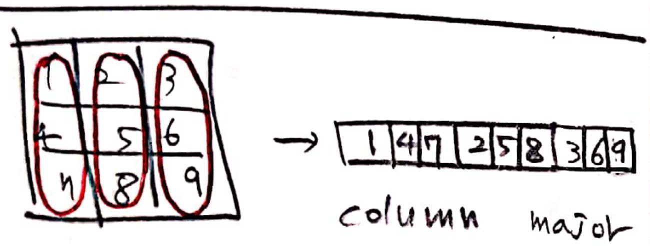
10/25

$\begin{bmatrix} a_1, a_2 \\ a_3, a_4 \end{bmatrix}$ 32 bits × 4 = 128 bits

$$\begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \cdot \begin{bmatrix} b_1 & b_2 \\ b_3 & b_4 \end{bmatrix} =$$

2×2

$$\begin{bmatrix} a_1 \cdot b_1 + a_2 \cdot b_3, & a_1 \cdot b_2 + a_2 \cdot b_4 \\ a_3 \cdot b_1 + a_4 \cdot b_3, & a_3 \cdot b_2 + a_4 \cdot b_4 \end{bmatrix}$$



column major

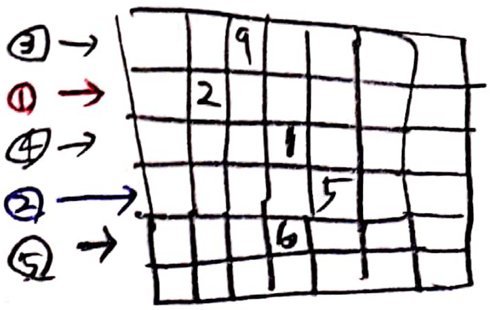
↳ If you ignore the sparsity you may store the matrix in either row-major order or column-major order.

Popular format = COO, CSR

COO:

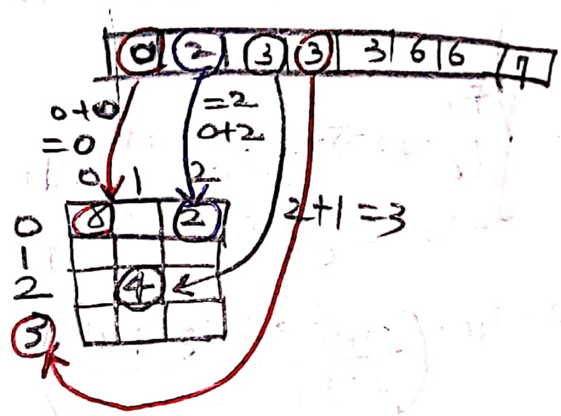
Row = [1 | 3 | 0 | 2 | 4]

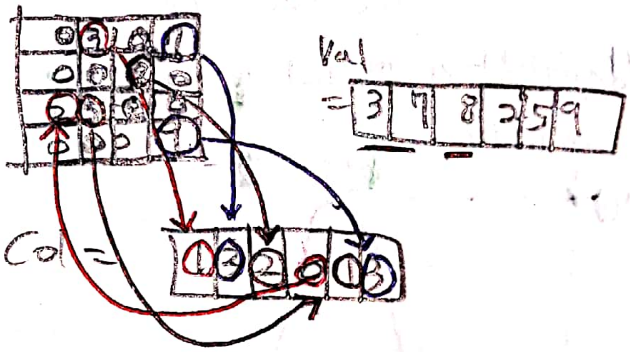
data = [2 | 5 | 9 | 1 | 6]

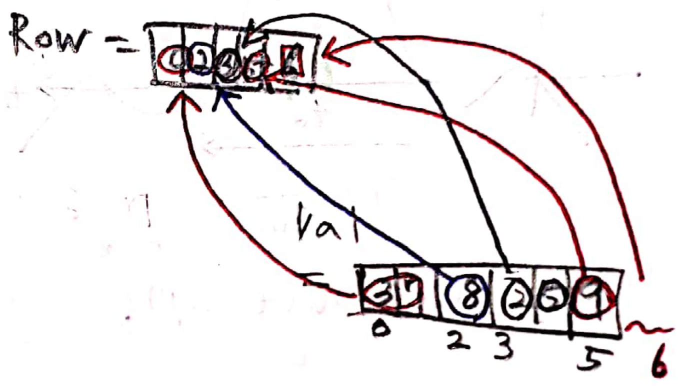


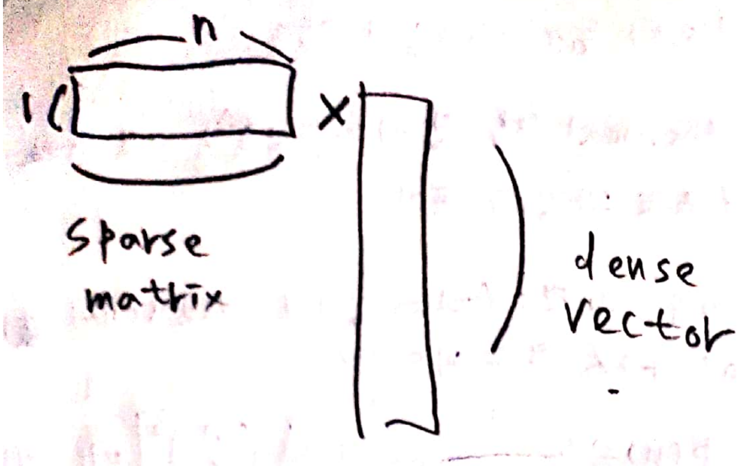
CSR = compressed sparse rows.

↳ store the column of each non-zero element, and which column indices belong to each row.



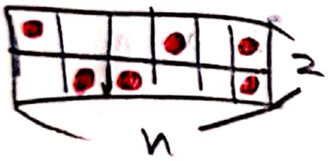






⇒ vector-wise operation.

grouping multiple vectors lead to more number of wasted computation.



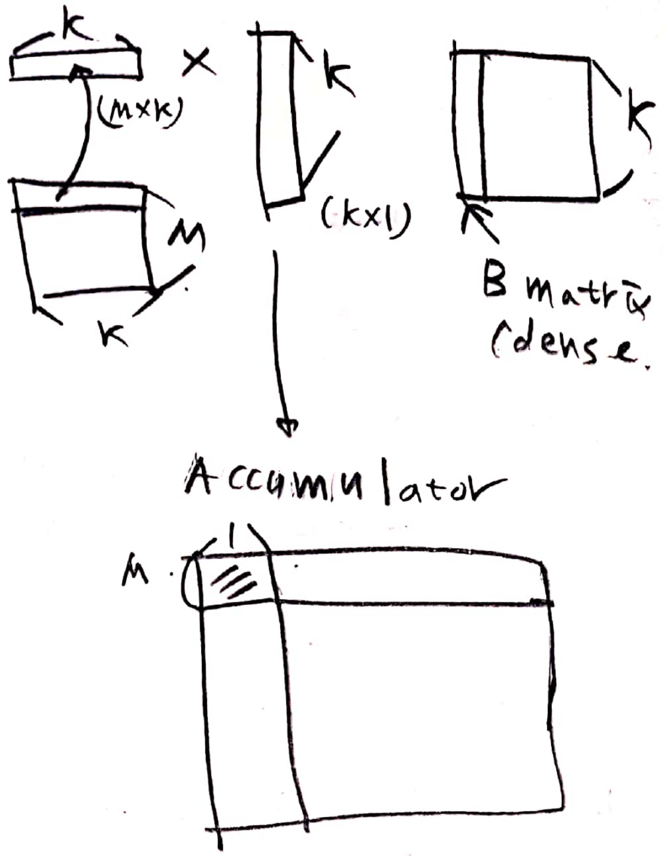
grouping multiple vectors.

● 위치다르니까 dense vector가 이전보다 계산 낭비↑↑.

meta data

<2>

「2-bits indices」



<pruning Initialization>

traing → Pruning → retraining

Folk knowledge

「without an initial training」

① ↳ The pruned model cannot be optimized well

② final model does not generalize well

「Can prune at the initial stage without performance drop.」

↳ There are other methods called GrasP / SynFlow, but SNIP perform best.

<3>

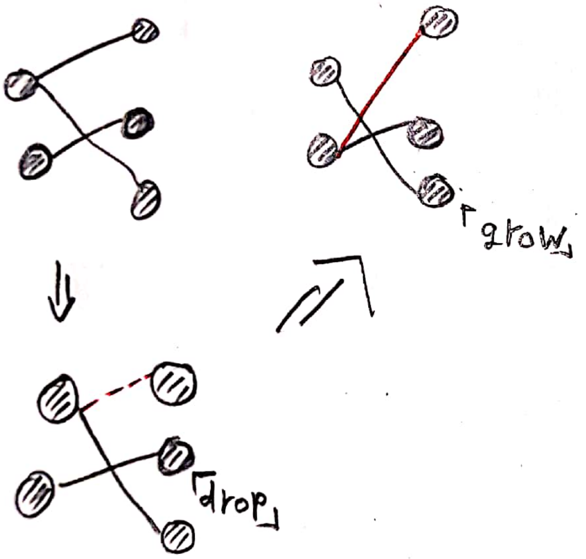
layerwise sparsity.

taylor approximation

↳ 「$L(\tilde{\theta}) \simeq L(\theta) + (\tilde{\theta}-\theta)^T G_\theta + \frac{1}{2}(\tilde{\theta}-\theta)^T H_\theta \cdot (\tilde{\theta}-\theta)$」

sparse training = Limitation

: Peak memory is still the same as dense.



Sparsity ↓↓ ⇒ test accuracy.
↑↑

pruning

① minimizing the loss after pruning

$\boxed{\text{② maximizing the re-train ability}}$

「good layerwise sparsity & schedule」

(1) = 더 중요 (예전)

↓

(2) is viewed as the most important decision criterion.