# CS4262/5462 Machine Learning Systems

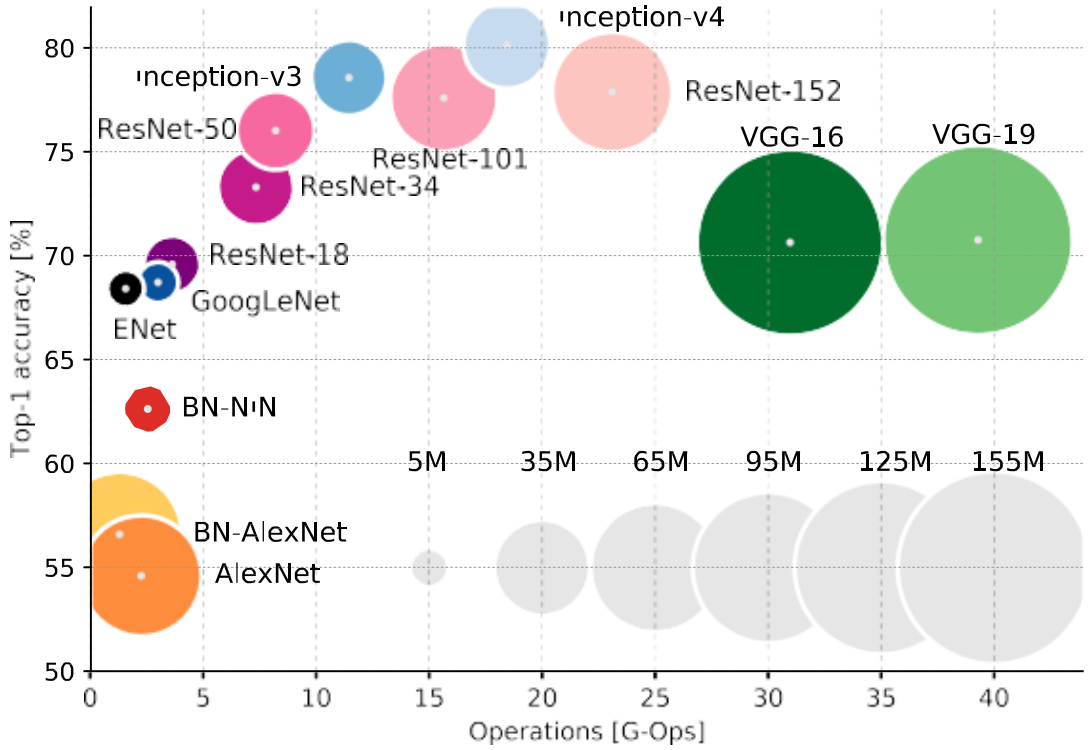
## Efficient AI

Yao LU

12 Mar 2026

National University of Singapore
School of Computing

# Memory and Computations

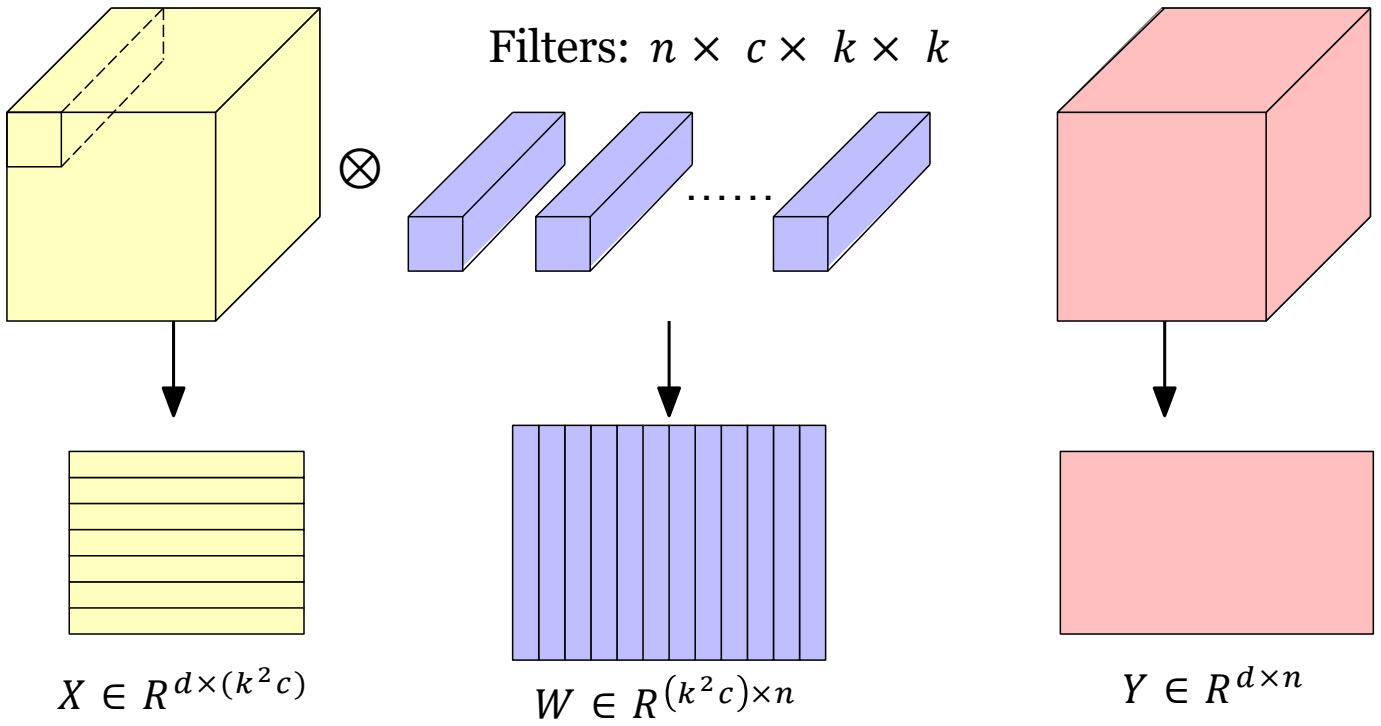



- The size of the blob is proportional to the number of network parameters.
- More than millions of parameters and billions of operations.
- Challenges in **memory** and **energy**, finally affect the performance.

# Agenda

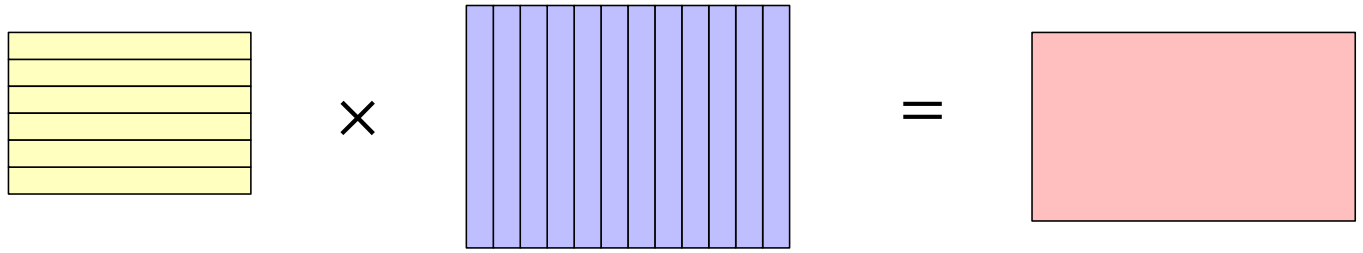
- Sparsity & Pruning
- SVD & Factorization based methods
- Quantization
- Distillation
- Cascade

# `Im2col` (Image2Column) Convolution




- Transform convolution to matrix multiplication
- Unified calculation for both convolution and fully-connected layers

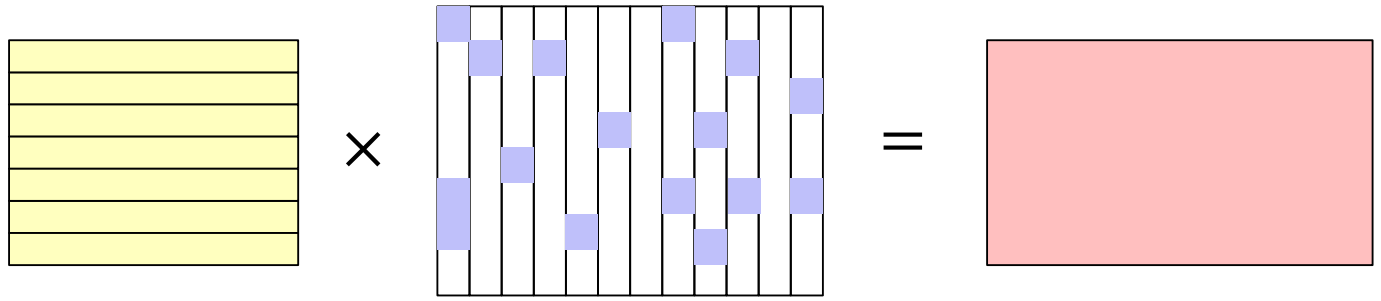
# Matrix Approximation or Matrix Regression?




Which is better?

- Matrix approximation: $\boldsymbol{W} \approx \boldsymbol{W'}$ $(\min \| W - W' \| = \min \| \Delta \|)$
- Matrix regression: $\boldsymbol{Y} = \boldsymbol{W} \cdot \boldsymbol{X} \approx \boldsymbol{W'} \cdot \boldsymbol{X}$ $(\min \| WX - W'X \| = \min \| \Delta \|)$

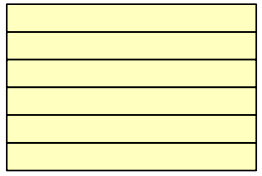
# Compression Approach 1: Random Sparsity



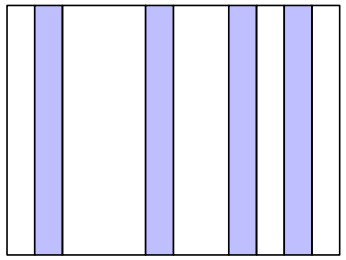

## Sparse DNN

- *Sparsification:* weight pruning;
- *Compression:* compressed sparse format for storage;
- *Potential acceleration:* sparse matrix multiplication algorithm.

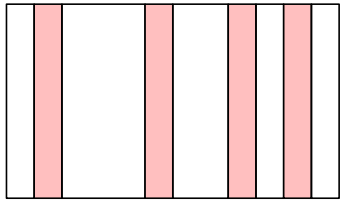
# Compression Approach 2: Structured Sparsity



$X \in R^{d \times (k^2 c)}$



$W \in R^{(k^2 c) \times n}$



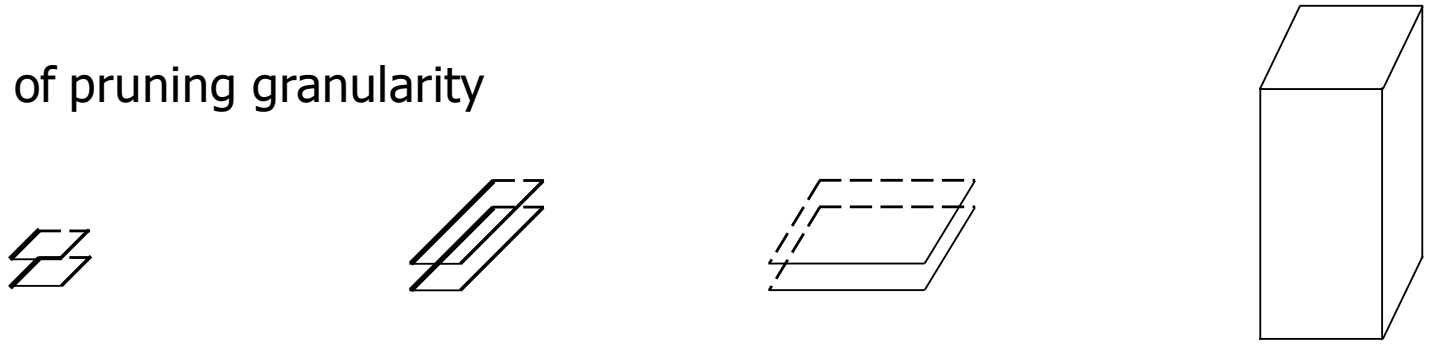
$Y \in R^{d \times n}$

## Structured Sparse DNN

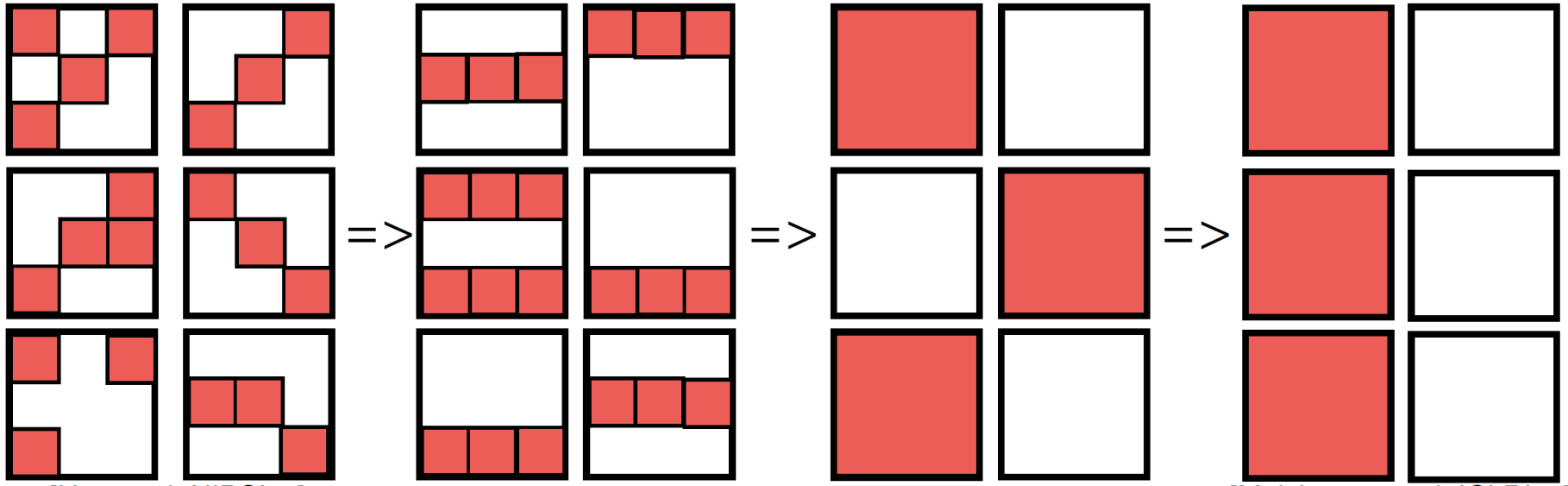
- *Potential acceleration:* GEMM or directed convolution.

# Exploring the Granularity of Sparsity that is Hardware-friendly

4 types of pruning granularity

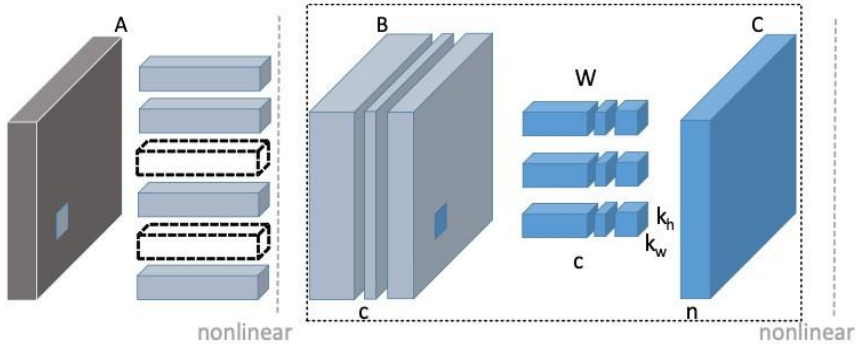


irregular sparsity | regular sparsity | more regular sparsity | fully-dense model




[Han et al, NIPS'15] [Molchanov et al, ICLR'17]

# Channel Pruning




We aim to reduce the width of feature map B, while minimizing the reconstruction error on feature map C. Our optimization algorithm performs within the dotted box, which does not involve nonlinearity. This figure illustrates the situation that two channels are pruned for feature map B. Thus corresponding channels of filters $W$ can be removed. Furthermore, even though not directly optimized by our algorithm, the corresponding filters in the previous layer can also be removed (marked by dotted filters). $c, n$: number of channels for feature maps B and C, $k_h \times k_w$: kernel size.

[2]Yihui He, Xiangyu Zhang, and Jian Sun (2017). "Channel Pruning for Accelerating Very Deep Neural Networks". In: *Proc. ICCV*.

# Channel Pruning

Formally, to prune a feature map with $c$ channels, we consider applying $n \times c \times k_h \times k_w$ convolutional filters W on $N \times c \times k_h \times k_w$ input volumes X sampled from this feature map, which produces $N \times n$ output matrix Y. Here, $N$ is the number of samples, $n$ is the number of output channels, and $k_h, k_w$ are the kernel size. For simple representation, bias term is not included in our formulation. To prune the input channels from $c$ to desired $c'$ ($0 \leq c' \leq c$), while minimizing reconstruction error, we formulate our problem as follow:

$$\begin{aligned} &\arg\min_{\boldsymbol{\beta}, \mathrm{W}} \frac{1}{2N} \left\| \mathrm{Y} - \sum_{i=1}^{c} \beta_i \mathrm{X_i} \mathrm{W_i}^\top \right\|_F^2 \\ &\text{subject to } \|\boldsymbol{\beta}\|_0 \leq c' \end{aligned} \tag{1}$$

$\|\cdot\|_F$ is Frobenius norm. $\mathrm{X_i}$ is $N \times k_h k_w$ matrix sliced from $i$th channel of input volumes X, $i = 1, \ldots, c$. $\mathrm{W_i}$ is $n \times k_h k_w$ filter weights sliced from $i$th channel of W. $\boldsymbol{\beta}$ is coefficient vector of length $c$ for channel selection, and $\beta_i$ is $i$th entry of $\boldsymbol{\beta}$. Notice that, if $\beta_i = 0$, $\mathrm{X_i}$ will be no longer useful, which could be safely pruned from feature map. $\mathrm{W_i}$ could also be removed.

[2]Yihui He, Xiangyu Zhang, and Jian Sun (2017). "Channel Pruning for Accelerating Very Deep Neural Networks". In: *Proc. ICCV*.

# Channel Pruning

Solving this $\ell_0$ minimization problem in Eqn. 1 is NP-hard. we relax the $\ell_0$ to $\ell_1$ regularization:

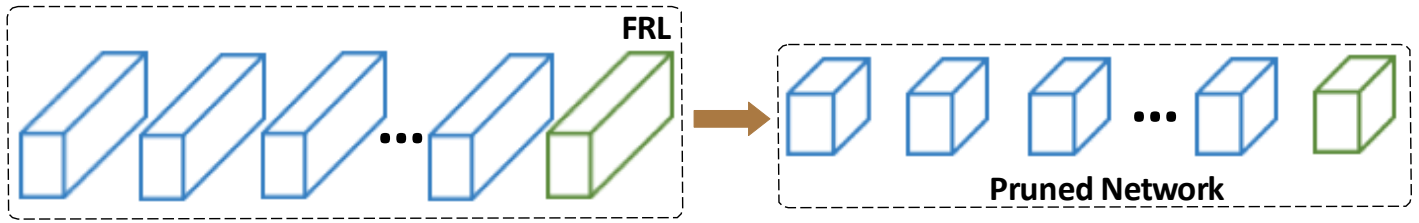
$$\begin{aligned} &\arg\min_{\beta,\mathrm{W}} \frac{1}{2N} \left\| \mathrm{Y} - \sum_{i=1}^{c} \beta_i \mathrm{X_i} \mathrm{W_i}^\top \right\|_F^2 + \lambda \|\beta\|_1 \\ &\text{subject to } \|\beta\|_0 \leq c', \forall i \, \|\mathrm{W_i}\|_F = 1 \end{aligned} \tag{2}$$

$\lambda$ is a penalty coefficient. By increasing $\lambda$, there will be more zero terms in $\beta$ and one can get higher speed-up ratio. We also add a constrain $\forall i \, \|\mathrm{W_i}\|_F = 1$ to this formulation, which avoids trivial solution. Now we solve this problem in two folds. First, we fix W, solve $\beta$ for channel selection. Second, we fix $\beta$, solve W to reconstruct error.

[2]Yihui He, Xiangyu Zhang, and Jian Sun (2017). "Channel Pruning for Accelerating Very Deep Neural Networks". In: *Proc. ICCV*.

# Feature Pruning

## Pruning Networks using Neuron Importance Score Propagation (NISP)



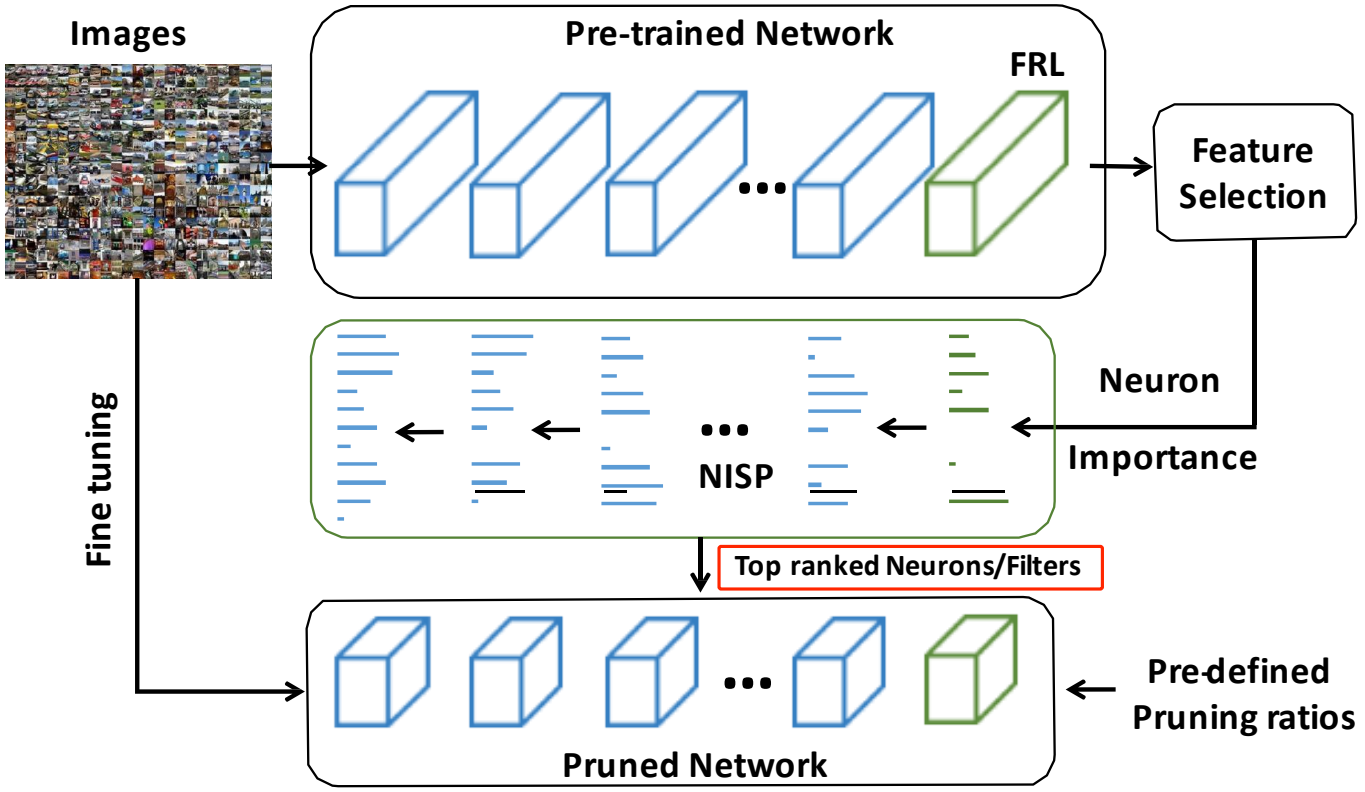

- FRL: final response layer
- Measure the importance of the neurons across the entire model;
- Rank features on the final response layer;
- Minimize the reconstruction errors of (important) final responses;
- Back-propagate the importance values and prune the neurons.

[3]Ruichi Yu et al. (2018). "NISP: Pruning networks using neuron importance score propagation".

# Feature Pruning

## Pruning Networks using Neuron Importance Score Propagation (NISP)

- Prune network using NISP.
- Fine-tune the pruned network.



# Feature Pruning

## Pruning Networks using Neuron Importance Score Propagation (NISP)

Some notations:

- The $l$-th layer $f^{(l)}(x)$ is represented as:

$$f^{(l)}(x) = \sigma^{(l)}(w^{(l)}x + b^{(l)}).$$

- A network with depth $n$ as a function $F^{(n)}$:

$$F^{(n)} = f^{(n)} \circ f^{(n-1)} \circ \cdots \circ f^{(1)}.$$

- The sub-network from $i$-th to $j$-th layer:

$$G^{(i,\,j)} = f^{(j)} \circ f^{(j-1)} \circ \cdots \circ f^{(i)}.$$

# Feature Pruning

## Pruning Networks using Neuron Importance Score Propagation (NISP)

- Define a binary vector $s_l^*$: neuron prune indicator for the $l$-th layer.
- The objective function for a single sample is defined as:

$$f\left(s_l^* \middle| x_l^{(m)}, s_n; F\right) = \langle s_n, |F(x) - F(s_l^* \odot x)| \rangle,$$

where $\langle \cdot, \cdot \rangle$ is dot product, $\odot$ is element-wise product, and $|\cdot|$ is element-wise absolute value.

- For all samples in the dataset:

$$\arg\min_{s_l^*} \sum_{m=1}^{M} f(s_l^* | x_l^{(m)}, s_n; G^{(l+1,n)})$$

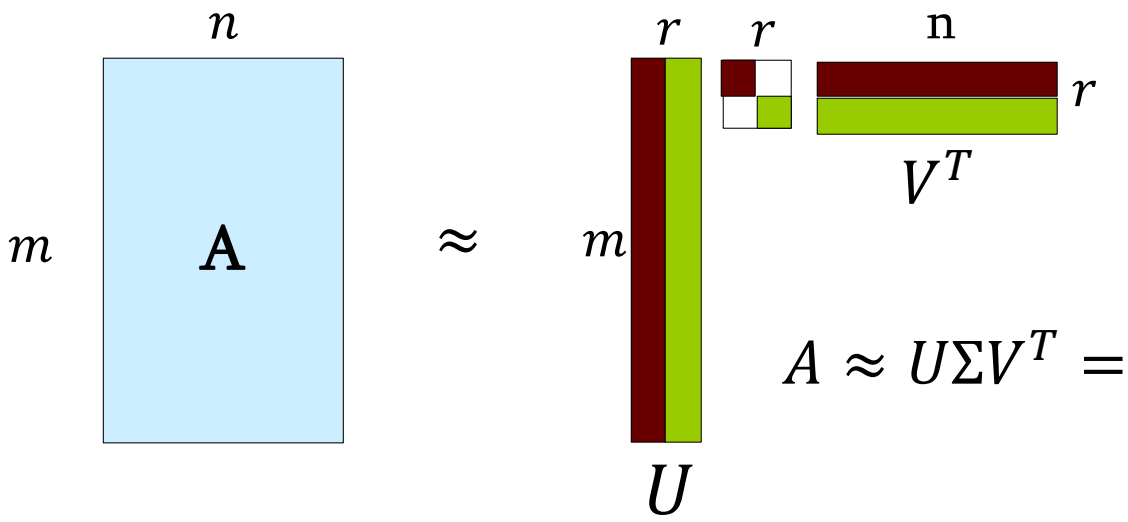
- Derive an upper-bound on this objective and minimize the upper-bound.

# Agenda

- Sparsity & Pruning
- SVD & Factorization based methods
- Quantization
- Distillation
- Cascade

# SVD: Singular Value Decomposition

- Given a decomposition of any matrix into a product of three matrices:

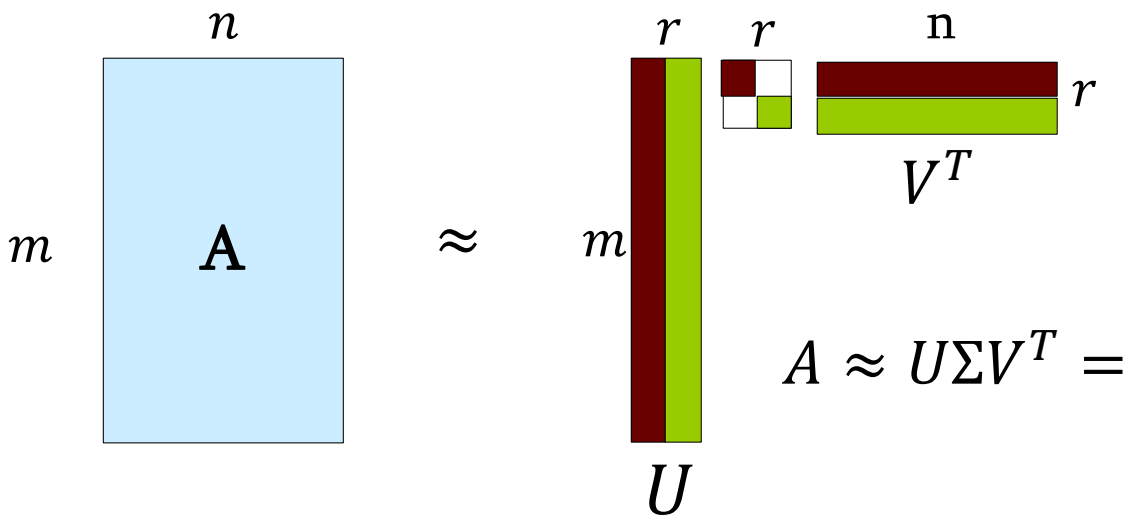



$$A \approx U\Sigma V^T = \sum_i \sigma_i u_i v_i^t$$

- **(Dimension reduction)** From the decomposition, you can choose any number of $m$ of intermediate concepts (latent factors) in a way that minimizes the reconstruction error.

# SVD: Singular Value Decomposition

- Given a decomposition of any matrix into a product of three matrices:

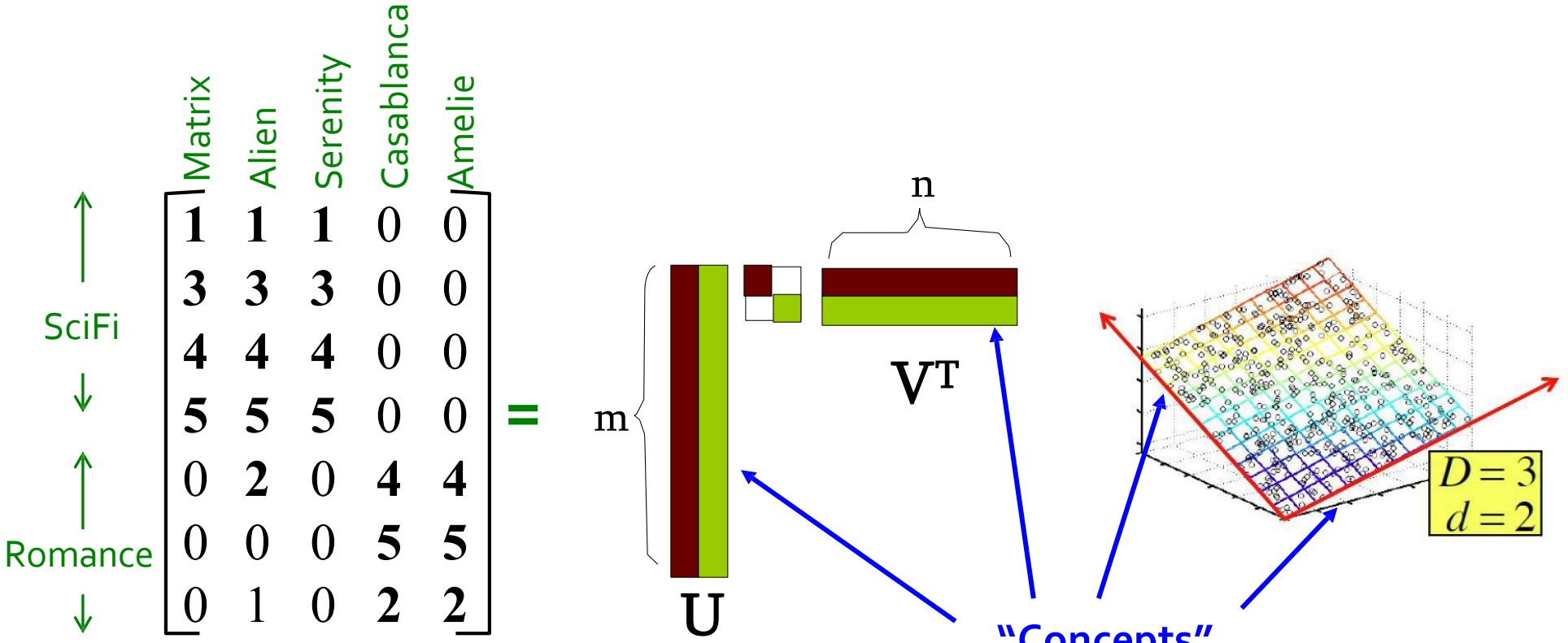



$$A \approx U\Sigma V^T = \sum_i \sigma_i u_i v_i^t$$

- **A:** Input data matrix (e.g., $m$ documents, $n$ terms)
- **U:** Left singular vectors ($m$ documents, $r$ concepts)
- **Σ:** Singular values ($r \times r$ diagonal matrix, strength of each concept, $r$: rank of A)
- ***V*:** Right singular vectors ($n$ terms, $r$ concepts)

# Example: users to movies

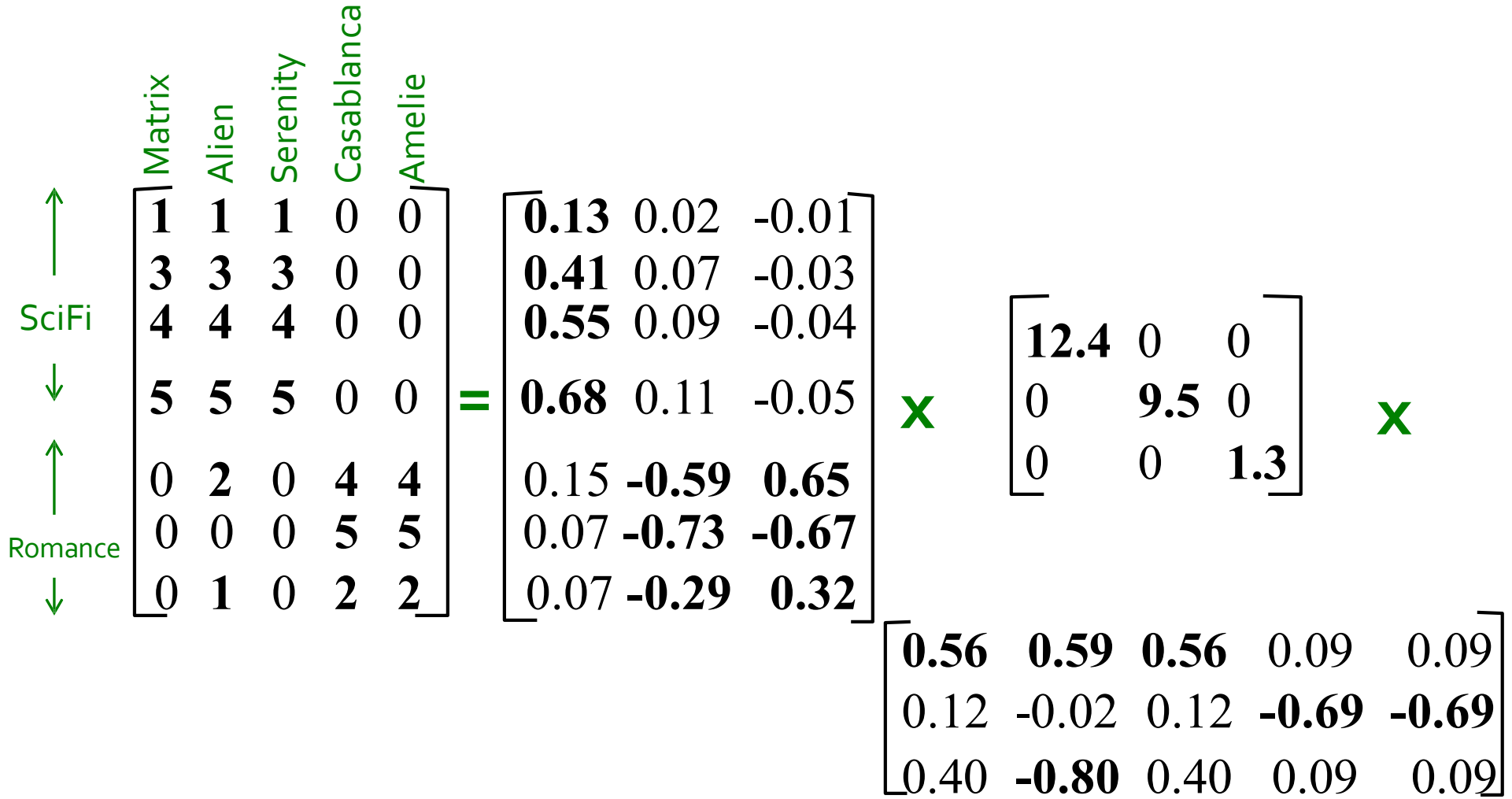
- Consider a matrix, what does SVD do?




Ratings matrix where each column corresponds to a movie and each row to a user. First 4 users prefer SciFi, while others prefer Romance.

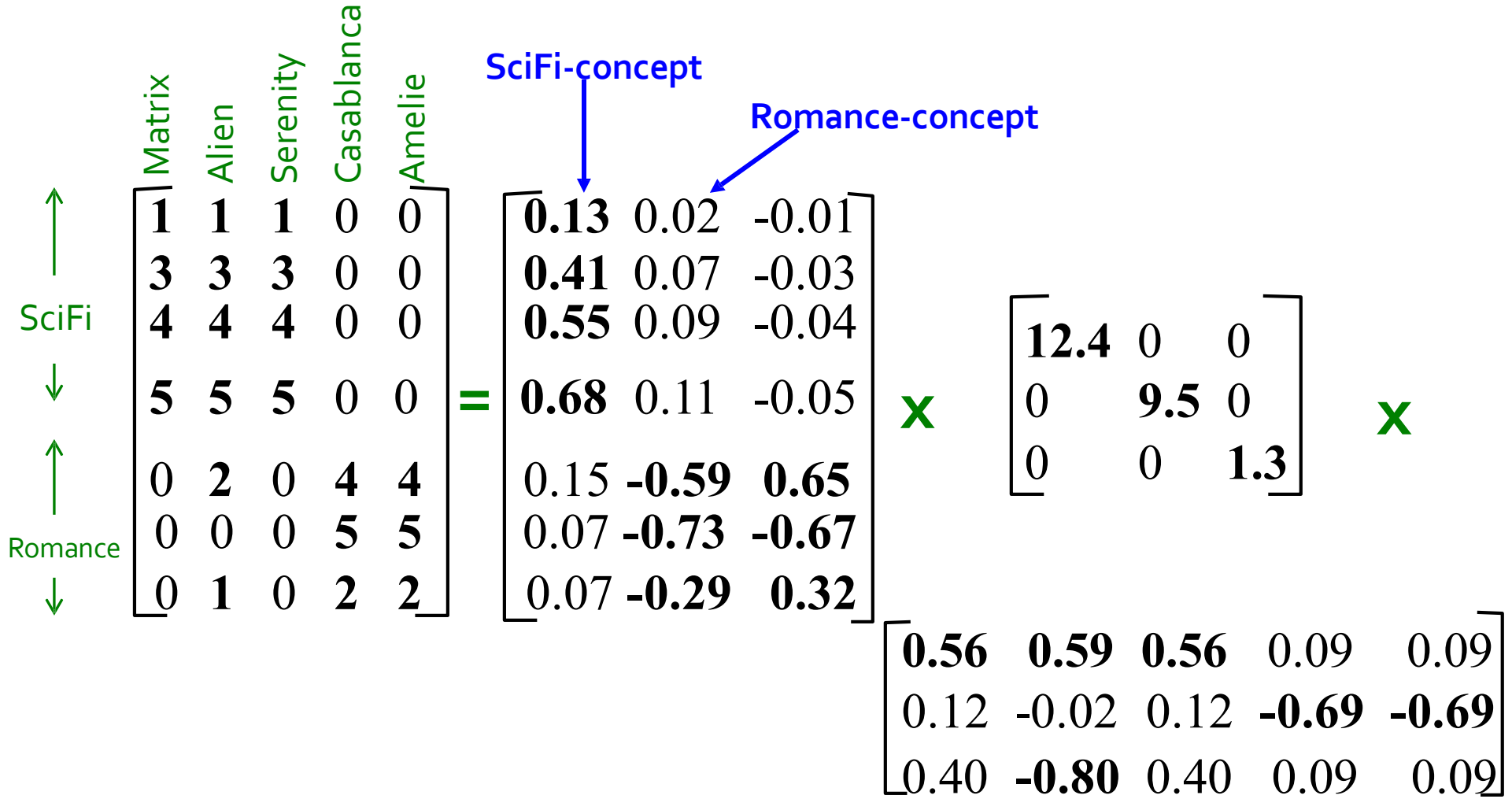
# Example: users to movies

- $A = U\Sigma V^T$ example



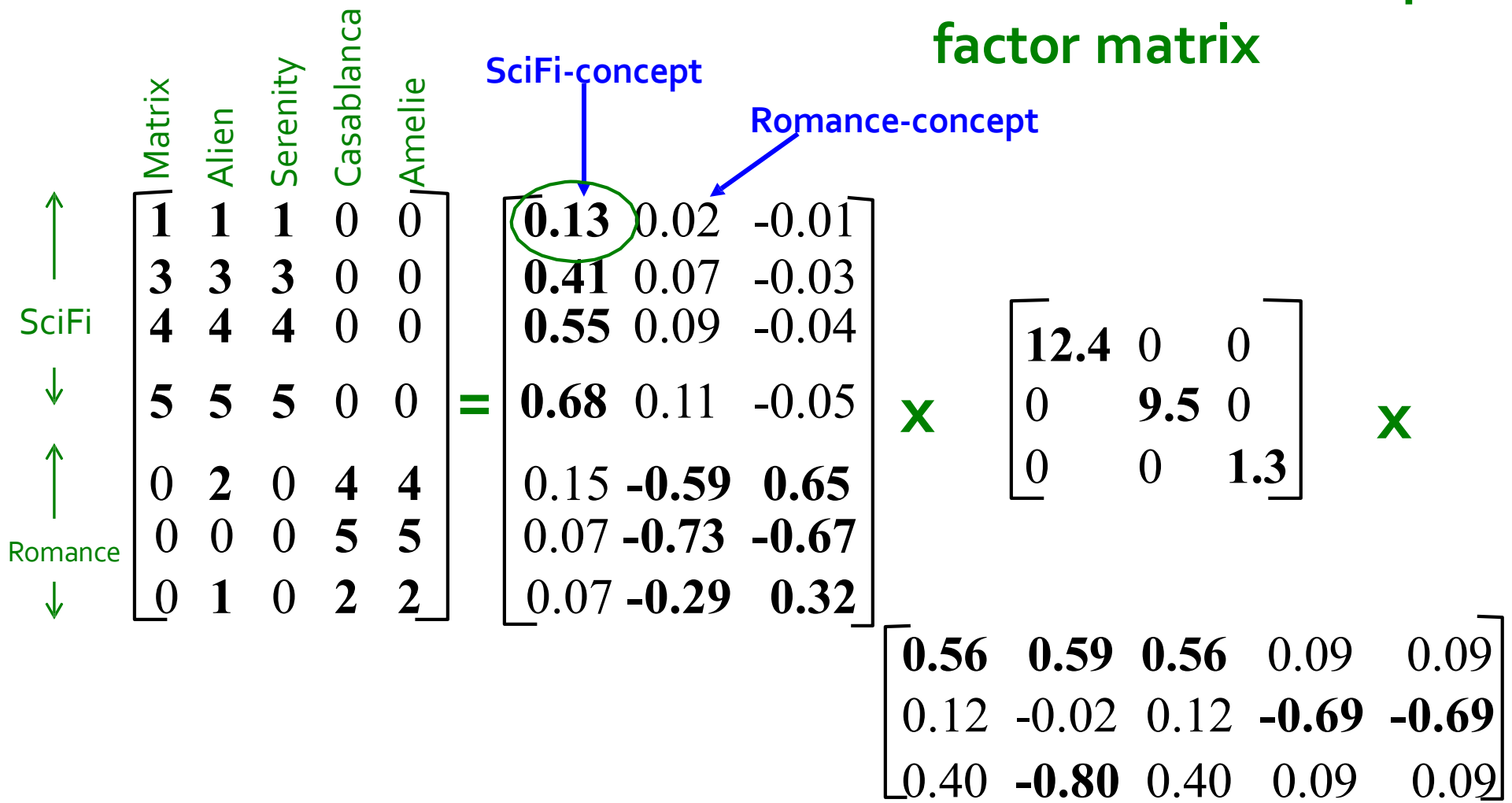
# Example: users to movies

- $A = U\Sigma V^T$ example



# Example: users to movies

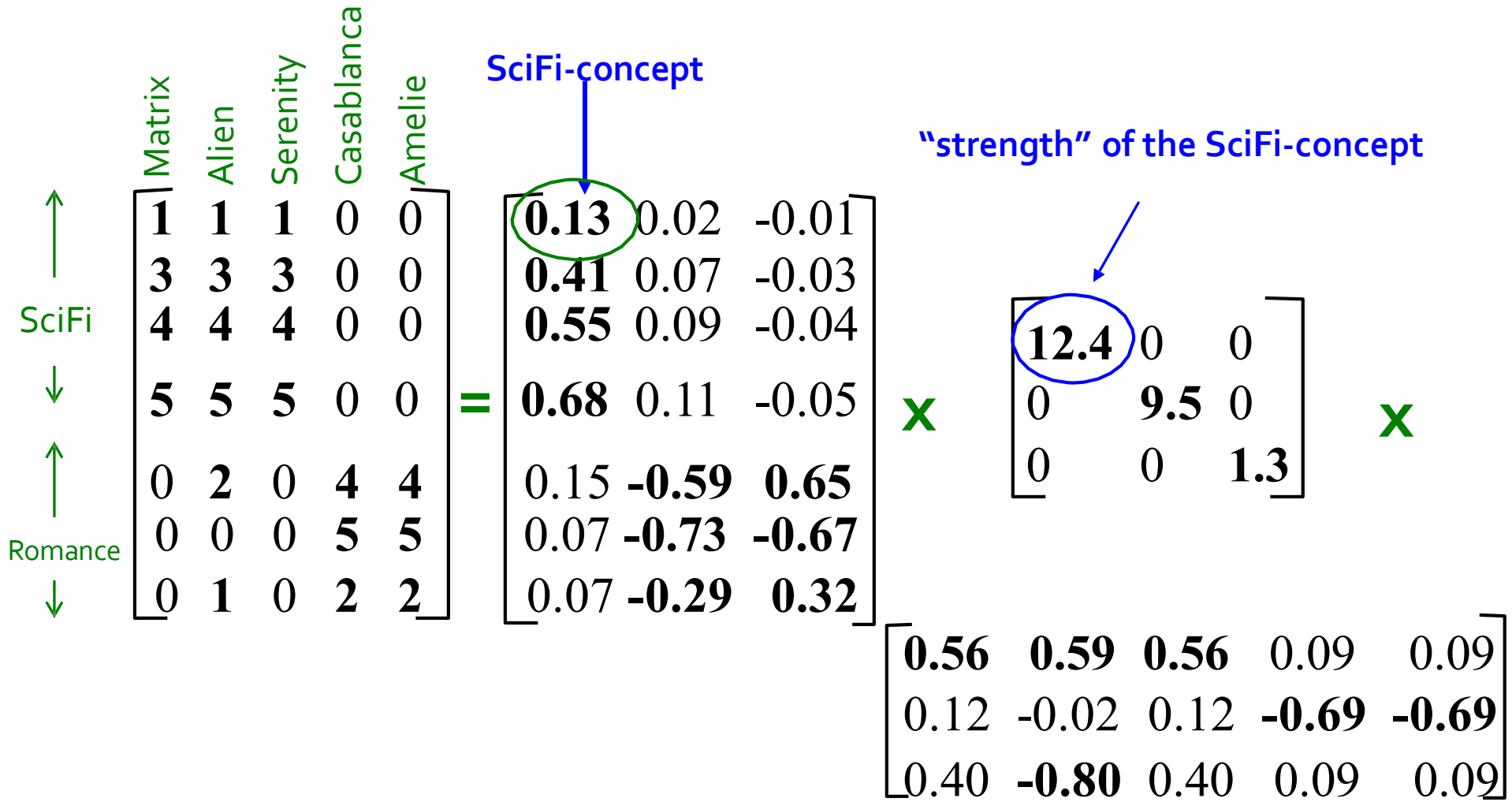
- $A = U\Sigma V^T$ example




***U* is "user-to-concept" factor matrix**

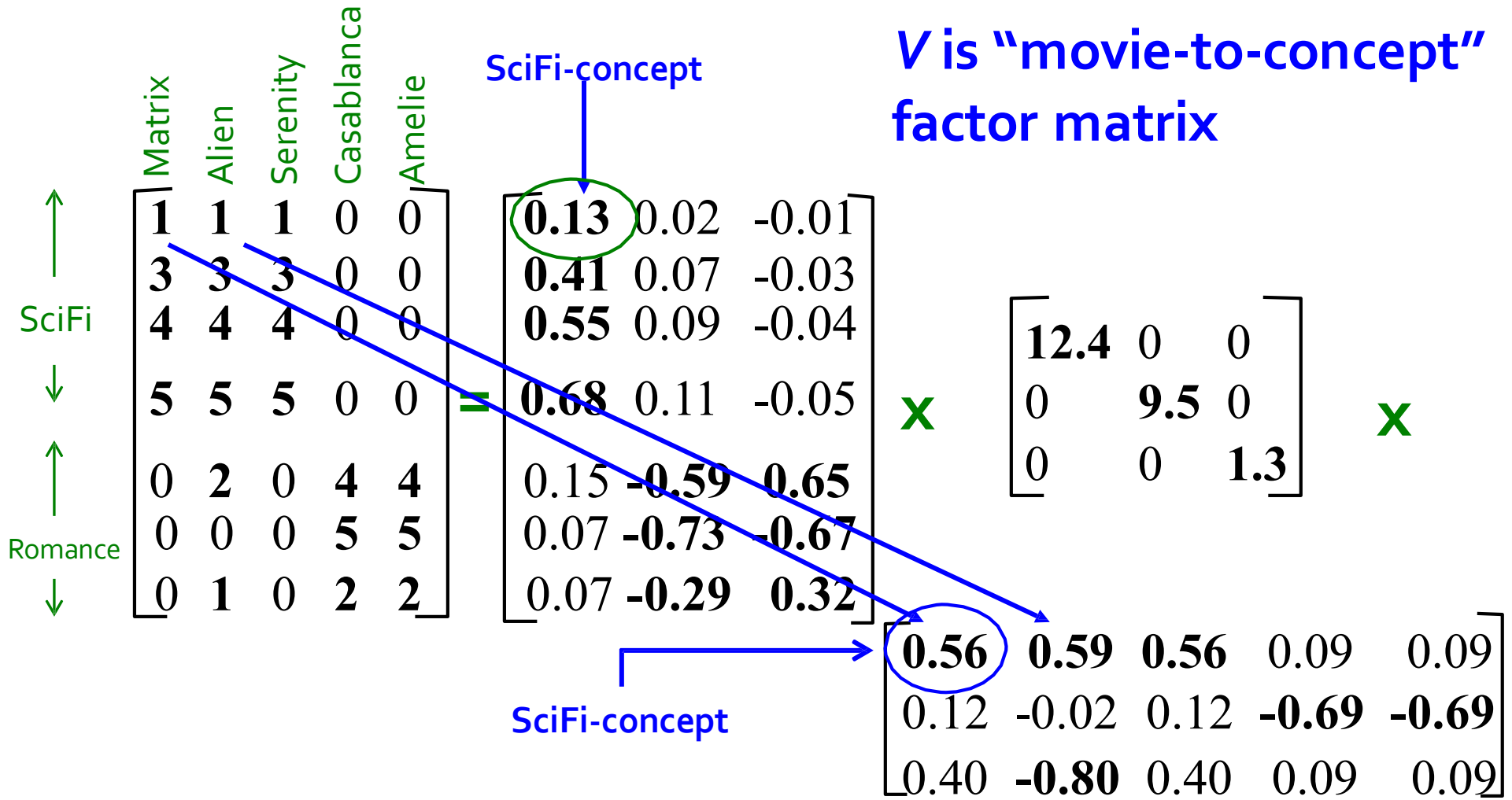
# Example: users to movies

- $A = U\Sigma V^T$ example



# Example: users to movies

- $A = U\Sigma V^T$ example

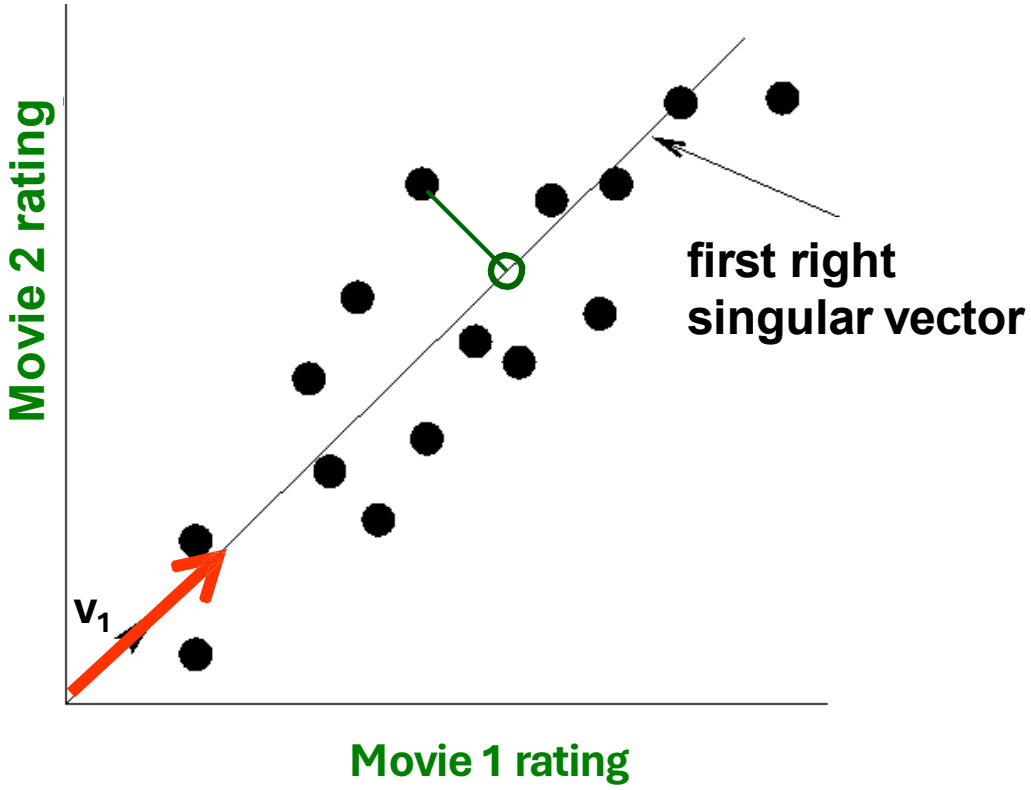


# SVD: Properties

- It is always possible to decompose a real matrix $A$ into $A = U\Sigma V^T = \sum_i \sigma_i u_i v_i^t$, where

- **$\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}$:** unique
- **$\mathbf{U}, \mathbf{V}$:** column orthonormal
  - $\mathrm{U}^{\mathrm{T}}\mathrm{U} = \mathrm{I}; \mathrm{V}^{\mathrm{T}}\mathrm{V} = \mathrm{I}$ (I: identity matrix)
  - Columns are orthogonal unit vectors
- **$\sum$:** Diagonal
  - Entries (**singular values**) are non-negative, and sorted in decreasing order ($\sigma_1 \geq \sigma_2 \geq \cdots \geq 0$)

# SVD: dimension reduction

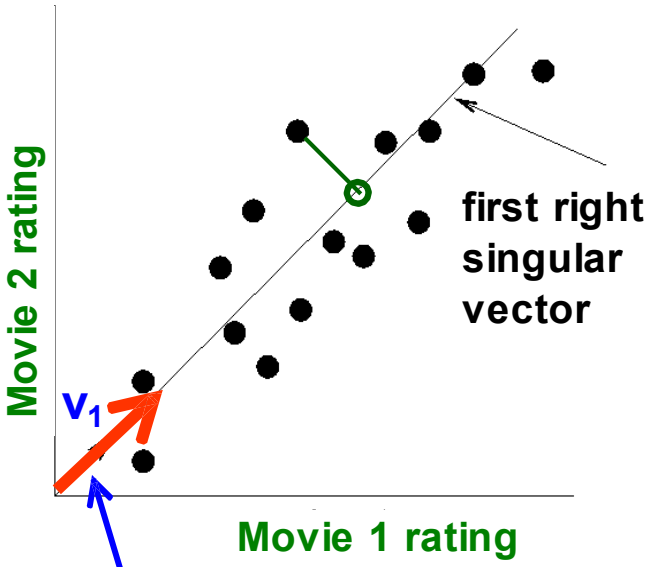



- Instead of using two coordinates $(x, y)$ to describe point positions, let's use only one coordinate.
- Point's position is its location along vector $v_1$

# Example: users to movies

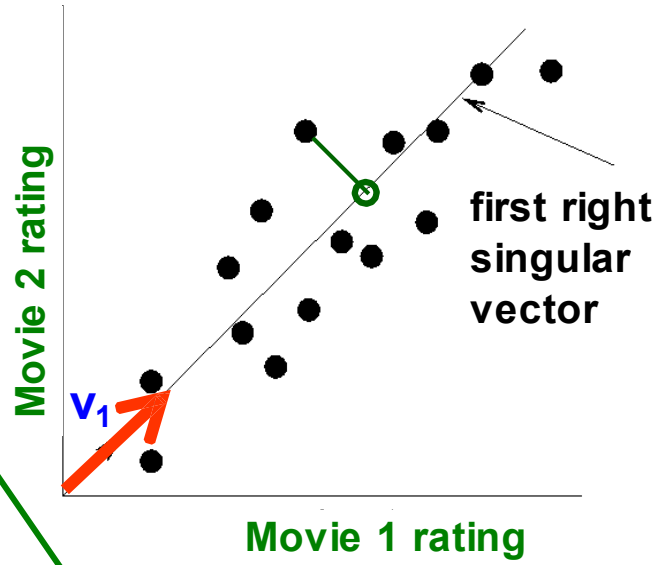
- $A = U\Sigma V^T$ example
  - **U**: "user-to-concept" matrix
  - **V**: "movie-to-concept" matrix

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 3 & 3 & 3 & 0 & 0 \\ 4 & 4 & 4 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 2 & 0 & 4 & 4 \\ 0 & 0 & 0 & 5 & 5 \\ 0 & 1 & 0 & 2 & 2 \end{bmatrix} = \begin{bmatrix} 0.13 & 0.02 & -0.01 \\ 0.41 & 0.07 & -0.03 \\ 0.55 & 0.09 & -0.04 \\ 0.68 & 0.11 & -0.05 \\ 0.15 & -0.59 & 0.65 \\ 0.07 & -0.73 & -0.67 \\ 0.07 & -0.29 & 0.32 \end{bmatrix} \times \begin{bmatrix} 12.4 & 0 & 0 \\ 0 & 9.5 & 0 \\ 0 & 0 & 1.3 \end{bmatrix} \times \begin{bmatrix} 0.56 & 0.59 & 0.56 & 0.09 & 0.09 \\ 0.12 & -0.02 & 0.12 & -0.69 & -0.69 \\ 0.40 & -0.80 & 0.40 & 0.09 & 0.09 \end{bmatrix}$$



# Example: users to movies

- $A = U\Sigma V^T$ example
  - **U**: "user-to-concept" matrix
  - **V**: "movie-to-concept" matrix

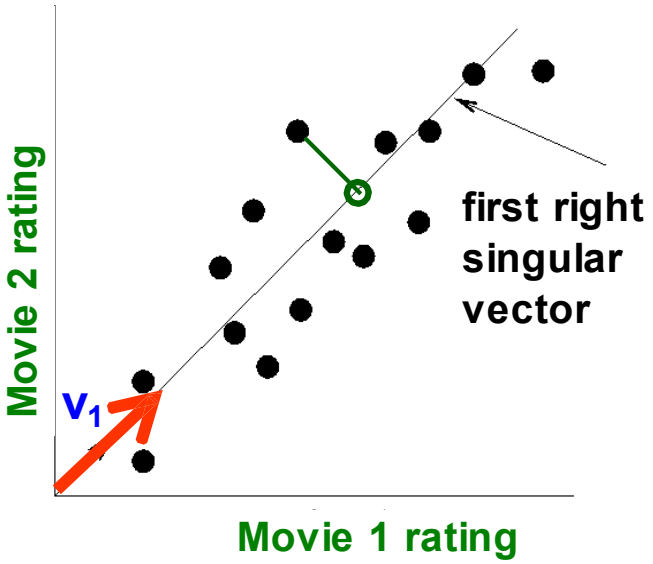



variance ('spread') on the $v_1$ axis

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 3 & 3 & 3 & 0 & 0 \\ 4 & 4 & 4 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 2 & 0 & 4 & 4 \\ 0 & 0 & 0 & 5 & 5 \\ 0 & 1 & 0 & 2 & 2 \end{bmatrix} = \begin{bmatrix} 0.13 & 0.02 & -0.01 \\ 0.41 & 0.07 & -0.03 \\ 0.55 & 0.09 & -0.04 \\ 0.68 & 0.11 & -0.05 \\ 0.15 & -0.59 & 0.65 \\ 0.07 & -0.73 & -0.67 \\ 0.07 & -0.29 & 0.32 \end{bmatrix} \times \begin{bmatrix} 12.4 & 0 & 0 \\ 0 & 9.5 & 0 \\ 0 & 0 & 1.3 \end{bmatrix} \times \begin{bmatrix} 0.56 & 0.59 & 0.56 & 0.09 & 0.09 \\ 0.12 & -0.02 & 0.12 & -0.69 & -0.69 \\ 0.40 & -0.80 & 0.40 & 0.09 & 0.09 \end{bmatrix}$$

# Example: users to movies

- $A = U\Sigma V^T$ example
  - **U Σ** : Gives the coordinates of the points in the projection axis




$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 3 & 3 & 3 & 0 & 0 \\ 4 & 4 & 4 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 2 & 0 & 4 & 4 \\ 0 & 0 & 0 & 5 & 5 \\ 0 & 1 & 0 & 2 & 2 \end{bmatrix}$$

**Projection of users on the "Sci-Fi" axis**

**U Σ:**

$$\begin{bmatrix} 1.61 & 0.19 & -0.01 \\ 5.08 & 0.66 & -0.03 \\ 6.82 & 0.85 & -0.05 \\ 8.43 & 1.04 & -0.06 \\ 1.86 & -5.60 & 0.84 \\ 0.86 & -6.93 & -0.87 \\ 0.86 & -2.75 & 0.41 \end{bmatrix}$$

# Example: users to movies

- $A = U\Sigma V^T$ example
  - How to do dimension reduction? Set smallest singular values to zero.

$$\begin{bmatrix} \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 \\ \mathbf{3} & \mathbf{3} & \mathbf{3} & 0 & 0 \\ \mathbf{4} & \mathbf{4} & \mathbf{4} & 0 & 0 \\ \mathbf{5} & \mathbf{5} & \mathbf{5} & 0 & 0 \\ 0 & \mathbf{2} & 0 & \mathbf{4} & \mathbf{4} \\ 0 & 0 & 0 & \mathbf{5} & \mathbf{5} \\ 0 & \mathbf{1} & 0 & \mathbf{2} & \mathbf{2} \end{bmatrix} = \begin{bmatrix} \mathbf{0.13} & 0.02 & -0.01 \\ \mathbf{0.41} & 0.07 & -0.03 \\ \mathbf{0.55} & 0.09 & -0.04 \\ \mathbf{0.68} & 0.11 & -0.05 \\ 0.15 & \mathbf{-0.59} & \mathbf{0.65} \\ 0.07 & \mathbf{-0.73} & \mathbf{-0.67} \\ 0.07 & \mathbf{-0.29} & \mathbf{0.32} \end{bmatrix} \times \begin{bmatrix} \mathbf{12.4} & 0 & 0 \\ 0 & \mathbf{9.5} & 0 \\ 0 & 0 & \mathbf{1.3} \end{bmatrix} \times \begin{bmatrix} \mathbf{0.56} & \mathbf{0.59} & \mathbf{0.56} & 0.09 & 0.09 \\ 0.12 & -0.02 & 0.12 & \mathbf{-0.69} & \mathbf{-0.69} \\ 0.40 & \mathbf{-0.80} & 0.40 & 0.09 & 0.09 \end{bmatrix}$$

# Example: users to movies

- $A = U\Sigma V^T$ example
  - How to do dimension reduction? Set smallest singular values to zero.

$$\begin{bmatrix} \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 \\ \mathbf{3} & \mathbf{3} & \mathbf{3} & 0 & 0 \\ \mathbf{4} & \mathbf{4} & \mathbf{4} & 0 & 0 \\ \mathbf{5} & \mathbf{5} & \mathbf{5} & 0 & 0 \\ 0 & \mathbf{2} & 0 & \mathbf{4} & \mathbf{4} \\ 0 & 0 & 0 & \mathbf{5} & \mathbf{5} \\ 0 & \mathbf{1} & 0 & \mathbf{2} & \mathbf{2} \end{bmatrix} \approx \begin{bmatrix} \mathbf{0.13} & 0.02 & -0.01 \\ \mathbf{0.41} & 0.07 & -0.03 \\ \mathbf{0.55} & 0.09 & -0.04 \\ \mathbf{0.68} & 0.11 & -0.05 \\ 0.15 & \mathbf{-0.59} & \mathbf{0.65} \\ 0.07 & \mathbf{-0.73} & \mathbf{-0.67} \\ 0.07 & \mathbf{-0.29} & \mathbf{0.32} \end{bmatrix} \times \begin{bmatrix} \mathbf{12.4} & 0 & 0 \\ 0 & \mathbf{9.5} & 0 \\ 0 & 0 & \mathbf{1.3} \end{bmatrix} \times \begin{bmatrix} \mathbf{0.56} & \mathbf{0.59} & \mathbf{0.56} & 0.09 & 0.09 \\ 0.12 & -0.02 & 0.12 & \mathbf{-0.69} & \mathbf{-0.69} \\ 0.40 & \mathbf{-0.80} & 0.40 & 0.09 & 0.09 \end{bmatrix}$$

# Example: users to movies

- $A = U\Sigma V^T$ example
  - How to do dimension reduction? Set smallest singular values to zero.

This is Rank 2 approximation to A. We could also do Rank 1 approx. The larger the rank the more accurate the approximation

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 3 & 3 & 3 & 0 & 0 \\ 4 & 4 & 4 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 2 & 0 & 4 & 4 \\ 0 & 0 & 0 & 5 & 5 \\ 0 & 1 & 0 & 2 & 2 \end{bmatrix} \quad \begin{bmatrix} 0.92 & 0.95 & 0.92 & 0.01 & 0.01 \\ 2.91 & 3.01 & 2.91 & -0.01 & -0.01 \\ 3.90 & 4.04 & 3.90 & 0.01 & 0.01 \\ 4.82 & 5.00 & 4.82 & 0.03 & 0.03 \\ 0.70 & 0.53 & 0.70 & 4.11 & 4.11 \\ -0.69 & 1.34 & -0.69 & 4.78 & 4.78 \\ 0.32 & 0.23 & 0.32 & 2.01 & 2.01 \end{bmatrix}$$

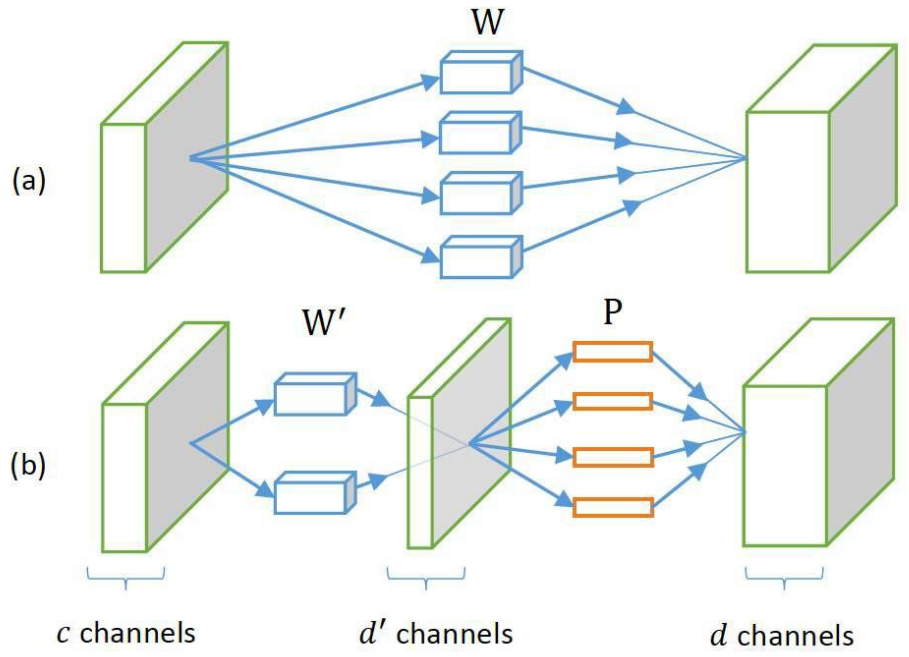
Reconstructed data matrix B

**Reconstruction Error is quantified by the Frobenius norm:**

$$|M|_F = \sqrt{\sum_{ij} M_{ij}^2}, \qquad |A - B|_F = \sqrt{\sum_{ij} (A_{ij} - B_{ij})^2} \quad \text{is "small"}$$

# Low rank approximation for Conv

- Layer responses lie in a low-rank subspace
- Decompose a convolutional layer with $d$ filters with filter size $k \times k \times c$ to
  - A layer with d' filters ($k \times k \times c$)
  - A layer with d filter ($1 \times 1 \times d$')




Zhang et al Efficient and Accurate Approximations of Nonlinear Convolutional Networks CVPR'15

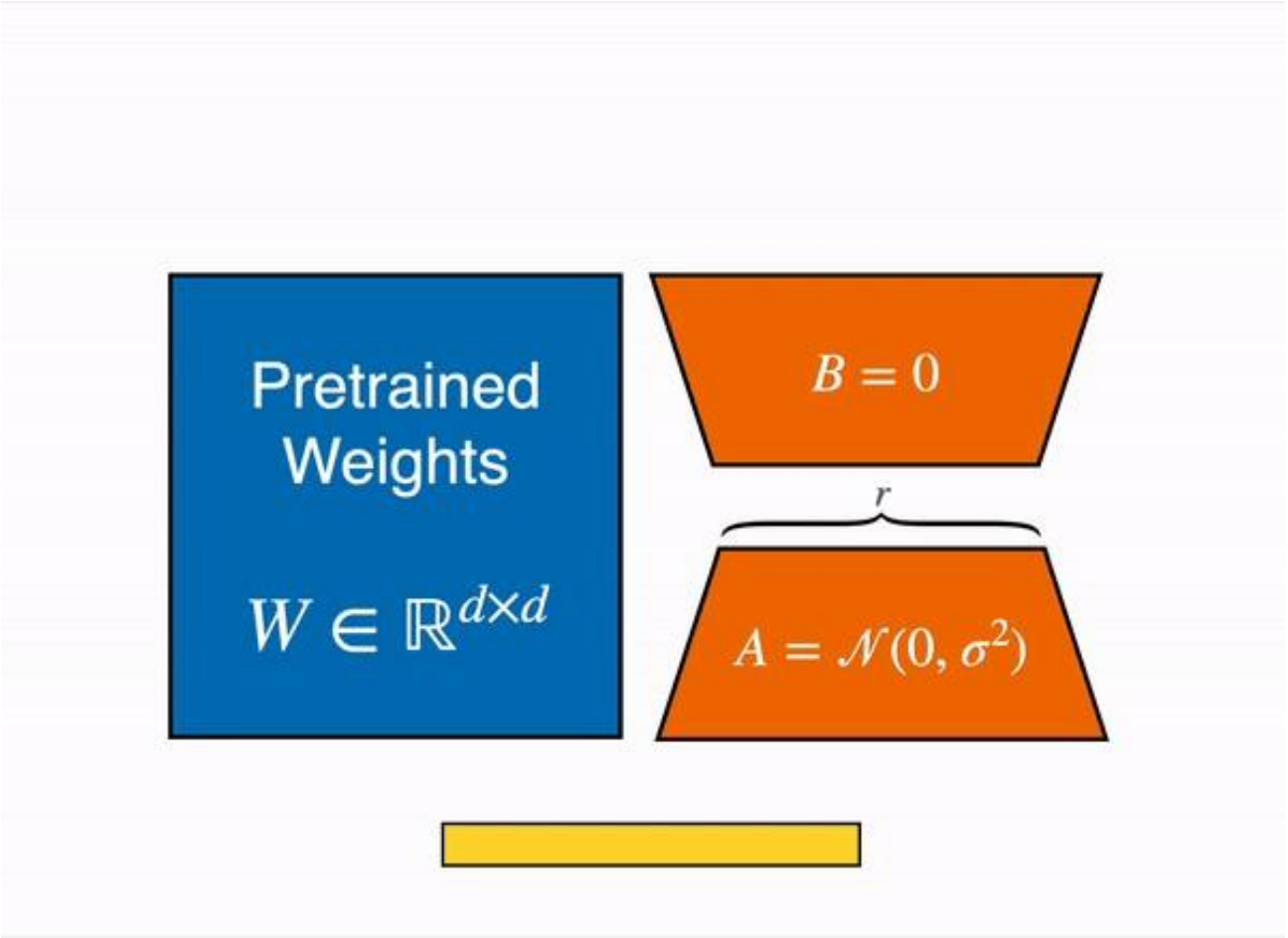
# Low rank approximation for Conv

| speedup | rank sel. | Conv1 | Conv2 | Conv3 | Conv4 | Conv5 | Conv6 | Conv7 | err. ↑ % |
|---|---|---|---|---|---|---|---|---|---|
| 2× | no | 32 | 110 | 199 | 219 | 219 | 219 | 219 | 1.18 |
| 2× | **yes** | 32 | 83 | 182 | 211 | 239 | 237 | 253 | **0.93** |
| 2.4× | no | 32 | 96 | 174 | 191 | 191 | 191 | 191 | 1.77 |
| 2.4× | **yes** | 32 | 74 | 162 | 187 | 207 | 205 | 219 | **1.35** |
| 3× | no | 32 | 77 | 139 | 153 | 153 | 153 | 153 | 2.56 |
| 3× | **yes** | 32 | 62 | 138 | 149 | 166 | 162 | 167 | **2.34** |
| 4× | no | 32 | 57 | 104 | 115 | 115 | 115 | 115 | 4.32 |
| 4× | **yes** | 32 | 50 | 112 | 114 | 122 | 117 | 119 | **4.20** |
| 5× | no | 32 | 46 | 83 | 92 | 92 | 92 | 92 | 6.53 |
| 5× | **yes** | 32 | 41 | 94 | 93 | 98 | 92 | 90 | **6.47** |

d' allocation at each layer

Zhang et al Efficient and Accurate Approximations of Nonlinear Convolutional Networks CVPR'15

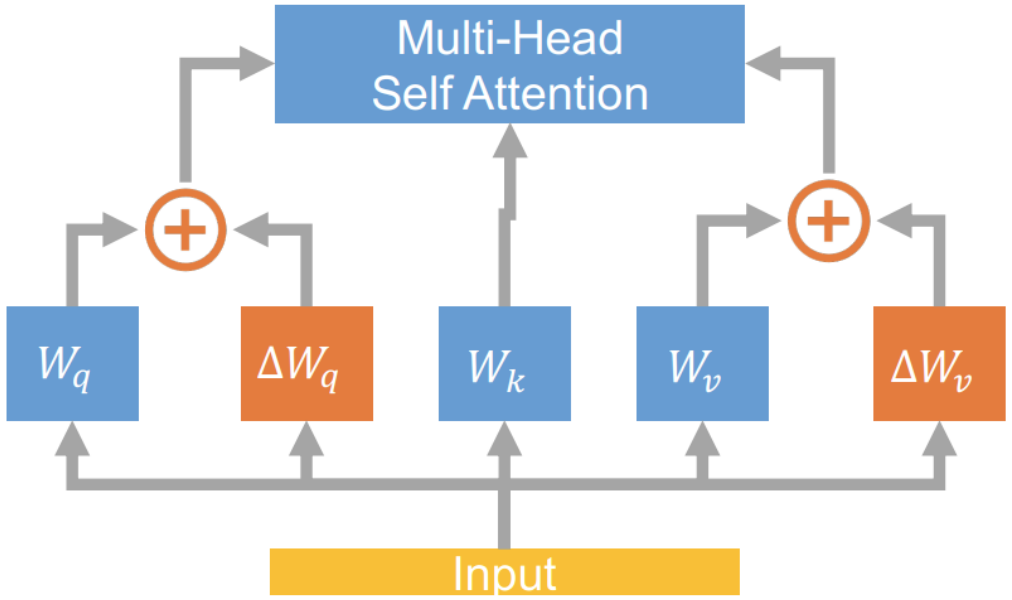
# Low-Rank Adaptation (LoRA) in LLMs



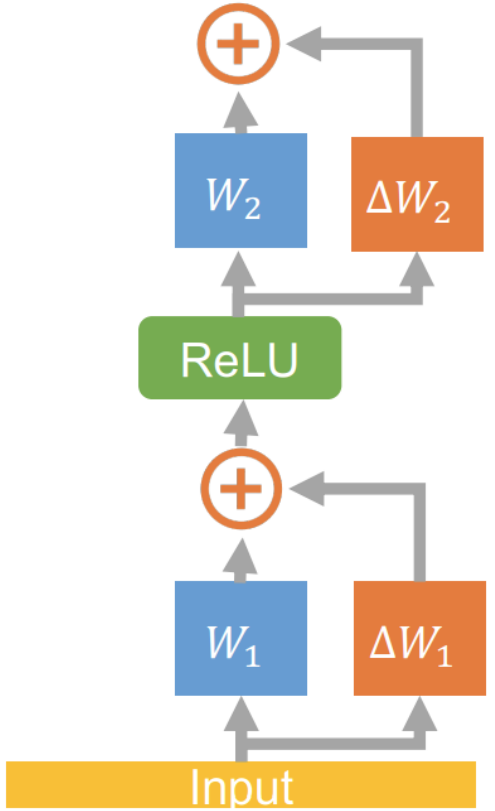

* LORA: Low-Rank Adaption of Large Language Models. Hu et al. 2021

# Low-Rank Adaptation (LoRA) in LLMs

- Apply LoRA to Attention

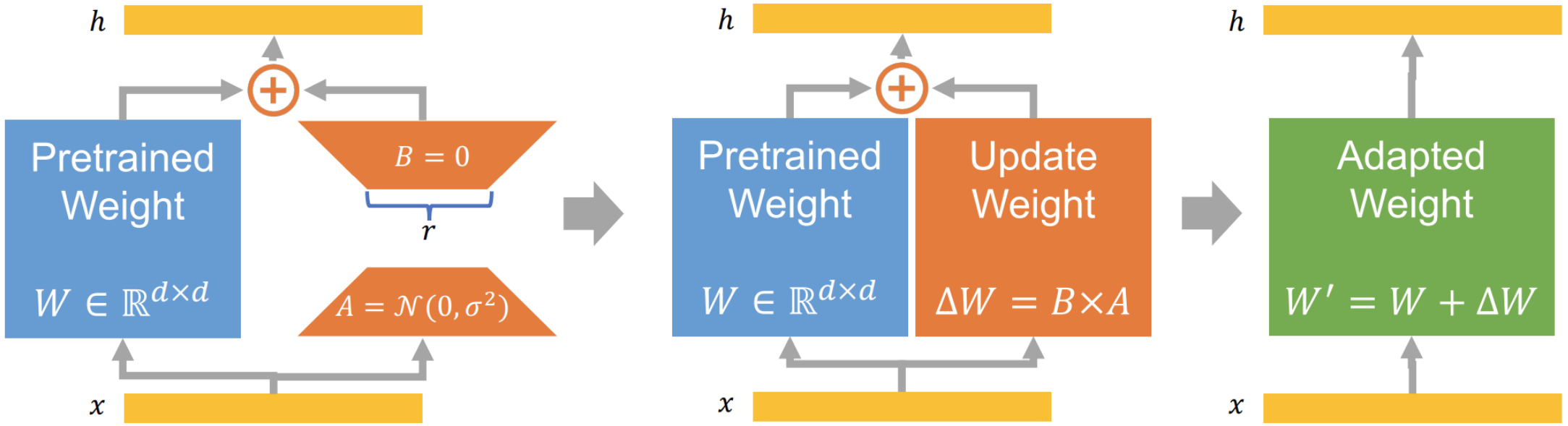



- Apply LoRA to MLP layer




* LORA: Low-Rank Adaption of Large Language Models. Hu et al. 2021

# LoRA Does Not Increase Inference Latency




* LORA: Low-Rank Adaption of Large Language Models. Hu et al. 2021

# Agenda

- Sparsity & Pruning
- SVD & Factorization based methods
- Quantization
- Distillation
- Cascade

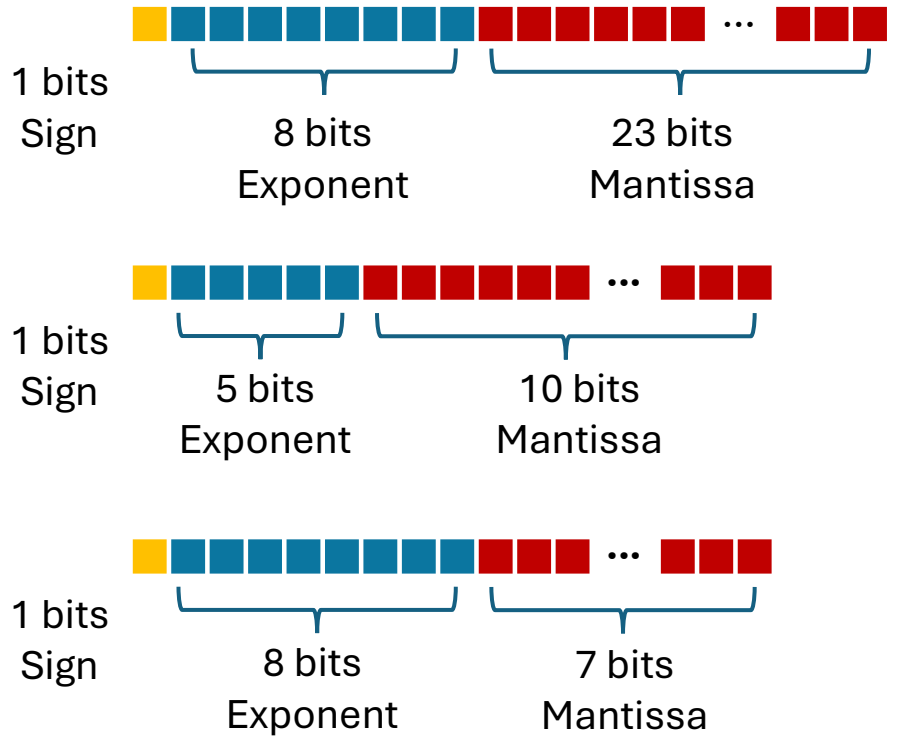
# Floating point representations

- Examples:

**Original value 0.0001**
FP16: 0.00010001659393 (Binary: 0|00001|1010001110, Hex: 068E)
BF16: 0.00010013580322 (Binary: 0|01110001|1010010, Hex: 38D2)

**Original value 1e-08**
FP16: 0.00000000000000 (Binary: 0|00000|0000000000, Hex: 0000)
BF16: 0.00000001001172 (Binary: 0|01100100|0101100, Hex: 322C)

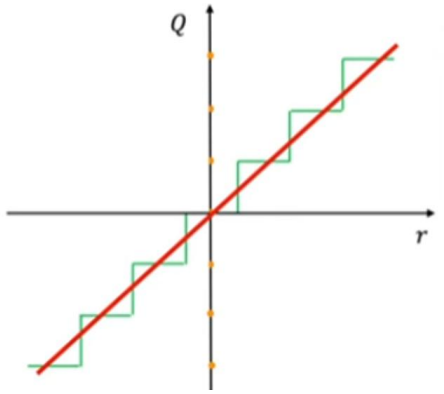
**Original value 100000.00001**
FP16: inf (Binary: 0|11111|0000000000, Hex: 7C00)
BF16: 99840.0000000000000 (Binary: 0|10001111|1000011, Hex: 47C3)

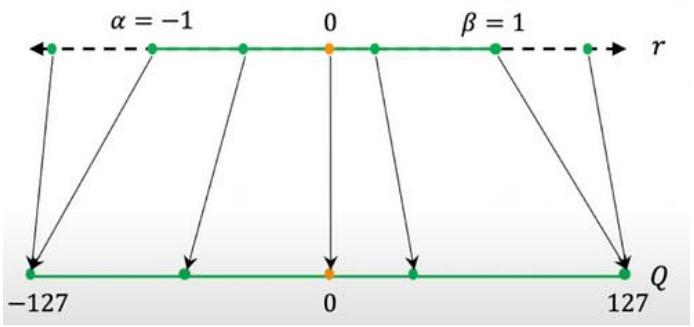



BF16 provides a wider range at a cost of some precision → balance between range & numerical stability

# Quantization

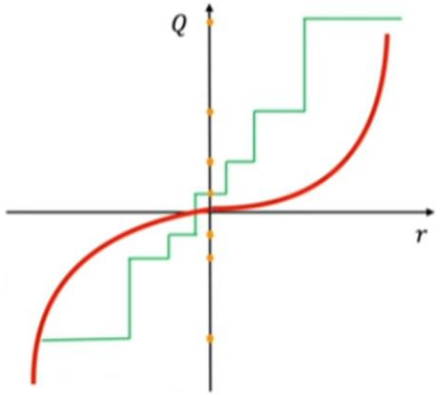
- Uniform, symmetric inputs

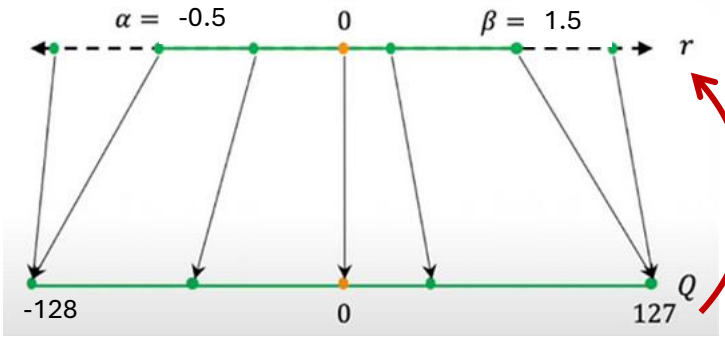



$$Q = round\left(\frac{r}{s}\right)$$

- Non-uniform, asymmetric inputs

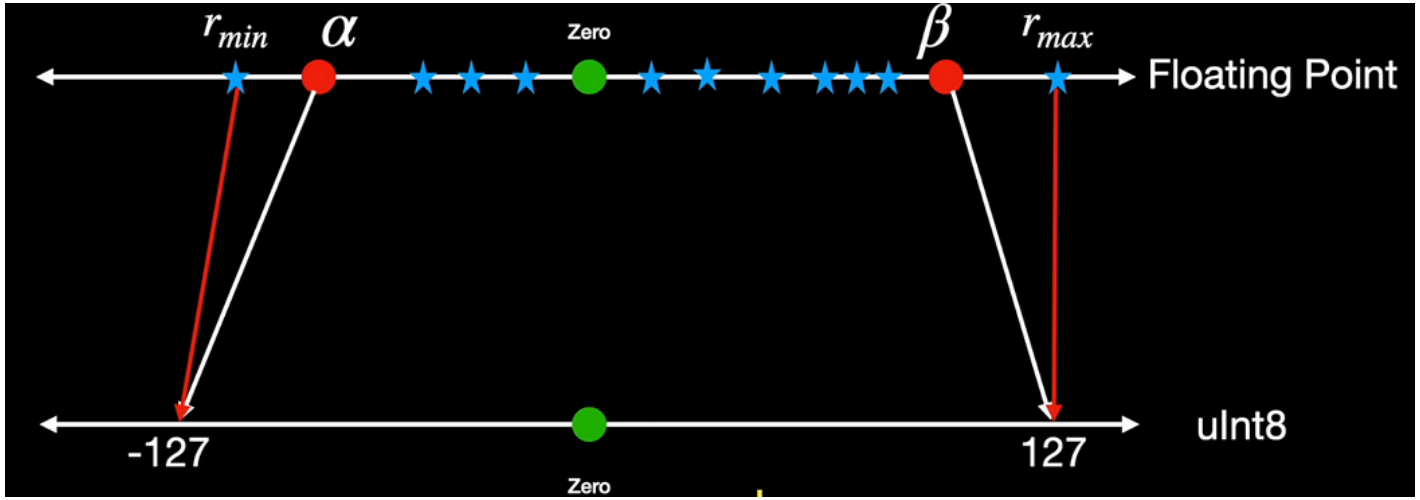



$$Q = round\left(\frac{r}{s}\right)+Z$$

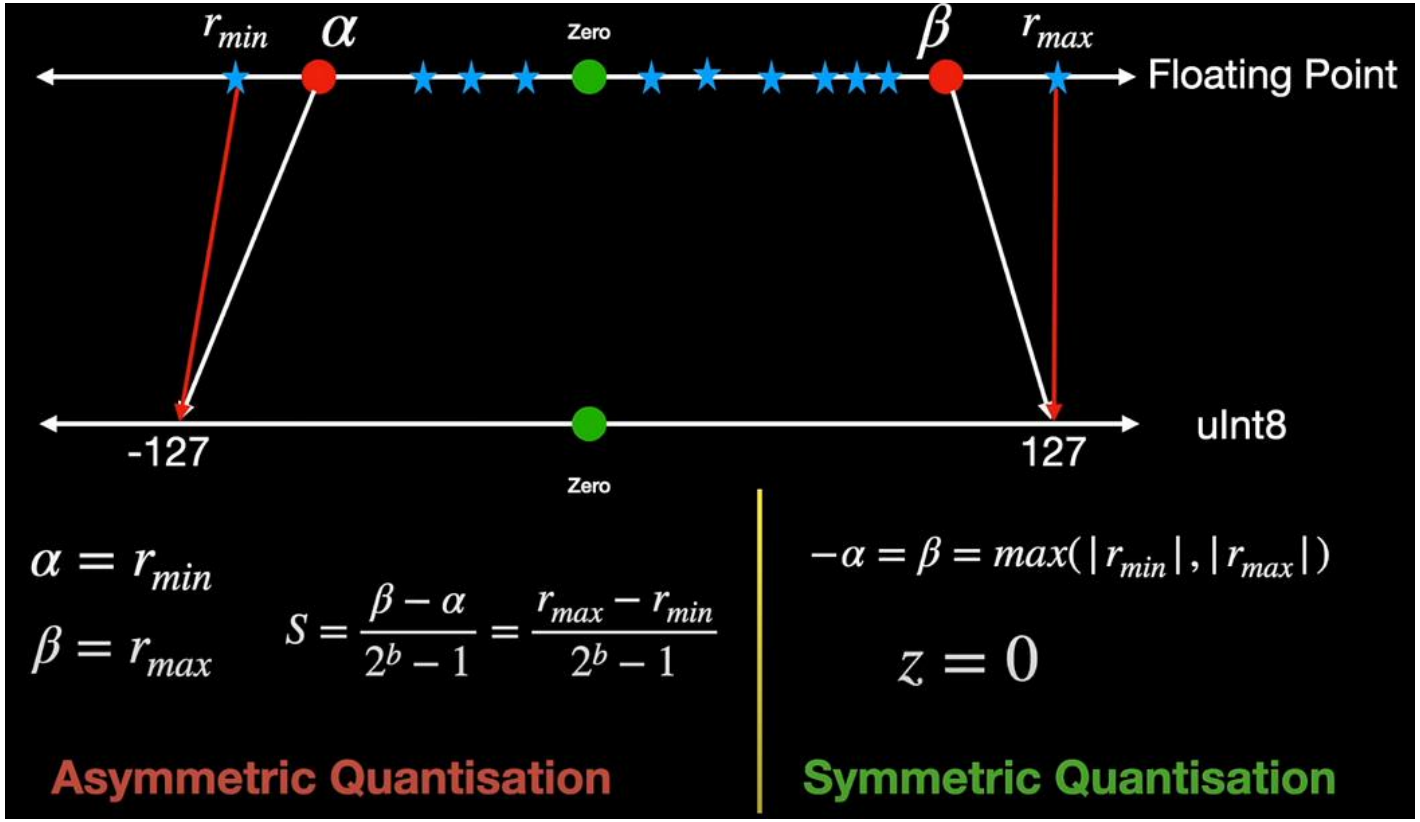
Dequantization $\tilde{r} = S(Q + Z)$,
Perplexity, error $\epsilon = \tilde{r} - \mathrm{r}$

# Calibration: choosing scale and zero factor

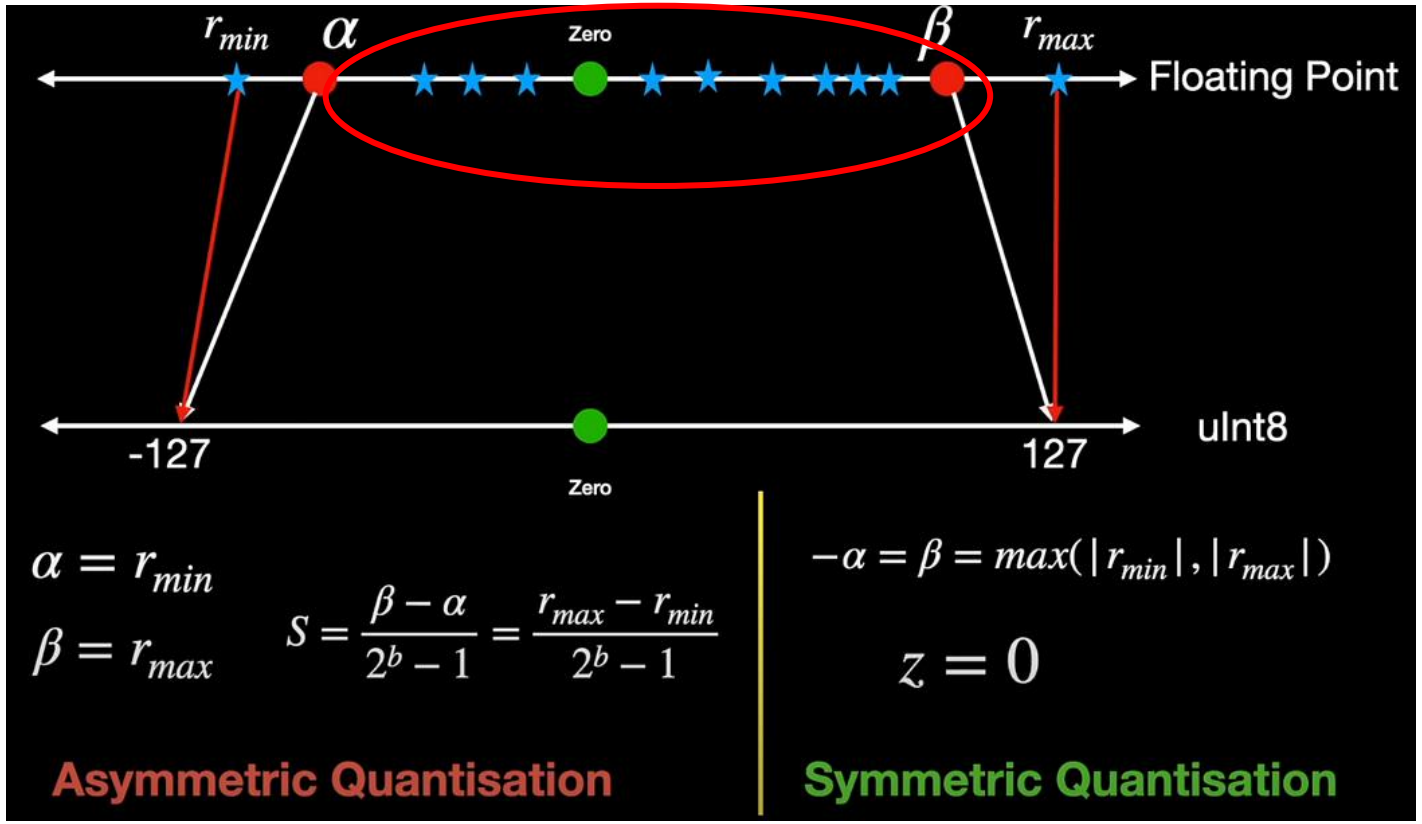



https://medium.com/@AIBites/model-quantization-in-deep-neural-networks-81df49f3c7d8

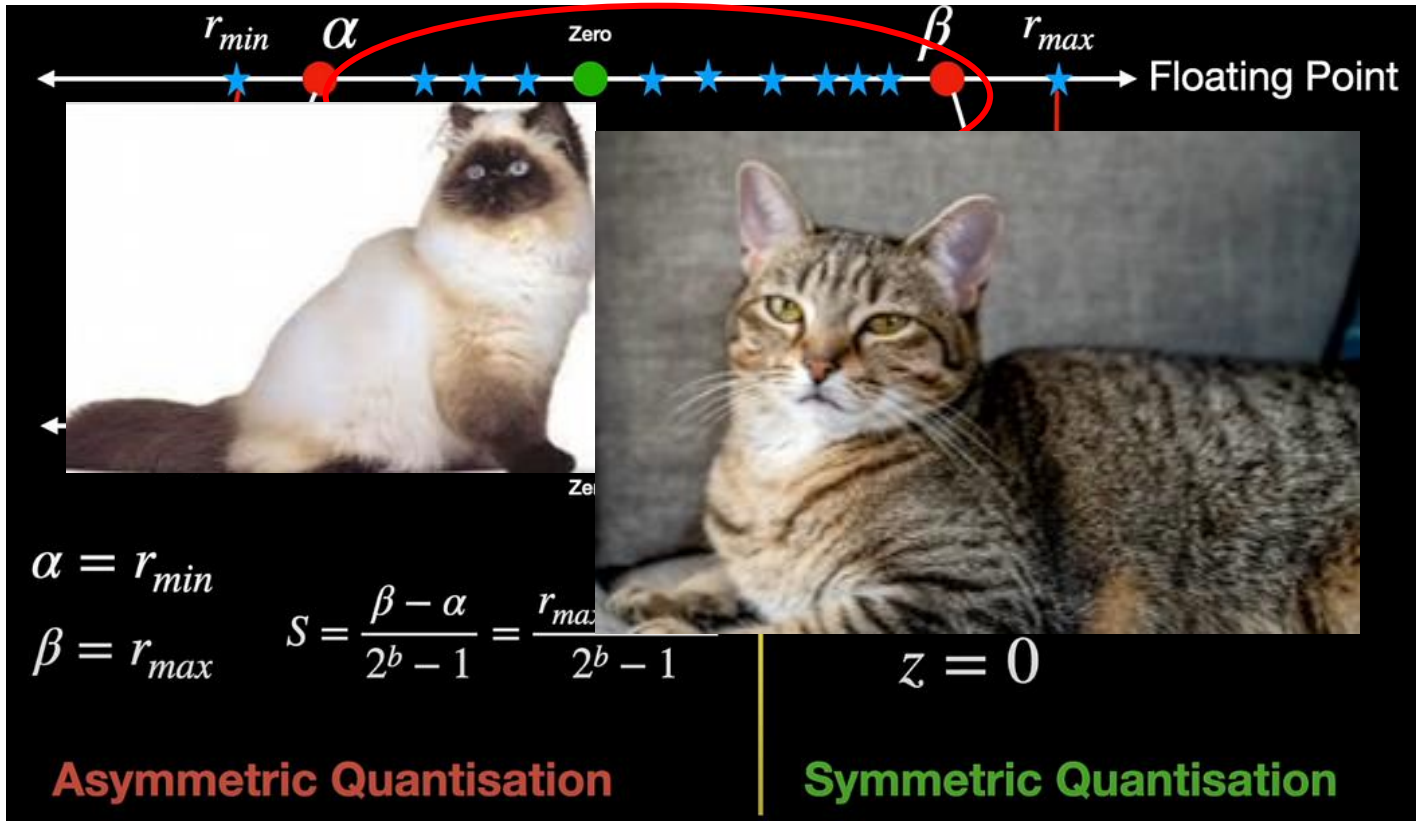
# Calibration: choosing scale and zero factor



# Calibration: rectifying skews



# Calibration: rectifying skews




**When & How to calibrate?**

During or after training?

Data skew is unknown a priori

# Quantization modes

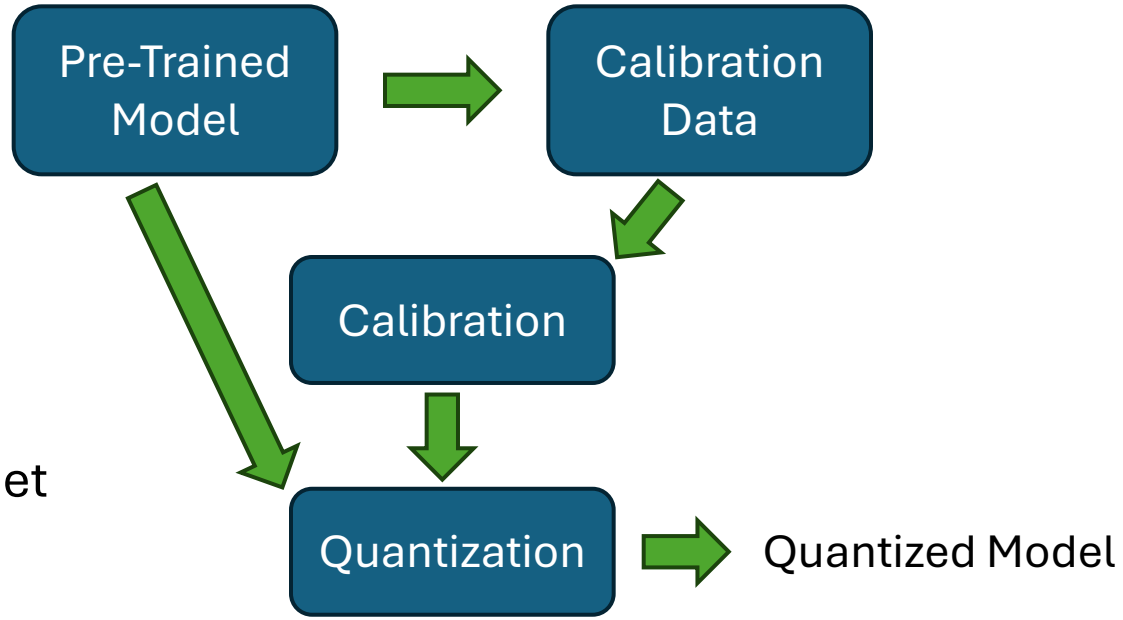
- Post Training Quantization (PTQ)
  - **Weight-only quantization:**

    Inflate model weights during computation

    May not need calibration dataset
  - **Full quantization:**

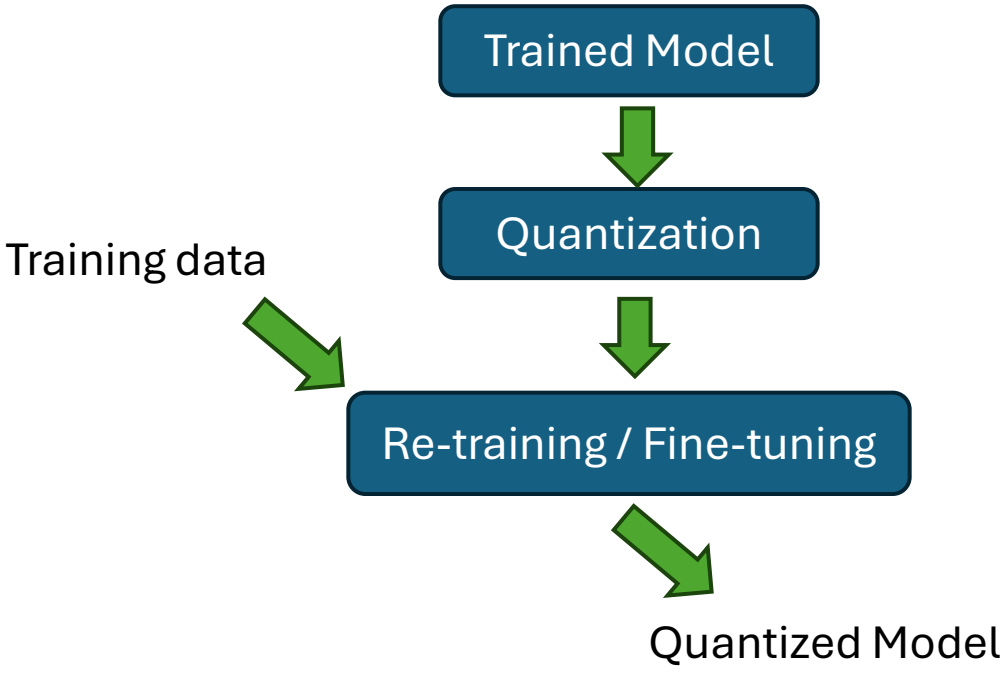
    Weights + activation,  need calibration dataset

    Calibrations include:
    - Output bias caused by quantization, add up to the final output
    - Weights, based on mean and variance before/after quantization



# Quantization modes

- Post Training Quantization (PTQ)




- Quantization Aware Training (QAT)
- Quantization Aware Fine-Tuning (QAF)
  - **Challenge:** quantization is not differentiable.
  - **Solution:**

    Insert fake quantization operators in the graph to compute statistics of the inputs

    Once the model is trained, update weights and remove the fake operators

# Quantization targets

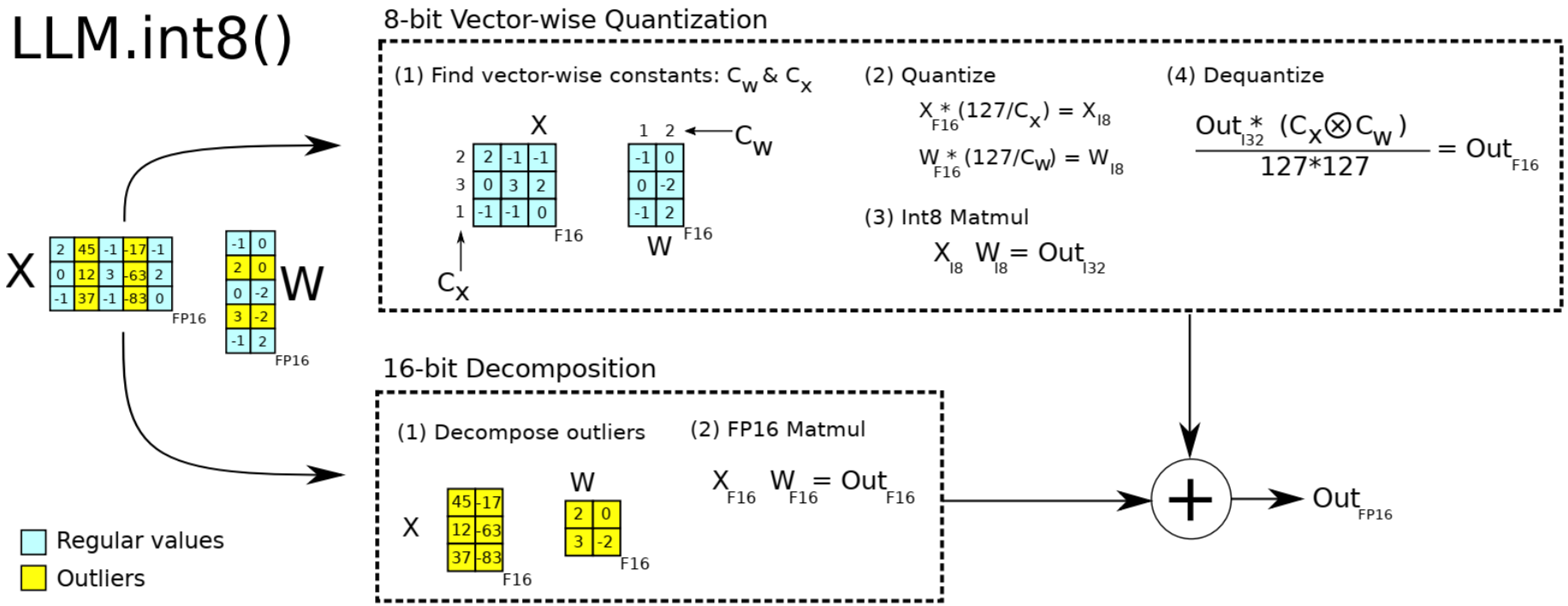
a) Weights (W): reduce model sizes and memory footprint

b) Activation (X): reduce memory footprint and improve speed

c) KV cache: improve throughput

d) Gradients: training only – reduce networking costs

# Weight-only quantization: LLM.int8()

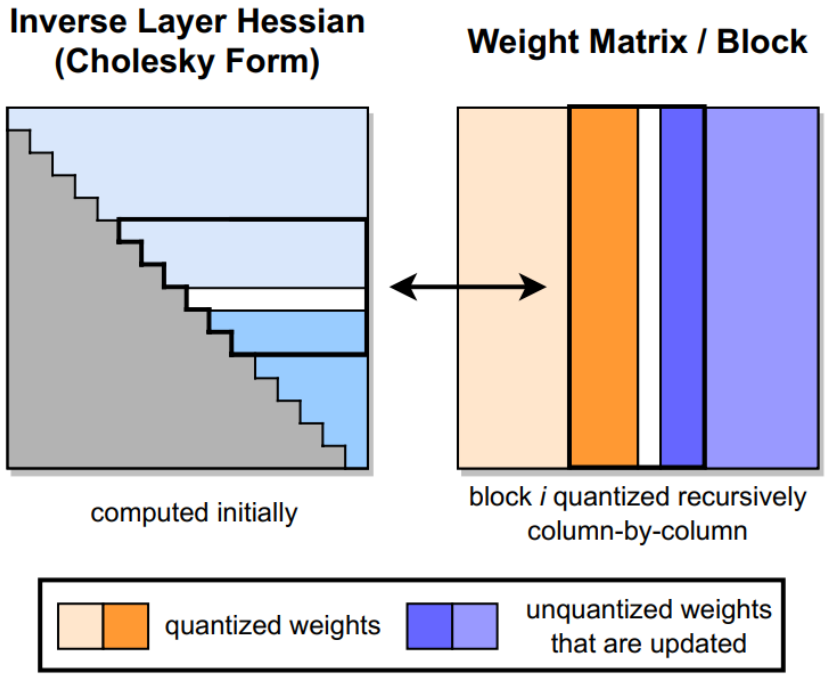



- Decompose the matrix
- Use 8bit quantization for the majority, 16bit for outliers

LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale. 2022

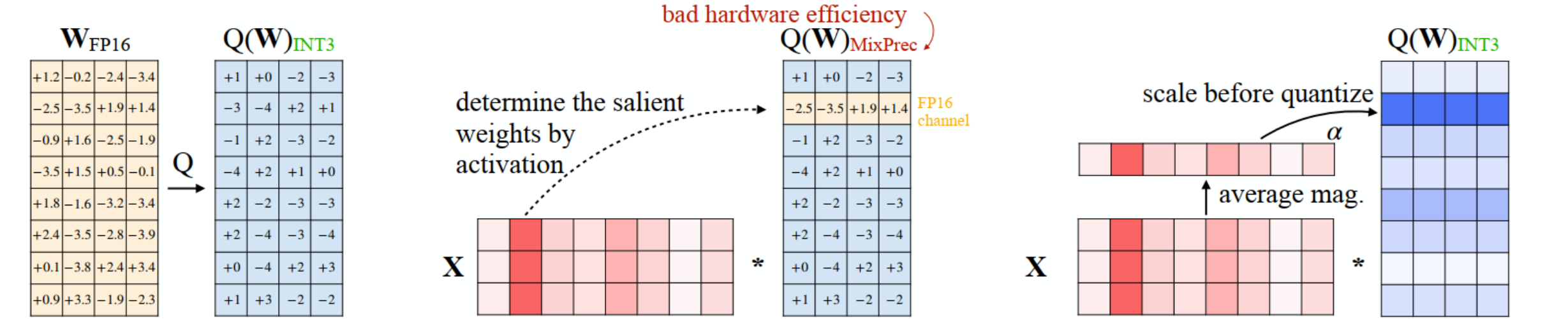
# Weight-only quantization: GPTQ

- Need calibration data
- Recursive process to
  - Quantize a block
  - Update the remaining weights to recover accuracy lost




GPTQ: Accurate Post-Training Quantization for Generative Pre-Trained Transformers. 2022

# Weight-only quantization: AWQ




(a) RTN quantization (**PPL 43.2**)
(Round To Nearest)

(b) Keep 1% salient weights in FP16 (**PPL 13.0**)

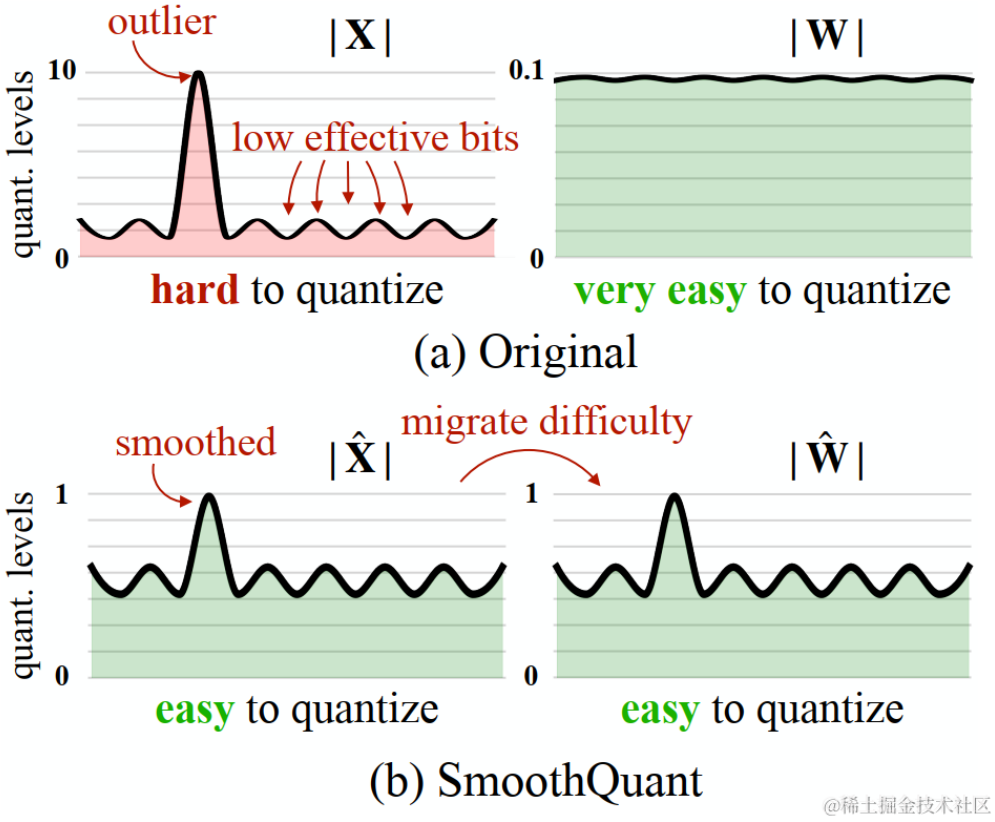
(c) Scale the weights before quantization (**PPL 13.0**)

- Keep salient weights (by observing the activation distribution) in FP16 can greatly reduce quantization error
- Use per-channel scaling

AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration. 2023

# Full quantization: SmoothQuant

- Activations are harder to quantize
- Propose to smooth activations by transformation on the weights
- Use per-token and per-channel quantization






Accurate and Efficient Post-Training Quantization for Large Language Models. 2022

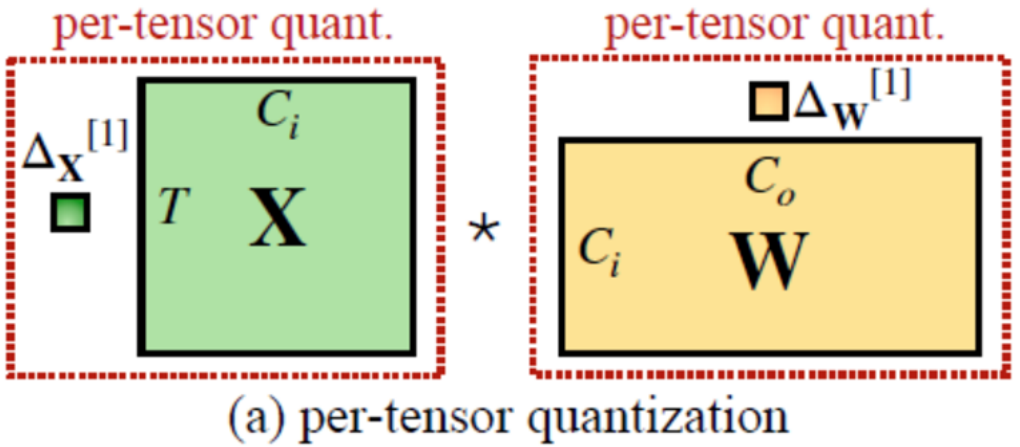
# Quantization granularity

a) **Per-tensor:** whole layer of input matrices

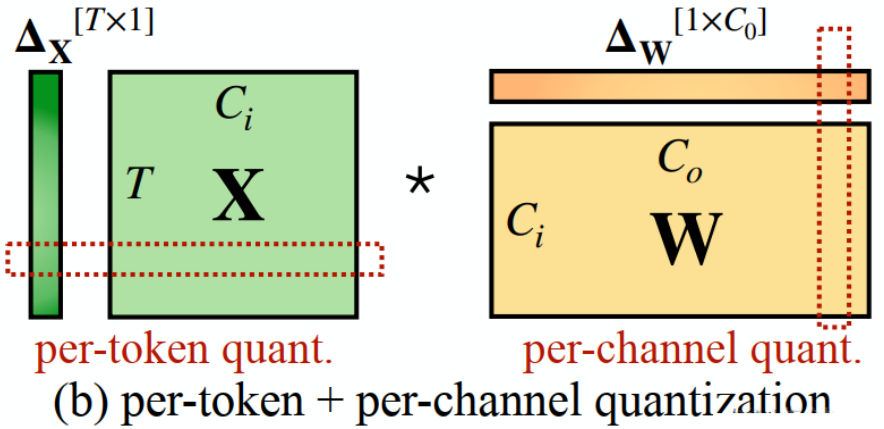
b) **Per-token & per-channel:** slices of input matrices

c) **Per-group:** combination of above

(b), (c) result in mixed-precision quantization schemes.

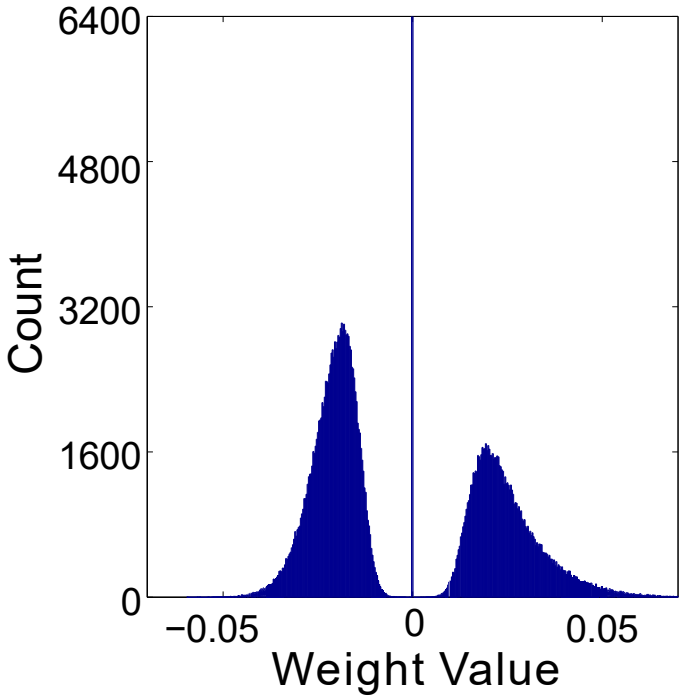


(a) per-tensor quantization

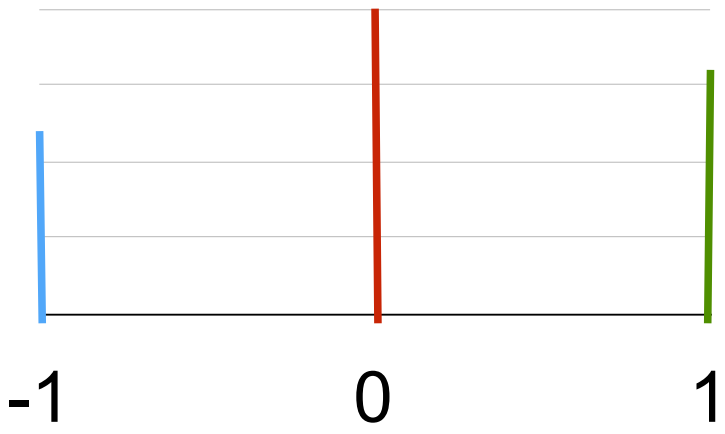


(b) per-token + per-channel quantization

# Binary / Ternary Network

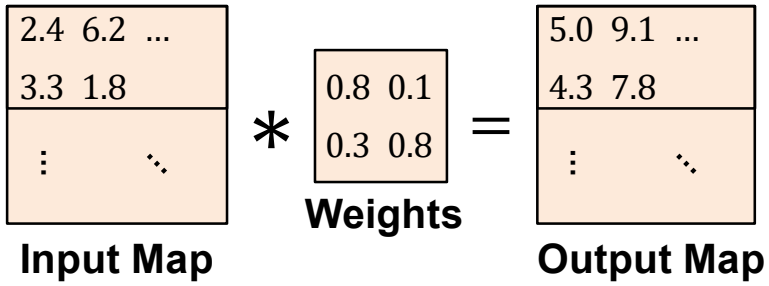



=>



# Binary / Ternary Network

**CNN**

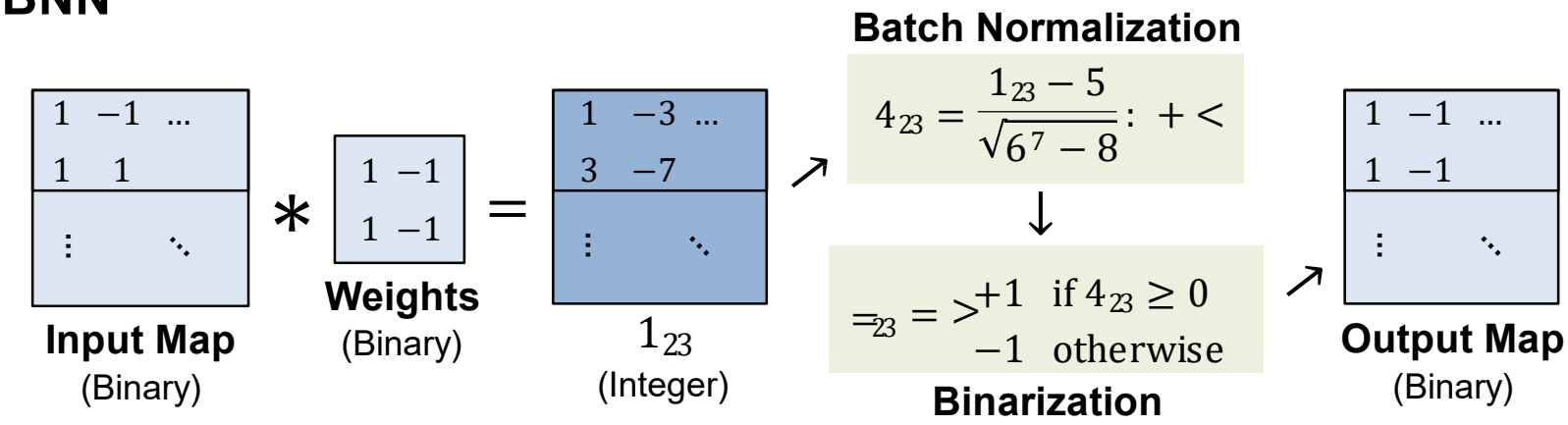



## Key Differences

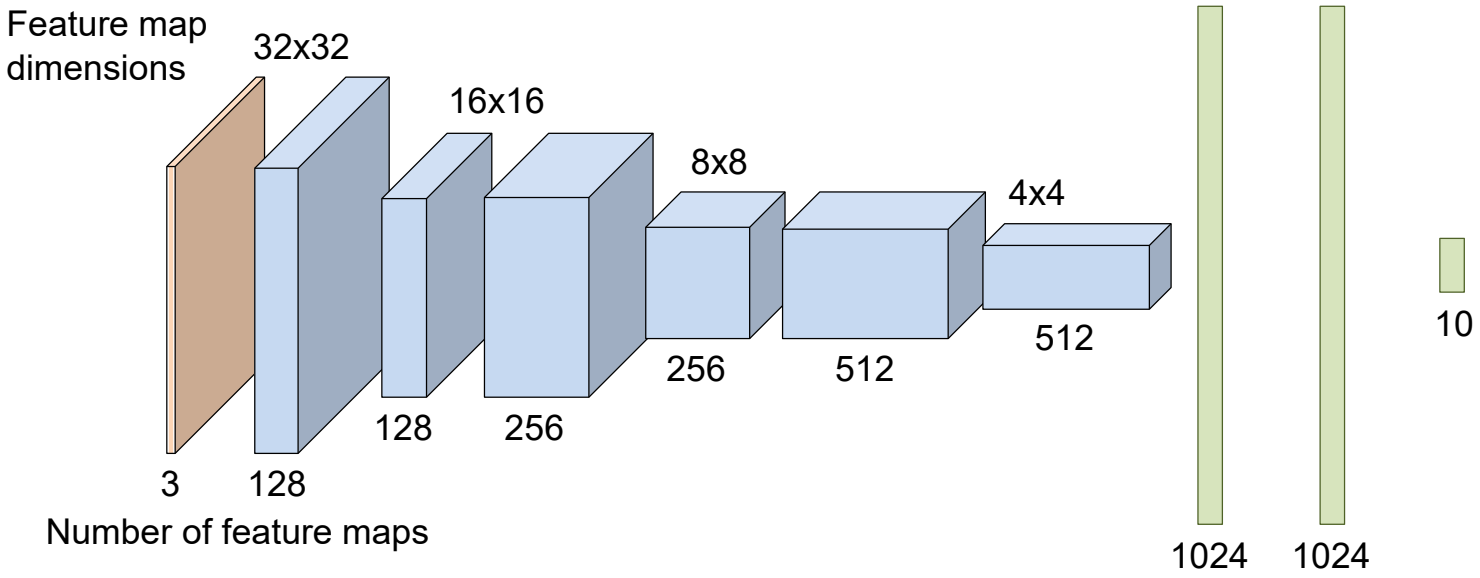
1. Inputs are binarized (−1 or +1)
2. Weights are binarized (−1 or +1)
3. Results are binarized after **batch normalization**

---

**BNN**



# BNN CIFAR-10 architecture




- 6 conv layers, 3 dense layers, 3 max pooling layers
- All conv filters are 3x3
- First conv layer takes in floating-point input
- **13.4 Mbits total model size** (after hardware optimizations)

[2] M. Courbariaux et al. **Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1**. *arXiv:1602.02830*, Feb 2016.

# BNN advantages

## 1. Floating point ops replaced with binary logic ops

| $b_1$ | $b_2$ | $b_1 \times b_2$ |
|---|---|---|
| $+1$ | $+1$ | $+1$ |
| $+1$ | $-1$ | $-1$ |
| $-1$ | $+1$ | $-1$ |
| $-1$ | $-1$ | $+1$ |

| $b_1$ | $b_2$ | $b_1$ XOR $b_2$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

- Encode {+1,−1} as {0,1}, multiplies become XORs
- Conv/dense layers do dot products, XOR and popcount
- Hardware accelerations

## 2. Binarized weights may reduce total model size

- Fewer bits per weight may be offset by having more weights

# BNN vs CNN: parameter efficiency

| Architecture | Depth | Param Bits (Float) | Param Bits (Fixed-Point) | Error Rate (%) |
|---|---|---|---|---|
| ResNet [3] (CIFAR-10) | 164 | 51.9M | 13.0M* | 11.26 |
| BNN [2] | 9 | - | 13.4M | 11.40 |

* Assuming each float param can be quantized to 8-bit fixed-point

- **Comparison:**
  - Conservative assumption: ResNet can use 8-bit weights
  - BNN is based on VGG (less advanced architecture)
  - BNN seems to hold promise!

[2] M. Courbariaux et al. **Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1**. *arXiv:1602.02830*, Feb 2016.
[3] K. He, X. Zhang, S. Ren, and J. Sun. **Identity Mappings in Deep Residual Networks.** *ECCV 2016.*

# Agenda

- Sparsity & Pruning
- SVD & Factorization based methods
- Quantization
- Distillation
- Cascade

# Distillation: introduction

Knowledge distillation (KD) is a model compression method in which a small model is trained to mimic a pre-trained, larger model (or ensemble of models).

- The method was first proposed by[1] then generalized by[2].
- This training setting is sometimes referred to as "teacher-student", where the large model is the teacher and the small model is the student.
- In distillation, knowledge is transferred from the teacher model to the student by minimizing a loss function in which the target is the distribution of class probabilities predicted by the teacher model.
- Specifically, KD is accomplished by minimzing the KL divergence between the predictions of teacher and student.

[1] Cristian Buciluǎ , Rich Caruana, and Alexandru Niculescu-Mizil (2006). “Model compression”. In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 535–541.

[2] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean (2015). “Distilling the knowledge in a neural network”.

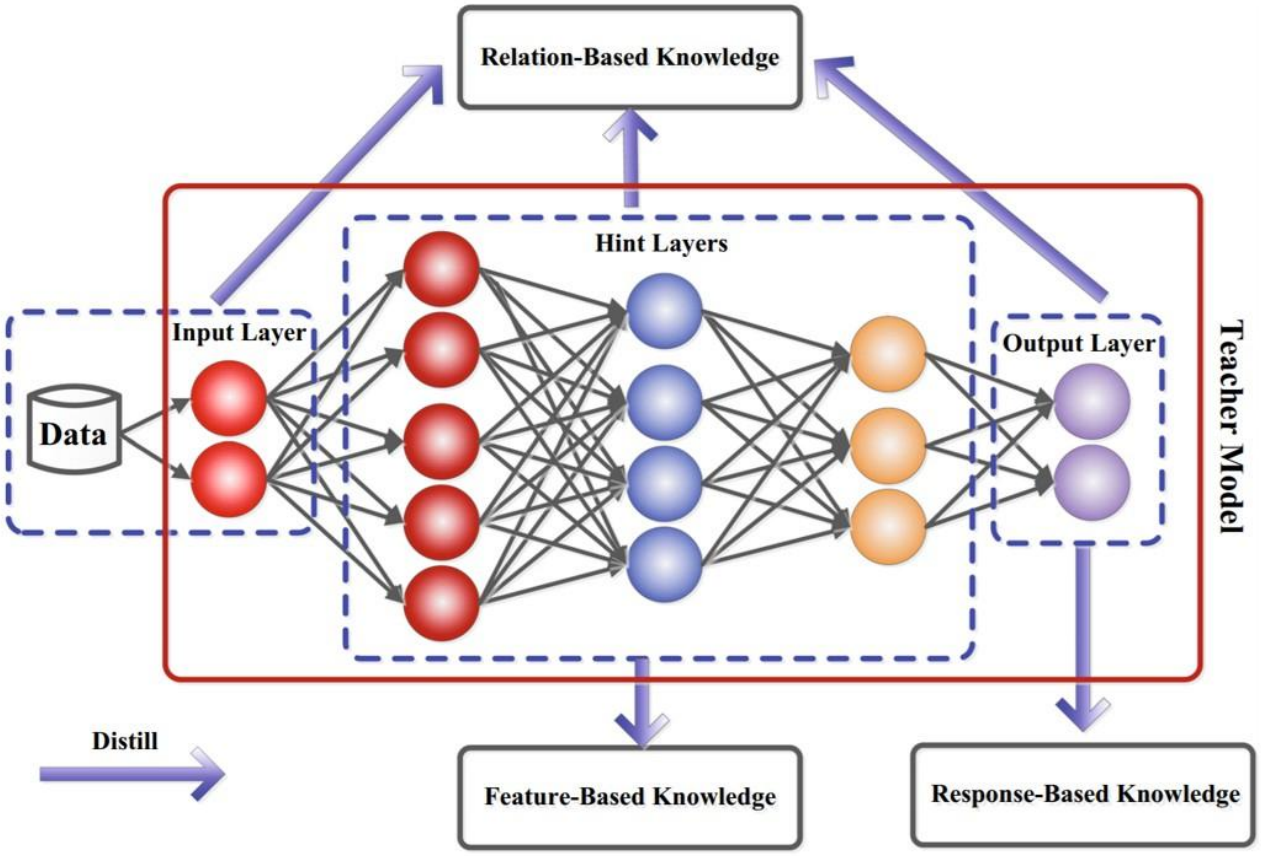
# KD loss

We define a teacher network $T(.)$ and a student network $S(.)$. Typically, networks can produce class probabilities that converts the logits $z_i$, computed for each class into a probability, $q_i$ by using softmax function. We define the class probabilities $p_i$ which is generated by teacher network $T(.)$ and $q_i$ which is generated by student network $S(.)$ correspondingly. Then the KD Loss can be given by:

$$\mathcal{L}_{KD} = -\sum_{i=1}^{N} p(x_i) \log(q(x_i))$$

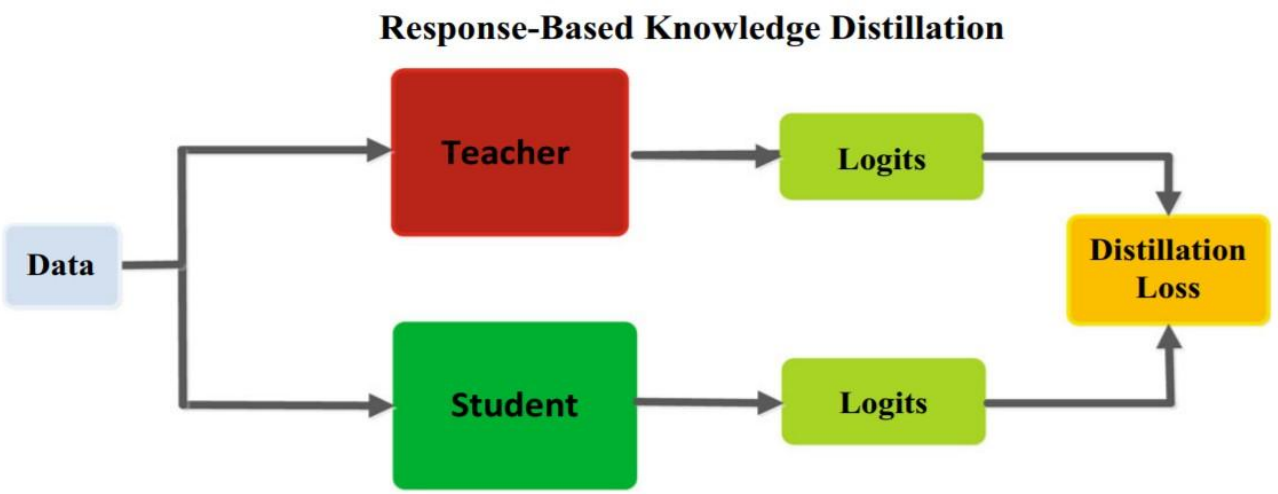
where $p(x_i) = \frac{\exp(v_i/\tau)}{\sum_{j=1}^{C-1} \exp(v_j/\tau)}$ and $q(x_i) = \frac{\exp(z_i/\tau)}{\sum_{j=1}^{C-1} \exp(z_j/\tau)}$. Here $C$ denotes the number of classes, $\tau$ iss the temperature parameter. $v$ and $z$ indicates the logits generated by teacher network and student network respectively.

# Knowledge modeling



# Response-Based Knowledge

- Response-based knowledge usually refers to the neural response of the last output layer of the teacher model. The main idea is to directly mimic the final prediction of the teacher model.

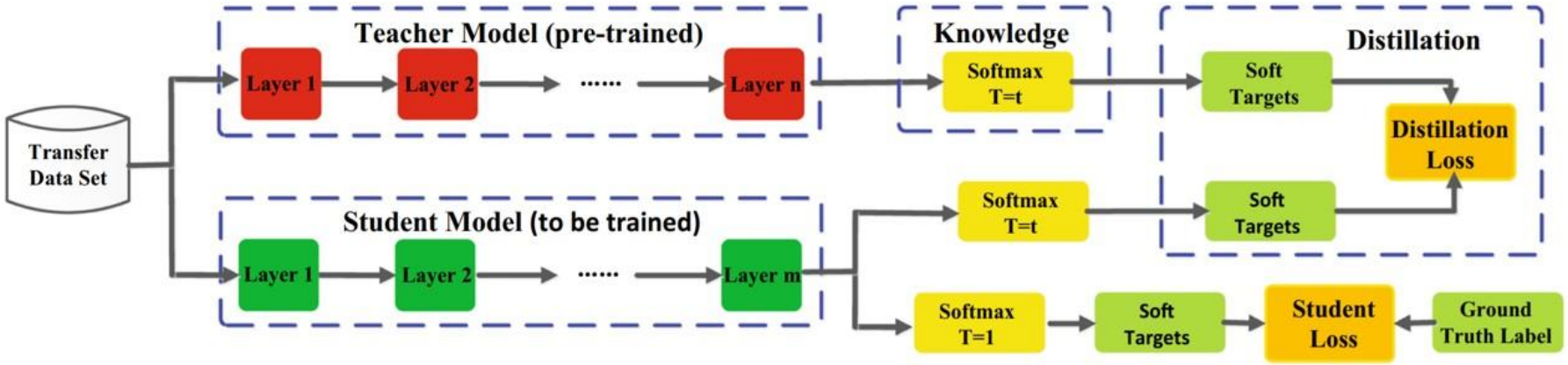



Given a vector of logits $z$ as the outputs of the last fully connected layer of a deep model, the distillation loss for response-based knowledge can be formulated as:

$$L_{ResD}(z_t, z_s) = L_R(z_t, z_s)$$

where $L_R(\cdot)$ indicates the divergence loss of logits, and $z_t$ and $z_s$ are logits of teacher and student respectively.

# Response-Based Knowledge




Knowledge distillation proposed by[4].

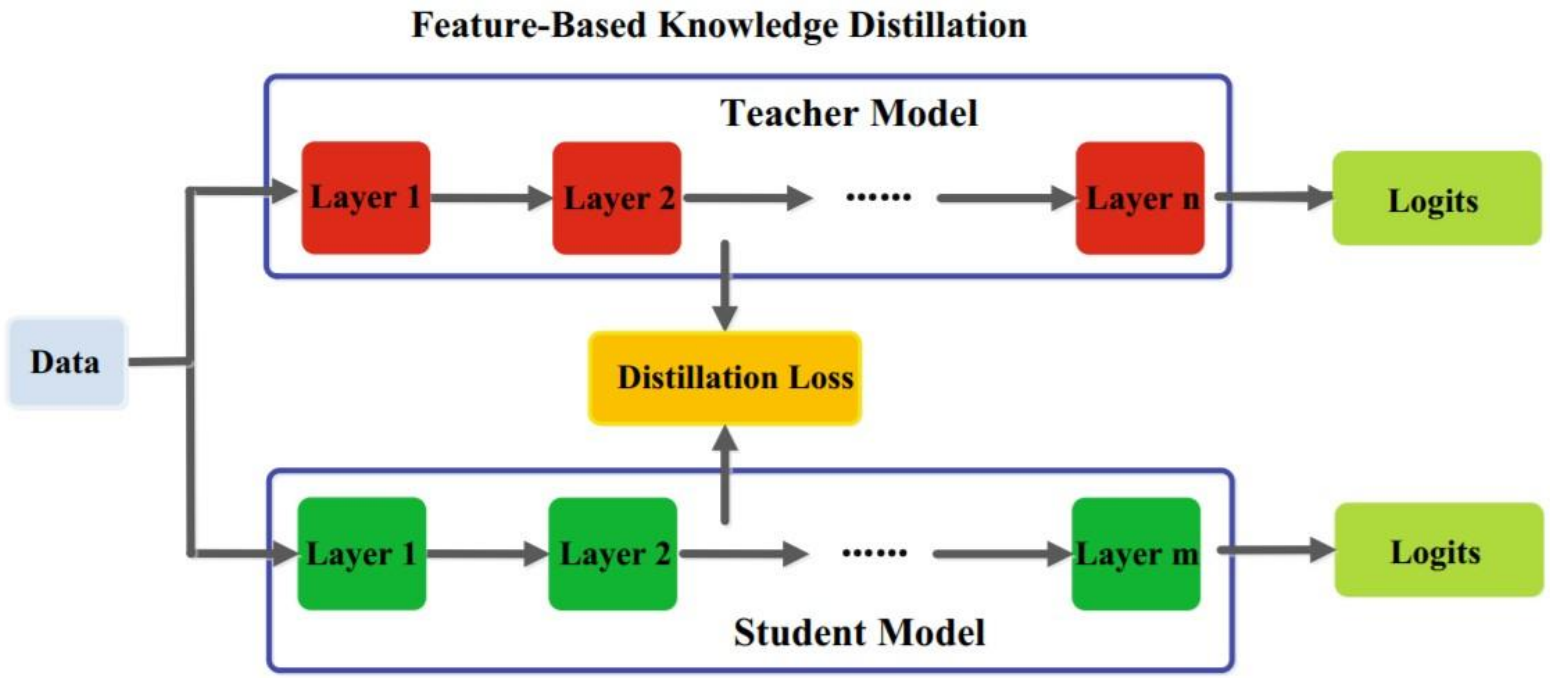
The KD loss for soft logits: $L_{ResD}(p(z_t,T),p(z_s,T)) = L_R(p(z_t,T),p(z_s,T))$,

where $p(z_t,T) = \frac{\exp\left(\frac{z_i}{T}\right)}{\sum_j \exp\left(\frac{z_j}{T}\right)}$ is the softmax and $T$ is a temperature factor to control the importance of each soft target.

[4]Geoffrey Hinton, Oriol Vinyals, and Jeff Dean (2015). "Distilling the knowledge in a neural network".

# Feature-Based Knowledge

The output of intermediate layers, *i.e.*, feature maps, can also be used as the knowledge to supervise the training of the student model, which forged feature-based knowledge distillation. It is the improvement of response-based knowledge distillation.



# Feature-Based Knowledge

The distillation loss for feature-based knowledge transfer can be formulated as

$$L_{FeaD}(f_t(x), f_s(x)) = L_F(\varphi_t(f_t(x)), \varphi_s(f_s(x)))$$

where $f_t(x)$ and $f_s(x)$ are the feature maps of the intermediate layers of teacher and student models, respectively. The transformation functions, $\varphi_t(f_t(x)\ and\ \varphi_s(f_s(x))$, are usually applied when the feature maps of teacher and student models are not in the same shape. $L_F(\cdot)$ indicates the similarity function used to match the feature maps of teacher and student models.$L_F(\cdot)$ can be $L_2(\cdot)$, $L_1(\cdot), L_{CE}(\cdot)$ and *etc.*

# Feature-Based Knowledge

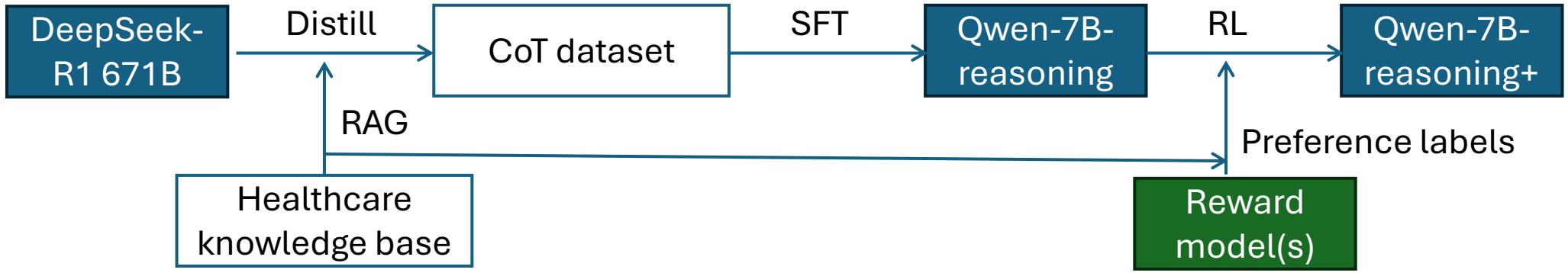
The work[6] proposed a feature-based knowledge distillation using attention mechanism. Specifically, the student network learns attention information from teacher network.


[6]Sergey Zagoruyko and Nikos Komodakis (2016). "Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer". In: *arXiv preprint arXiv:1612.03928*.

# LLM-based distillation

- The same idea can be applied to LLMs as well, and, sometimes even more easily.

- Example task: give me a 7B reasoning model specialized in healthcare
  - DeepSeek-R1 671B too big & expensive
  - I don't care about astrophysics or Shakespeare

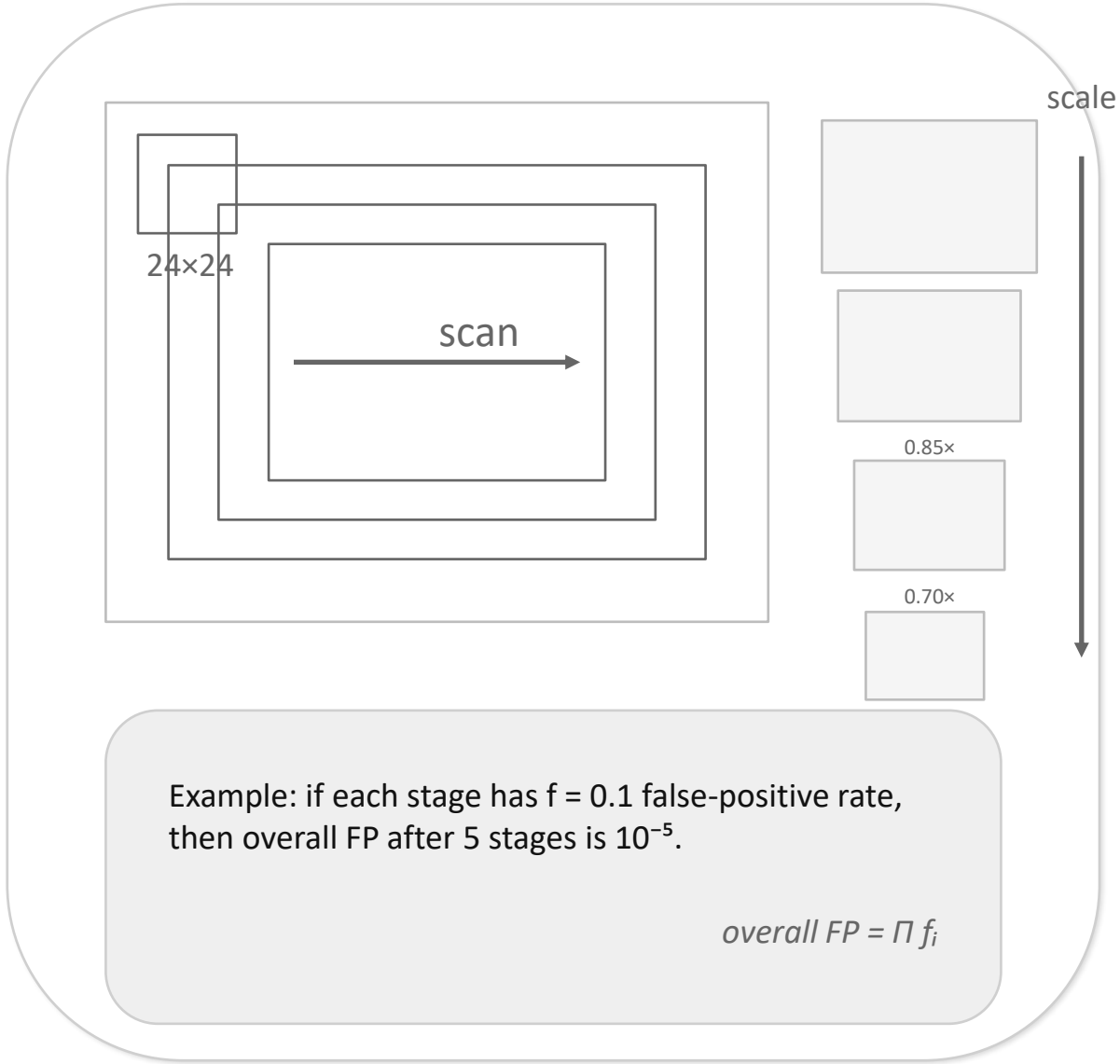


# Agenda

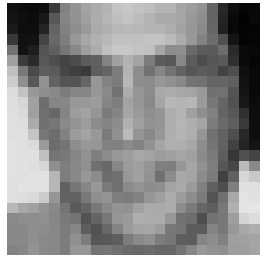
- Sparsity & Pruning
- SVD & Factorization based methods
- Quantization
- Distillation
- Cascade

# Object detection as exhaustive search

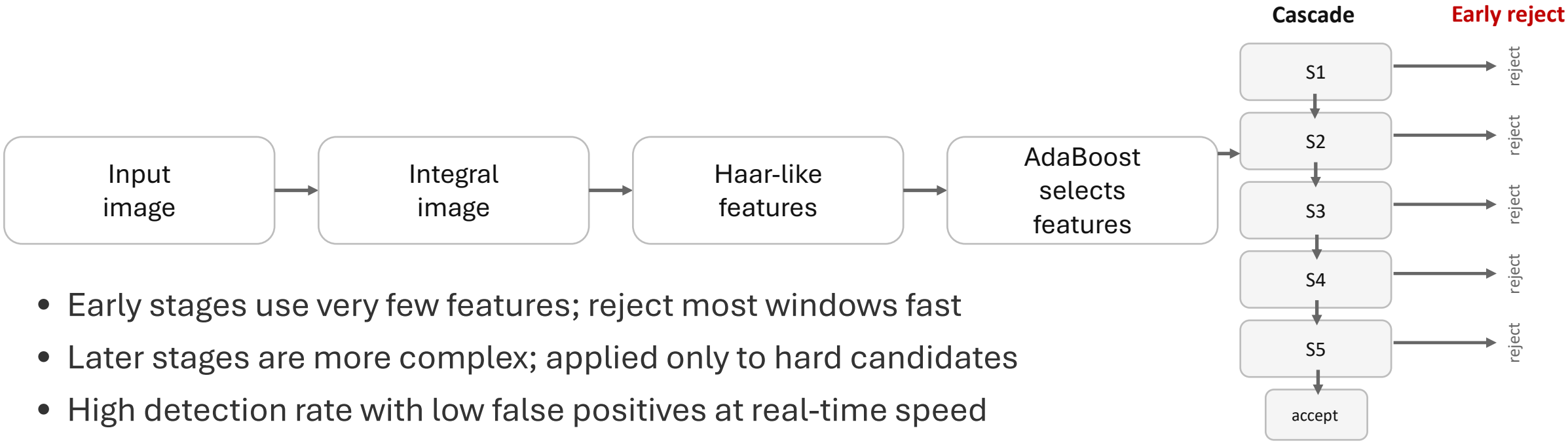
- Slide a fixed window over an image pyramid (scale space)
- Millions of candidate windows per frame → cost must be O(1) per feature
- False positives compound across stages: overall FP = $\Pi\ f_i$



# Model cascade: Viola–Jones Face Detection (2001)



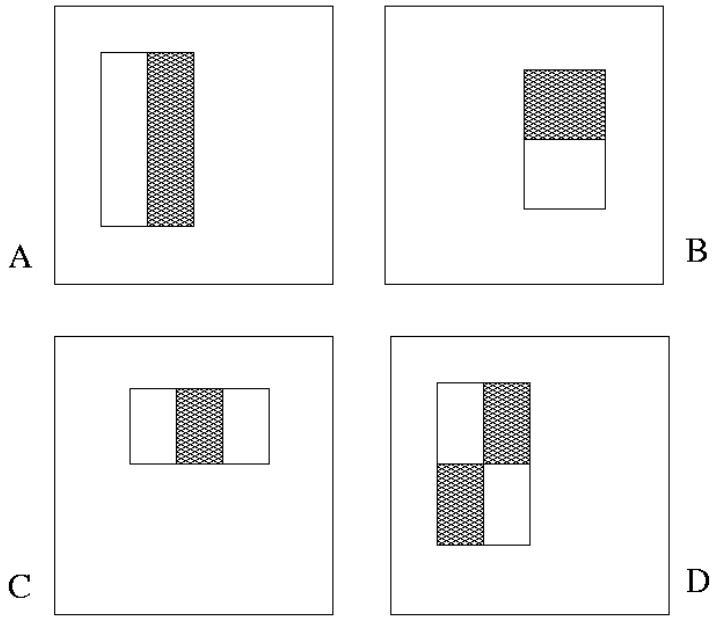
- Real-time detection by scanning a 24×24 window over many scales/positions
- Make per-window computation extremely cheap
- Key ingredients: integral image, Haar features, AdaBoost selection, cascade




- Early stages use very few features; reject most windows fast
- Later stages are more complex; applied only to hard candidates
- High detection rate with low false positives at real-time speed

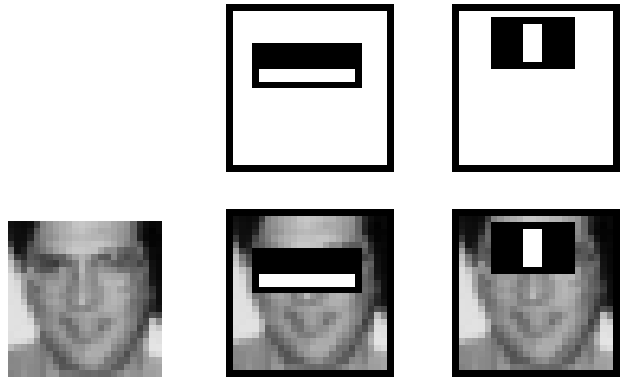
# Haar-like features via integral image

- Rectangle features approximate intensity differences (edges / lines)
- Integral image gives any rectangle sum in O(1)
- Evaluate a feature using 4 corner lookups



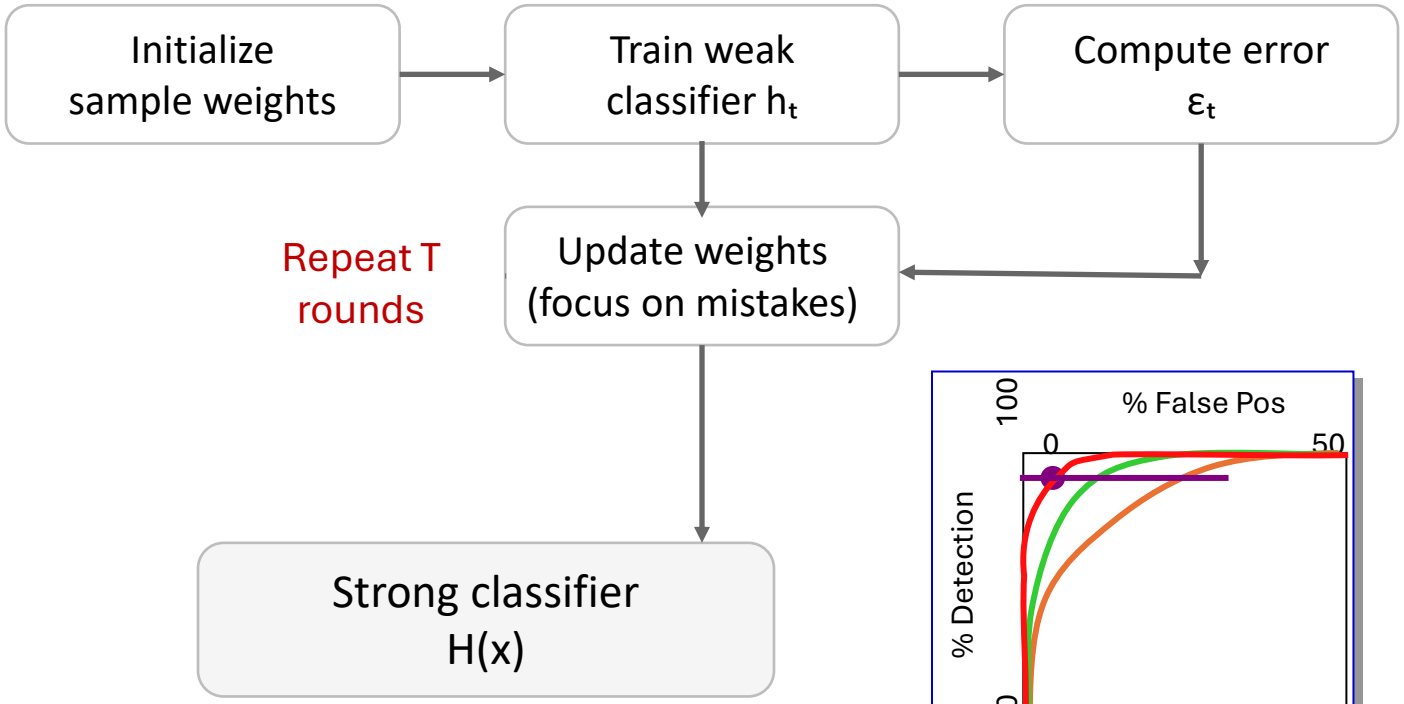

$$h_t(x_i) = \begin{cases} \alpha_t & \text{if } f_t(x_i) > \theta_t \\ \beta_t & \text{otherwise} \end{cases}$$

$$C(x) = \theta\left(\sum_t h_t(x) + b\right)$$



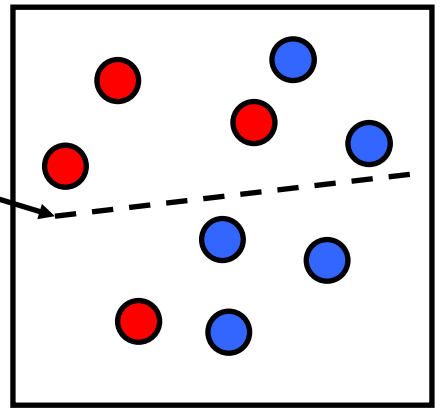
# AdaBoost: Learn + Select Features

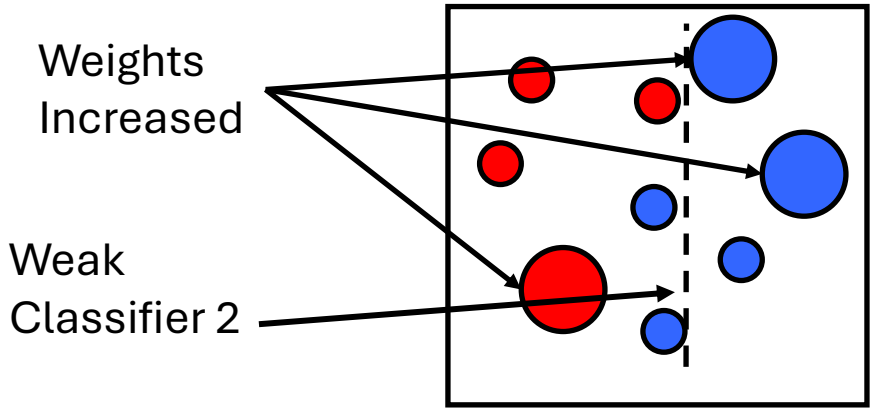
- Each Haar feature is a weak learner (threshold / stump)
- AdaBoost iteratively picks the best feature under reweighted samples
- Output strong classifier: $H(x) = \text{sign}(\Sigma\, \alpha_t\, h_t(x))$

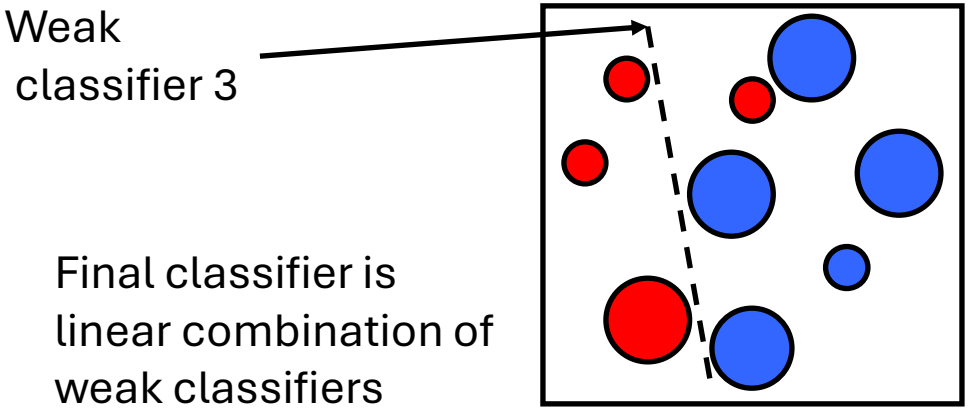









Final classifier is linear combination of weak classifiers

# Applying cascade in data analytics

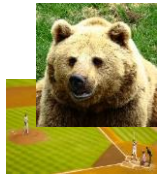
(Streaming inputs)

Images → rcnn → filter: suv → Result

→ rcnn → has person? N
→ has bear? Y
→ has suv? N
→ …

Query is costly because each input → rcnn

Images → filter′ → rcnn → filter: suv → Result

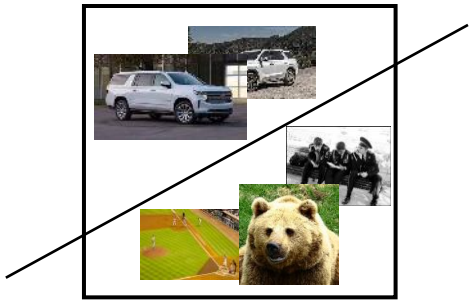


↓ 50-1K fps

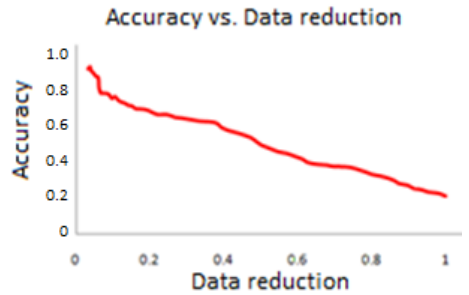
10-30 fps

# Probabilistic Predicates (PPs) w/ a modern query optimizer [SIGMOD18]

Images → $PP_{suv}$ → rcnn → filter: suv → Result



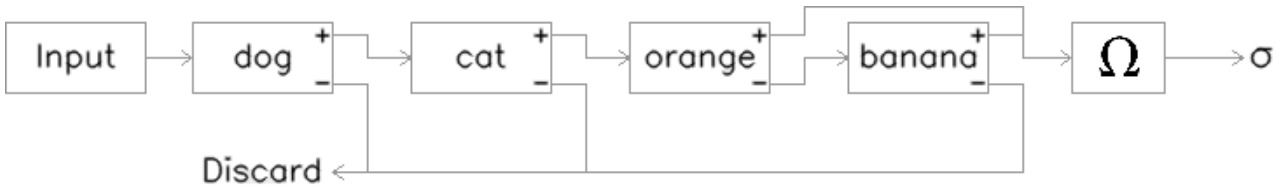
- PPs are binary predicate-specific classifiers:
  - 5x-1000x faster than complete model.
  - {Accuracy, data-reduction} tradeoffs.




- But, too costly to build a PP for every predicate combination:
  - A large combinatorial space. e.g., ¬person ∧ (bike ∨ car).

- Key idea: $PP_{bike}$ $PP_{person}$ $PP_{car}$ $(PP_{dog} \wedge PP_{cat})$ $\wedge$ $(PP_{orange} \vee PP_{banana})$

  Build a PP for each predicate clause & optimizer assembles PPs for complex predicates.




Yao Lu, Aakanksha Chowdhery, Srikanth Kandula, Surajit Chaudhuri. Accelerating Machine Learning Inference with Probabilistic Predicates. SIGMOD 2018.

# Overview

- Sparsity & Pruning
- SVD & Factorization based methods
- Quantization
- Distillation
- Cascade

- Take home messages
  - There is always a tradeoff between accuracy and speed
  - Hardware-software co-design to fully leverage everything

# Questions & comments?

https://PollEv.com/yaolu720