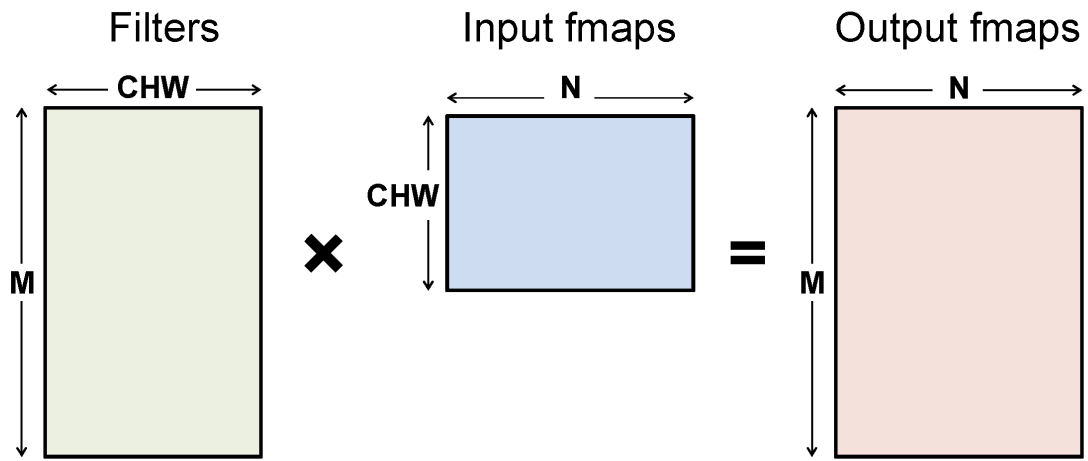


FC Layer – Batch (N)



FC Compute → Matrix-Matrix Multiply

$$O_{n,m} = I_{n,chw} \times F_{m,chw}$$



After flattening, having a batch size of N turns the **matrix-vector** multiply into a **matrix-matrix** multiply

FC Compute → Matrix-Matrix Multiply

$$O_{n,m} = I_{n,chw} \times F_{m,chw}$$

reduction on rank **chw**

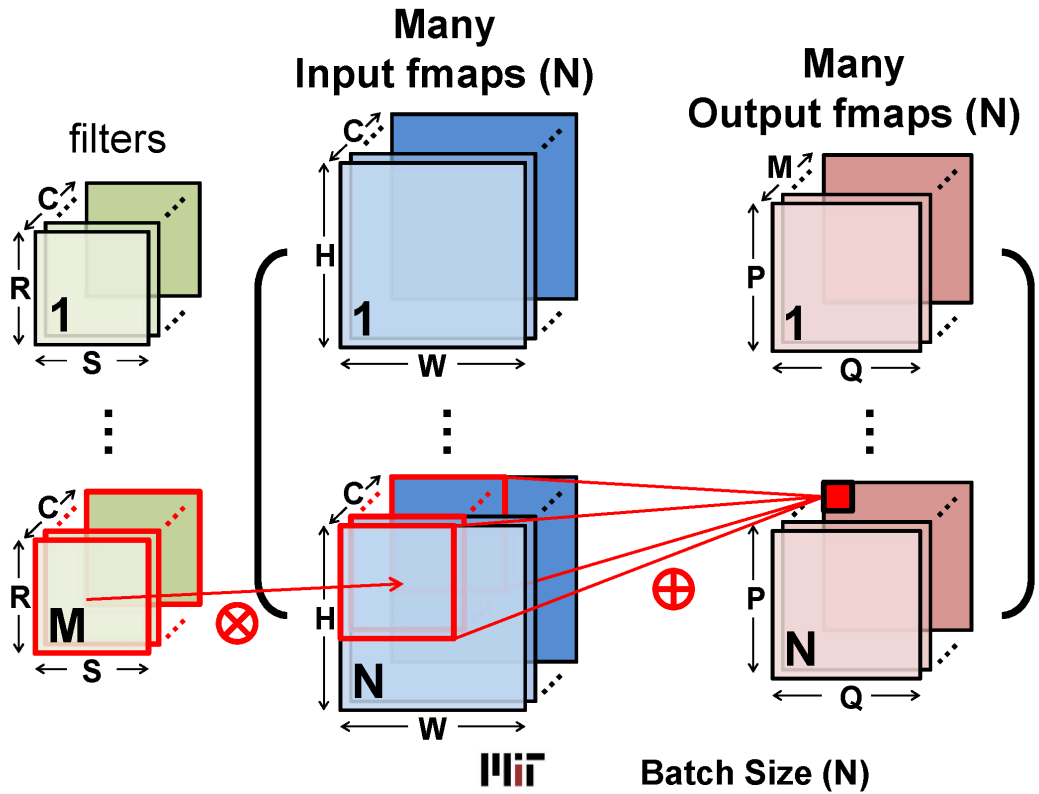
Typical matrix multiplication notation (used in Lab)

$$Z_{m,n} = A_{m,k} \times B_{k,n}$$

reduction on rank **k**

Note: for Einsum, the order of ranks does not matter

Convolution (CONV) Layer



Convolution Einsum

Algebraic Notation

$$\mathbf{o}[n][m][p][q] = \mathbf{b}[m] + \sum_{c=0}^{C-1} \sum_{r=0}^{R-1} \sum_{s=0}^{S-1} \mathbf{i}[n][c][Up+r][Uq+s] \times \mathbf{f}[m][c][r][s]$$

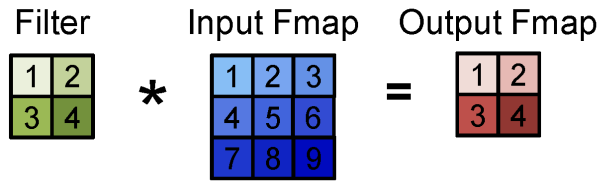
Einsum Notation

$$O_{n,m,p,q} = B_m + I_{n,c,U\times p+r,U\times q+s} \times F_{m,c,r,s}$$

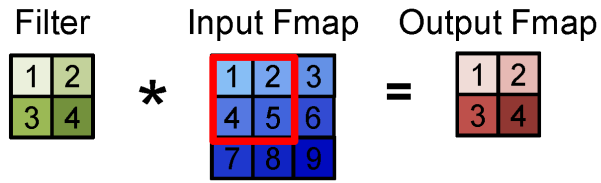
Note:

- There is a relationship between the ranks for the tensors used for convolution
  - *Not the case for fully connected*
- Specifically, h=U×p+r and w=U×q+s → Use change of variables to capture this relationship

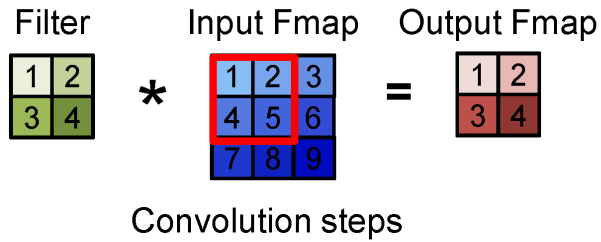
Convolution (CONV) Layer



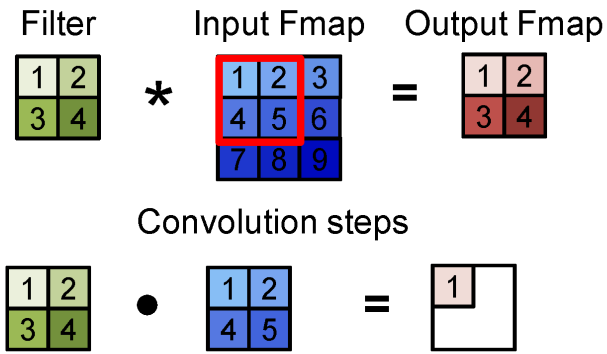
Convolution (CONV) Layer



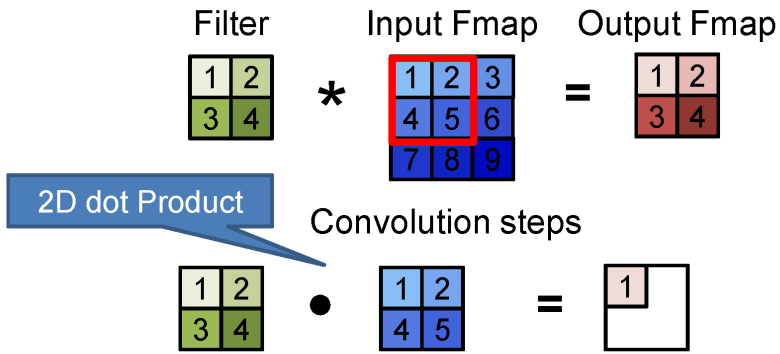
Convolution (CONV) Layer



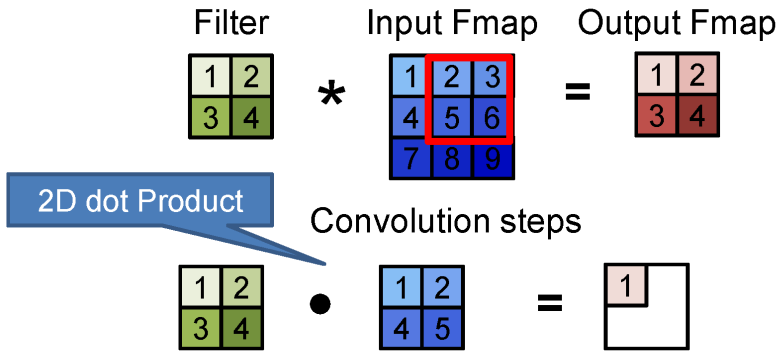
Convolution (CONV) Layer



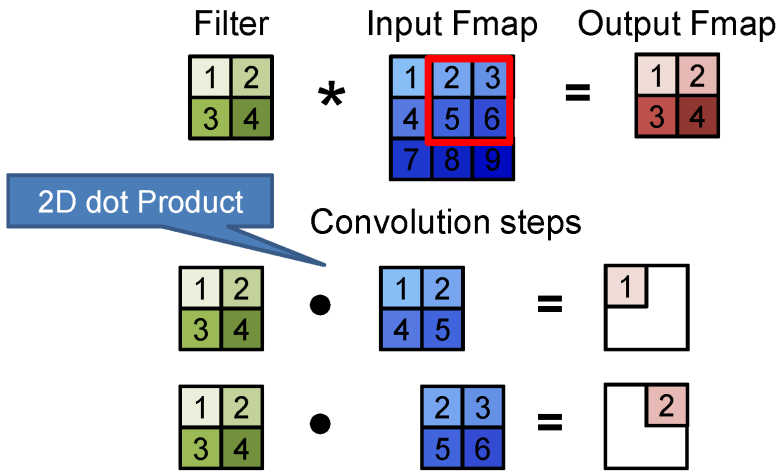
Convolution (CONV) Layer



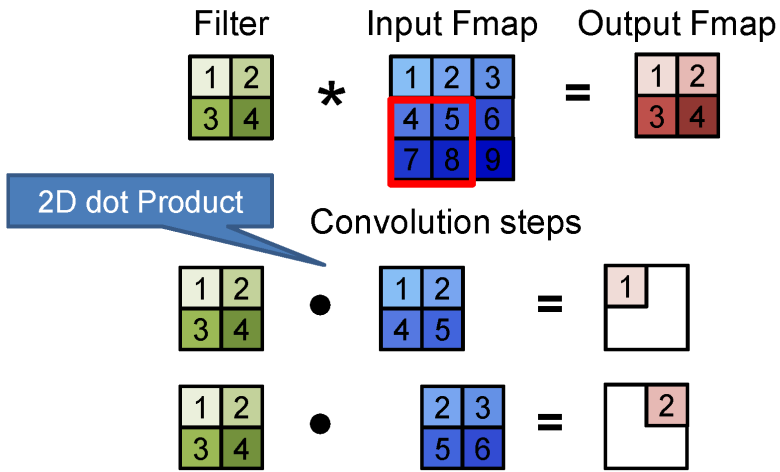
Convolution (CONV) Layer



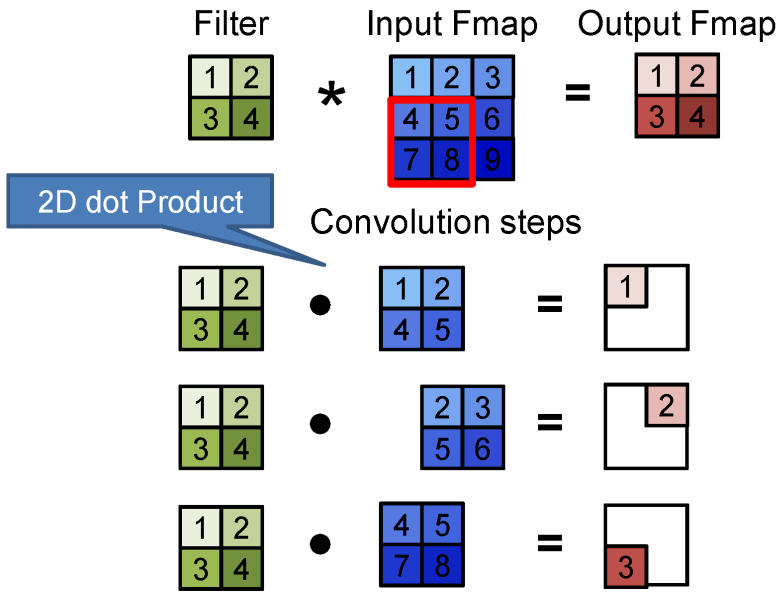
Convolution (CONV) Layer



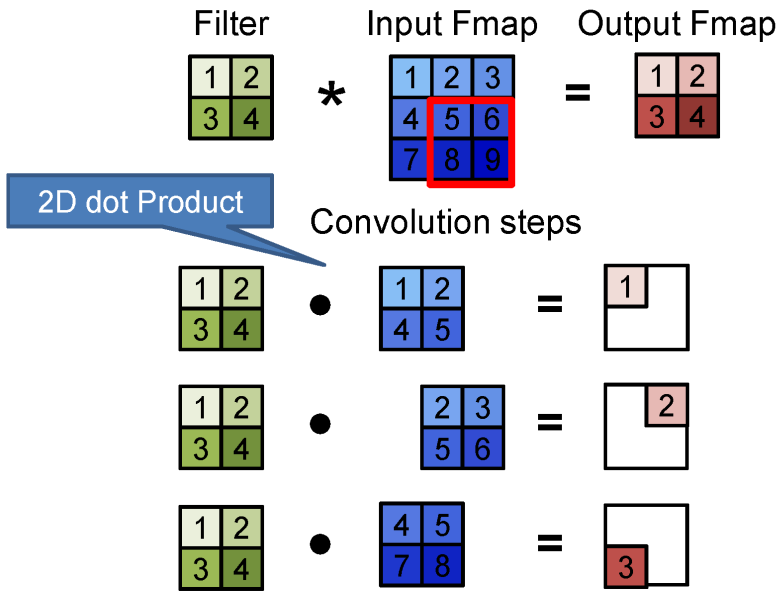
Convolution (CONV) Layer



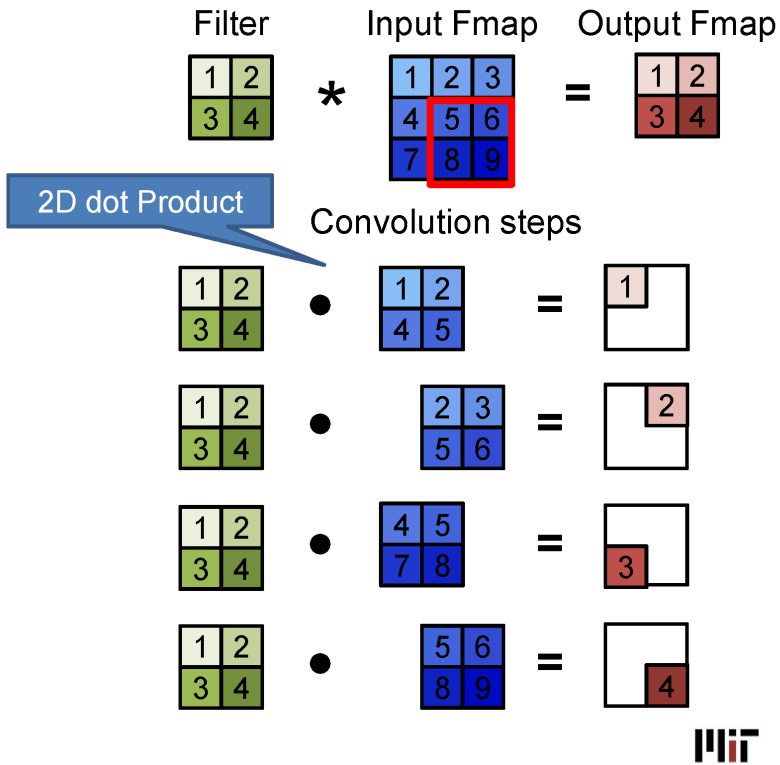
Convolution (CONV) Layer



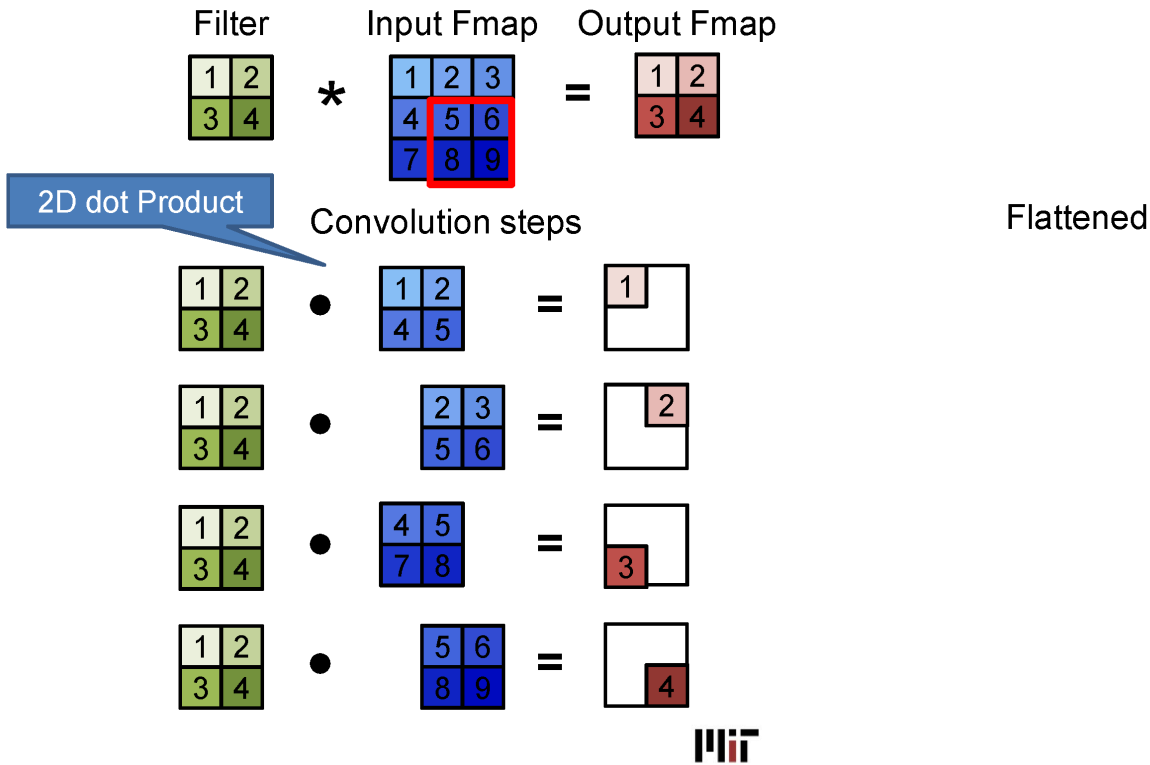
Convolution (CONV) Layer



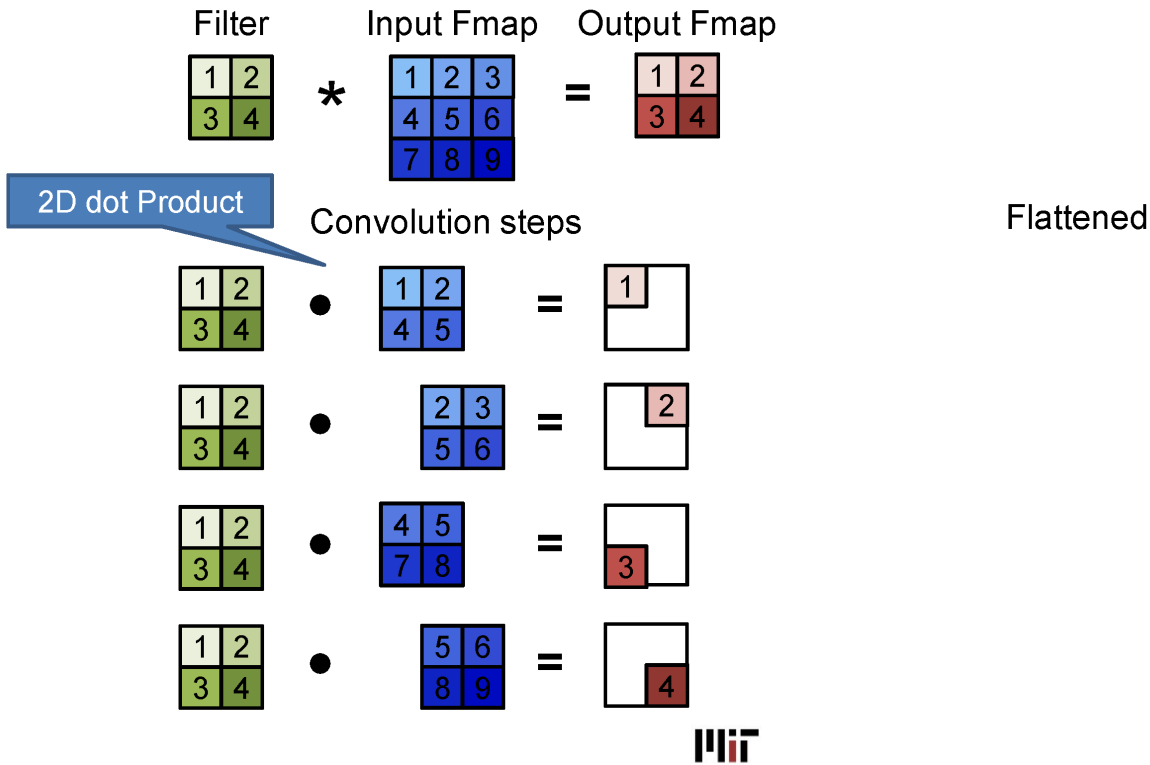
Convolution (CONV) Layer



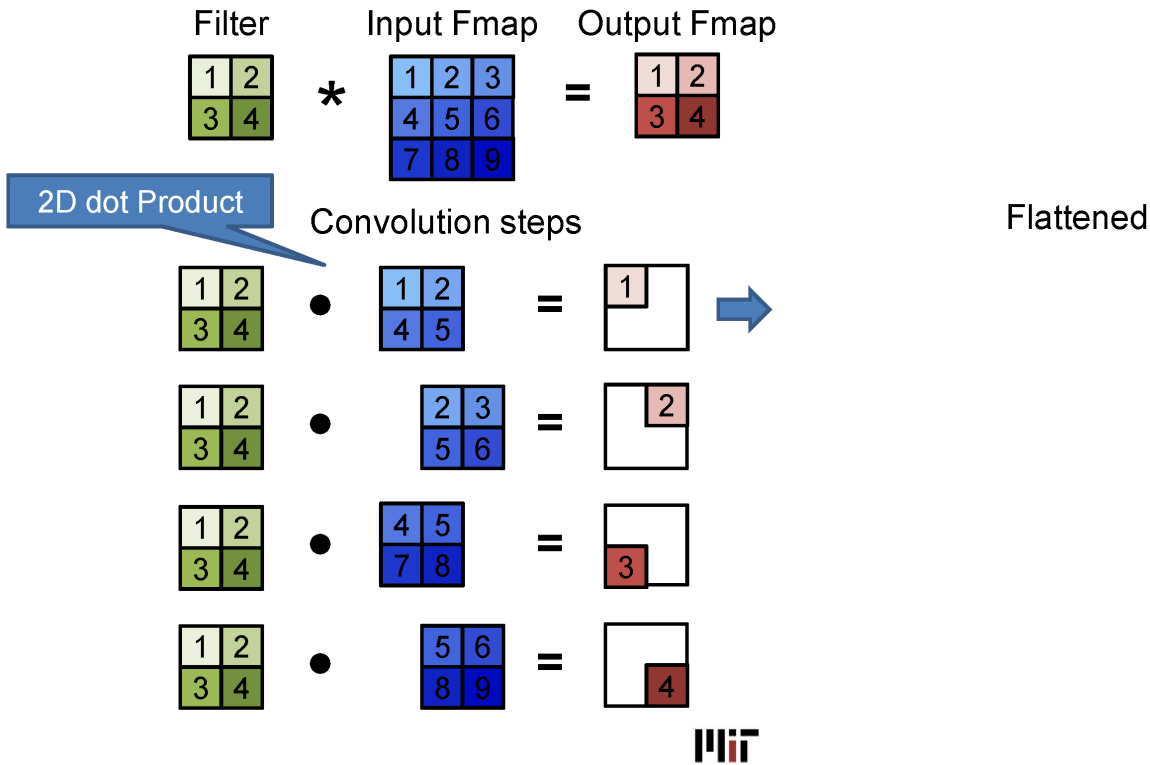
Convolution (CONV) Layer



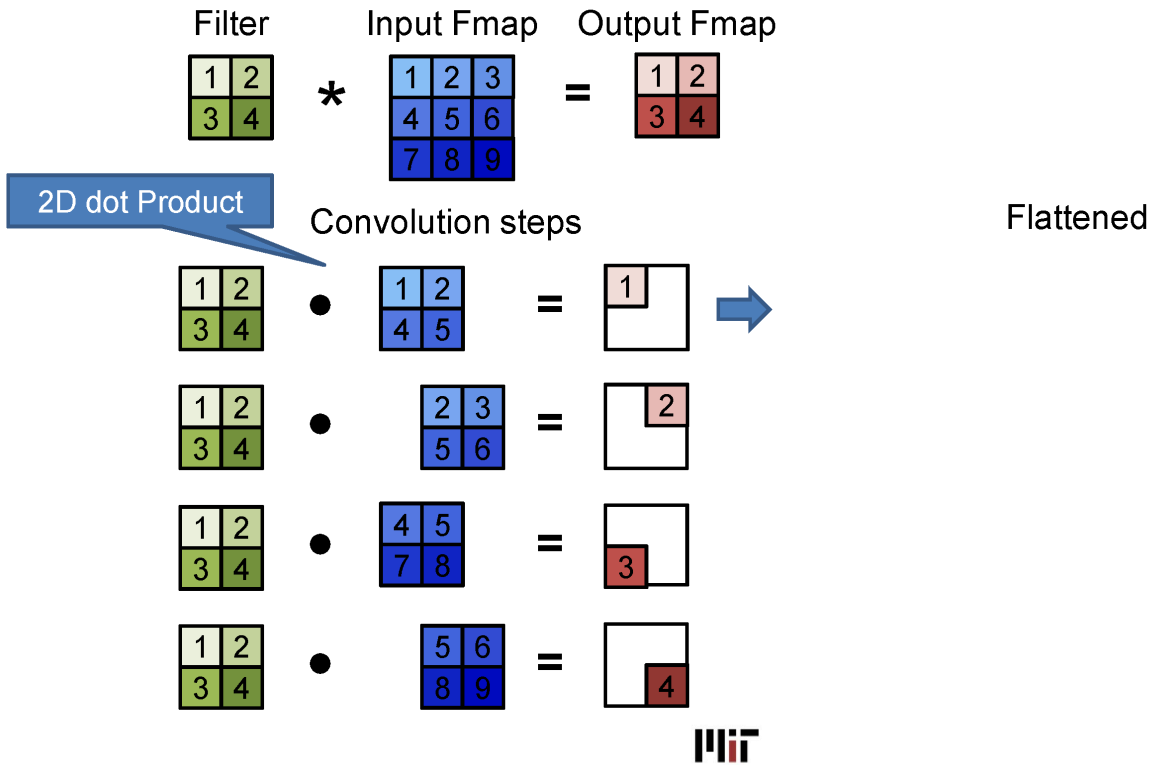
Convolution (CONV) Layer



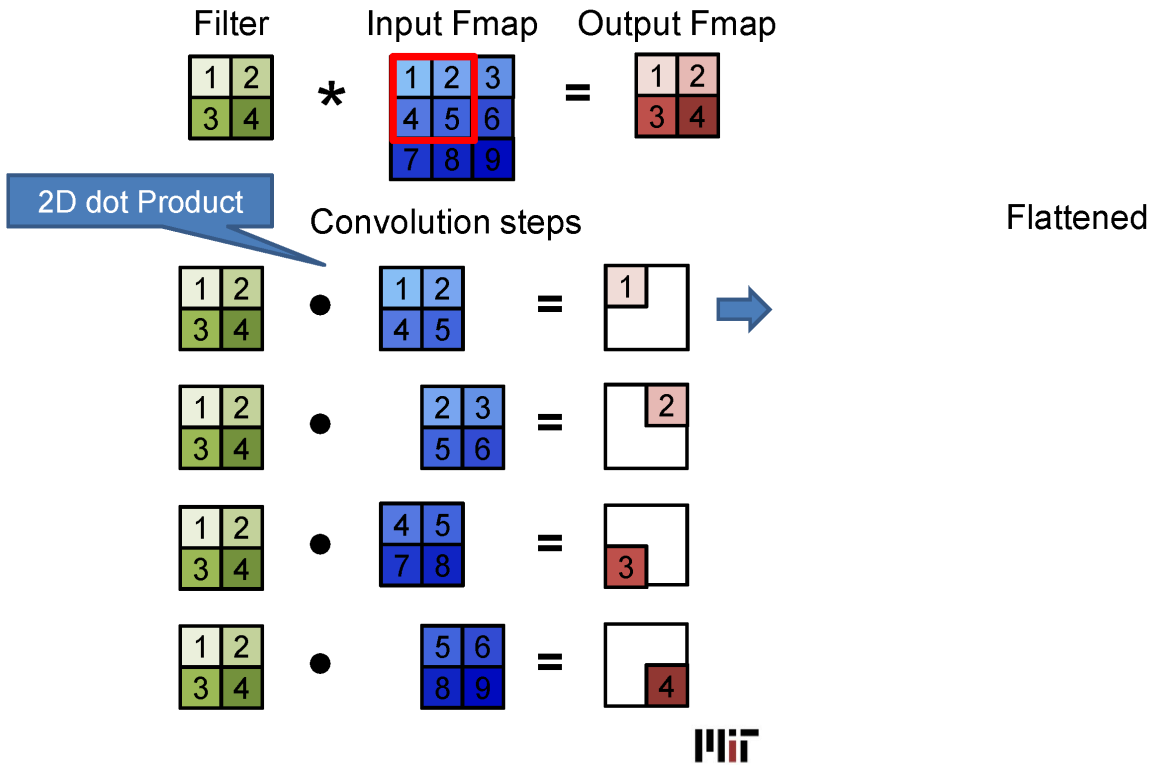
Convolution (CONV) Layer



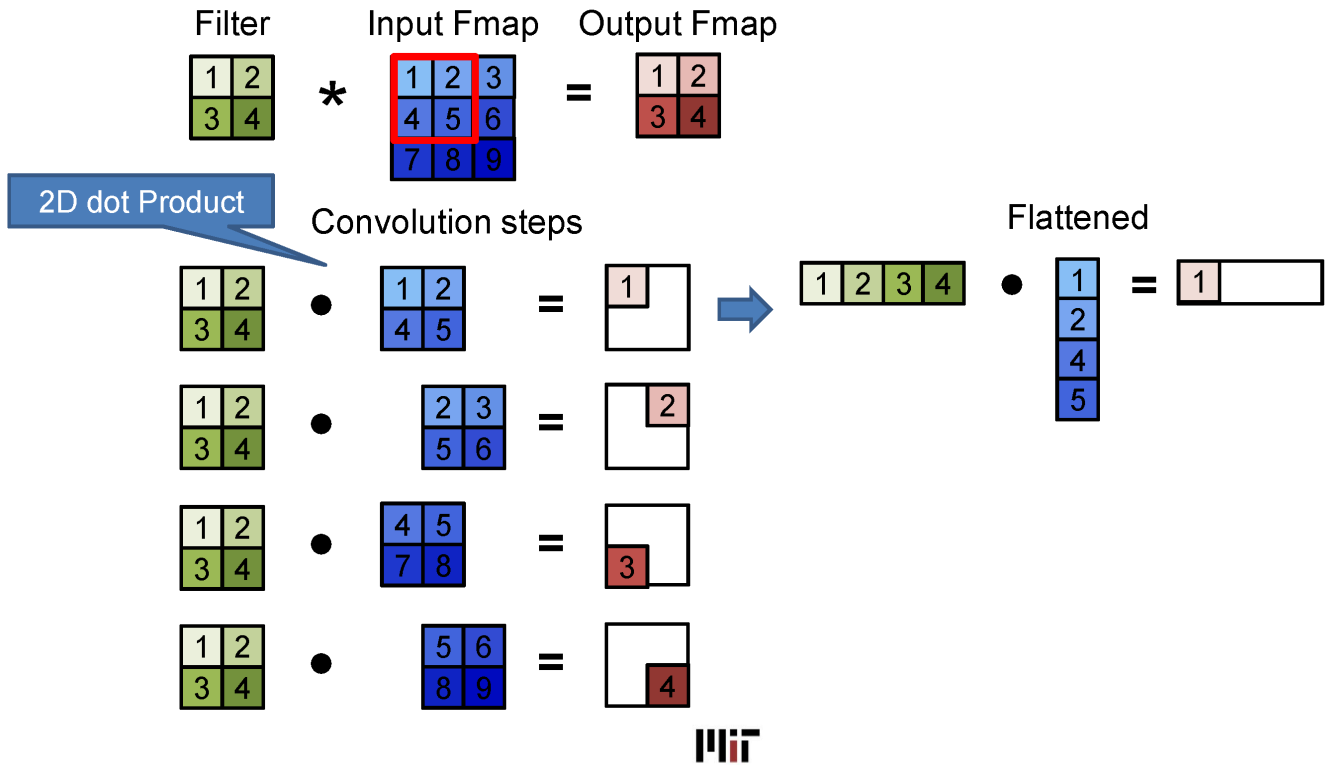
Convolution (CONV) Layer



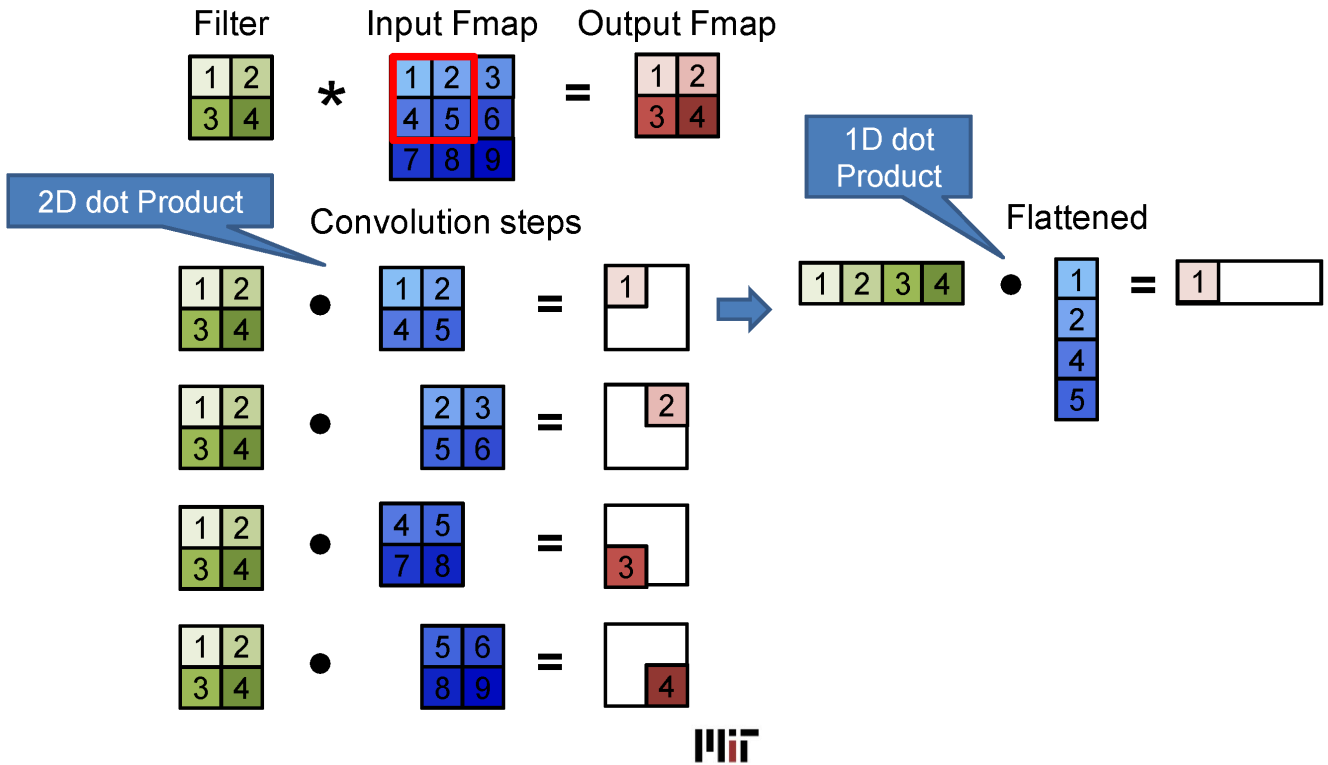
Convolution (CONV) Layer



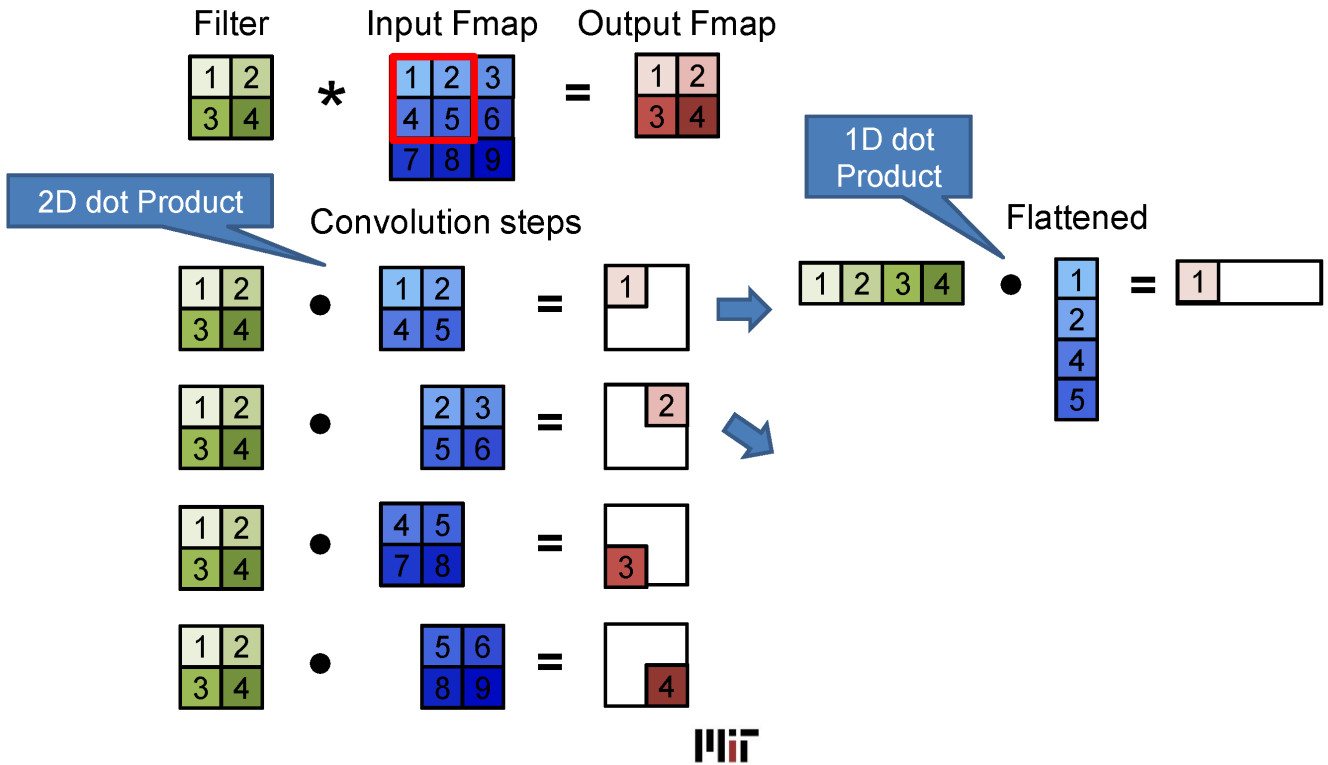
Convolution (CONV) Layer



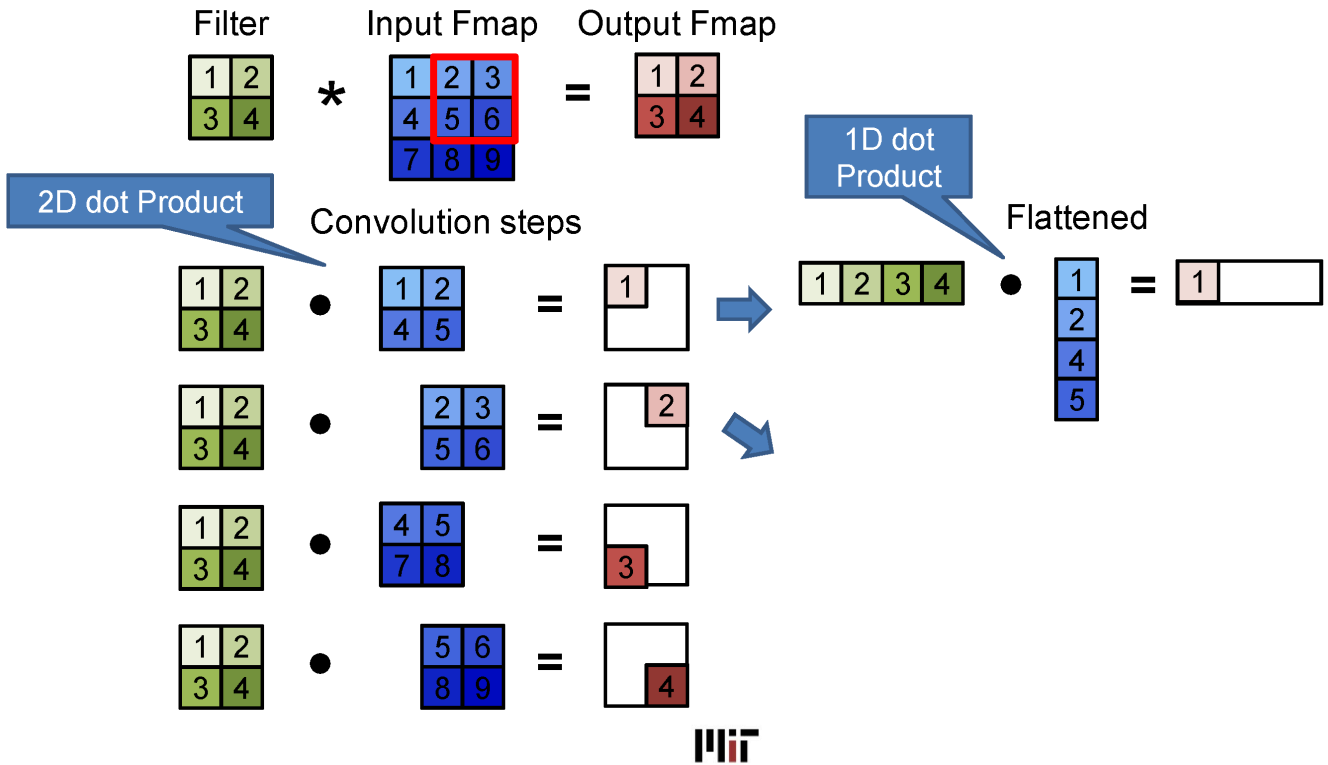
Convolution (CONV) Layer



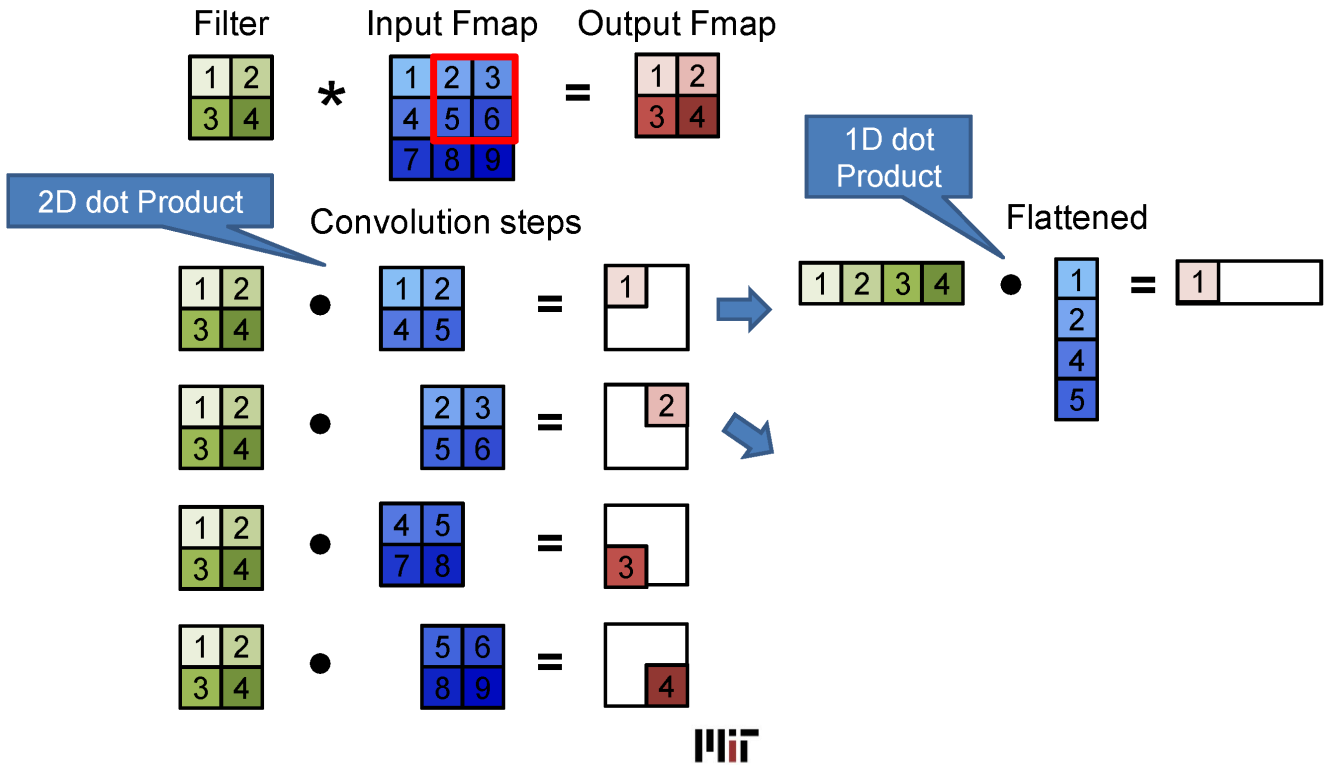
Convolution (CONV) Layer



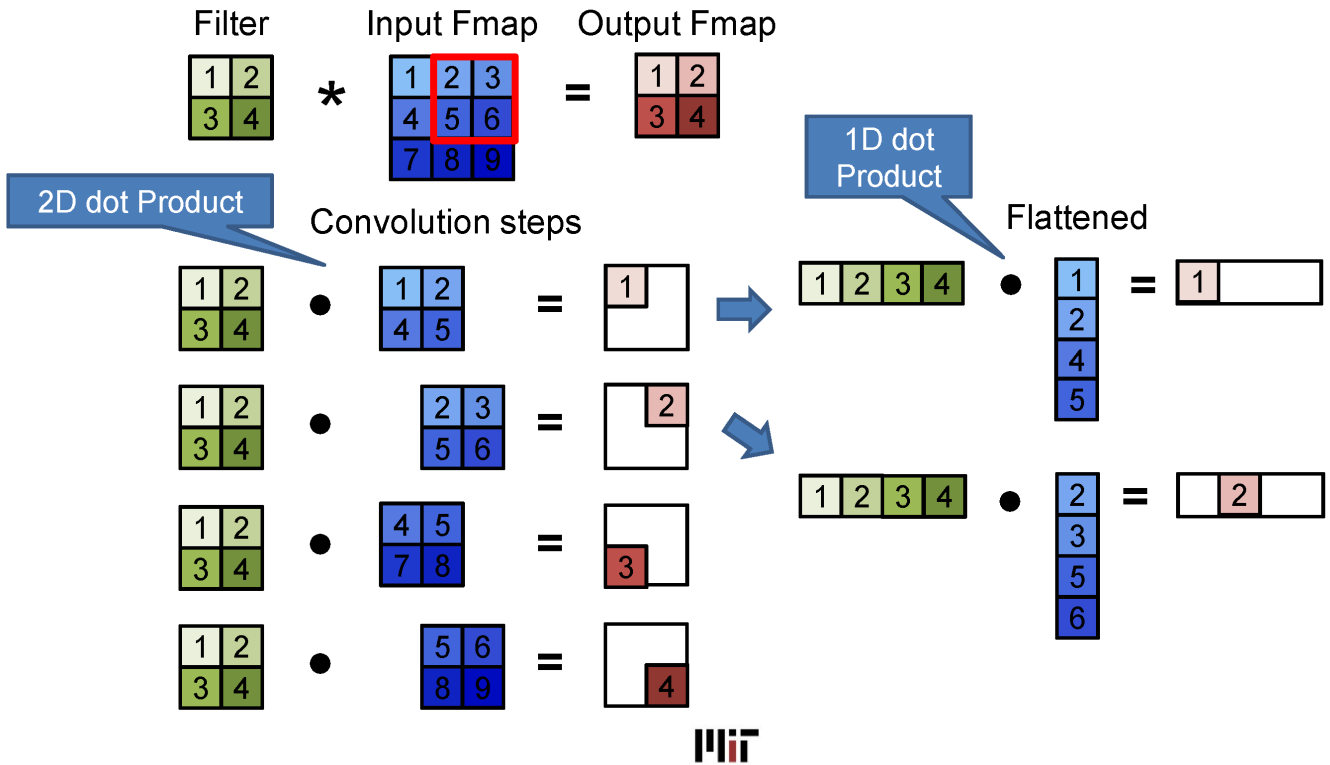
Convolution (CONV) Layer



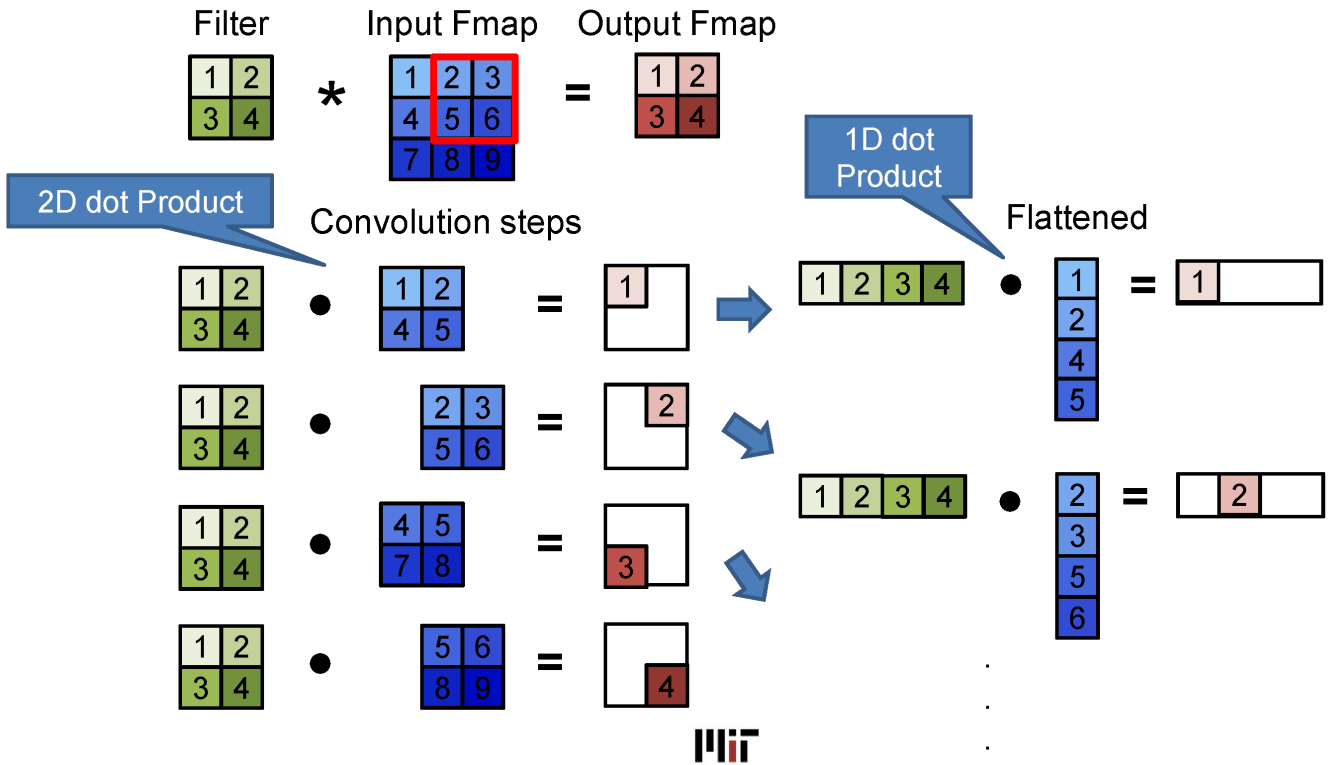
Convolution (CONV) Layer



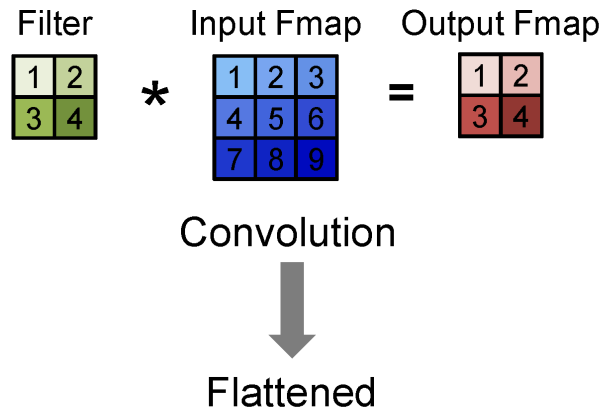
Convolution (CONV) Layer



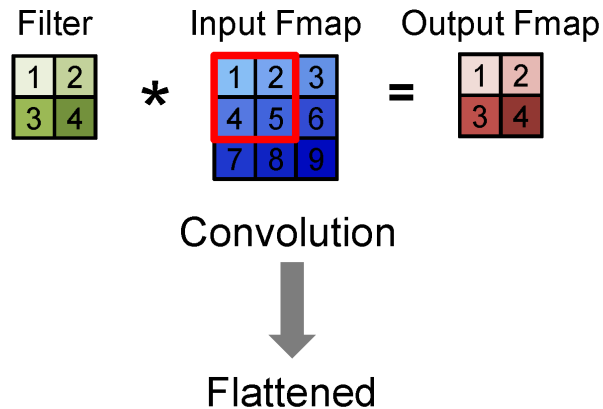
Convolution (CONV) Layer



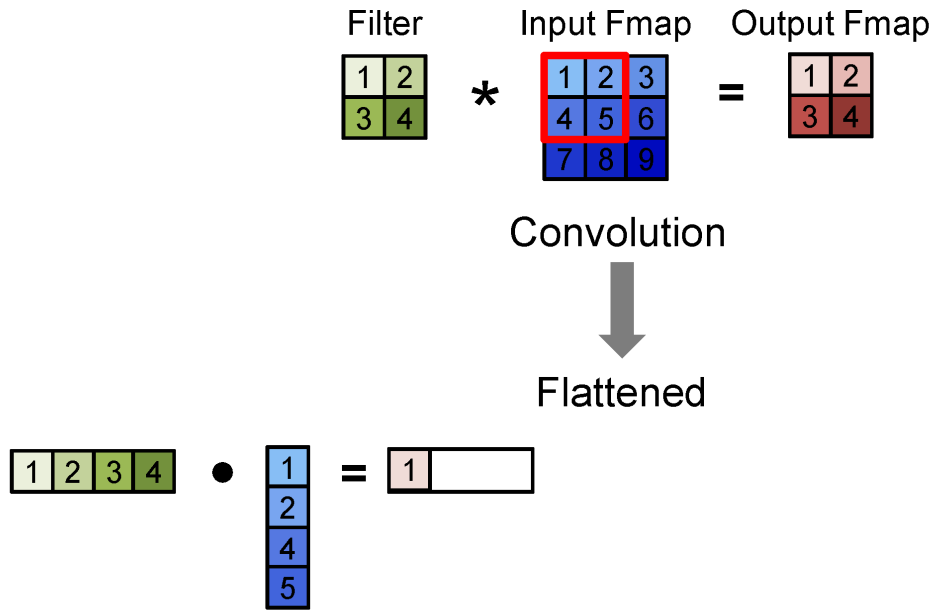
Convolution (CONV) Layer



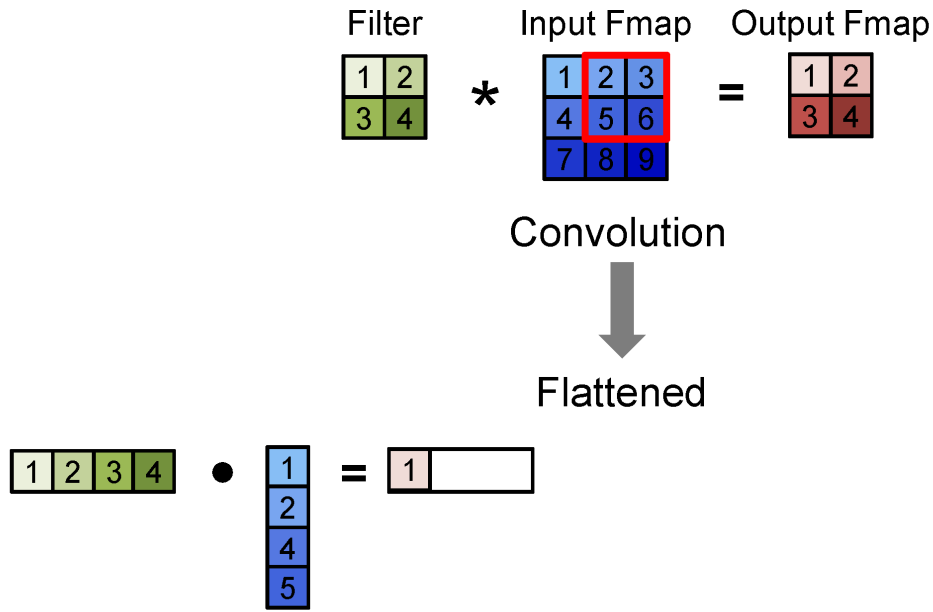
Convolution (CONV) Layer



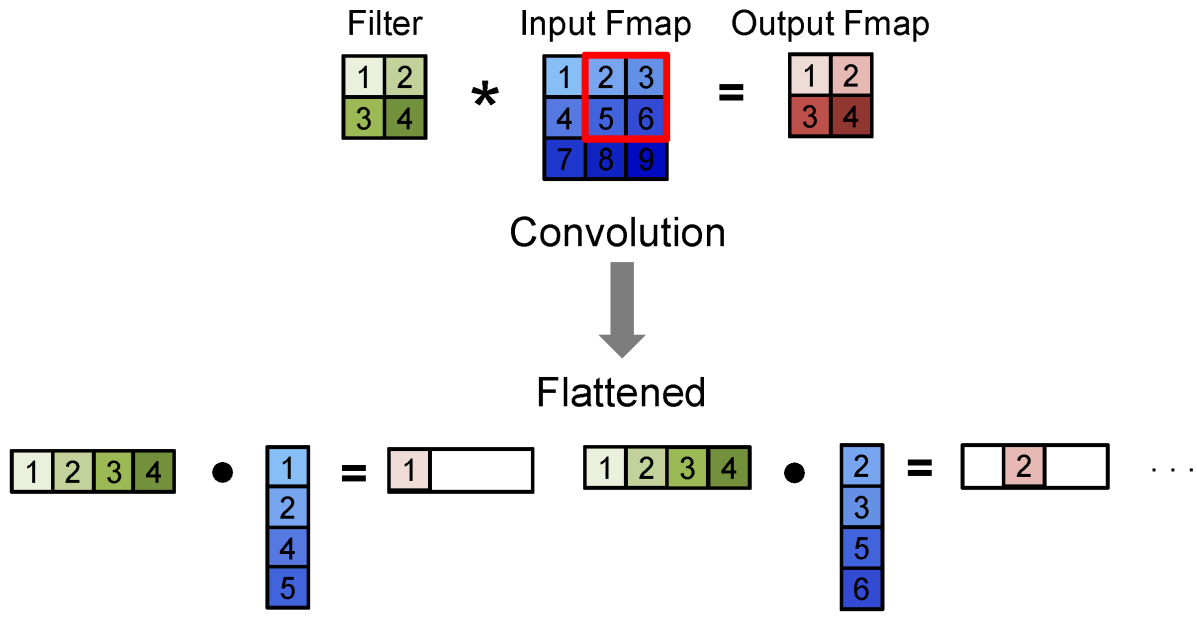
Convolution (CONV) Layer



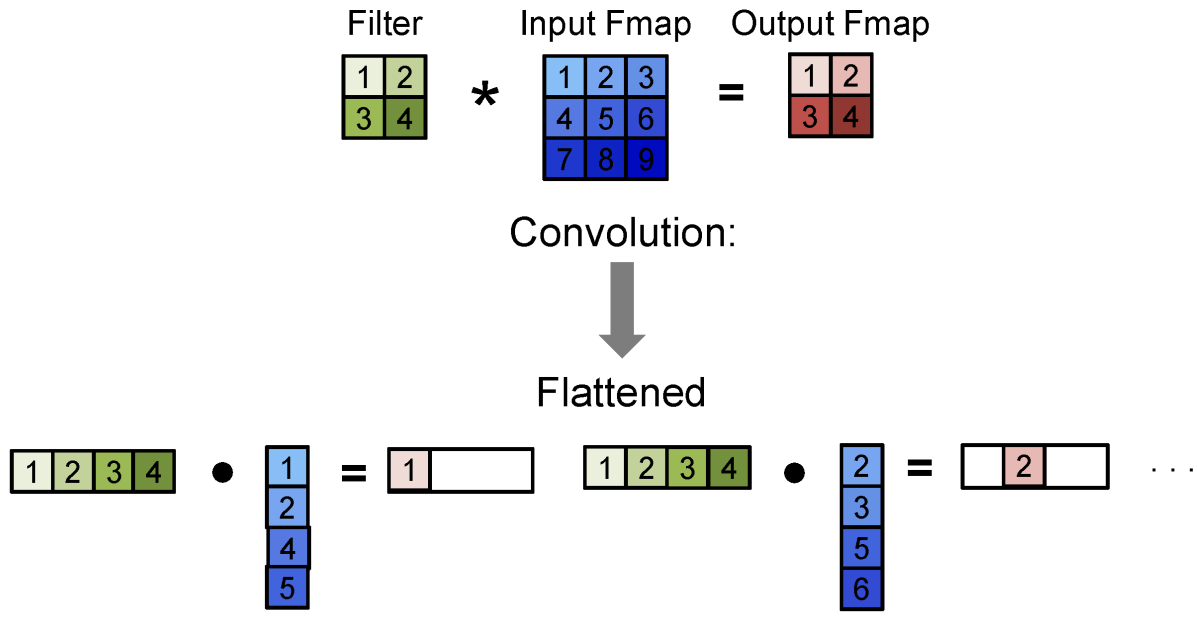
Convolution (CONV) Layer



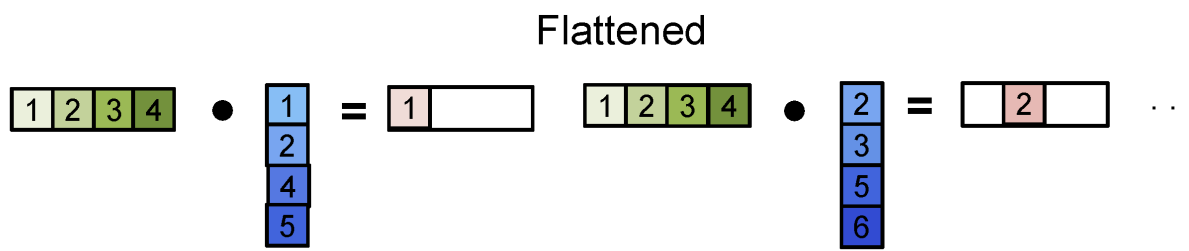
Convolution (CONV) Layer



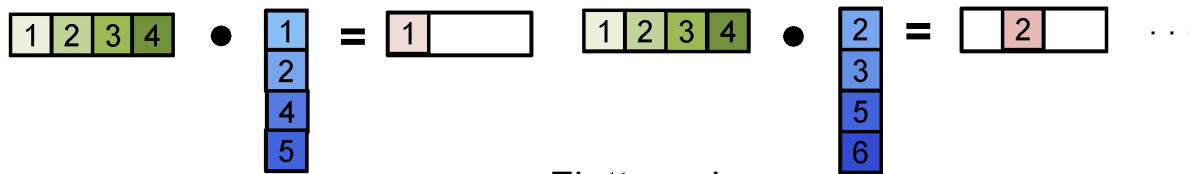
Convolution (CONV) Layer



Convolution (CONV) Layer

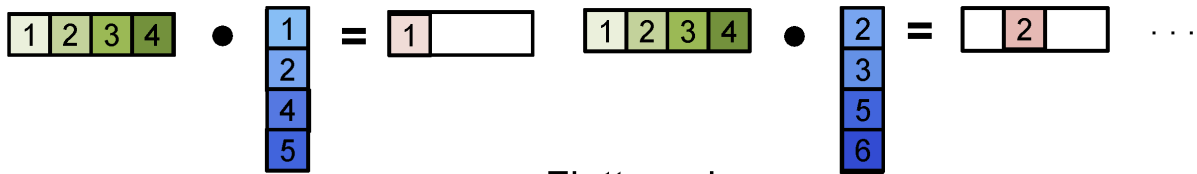


Convolution (CONV) Layer



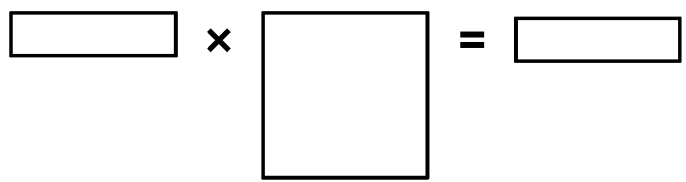
Flattened

Convolution (CONV) Layer

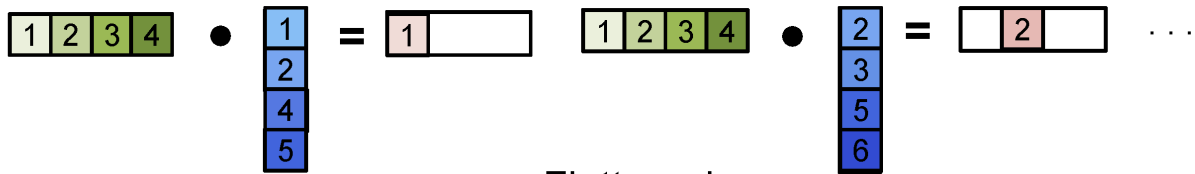


Flattened

Matrix Multiply (by Toeplitz Matrix)

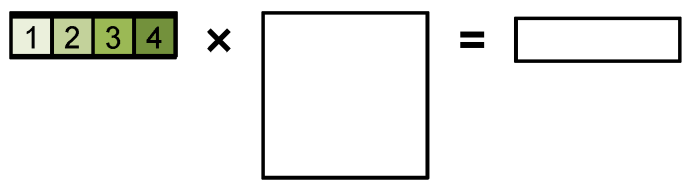


Convolution (CONV) Layer

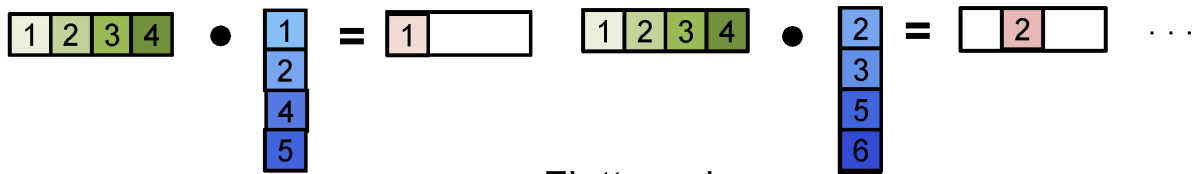


Flattened

Matrix Multiply (by Toeplitz Matrix)

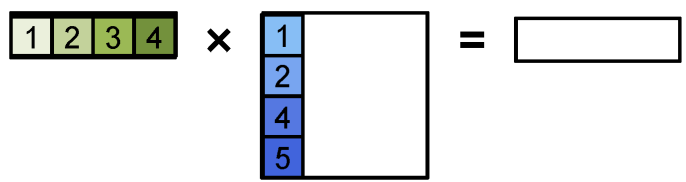


Convolution (CONV) Layer

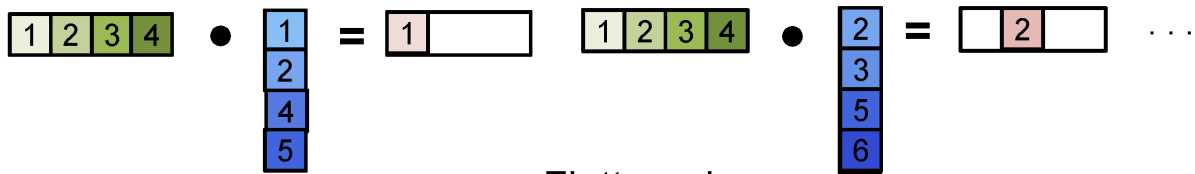


Flattened

Matrix Multiply (by Toeplitz Matrix)

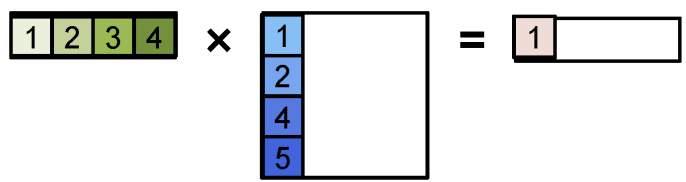


Convolution (CONV) Layer

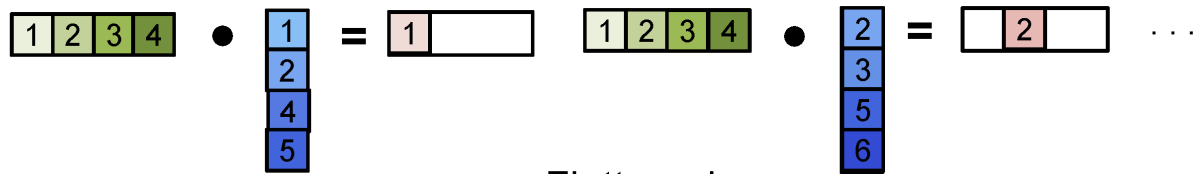


Flattened

Matrix Multiply (by Toeplitz Matrix)

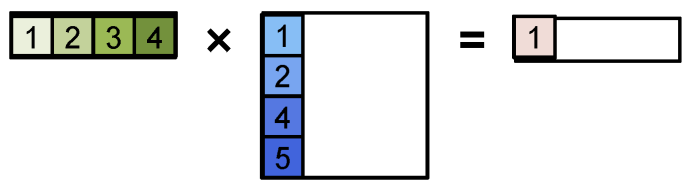


Convolution (CONV) Layer

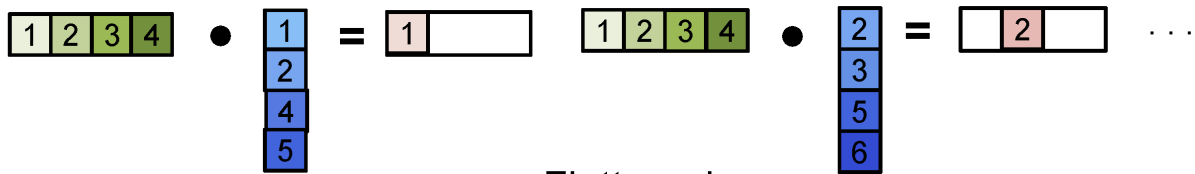


Flattened

Matrix Multiply (by Toeplitz Matrix)

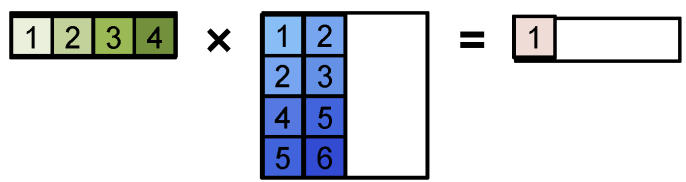


Convolution (CONV) Layer

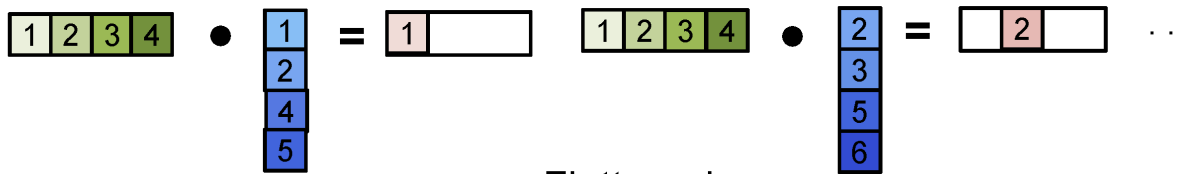


Flattened

Matrix Multiply (by Toeplitz Matrix)

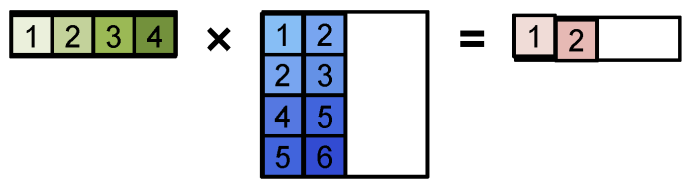


Convolution (CONV) Layer

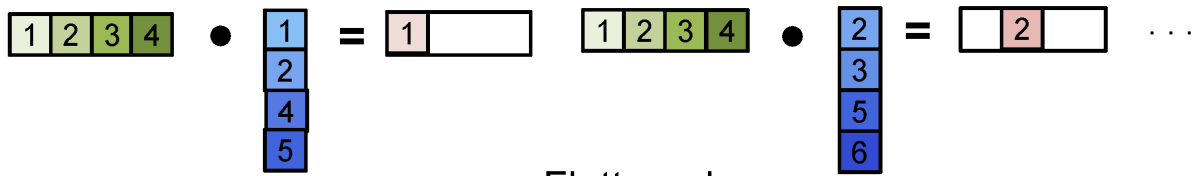


Flattened

Matrix Multiply (by Toeplitz Matrix)

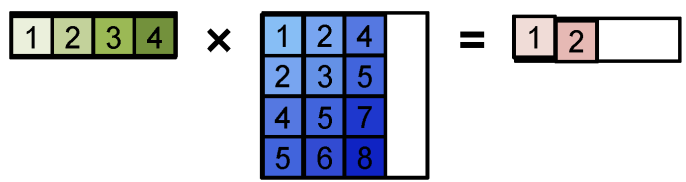


Convolution (CONV) Layer

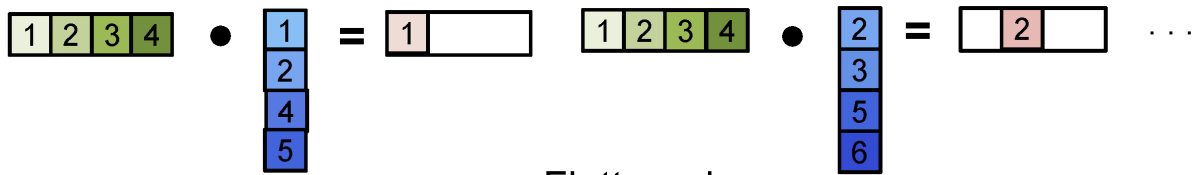


Flattened

Matrix Multiply (by Toeplitz Matrix)

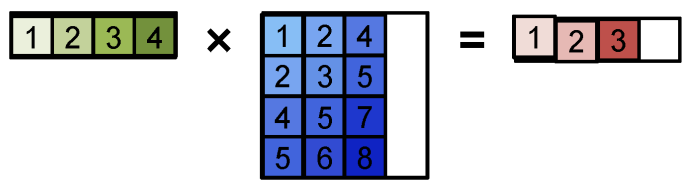


Convolution (CONV) Layer

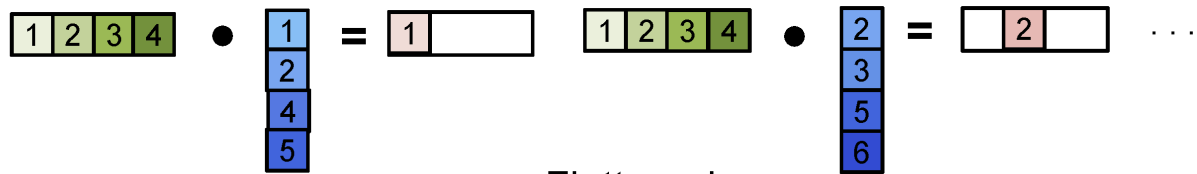


Flattened

Matrix Multiply (by Toeplitz Matrix)

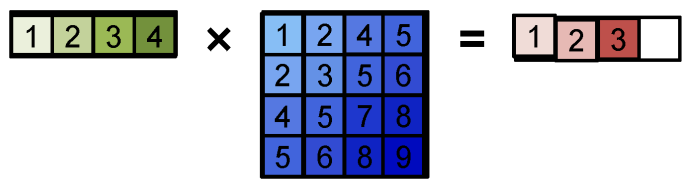


Convolution (CONV) Layer

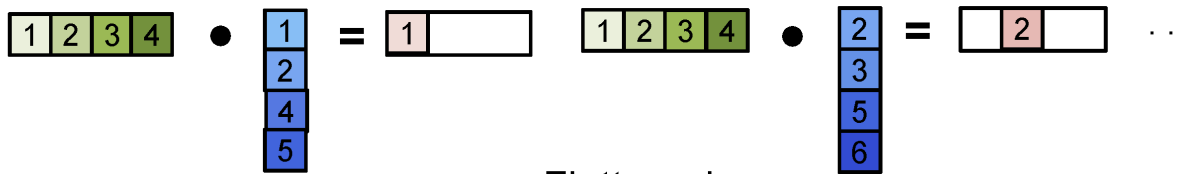


Flattened

Matrix Multiply (by Toeplitz Matrix)

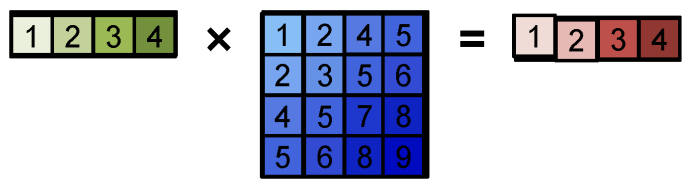


Convolution (CONV) Layer

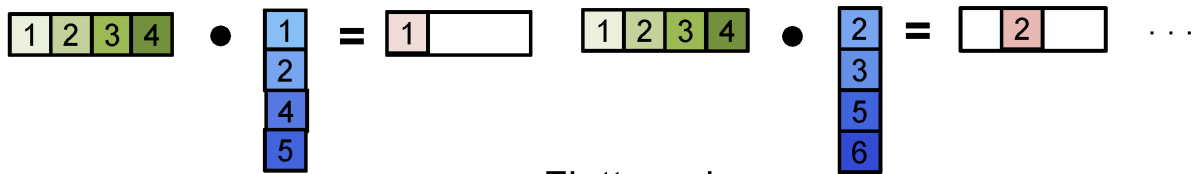


Flattened

Matrix Multiply (by Toeplitz Matrix)

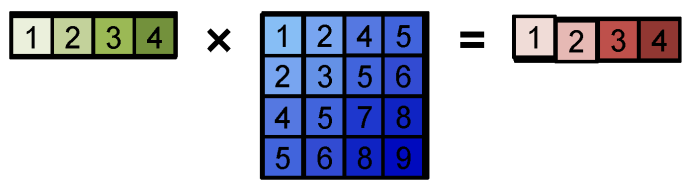


Convolution (CONV) Layer

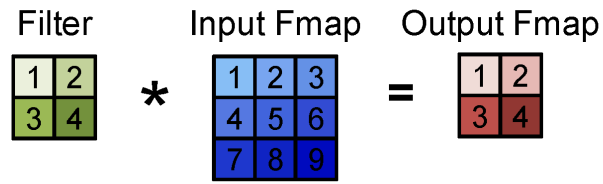


Flattened

Matrix Multiply (by Toeplitz Matrix)

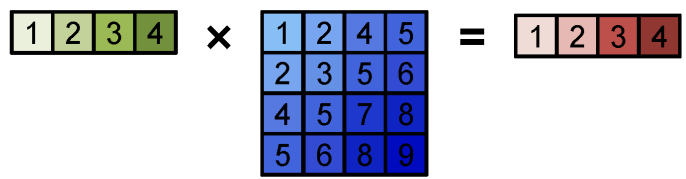


Convolution (CONV) Layer



Convolution:

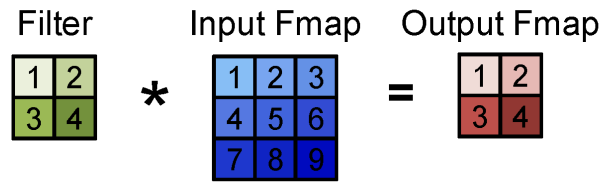
Matrix Multiply (by Toeplitz Matrix)



Convert to matrix multiply using the **Toeplitz Matrix**

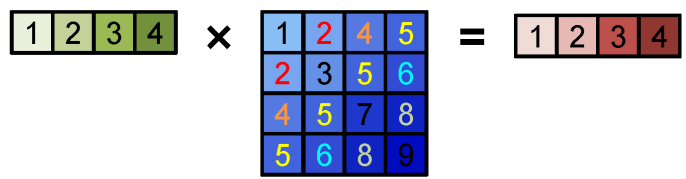
(aka 'im2col')

Convolution (CONV) Layer



Convolution:

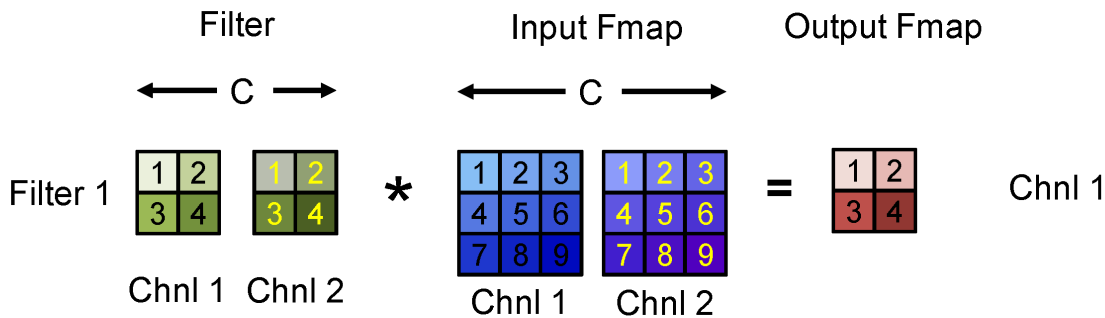
Matrix Multiply (by Toeplitz Matrix)



Data is repeated

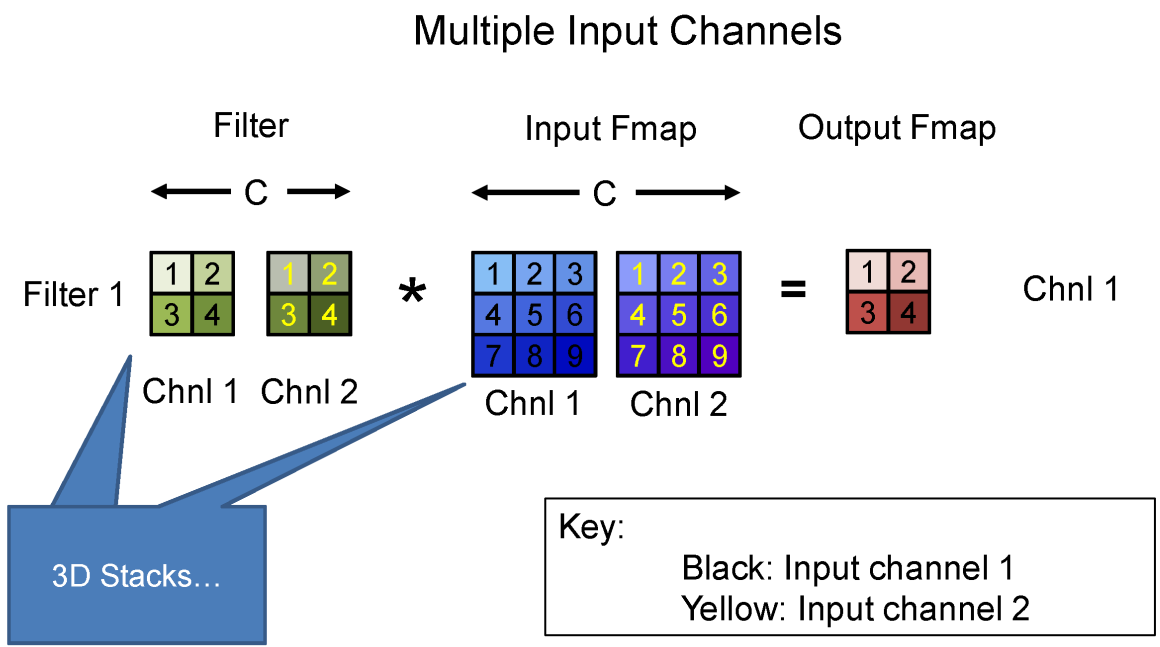
Convolution (CONV) Layer

Multiple Input Channels



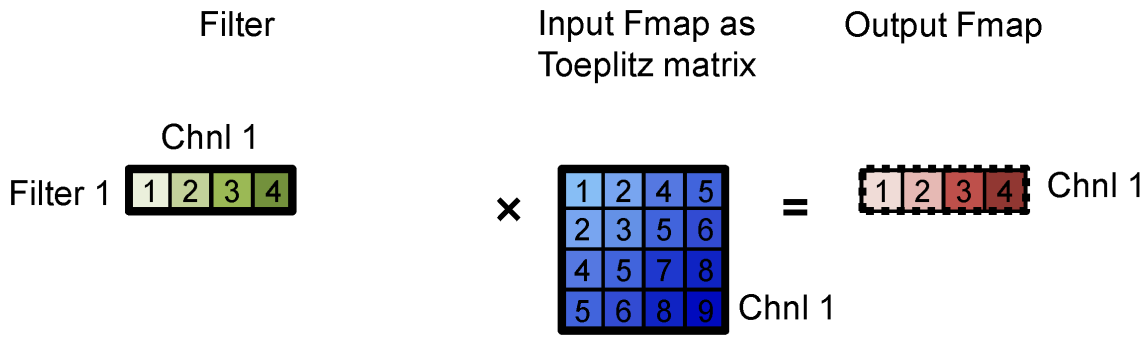
Key:
Black: Input channel 1
Yellow: Input channel 2

Convolution (CONV) Layer

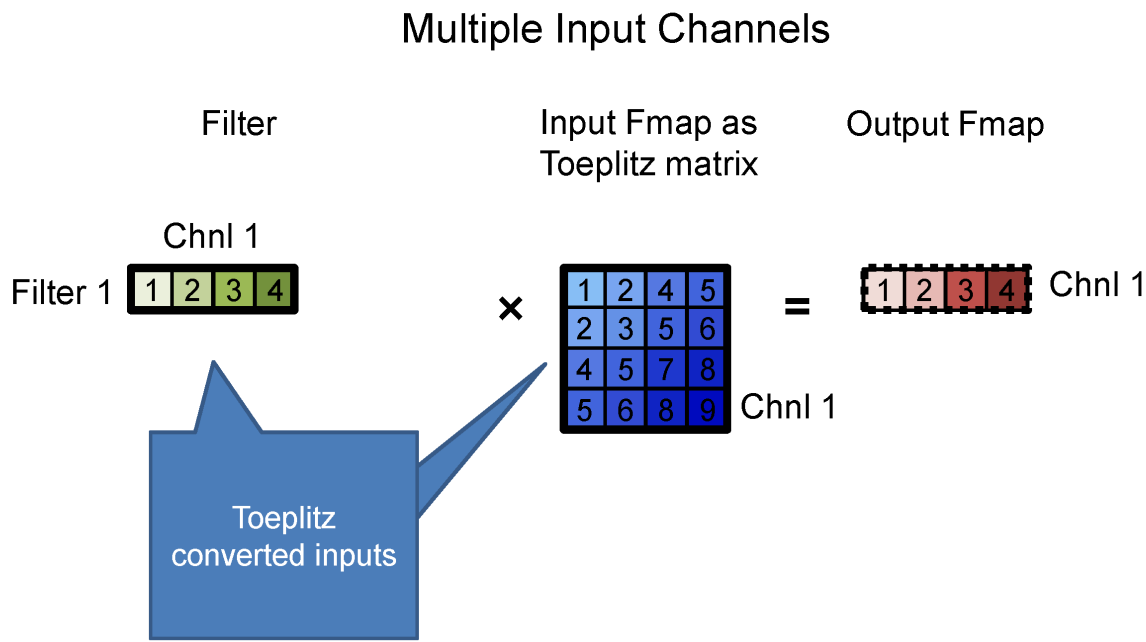


Convolution (CONV) Layer

Multiple Input Channels

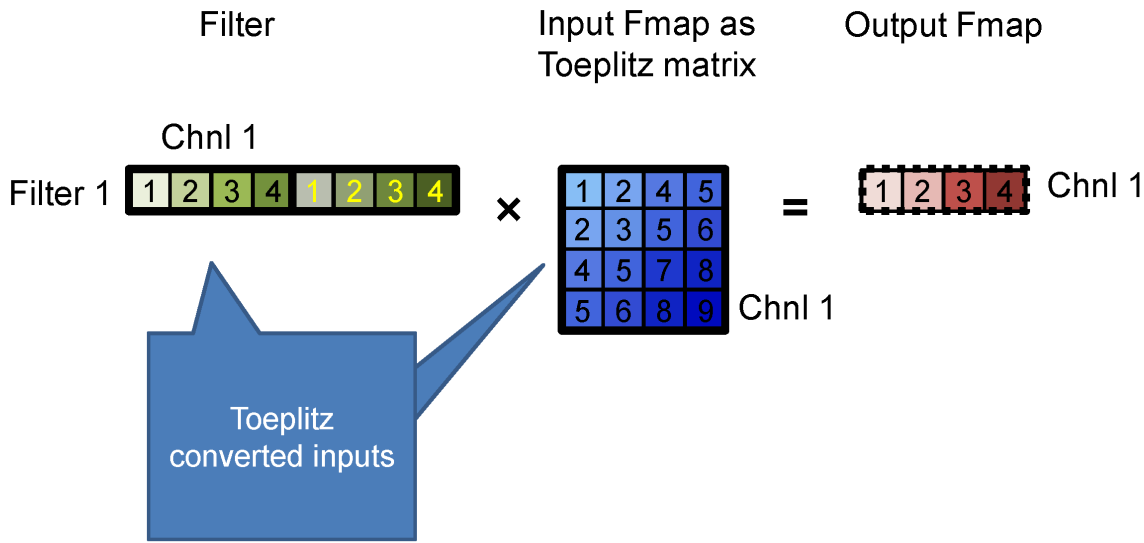


Convolution (CONV) Layer



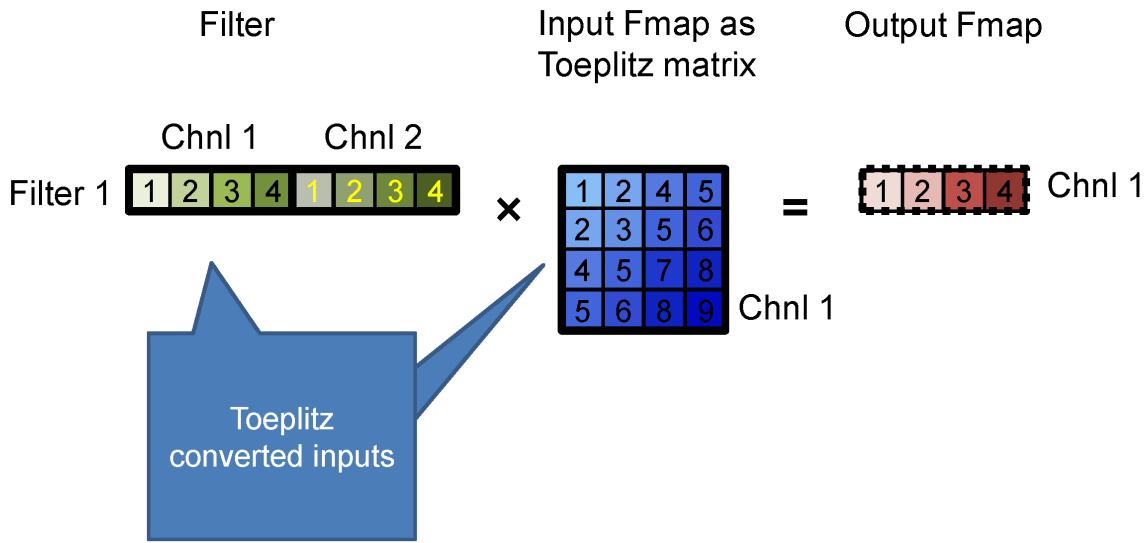
Convolution (CONV) Layer

Multiple Input Channels



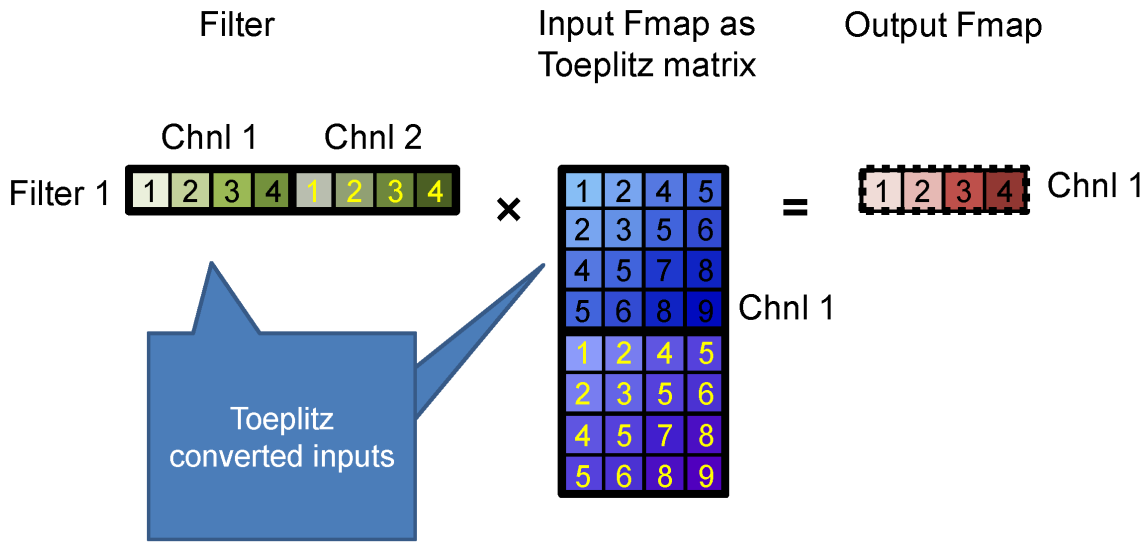
Convolution (CONV) Layer

Multiple Input Channels



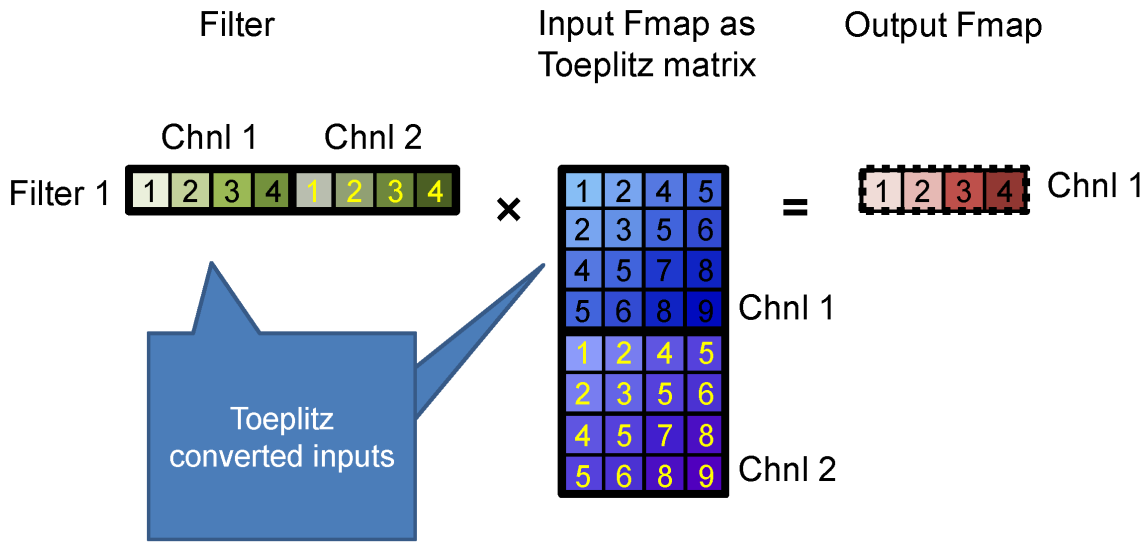
Convolution (CONV) Layer

Multiple Input Channels



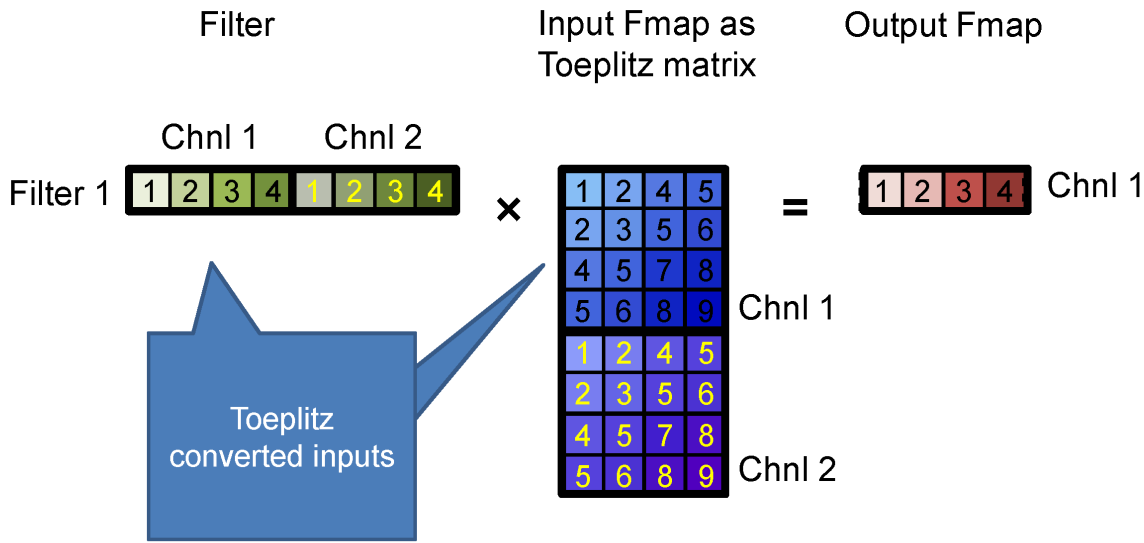
Convolution (CONV) Layer

Multiple Input Channels



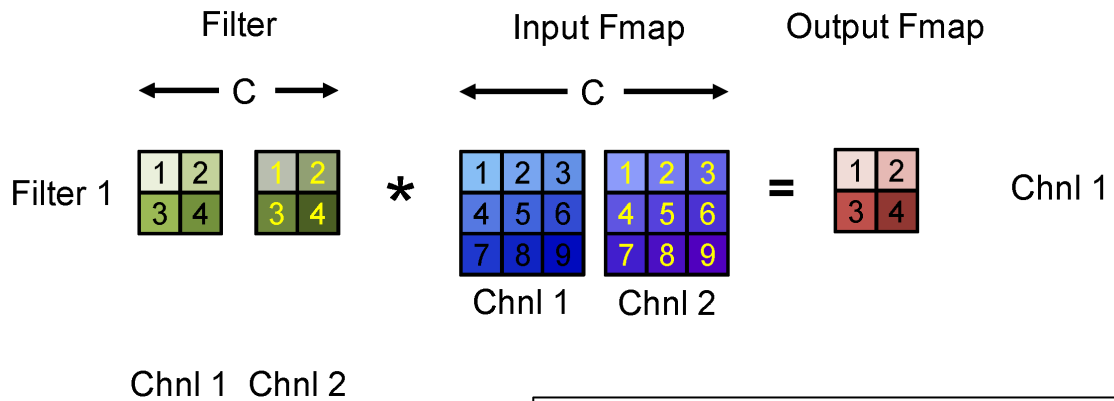
Convolution (CONV) Layer

Multiple Input Channels



Convolution (CONV) Layer

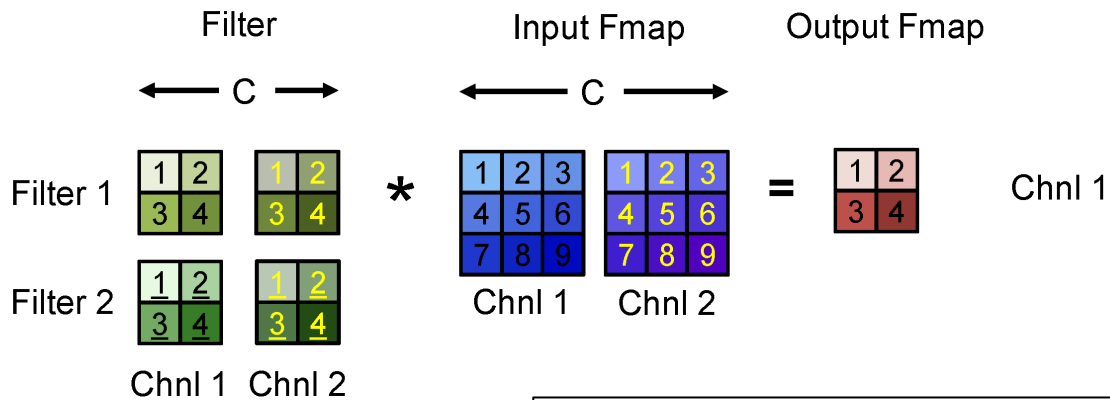
Multiple Input Channels and Output Channels



Key:
- Black: Input channel 1
- Yellow: Input channel 2
- Underlined: Output channel 2

Convolution (CONV) Layer

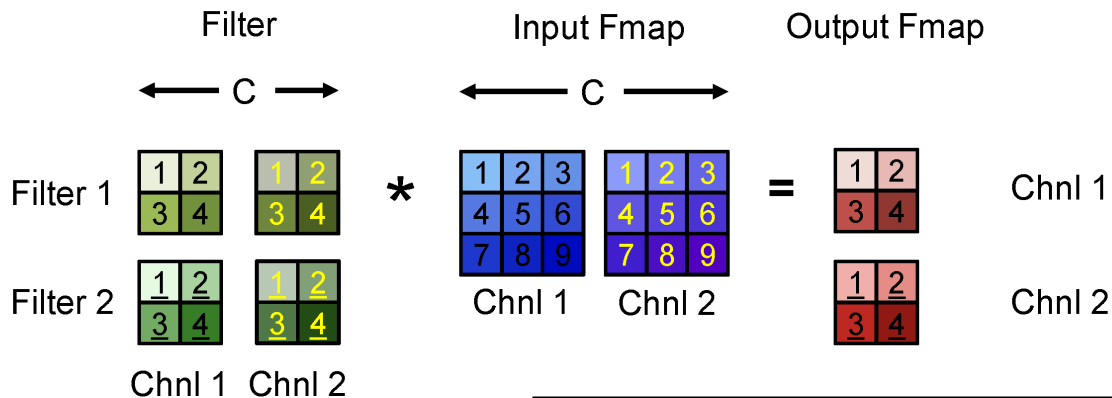
Multiple Input Channels and Output Channels



Key:
- Black: Input channel 1
- Yellow: Input channel 2
- Underlined: Output channel 2

Convolution (CONV) Layer

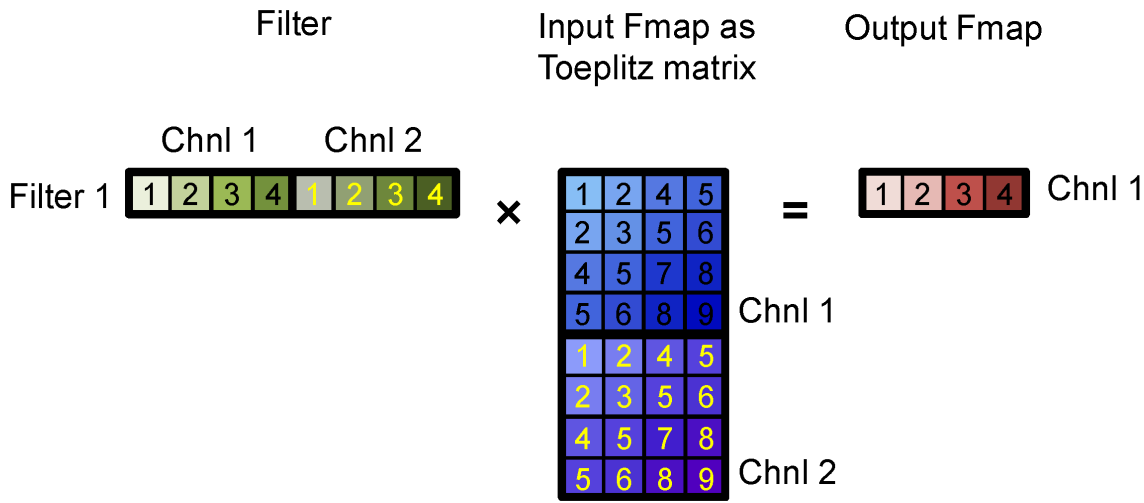
Multiple Input Channels and Output Channels



Key:
- Black: Input channel 1
- Yellow: Input channel 2
- Underlined: Output channel 2

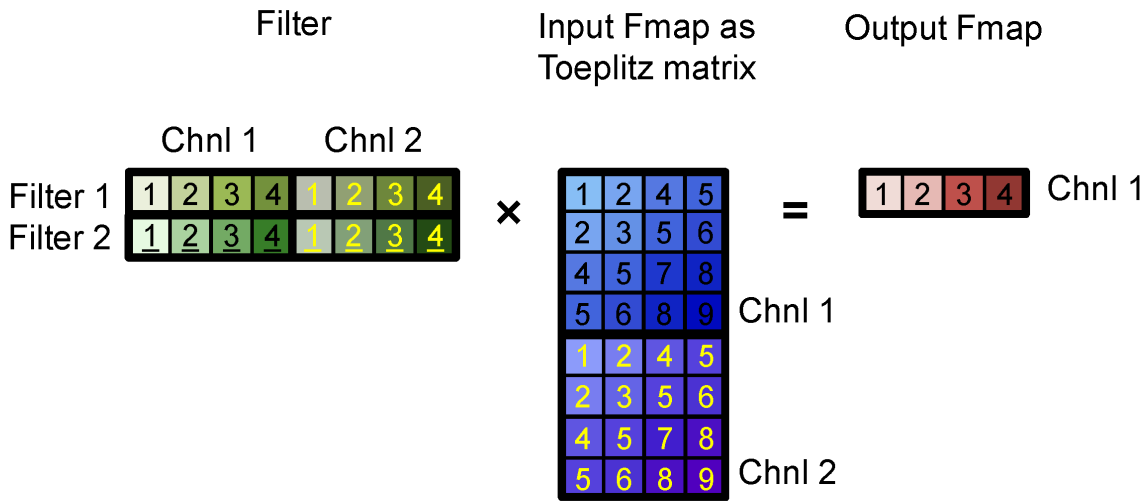
Convolution (CONV) Layer

Multiple Input Channels and Output Channels



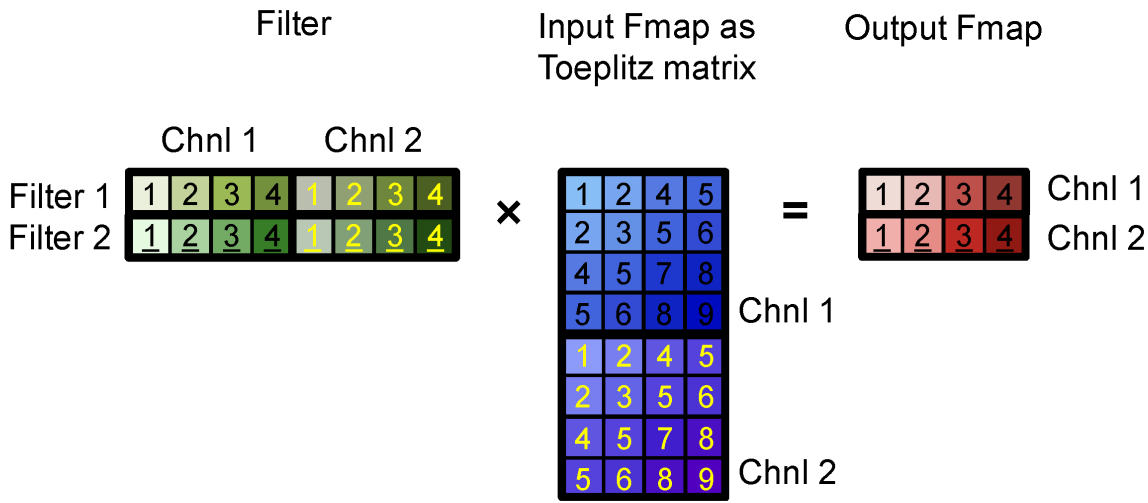
Convolution (CONV) Layer

Multiple Input Channels and Output Channels



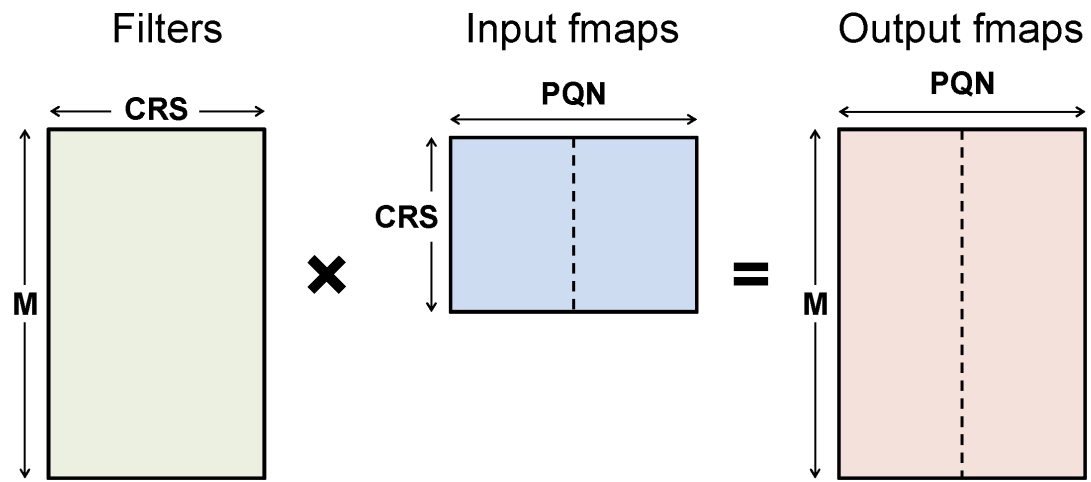
Convolution (CONV) Layer

Multiple Input Channels and Output Channels



Convolution (CONV) Layer → Matrix Multiplication

Dimensions of matrices for matrix multiply in convolution layers with batch size N



where P=H-R+1 and Q=W-S+1

N=2 in example

1-D Toeplitz Convolution Einsum

1-D Toeplitz Convolution Einsum

$$O_{n,m,p,q} = I_{n,c,U\times p+r,U\times q+s} \times F_{m,c,r,s}$$

1-D Toeplitz Convolution Einsum

$$O_{n,m,p,q} = I_{n,c,U\times p+r,U\times q+s} \times F_{m,c,r,s}$$

Simplify to 1-D with N=1, C=1, M=1, U=1

1-D Toeplitz Convolution Einsum

$$O_{n,m,p,q} = I_{n,c,U\times p+r,U\times q+s} \times F_{m,c,r,s}$$

Simplify to 1-D with N=1, C=1, M=1, U=1

$$O_q = I_{q+s} \times F_s$$

1-D Toeplitz Convolution Einsum

$$O_{n,m,p,q} = I_{n,c,U\times p+r,U\times q+s} \times F_{m,c,r,s}$$

Simplify to 1-D with N=1, C=1, M=1, U=1

$$O_q = I_{q+s} \times F_s$$

Convolution can be represented in **two** steps

1-D Toeplitz Convolution Einsum

$$O_{n,m,p,q} = I_{n,c,U\times p+r,U\times q+s} \times F_{m,c,r,s}$$

Simplify to 1-D with N=1, C=1, M=1, U=1

$$O_q = I_{q+s} \times F_s$$

Convolution can be represented in **two** steps

1-D Toeplitz Convolution Einsum

$$O_{n,m,p,q} = I_{n,c,U\times p+r,U\times q+s} \times F_{m,c,r,s}$$

Simplify to 1-D with N=1, C=1, M=1, U=1

$$O_q = I_{q+s} \times F_s$$

Convolution can be represented in **two** steps

$$T_{q,s} = I_{q+s}$$

1-D Toeplitz Convolution Einsum

$$O_{n,m,p,q} = I_{n,c,U\times p+r,U\times q+s} \times F_{m,c,r,s}$$

Simplify to 1-D with N=1, C=1, M=1, U=1

$$O_q = I_{q+s} \times F_s$$

Convolution can be represented in **two** steps

$$T_{q,s} = I_{q+s}$$

$$O_q = T_{q,s} \times F_s$$

1-D Toeplitz Convolution Einsum

$$O_{n,m,p,q} = I_{n,c,U\times p+r,U\times q+s} \times F_{m,c,r,s}$$

Simplify to 1-D with N=1, C=1, M=1, U=1

$$O_q = I_{q+s} \times F_s$$

Convolution can be represented in **two** steps

Step (1) $$T_{q,s} = I_{q+s}$$

$$O_q = T_{q,s} \times F_s$$

1-D Toeplitz Convolution Einsum

$$O_{n,m,p,q} = I_{n,c,U\times p+r,U\times q+s} \times F_{m,c,r,s}$$

Simplify to 1-D with N=1, C=1, M=1, U=1

$$O_q = I_{q+s} \times F_s$$

Convolution can be represented in **two** steps

Step (1) $$T_{q,s} = I_{q+s}$$

$$O_q = T_{q,s} \times F_s$$

1-D Toeplitz Convolution Einsum

$$O_{n,m,p,q} = I_{n,c,U\times p+r,U\times q+s} \times F_{m,c,r,s}$$

Simplify to 1-D with N=1, C=1, M=1, U=1

$$O_q = I_{q+s} \times F_s$$

Convolution can be represented in **two** steps

Step (1) $T_{q,s} = I_{q+s}$

Step (2) $O_q = T_{q,s} \times F_s$

1-D Toeplitz Convolution Einsum

$$O_{n,m,p,q} = I_{n,c,U\times p+r,U\times q+s} \times F_{m,c,r,s}$$

Simplify to 1-D with N=1, C=1, M=1, U=1

$$O_q = I_{q+s} \times F_s$$

Convolution can be represented in **two** steps

Step (1) $T_{q,s} = I_{q+s}$ Toeplitz conversion

Step (2) $O_q = T_{q,s} \times F_s$

1-D Toeplitz Convolution Einsum

$$O_{n,m,p,q} = I_{n,c,U\times p+r,U\times q+s} \times F_{m,c,r,s}$$

Simplify to 1-D with N=1, C=1, M=1, U=1

$$O_q = I_{q+s} \times F_s$$

Convolution can be represented in **two** steps

Step (1) $T_{q,s} = I_{q+s}$ Toeplitz conversion

Step (2) $O_q = T_{q,s} \times F_s$ Matrix multiplication

1-D Toeplitz Convolution Einsum

$$O_{n,m,p,q} = I_{n,c,U\times p+r,U\times q+s} \times F_{m,c,r,s}$$

Simplify to 1-D with N=1, C=1, M=1, U=1

$$O_q = I_{q+s} \times F_s$$ 1-D convolution

Convolution can be represented in **two** steps

Step (1) $T_{q,s} = I_{q+s}$ Toeplitz conversion

Step (2) $O_q = T_{q,s} \times F_s$ Matrix multiplication

1-D Toeplitz Convolution Einsum

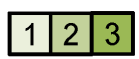
Review: Illustrating **two** steps for 1-D convolution

Filter (F_s) Input Fmap (I_{q+s}) Output Fmap (O_q)

1-D Toeplitz Convolution Einsum

Review: Illustrating **two** steps for 1-D convolution

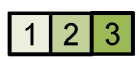
Filter (F_s) Input Fmap (I_{q+s}) Output Fmap (O_q)



1-D Toeplitz Convolution Einsum

Review: Illustrating **two** steps for 1-D convolution

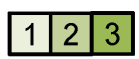
Filter (F_s) Input Fmap (I_{q+s}) Output Fmap (O_q)



1-D Toeplitz Convolution Einsum

Review: Illustrating **two** steps for 1-D convolution

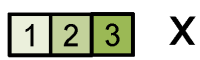
Filter (F_s) Input Fmap (I_{q+s}) Output Fmap (O_q)



1-D Toeplitz Convolution Einsum

Review: Illustrating **two** steps for 1-D convolution

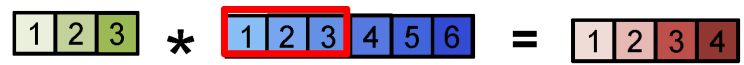
Filter (F_s) Input Fmap (I_{q+s}) Output Fmap (O_q)

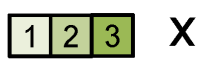


1-D Toeplitz Convolution Einsum

Review: Illustrating **two** steps for 1-D convolution

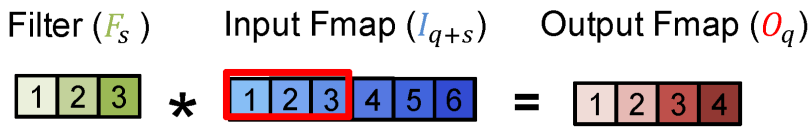
Filter (F_s) Input Fmap (I_{q+s}) Output Fmap (O_q)

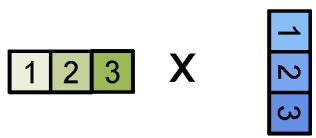




1-D Toeplitz Convolution Einsum

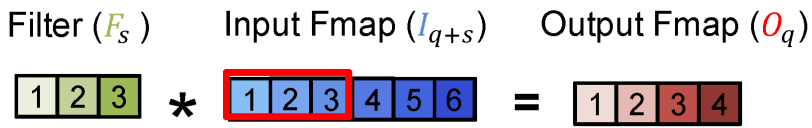
Review: Illustrating **two** steps for 1-D convolution

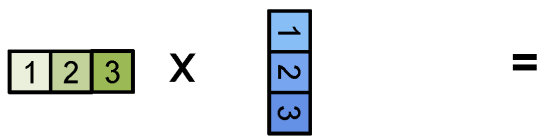




1-D Toeplitz Convolution Einsum

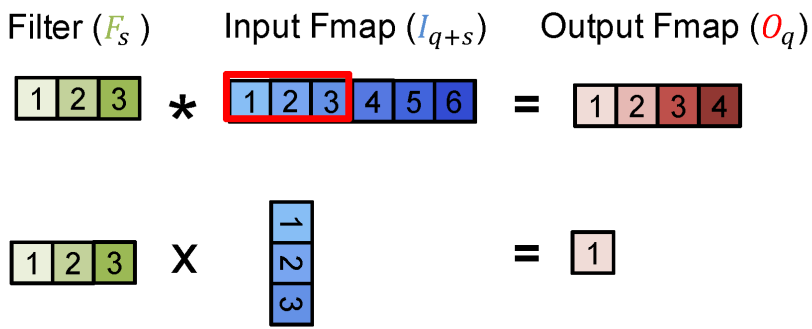
Review: Illustrating **two** steps for 1-D convolution





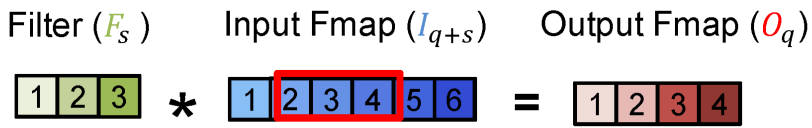
1-D Toeplitz Convolution Einsum

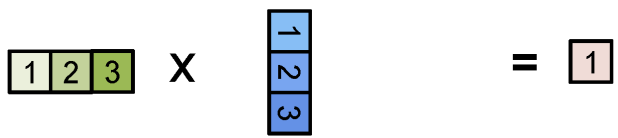
Review: Illustrating **two** steps for 1-D convolution



1-D Toeplitz Convolution Einsum

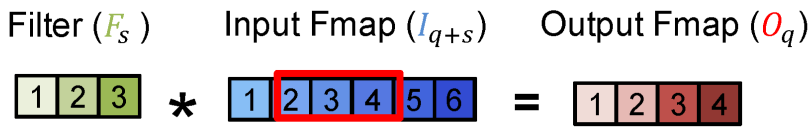
Review: Illustrating **two** steps for 1-D convolution

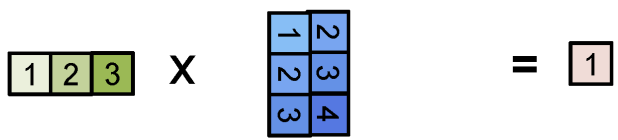




1-D Toeplitz Convolution Einsum

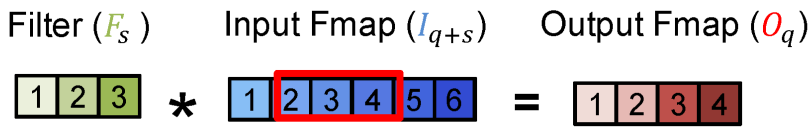
Review: Illustrating **two** steps for 1-D convolution

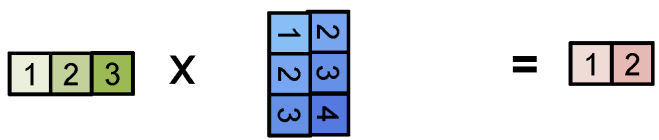




1-D Toeplitz Convolution Einsum

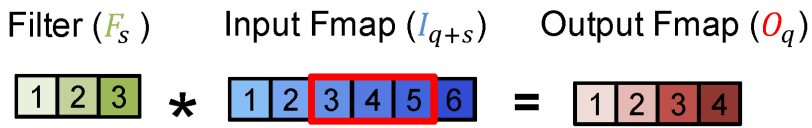
Review: Illustrating **two** steps for 1-D convolution

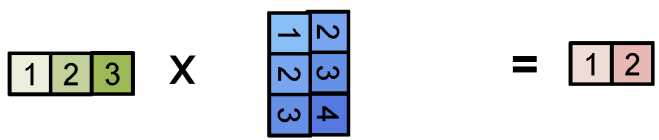




1-D Toeplitz Convolution Einsum

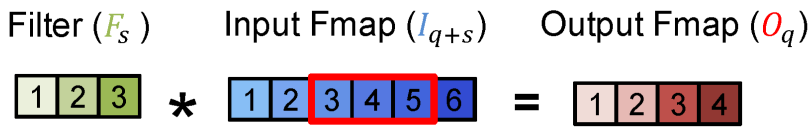
Review: Illustrating **two** steps for 1-D convolution

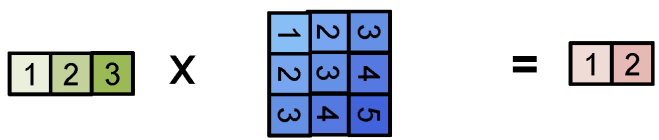




1-D Toeplitz Convolution Einsum

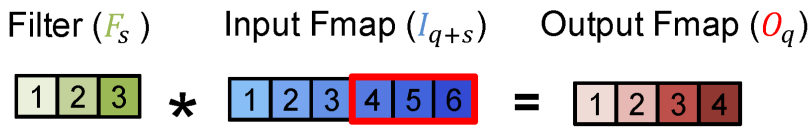
Review: Illustrating **two** steps for 1-D convolution

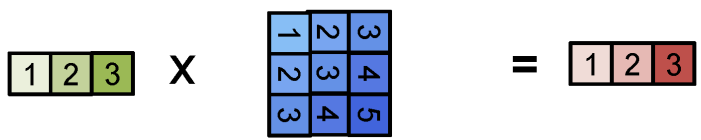




1-D Toeplitz Convolution Einsum

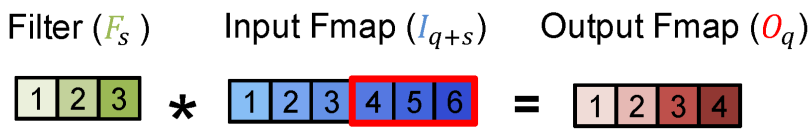
Review: Illustrating **two** steps for 1-D convolution

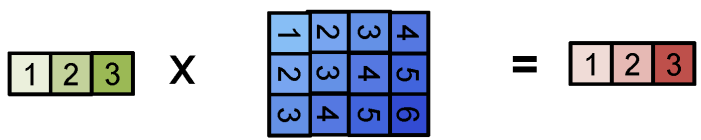




1-D Toeplitz Convolution Einsum

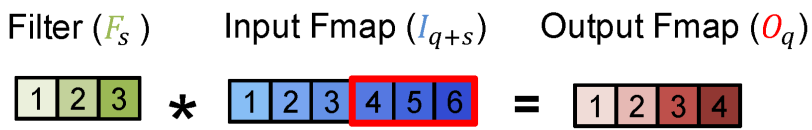
Review: Illustrating **two** steps for 1-D convolution

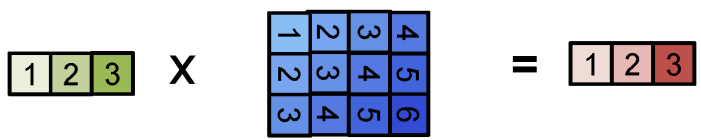




1-D Toeplitz Convolution Einsum

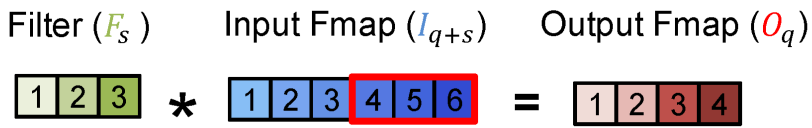
Review: Illustrating **two** steps for 1-D convolution

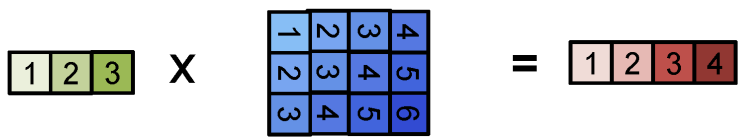




1-D Toeplitz Convolution Einsum

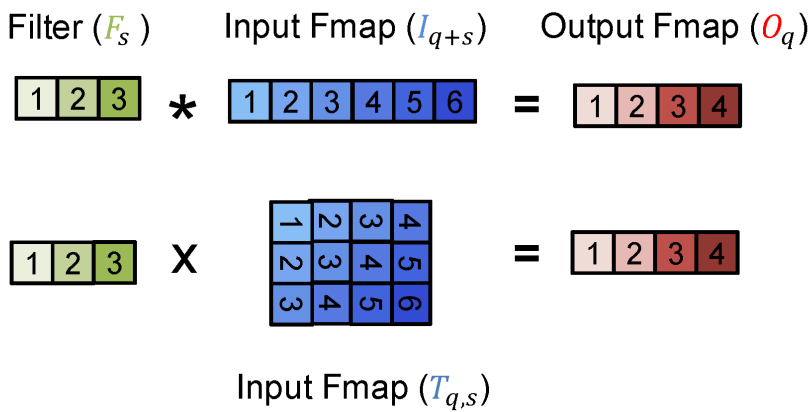
Review: Illustrating **two** steps for 1-D convolution





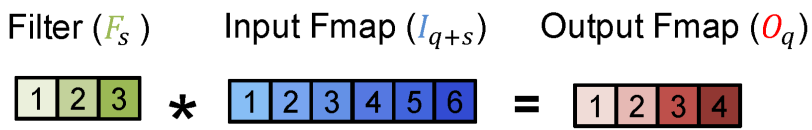
1-D Toeplitz Convolution Einsum

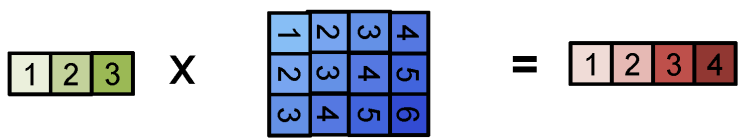
Review: Illustrating **two** steps for 1-D convolution



1-D Toeplitz Convolution Einsum

Review: Illustrating **two** steps for 1-D convolution

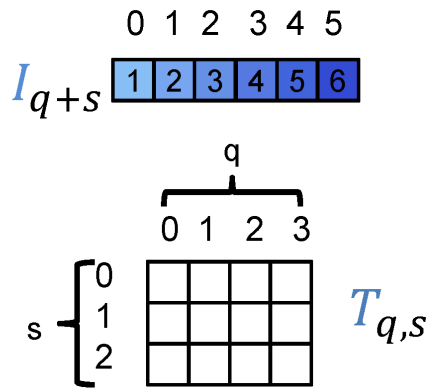


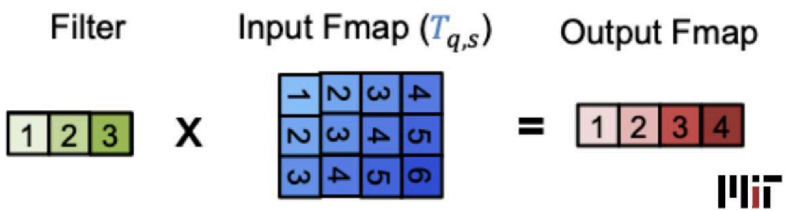


Let's show how the Einsum performs this process…

1-D Toeplitz Convolution Einsum

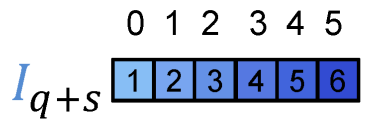
Example of execution of **step (1)** Topelitz conversion $T_{q,s} = I_{q+s}$



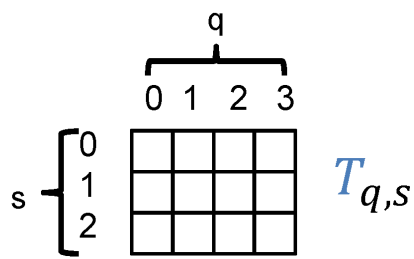


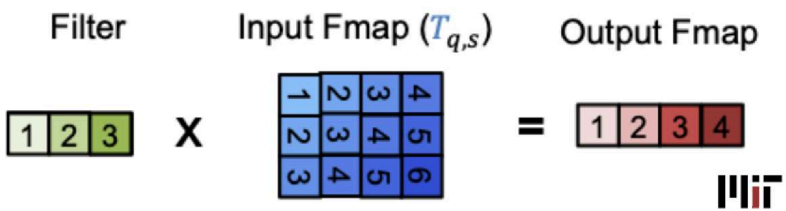
1-D Toeplitz Convolution Einsum

Example of execution of **step (1)** Topelitz conversion $T_{q,s} = I_{q+s}$



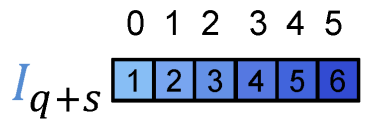
q	s	q+s
0	0	0



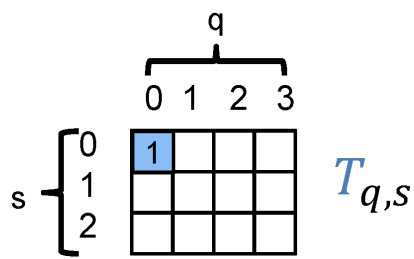


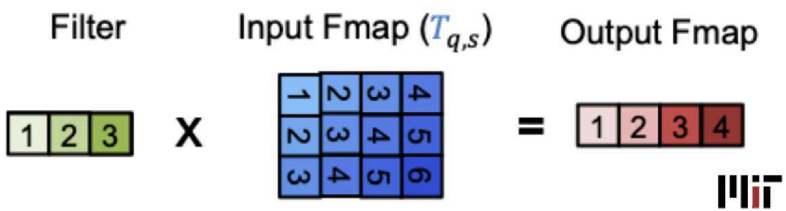
1-D Toeplitz Convolution Einsum

Example of execution of **step (1)** Topelitz conversion $T_{q,s} = I_{q+s}$



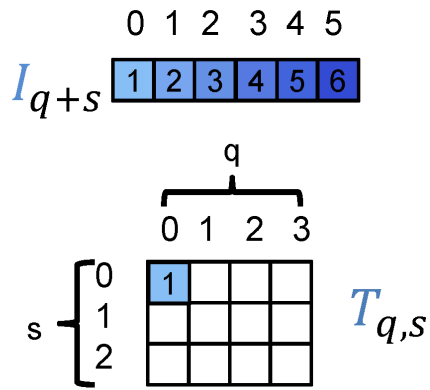
q	s	q+s
0	0	0



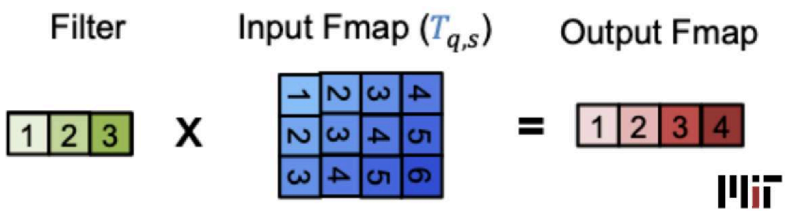


1-D Toeplitz Convolution Einsum

Example of execution of **step (1)** Topelitz conversion $T_{q,s} = I_{q+s}$

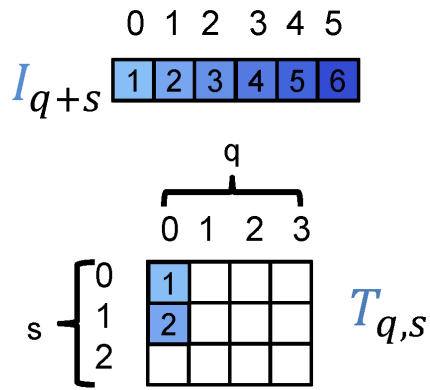


q	s	q+s
0	0	0
0	1	1

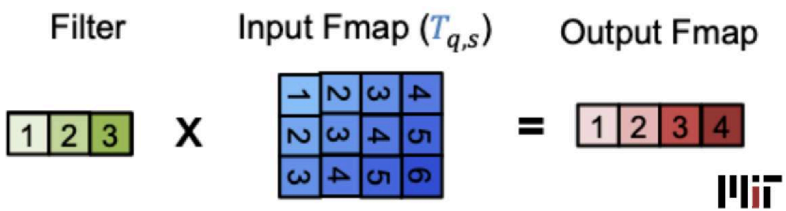


1-D Toeplitz Convolution Einsum

Example of execution of **step (1)** Topelitz conversion $T_{q,s} = I_{q+s}$

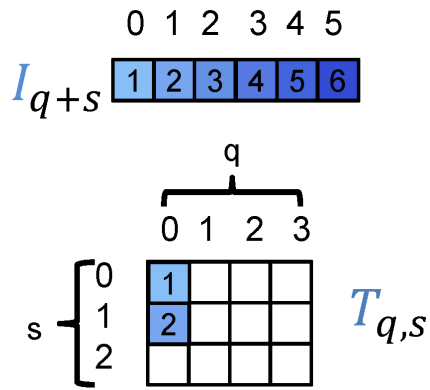


q	s	q+s
0	0	0
0	1	1

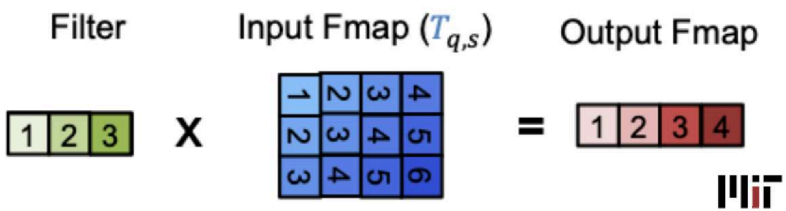


1-D Toeplitz Convolution Einsum

Example of execution of **step (1)** Topelitz conversion $T_{q,s} = I_{q+s}$

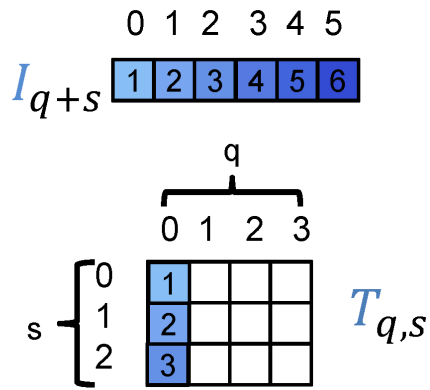


q	s	q+s
0	0	0
0	1	1
0	2	2

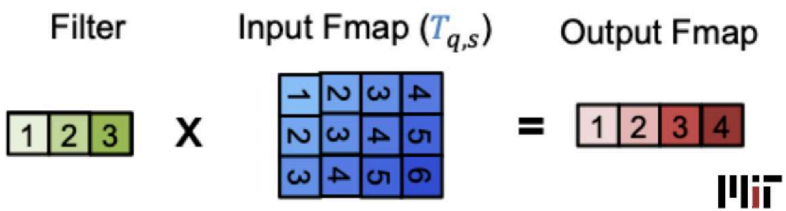


1-D Toeplitz Convolution Einsum

Example of execution of **step (1)** Topelitz conversion $T_{q,s} = I_{q+s}$

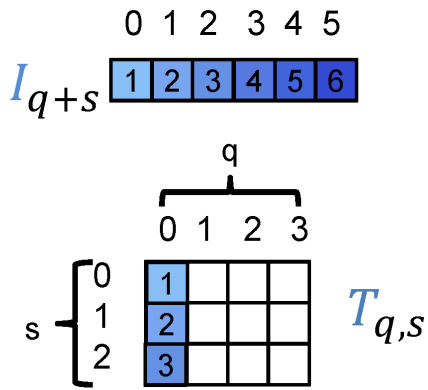


q	s	q+s
0	0	0
0	1	1
0	2	2

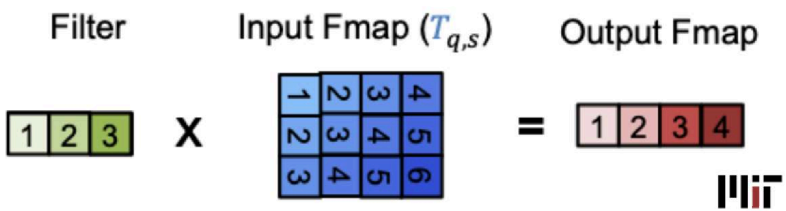


1-D Toeplitz Convolution Einsum

Example of execution of **step (1)** Topelitz conversion $T_{q,s} = I_{q+s}$

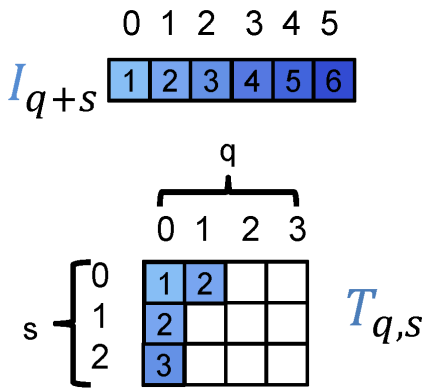


q	s	q+s
0	0	0
0	1	1
0	2	2
1	0	1

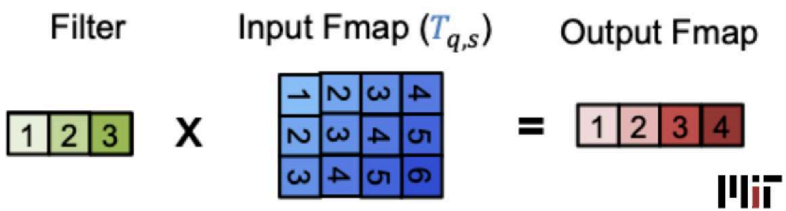


1-D Toeplitz Convolution Einsum

Example of execution of **step (1)** Topelitz conversion $T_{q,s} = I_{q+s}$

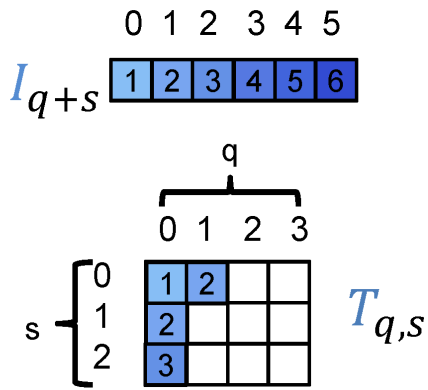


q	s	q+s
0	0	0
0	1	1
0	2	2
1	0	1

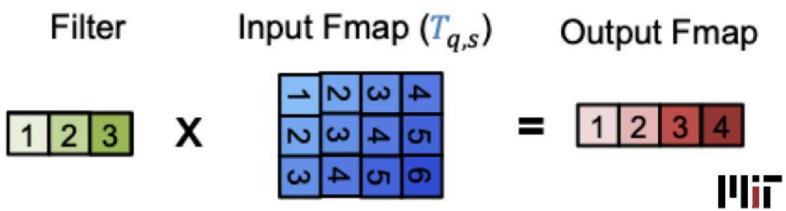


1-D Toeplitz Convolution Einsum

Example of execution of **step (1)** Topelitz conversion $T_{q,s} = I_{q+s}$

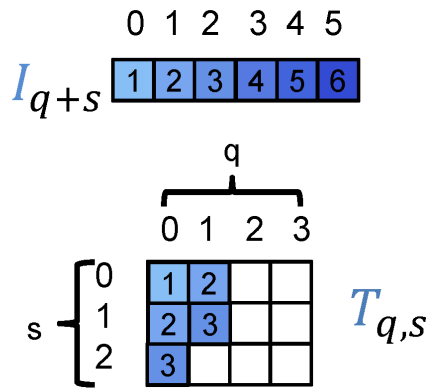


q	s	q+s
0	0	0
0	1	1
0	2	2
1	0	1
1	1	2

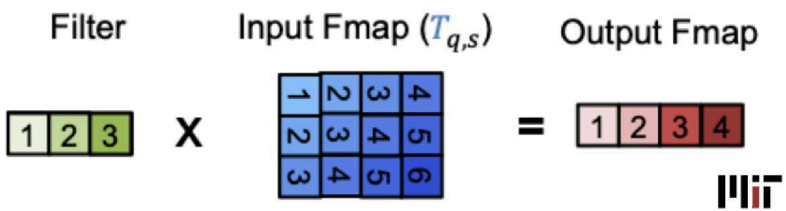


1-D Toeplitz Convolution Einsum

Example of execution of **step (1)** Topelitz conversion $T_{q,s} = I_{q+s}$

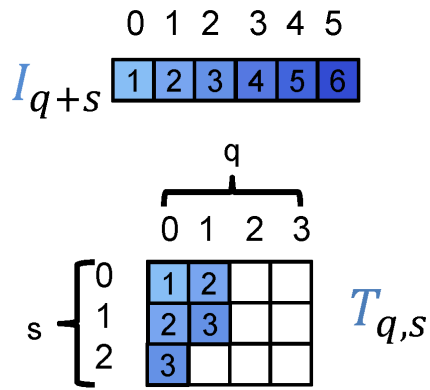


q	s	q+s
0	0	0
0	1	1
0	2	2
1	0	1
1	1	2

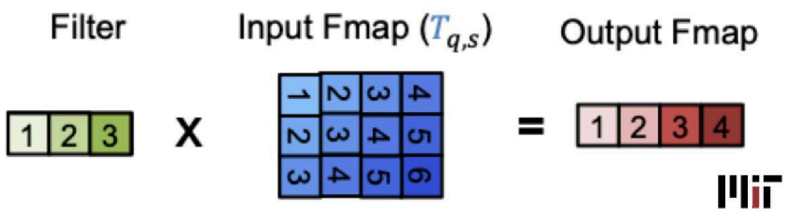


1-D Toeplitz Convolution Einsum

Example of execution of **step (1)** Topelitz conversion $T_{q,s} = I_{q+s}$

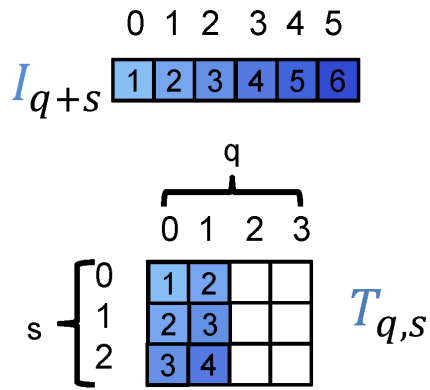


q	s	q+s
0	0	0
0	1	1
0	2	2
1	0	1
1	1	2
1	2	3

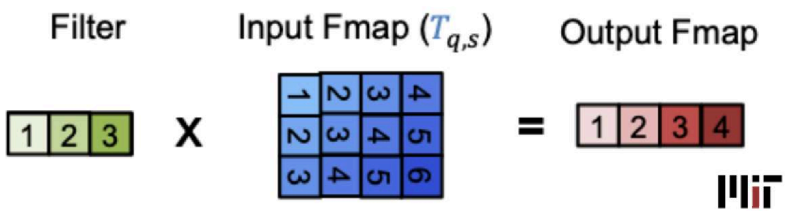


1-D Toeplitz Convolution Einsum

Example of execution of **step (1)** Topelitz conversion $T_{q,s} = I_{q+s}$

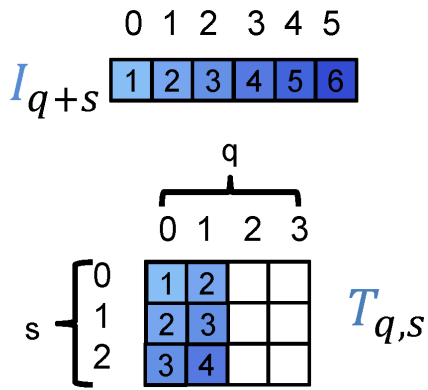


q	s	q+s
0	0	0
0	1	1
0	2	2
1	0	1
1	1	2
1	2	3

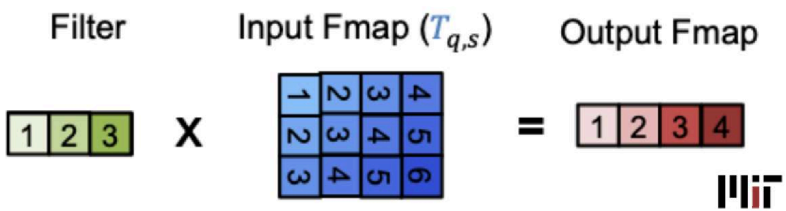


1-D Toeplitz Convolution Einsum

Example of execution of **step (1)** Topelitz conversion $T_{q,s} = I_{q+s}$

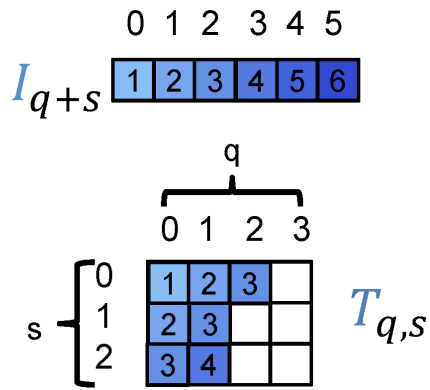


q	s	q+s
0	0	0
0	1	1
0	2	2
1	0	1
1	1	2
1	2	3
2	0	2

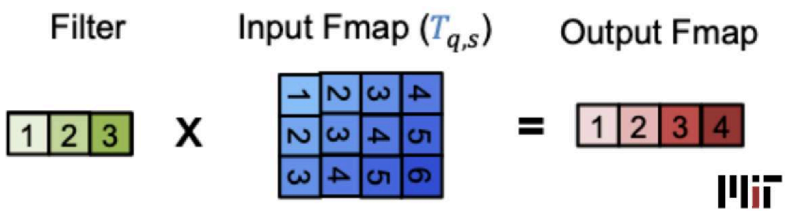


1-D Toeplitz Convolution Einsum

Example of execution of **step (1)** Topelitz conversion $T_{q,s} = I_{q+s}$

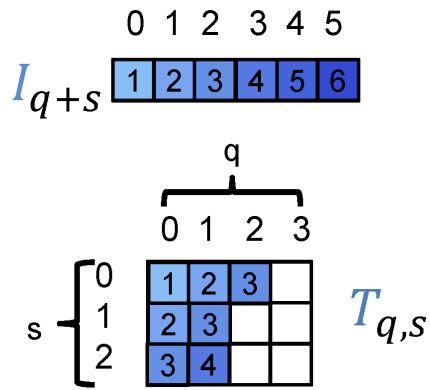


q	s	q+s
0	0	0
0	1	1
0	2	2
1	0	1
1	1	2
1	2	3
2	0	2

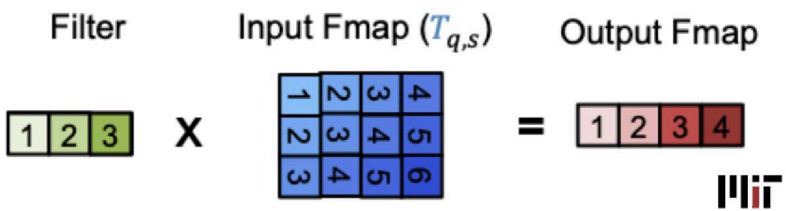


1-D Toeplitz Convolution Einsum

Example of execution of **step (1)** Topelitz conversion $T_{q,s} = I_{q+s}$

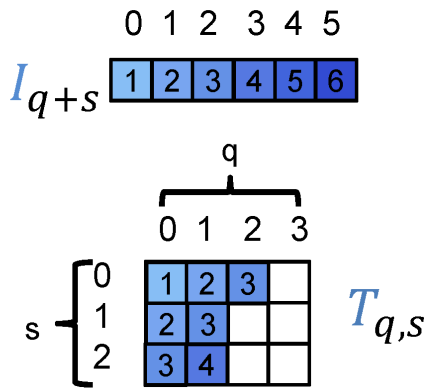


q	s	q+s
0	0	0
0	1	1
0	2	2
1	0	1
1	1	2
1	2	3
2	0	2

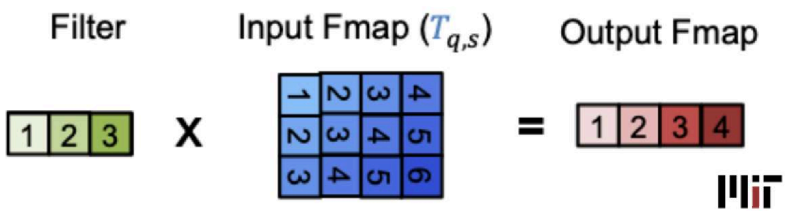


1-D Toeplitz Convolution Einsum

Example of execution of **step (1)** Topelitz conversion $T_{q,s} = I_{q+s}$

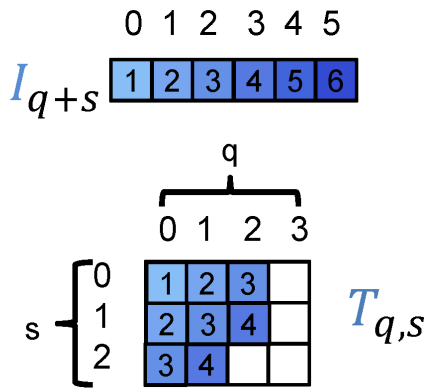


q	s	q+s
0	0	0
0	1	1
0	2	2
1	0	1
1	1	2
1	2	3
2	0	2
2	1	3

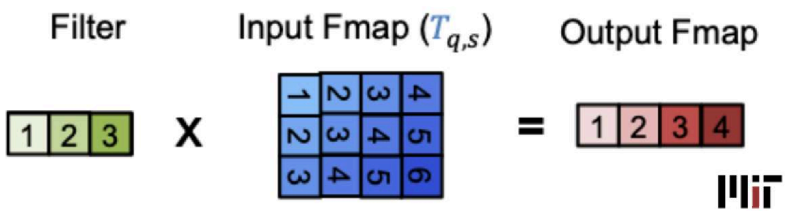


1-D Toeplitz Convolution Einsum

Example of execution of **step (1)** Topelitz conversion $T_{q,s} = I_{q+s}$

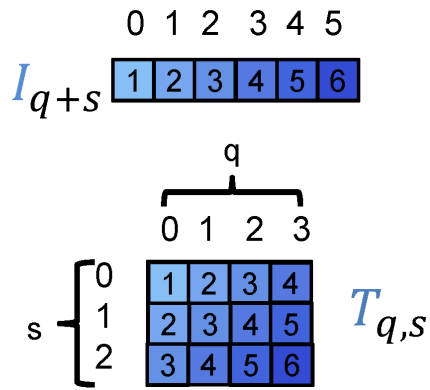


q	s	q+s
0	0	0
0	1	1
0	2	2
1	0	1
1	1	2
1	2	3
2	0	2
2	1	3

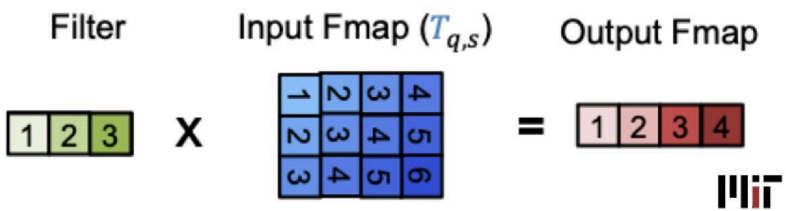


1-D Toeplitz Convolution Einsum

Example of execution of **step (1)** Topelitz conversion $T_{q,s} = I_{q+s}$



q	s	q+s
0	0	0
0	1	1
0	2	2
1	0	1
1	1	2
1	2	3
2	0	2
2	1	3
…	…	…



2-D Toeplitz Convolution Einsum

2-D Toeplitz Convolution Einsum

$$O_{m,p,q} = I_{c,p+r,q+s} \times F_{m,c,r,s}$$

2-D Toeplitz Convolution Einsum

$$O_{m,p,q} = I_{c,p+r,q+s} \times F_{m,c,r,s}$$

Break out Toeplitz conversion

2-D Toeplitz Convolution Einsum

$$O_{m,p,q} = I_{c,p+r,q+s} \times F_{m,c,r,s}$$

Break out Toeplitz conversion

$$T_{c,p,q,r,s} = I_{c,p+r,q+s}$$

2-D Toeplitz Convolution Einsum

$$O_{m,p,q} = I_{c,p+r,q+s} \times F_{m,c,r,s}$$

Break out Toeplitz conversion

$$T_{c,p,q,r,s} = I_{c,p+r,q+s}$$

Flatten ranks

2-D Toeplitz Convolution Einsum

$$O_{m,p,q} = I_{c,p+r,q+s} \times F_{m,c,r,s}$$

Break out Toeplitz conversion

$$T_{c,p,q,r,s} = I_{c,p+r,q+s}$$

Flatten ranks

$$T_{pq,crs} \rightarrow T_{c,p,q,r,s}$$

2-D Toeplitz Convolution Einsum

$$O_{m,p,q} = I_{c,p+r,q+s} \times F_{m,c,r,s}$$

Break out Toeplitz conversion

$$T_{c,p,q,r,s} = I_{c,p+r,q+s}$$

Flatten ranks

$$T_{pq,crs} \rightarrow T_{c,p,q,r,s}$$

2-D Toeplitz Convolution Einsum

$$O_{m,p,q} = I_{c,p+r,q+s} \times F_{m,c,r,s}$$

Break out Toeplitz conversion

$$T_{c,p,q,r,s} = I_{c,p+r,q+s}$$

Flatten ranks

$$T_{pq,crs} \rightarrow T_{c,p,q,r,s}$$

$$F_{m,crs} \rightarrow F_{m,c,r,s}$$

2-D Toeplitz Convolution Einsum

$$O_{m,p,q} = I_{c,p+r,q+s} \times F_{m,c,r,s}$$

Break out Toeplitz conversion

$$T_{c,p,q,r,s} = I_{c,p+r,q+s}$$

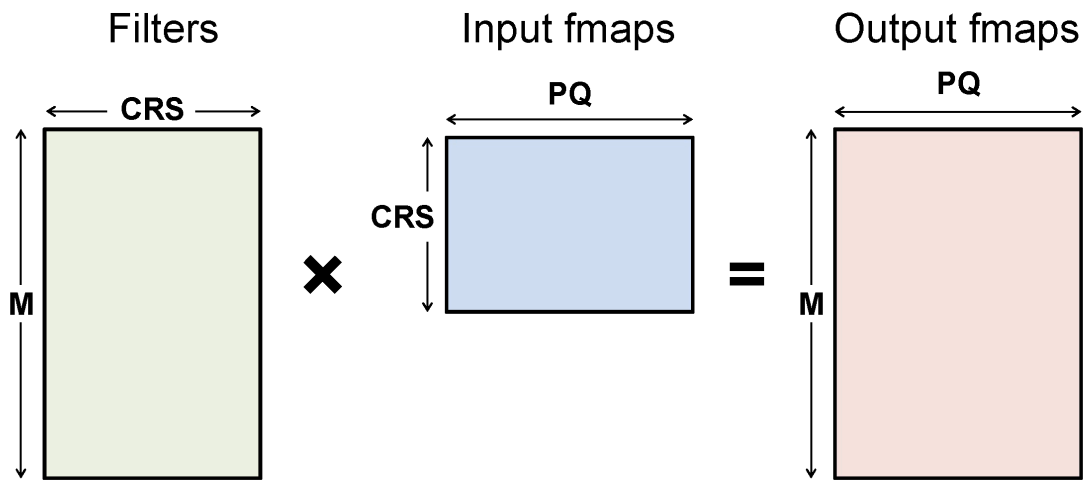
Flatten ranks

$$T_{pq,crs} \rightarrow T_{c,p,q,r,s}$$

$$F_{m,crs} \rightarrow F_{m,c,r,s}$$

$$O_{m,pq} = T_{pq,crs} \times F_{m,crs}$$

Convolution (CONV) Layer → Matrix Multiplication

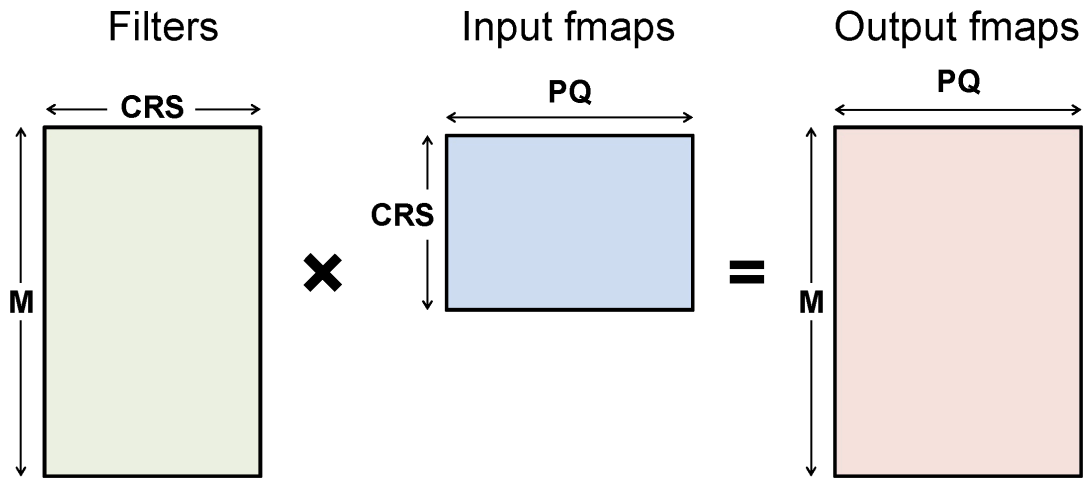


where P=H-R+1 and Q=W-S+1

N=1 in example

Convolution (CONV) Layer → Matrix Multiplication

$$O_{m,pq} = T_{pq,crs} \times F_{m,crs}$$

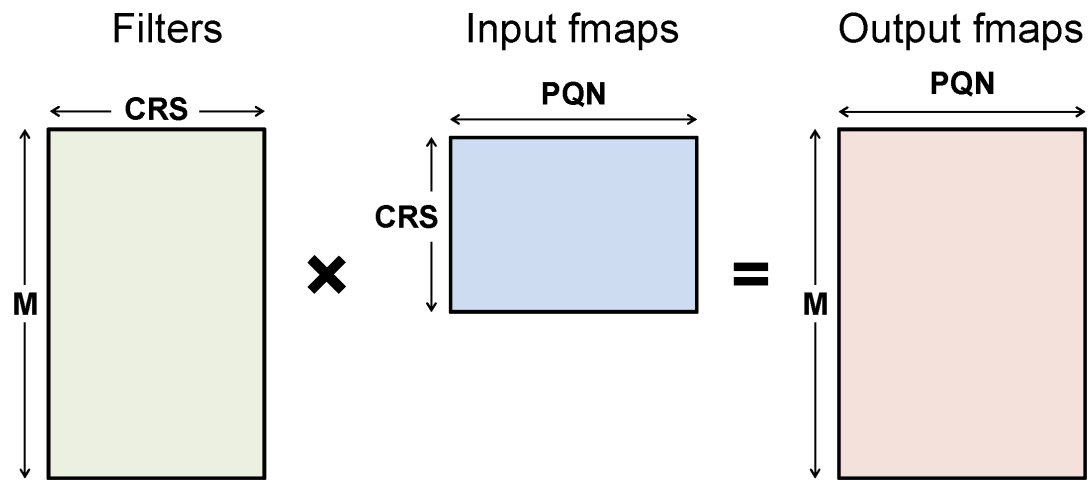


where P=H-R+1 and Q=W-S+1

N=1 in example

Convolution (CONV) Layer → Matrix Multiplication

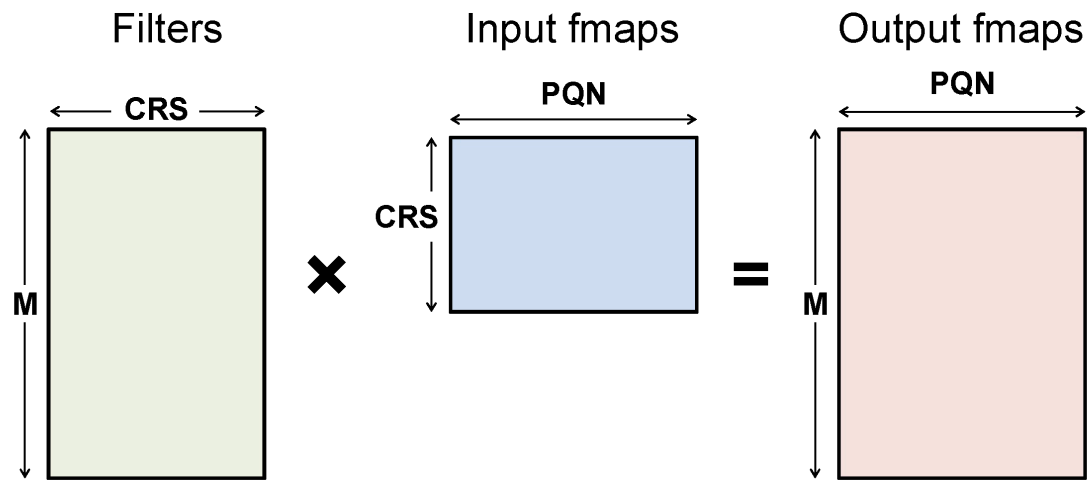
Include **N** rank



where P=H-R+1 and Q=W-S+1

Convolution (CONV) Layer → Matrix Multiplication

Include **N** rank

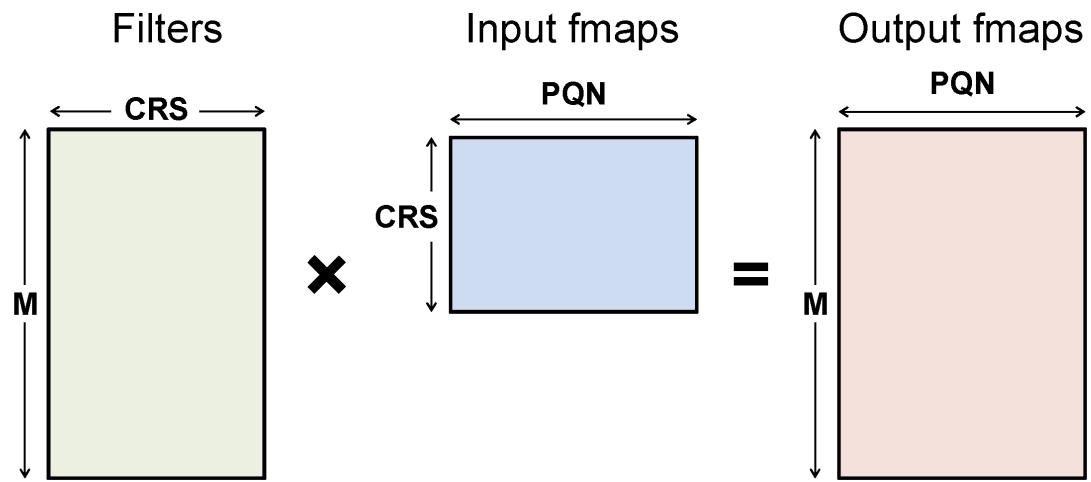


where P=H-R+1 and Q=W-S+1

Convolution (CONV) Layer → Matrix Multiplication

Include **N** rank

$$O_{m,pqn} = T_{pqn,crs} \times F_{m,crs}$$



where P=H-R+1 and Q=W-S+1

Summary

- We have shown how FC and CONV layers can be converted into matrix multiplication
- There are many ways to compute this matrix multiplication
  - i.e., the MACs can be performed in many different orders and achieve the same output
- However, the processing order impacts hardware metrics such as energy and latency
  - e.g., processing order can affect the amount of data movement

Attention

Convolution vs Attention

Convolution

- Only models dependencies between spatial neighbors
- Use sparsely connected layer to spatial neighbors; no support for dependencies outside of spatial dimensions of filter (R x S)
- Fixed region of interest in input for given output

Attention

- "Allows modeling of global dependencies without regard to their distance" (Vaswani, NeurIPS 2017)
- However, fully connected layer is too expensive; develop mechanism to bias "the allocation of available computational resources towards the most informative components of a signal" (Hu, CVPR 2018)
- Dynamically select region of interest in input for given output

Transformers

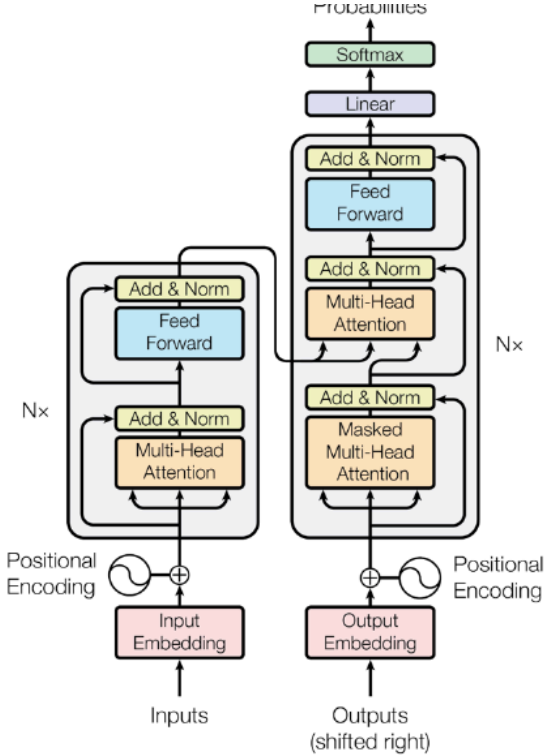


Image Source: [**Vaswani**, *NeurIPS* 2017]

- Built from Attention Mechanism Vaswani,NeurIPS 2017
- Widely used for natural language processing (e.g., GPT-3 Brown, NeurIPS 2020), since long dependencies between words exist
- Also used for other forms of data including – audio (e.g., AST Gong, Interspeech 2021) – vision (e.g., ViT Dosovitskiy, ICLR 2021)

Format of Input into Attention Mechanism

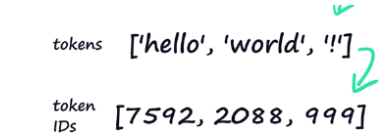
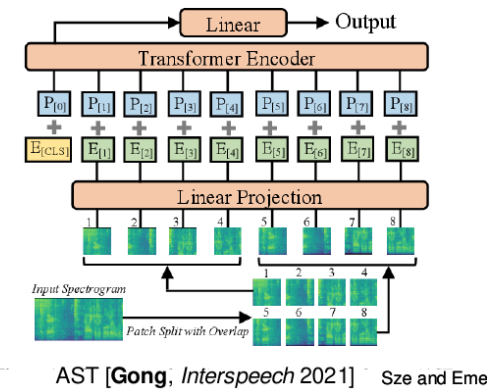


Image Source: https://towardsdatascience.com/why-are-there-so-many-tokenization-methods-for-transformers-a340e493b3a8



AST [**Gong**, *Interspeech* 2021]

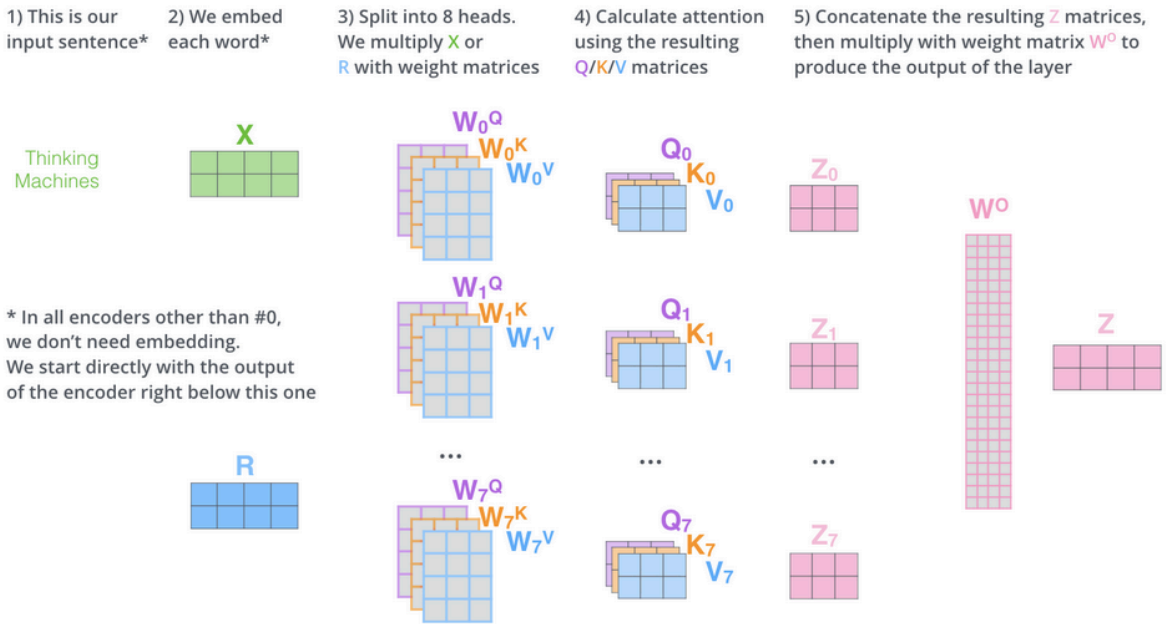
Sze and Eme

Break input into chunks referred to as tokens

– For a sentence, each word is a token
– For an image, each patch of pixels is a token
– For audio, each patch of spectrogram is a token

Support variable sized input by processing a sequence of "tokens" one at a time

Multi-head attention



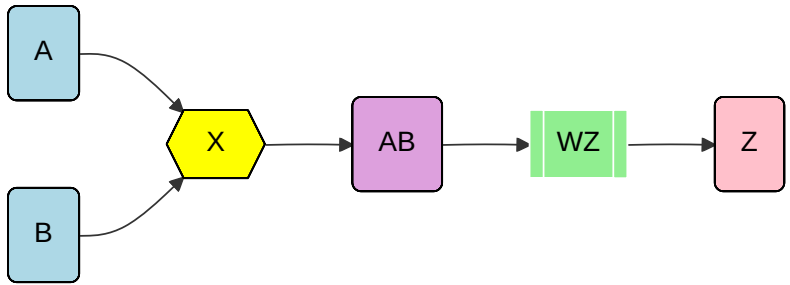
Source: https://jalammar.github.io/illustrated-transformer

Attention Mechanisms as Einsums

Based on collaborations with:

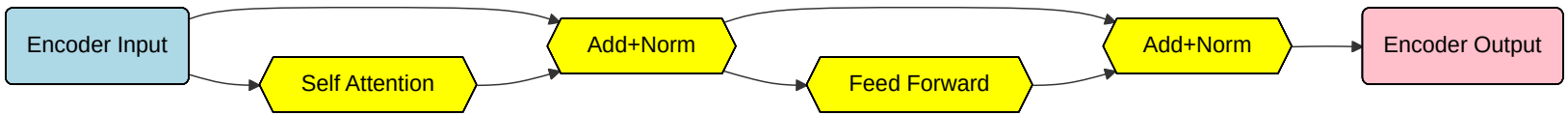
- Nandeeka Nayak (UC Berkeley)
- Alice Wu (UCLA)
- Chris Fletcher (UC Berkeley)
- Michael Pellauer (NVIDIA)

Diagram Conventions

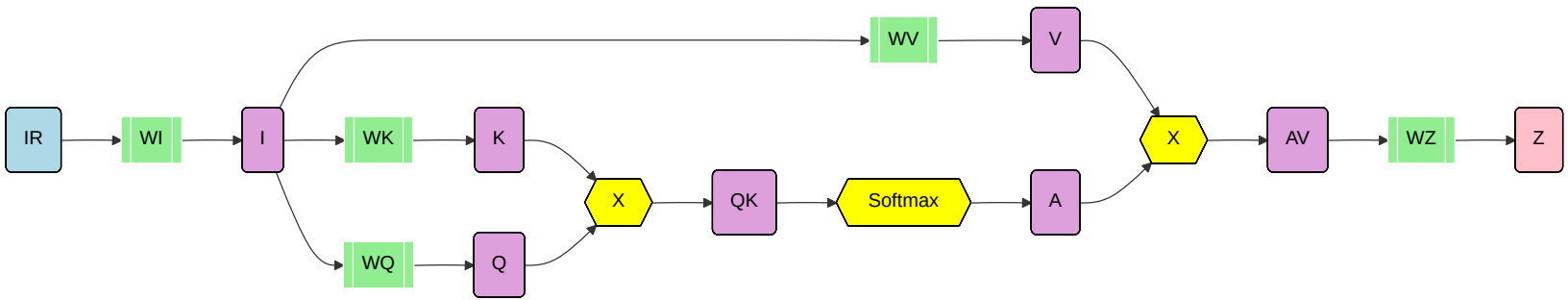


- Rounded box → Tensor
  - Examples: A, B, AB, Z
- Hexagonal box → Operation
  - Examples: $\times$, $softmax$
- Vertical lined box → Projection
  - Examples: $\times WZ$

Overall Encoder Structure



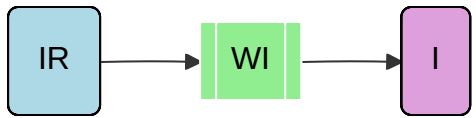
Basic Self-Attention Encoder Computation



Note: Some constant scaling steps are not illustrated.

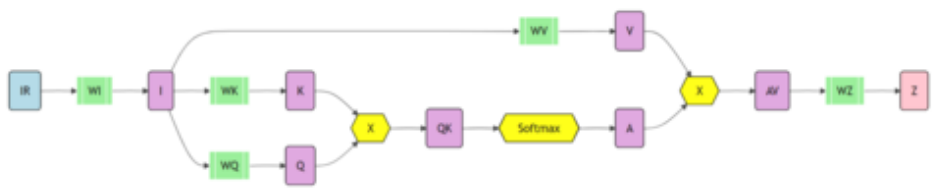
Inputs (Vocabulary Embedding)

Convert word sequence in one-hot representation to tokens in embedding space d.



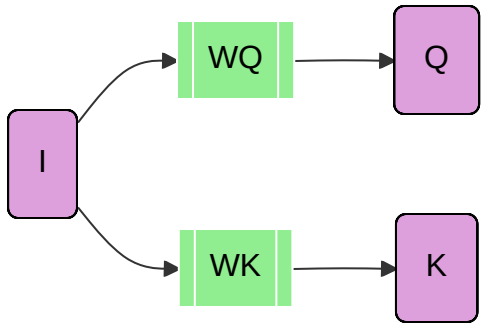
$$I_{m,d} = IR_{m,c} \times WI_{c,d}$$

Note: Only relevant to first attention computation in a chain of transformers



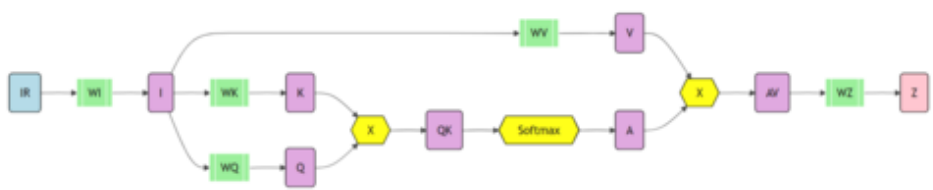
Calculate Key (K) and Query (Q)

Compute projections of I from d-space for "key" (K) and "query" (Q) into e-space



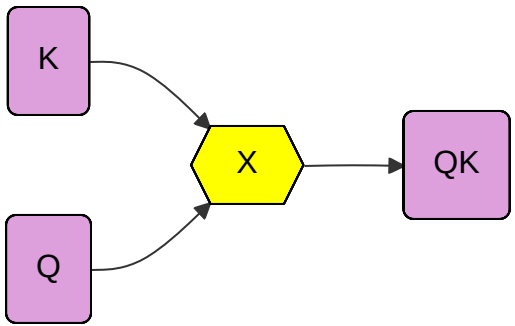
$$Q_{m,e} = I_{m,d} \times WQ_{d,e}$$

$$K_{m,e} = I_{m,d} \times WK_{d,e}$$



Multiply Key and Query

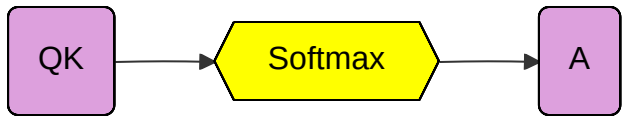
Create pre-softmax attention matrix



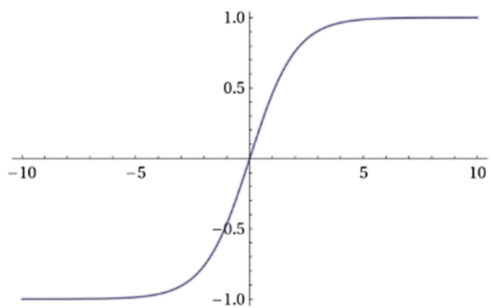
$$QK_{m,p}^{M,P=M} = Q_{p,e}^{M,E} \times K_{m,e}$$

Calculate Softmax

Create numerator (SN) and denominator (SD) for softmax and then do scaling…



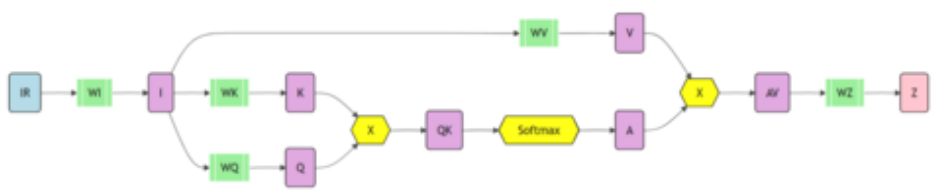
Softmax Activation Function



$$SN_{m,p} = exp(QK_{m,p})$$

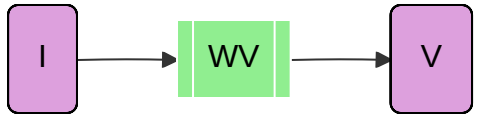
$$SD_p = SN_{m,p}$$

$$A_{m,p} = SN_{m,p}/SD_p$$



Calculate Value (V)

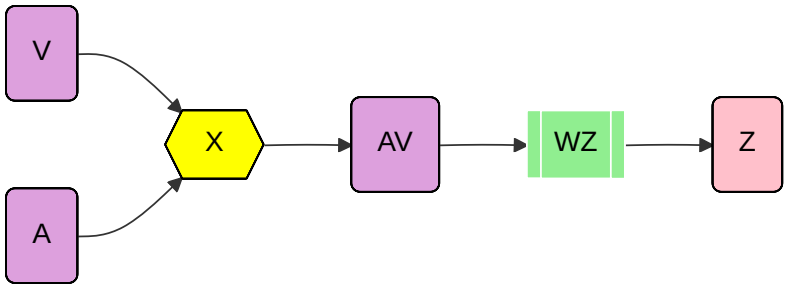
Project I from d-space to f-space to form V.



$$V_{m,f} = I_{m,d} \times WV_{d,f}$$

Create output (Z)

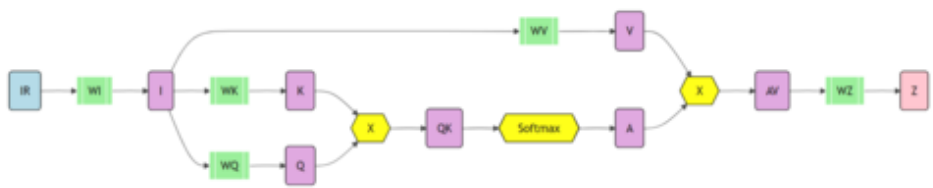
Multiply V and A and project from f-space into g-space



$$AV_{p,f}^{P=M,F} = A_{m,p} \times V_{m,f}$$

$$Z_{p,g} = AV_{p,f} \times WZ_{f,g}$$

Note: Embedding g is often the same as embedding d.



Full Cascade

$$I_{m,d} = IR_{m,c} \times WI_{c,d}$$

$$K_{m,e} = I_{m,d} \times WK_{d,e}$$

$$Q_{m,e} = I_{m,d} \times WQ_{d,e}$$

$$QK_{m,p}^{M,P=M} = Q_{p,e}^{M,E} \times K_{m,e}$$

$$SN_{m,p} = exp(QK_{m,p})$$

$$SD_p = SN_{m,p}$$

$$A_{m,p} = SN_{m,p} / SD_p$$

$$V_{m,f} = I_{m,d} \times WV_{d,f}$$

$$AV_{p,f}^{P=M,F} = A_{m,p} \times V_{m,f}$$

$$Z_{p,g} = AV_{p,f} \times WZ_{f,g}$$

Computation Properties of Attention Mechanism

Many matrix multiplications

– Three matrix multiplications for input projections

– One matrix multiplication for output projections

– Two matrix multiplications for computing scaled-dot product attention

– A total of five matrix multiplications plus one for output projection

Parallel processing opportunities

– Parallel multiplications across projections of Q, K, and V

Sequential dependency between matrix multiplications

– Within attention: QK then multiply by V

– Between projection and attention: Input projections → Attention → Output projection

Computation Properties of Attention Mechanism

Do operands change?

- Matrices WQ, WK, WV, and WZ **do not change** with input (static)
- Reuse WQ, WK, WV , and WZ across input tokens
- All other matrices (including Q, K, and V) change with input (dynamic)

Compute:

- Complexity based on size of matrices and number of tokens
- Number of MACs scales quadratically with number of input tokens

Storage:

- Input token matrix I,
- Weight storage for WQ, WK, WV and WZ
- Intermediate matrices (including Q, K, V),
- Output token matrix Z

Batched Attention

$$I_{b,m,d} = IR_{b,m,c} \times WI_{c,d}$$

$$K_{b,m,e} = I_{b,m,d} \times WK_{d,e}$$

$$Q_{b,m,e} = I_{b,m,d} \times WQ_{d,e}$$

$$QK_{b,m,p}^{B,M,P=M} = Q_{b,p,e}^{B,M,E} \times K_{b,m,e}$$

$$SN_{b,m,p} = exp(QK_{b,m,p})$$

$$SD_{b,p} = SN_{b,m,p}$$

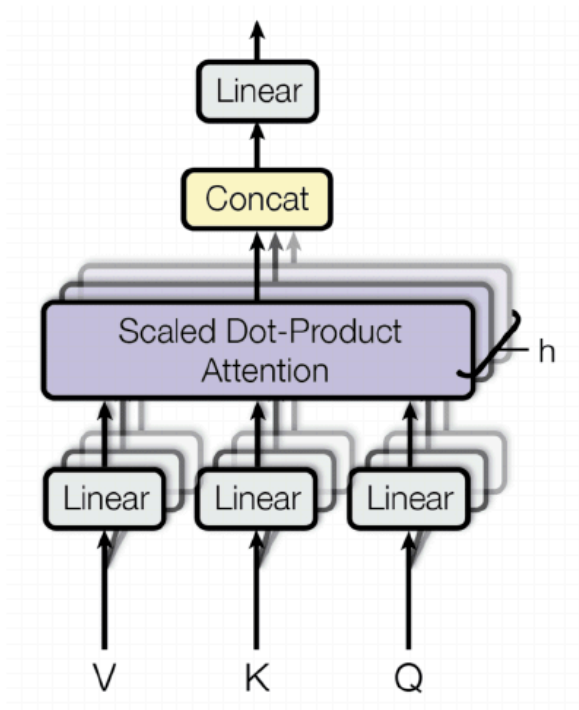
$$A_{b,m,p} = SN_{b,m,p} / SD_{b,p}$$

$$V_{b,m,f} = I_{b,m,d} \times WV_{d,f}$$

$$AV_{b,p,f}^{B,P=M,F} = A_{b,m,p} \times V_{b,m,f}$$

$$Z_{b,p,g} = AV_{b,p,f} \times WZ_{f,g}$$

Attention Mechanism: Multi-Head Attention



Source: https://d2l.ai/

Desirable to capture different behaviors
– e.g., shorter range versus longer range

Allow for different transforms of Q, K, and V.
– This is referred to as multi-head attention, where h is the number of heads.

Transforms are performed with linear projections
– Three matrix multiplications (WQ, WK, WV) per head

Outputs are concatenated and undergoes a linear projection
– Another matrix multiplication (WZ)

Multi-head Attention

(skipping initial embedding step)

$$K_{b,h,m,e} = I_{b,m,d} \times WK_{d,h,e}$$

$$Q_{b,h,m,e} = I_{b,m,d} \times WQ_{d,h,e}$$

$$QK_{b,h,m,p}^{B,H,M,P=M} = Q_{b,h,p,e}^{B,H,M,E} \times K_{b,h,m,e}$$

$$SN_{b,h,m,p} = exp(QK_{b,h,m,p})$$

$$SD_{b,h,p} = SN_{b,h,m,p}$$

$$A_{b,h,m,p} = SN_{b,h,m,p} / SD_{b,h,p}$$

$$V_{b,h,m,f} = I_{b,m,d} \times WV_{d,h,f}$$

$$AV_{b,h,p,f}^{B,H,P=M,F} = A_{b,h,m,p} \times V_{b,h,m,f}$$

$$C_{b,p,h\times F+f}^{B,P=M,G=H\times F} = AV_{b,h,p,f}$$

$$Z_{b,p,d} = C_{b,p,f} \times WZ_{g,d}$$

Rank Names

Word/token length ranks

- M - sequence length for the query, key and value in self-attention
- P - alias of sequence length for the rank of QK coming from Q
- R - sequence length for the query in non-self-attention

Vocabulary/embedding ranks

- C - dictionary size (words in vocabulary)
- D - input global space embedding (d_{model})
- E - query and key's local space embedding (d_k)
- F - value's local space embedding (d_v)
- G - output embedding

Other ranks

- B - batch size
- H - heads in multi-head attention

Input and Trained Tensors

Input tensors

- $IR^{M,C}$ - raw input (input to first layer only)
- $I^{M,D}$ - input after embedding (input to subsequent layers)

Weight tensors

- $WI^{C,D}$ - Weight tensor to create I
- $WK^{D,E}$ - Weight tensor to create K
- $WQ^{D,E}$ - Weight tensor to create Q
- $WV^{D,F}$ - Weight tensor to create V
- $WZ^{F,G}$ - Weight tensor to create Z

Note: Superscripts are names of the ranks

Principal Intermediate Tensors

Input projections

- $K^{M,E}$ - key
- $Q^{M,E}$ - query
- $V^{M,F}$ - value

Other tensors

- $A^{M,P}$ - Attention tensor
- $Z^{P,G}$ - Output tensor

Other Intermediate Tensors

Product tensors

- $QK^{M,P}$ - $Q \times K$
- $AV^{P,F}$ - $A \times V$

Softmax component tensors

- $SN^{M,P}$ - Softmax numerator
- SD^{P} - Softmax denominator