



Lecture 04:

Pruning Strategies for Efficient DNN Implementation

Notes

- Course website: <https://www.saiqianzhang.com/COURSE/>
- I use Brightspace to post announcements and grades
- I provide an [online zoom meeting](#) option for people interested in auditing the class. However, enrolled students are required to attend in person unless special condition.
- Discussion groups has been created in the Brightspace
- Course email: [`efficientaiaccelerator@gmail.com`](mailto:efficientaiaccelerator@gmail.com)

Notes

- Lab1 will be released this weekend!
- Lab0 will be posted to help you understand DNN pruning.
- In-course quiz today, covering materials of transformer.

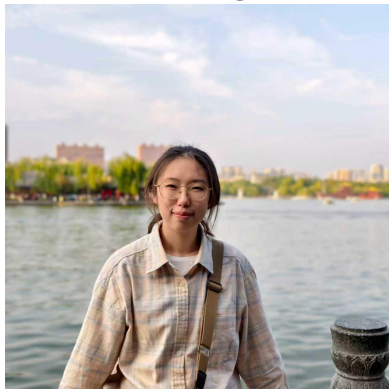
Course Assistant/Grader

Shubham Ojha (CA)



Office hour: Thursday
1:30pm-2:30pm
([Zoom](#))

Yifei Feng (CA)



Office hour: Tuesday
11:00am-12:00pm
([Zoom](#))

Raj Appana (CA)



Office hour: Wednesday
5:00pm-6:00pm
([Zoom](#))

Handong Ji (Grader)



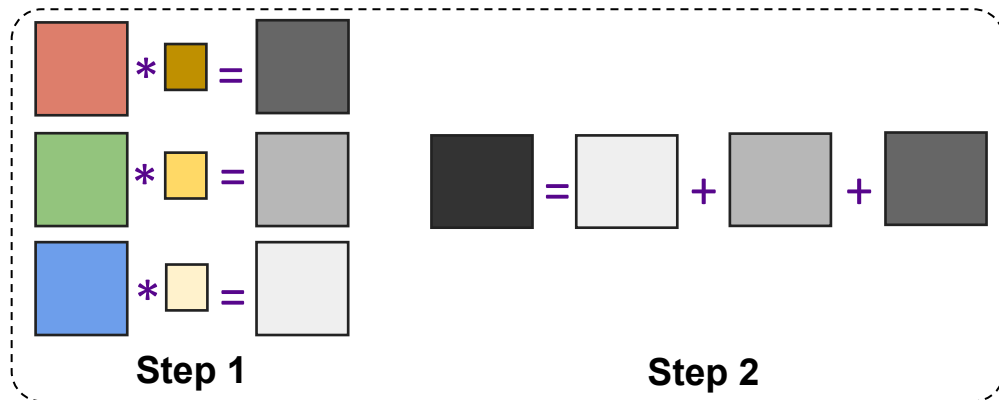
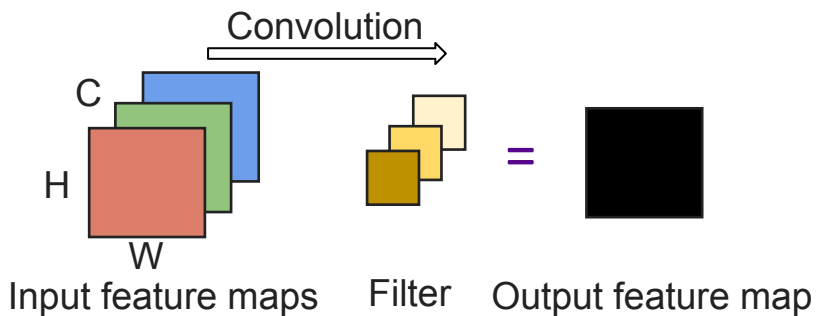
Recap

- Transformer basics
- Bert
- Vision transformer
- Large Language Model

Topics

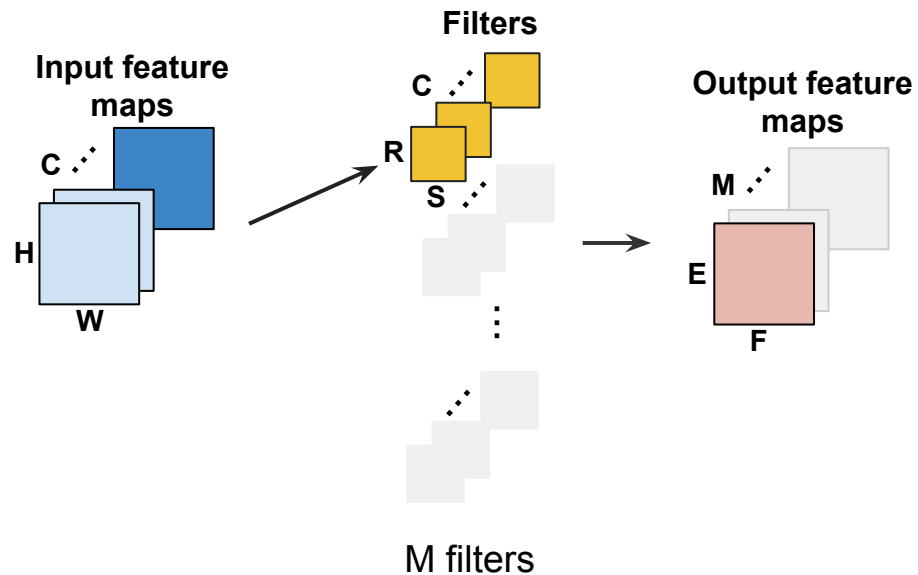
- Why pruning?
 - Reduce running cost
 - Reduce storage
- General pruning techniques
- Transformer pruning
- Large model pruning

Convolutional Layers

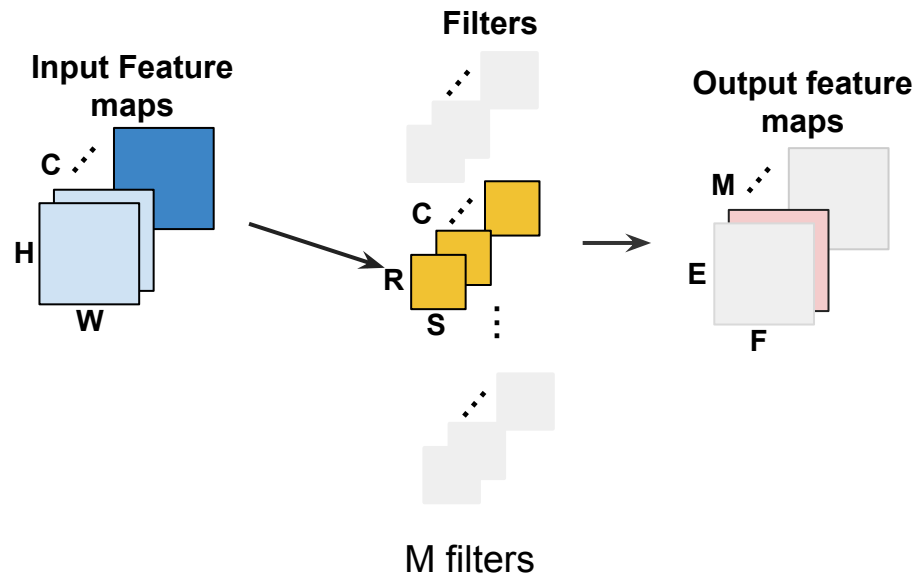


- Core building block of a CNN, it is also the most computational intensive layer.

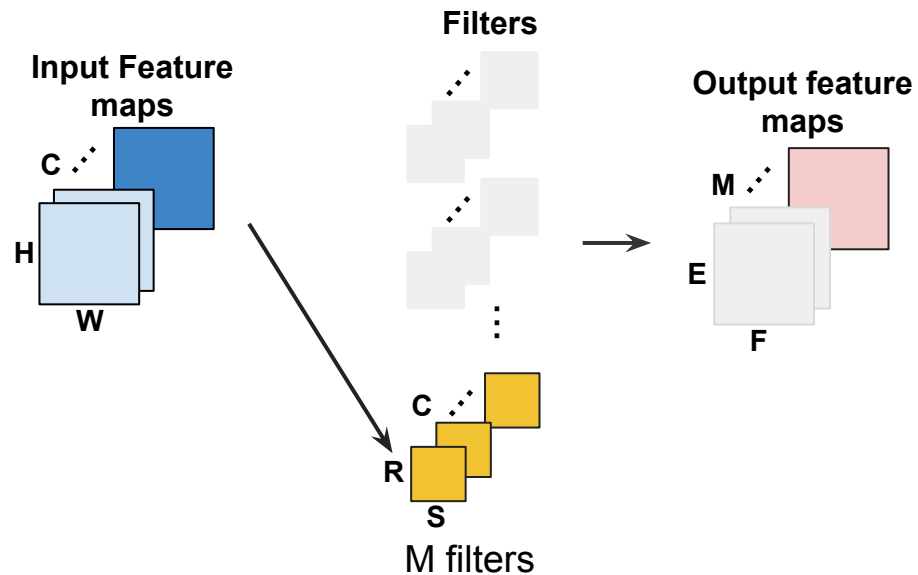
Convolution



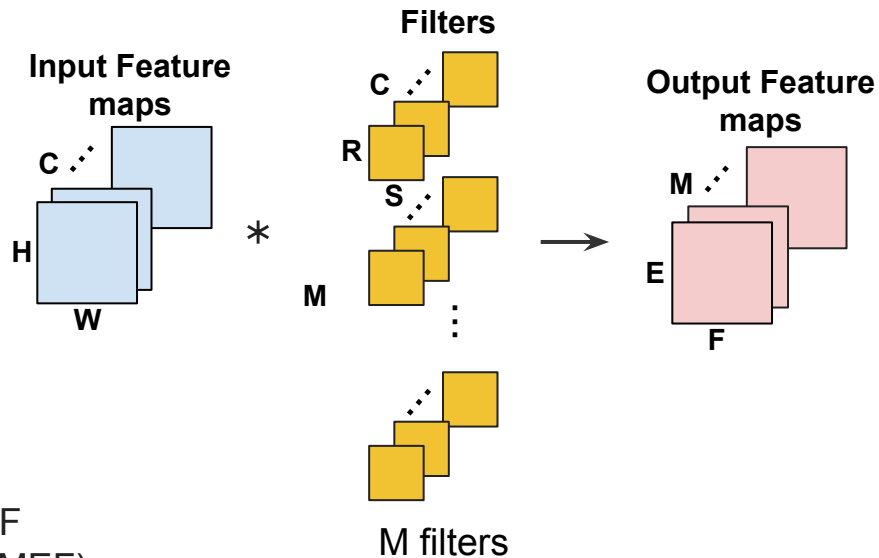
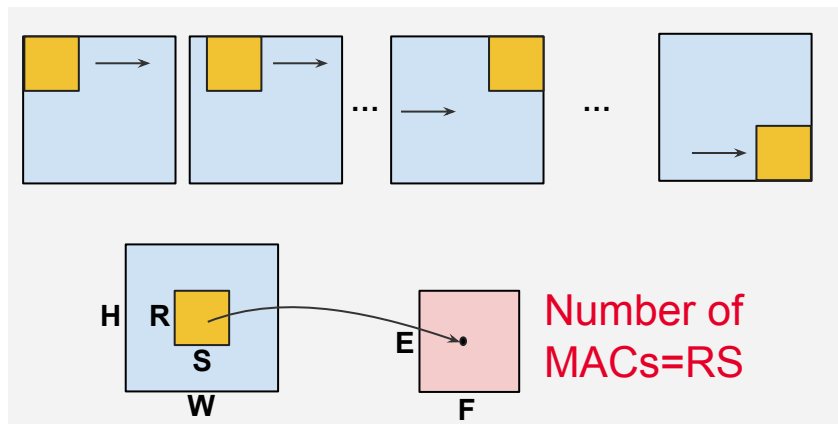
Convolution



Convolution

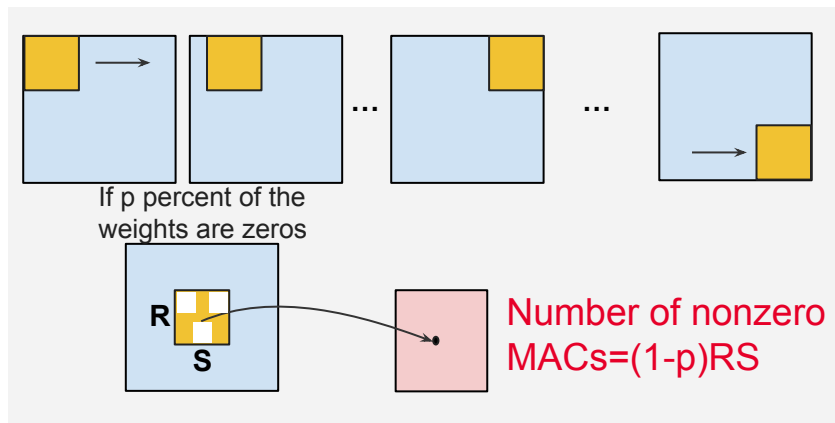


Computational Cost of Convolution



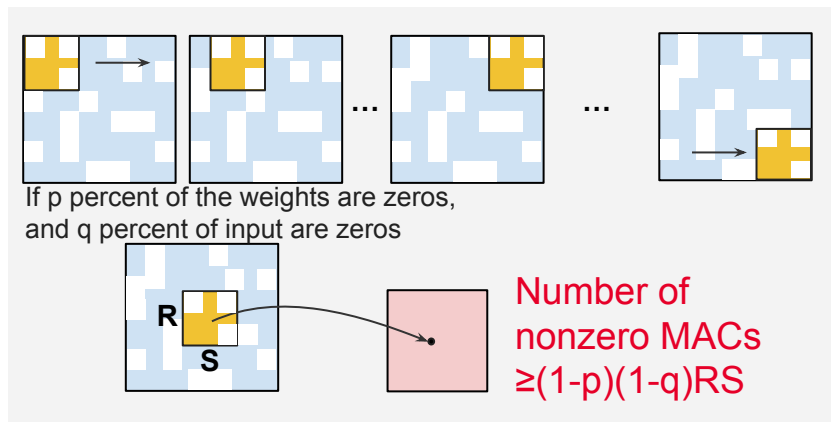
- Number of MACs: $M \times C \times R \times S \times E \times F$
- Storage cost: $32 \times (MCRS + CHW + MEF)$
- The input activation and output activations are transient storage, can be eliminated once this layer is finished processing.

Convolution with Sparse Weight



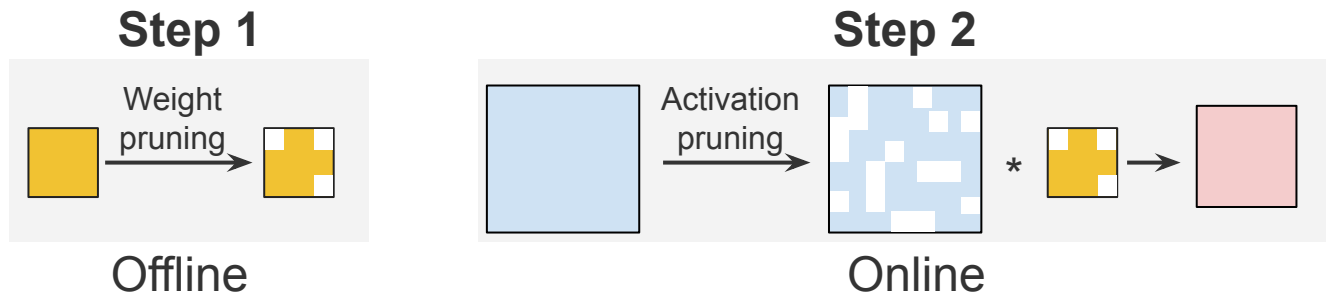
- Number of MACs: $(1-p) \times M \times C \times R \times S \times E \times F$
- Weight pruning can reduce the computations.
- Sparse weight matrices can be stored more efficiently, which helps minimize memory usage.

Convolution with Sparse Weight



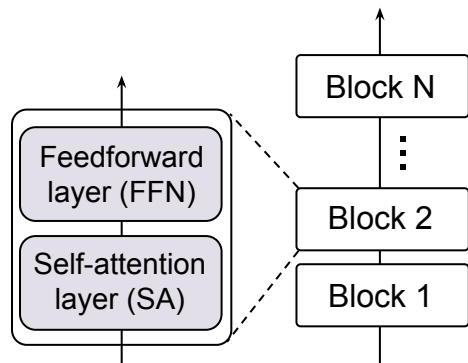
- Number of MACs $\geq (1-p) \times (1-q) \times M \times C \times R \times S \times E \times F$
- Input pruning can also reduce the computations.
- Sparse input and weight matrices can be stored more efficiently, which helps minimize memory storage.

Convolution with Sparse Weight

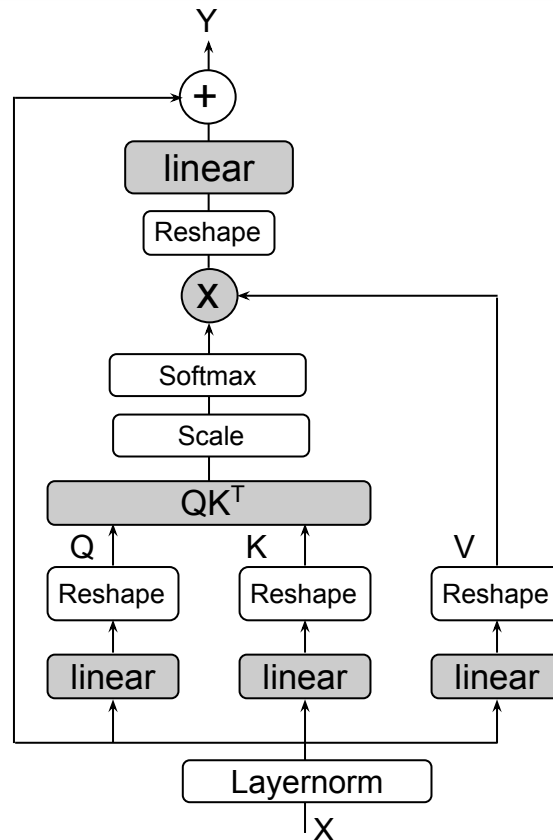


- Activation sparsity requires online pruning, which leads to additional overhead for sorting.

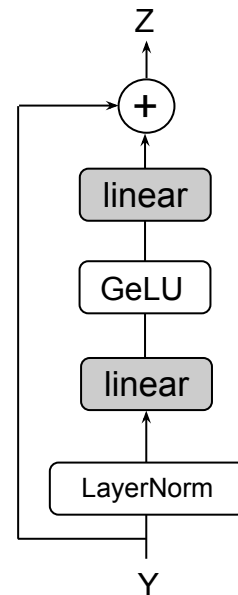
Transformers



- The input sentence has three dimensions:
 - B: batch
 - L: sequence length (number of words)
 - E: embeddings

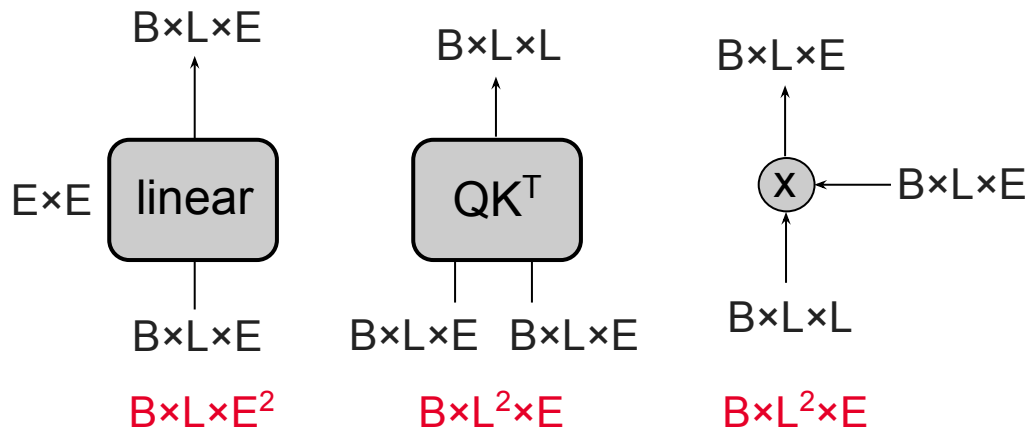


**Self attention block
(SA)**

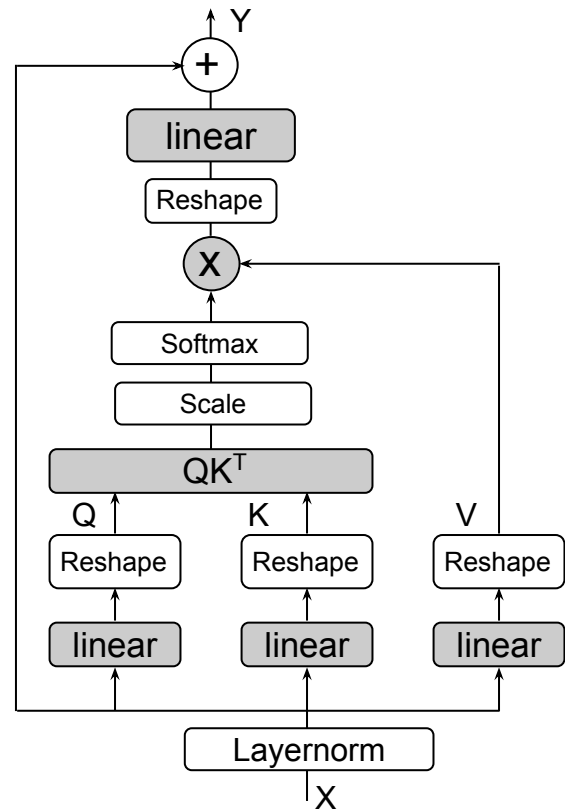


**Feed forward block
(FFN)**

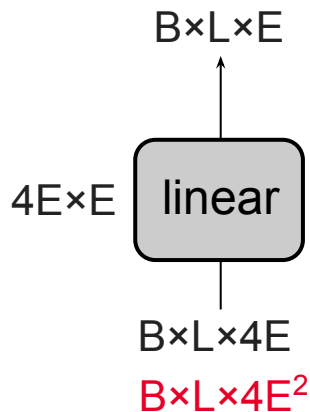
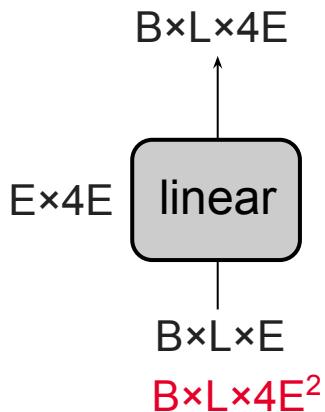
Computational Cost of Transformer



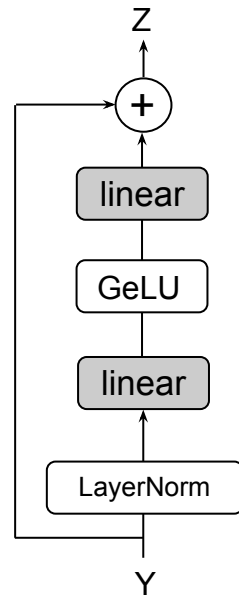
$$\text{Total} = 4B \times L \times E^2 + 2B \times L^2 \times E$$



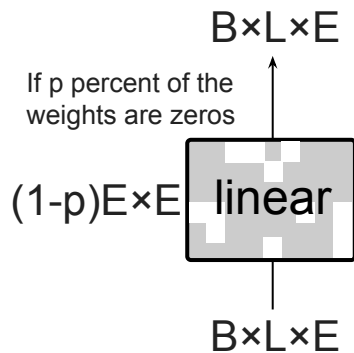
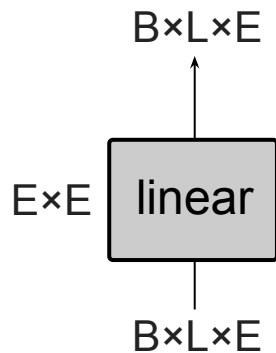
Computational Cost of Transformer



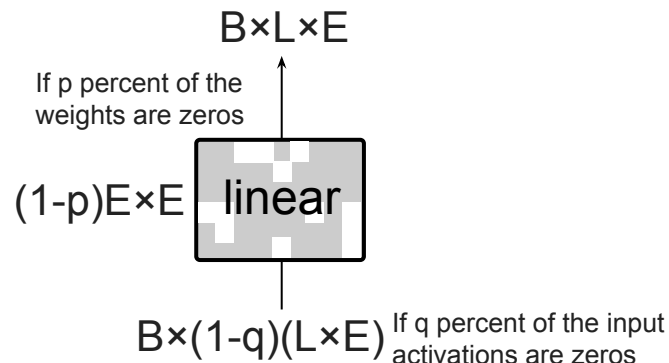
$$\begin{aligned}\text{Total} &= 4B \times L \times E^2 + 2B \times L^2 \times E + 8B \times L \times E^2 \\ &= 12B \times L \times E^2 + 2B \times L^2 \times E\end{aligned}$$



Computational Cost of Transformer



Nonzero MACs
 $= (1-p) \times B \times L \times E^2$



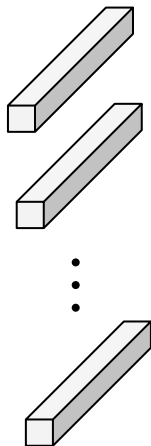
Nonzero MACs
 $\geq (1-p)(1-q) \times B \times L \times E^2$

Topics

- Why pruning?
 - Reduce running cost
 - **Reduce storage**
- General pruning techniques
- Transformer pruning
- Large model pruning

Pruning

weight filter



0.1	3	0.2	1	0.4	0.2	-1	3	0.4	0.6
-----	---	-----	---	-----	-----	----	---	-----	-----

0.2	1.2	0.2	-1	0.2	-3	0.5	-3	3	-1
-----	-----	-----	----	-----	----	-----	----	---	----

⋮

-8	-1	0.6	1.4	0.1	0.1	-2	0.1	-7	-1
----	----	-----	-----	-----	-----	----	-----	----	----

Prune if $|w| < 1$
→

0	3	0	1	0	0	-1	3	0	0
---	---	---	---	---	---	----	---	---	---

0	1.2	0	-1	0	-3	0	-3	3	-1
---	-----	---	----	---	----	---	----	---	----

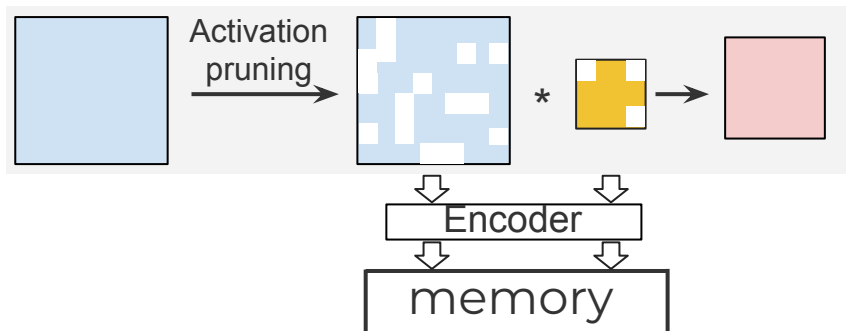
⋮

-8	-1	0	1.4	0	0	-2	0	-7	-1
----	----	---	-----	---	---	----	---	----	----

- Pruning reduces both computational demands and storage costs.

Benefit of Pruning

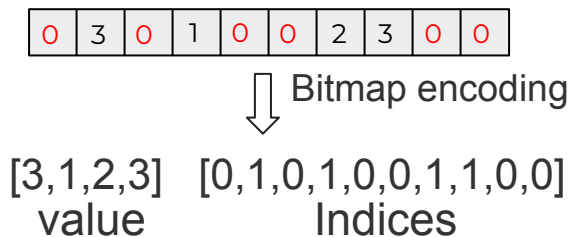
- Reduce computational complexity
 - To support sparse matrix with random sparsity pattern, specialized hardware is required.
- Reduce the storage complexity
 - To achieve it, we need to encode the sparse weights.
 - Encoding the activations requires additional computation.



Sparse Matrices Encodings

- Efficient encoding scheme for sparse matrix storage.
 - Bitmap
 - Run Length Encoding (RLE)
 - Coordinate format (COO)
 - Compressed sparse row (CSR), Compressed sparse column (CSC)

Bitmap Encoding

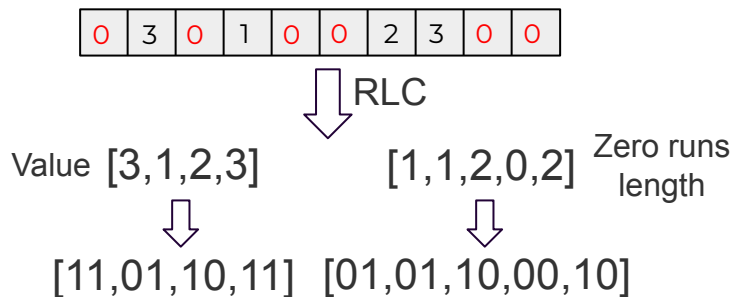


- In summary, the storage cost of bitmap encoding (in bits) is:
 $(1-p) \times L \times n + L$
- n : number bits per value
- L : number of elements
- p : sparsity (%)

- Bitmap is effective for compressing the tensors of low or moderate sparsity.
- Encoding cost is low.

Run Length Encoding (RLC)

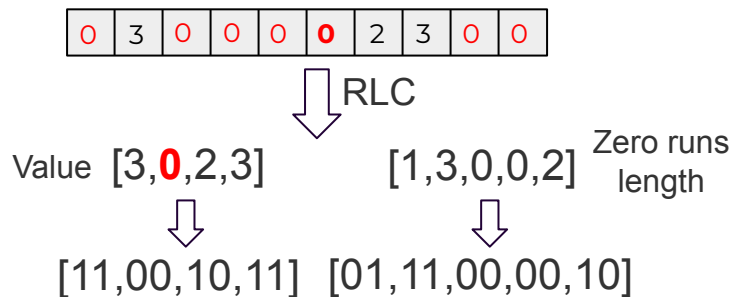
- Record the values and length of zero runs between the values.
- Assume 2 bits are used to encode the length of zero runs (0-3).
- Each value requires 2 bits.



- RLC can reduce storage requirement when sparsity is moderate.

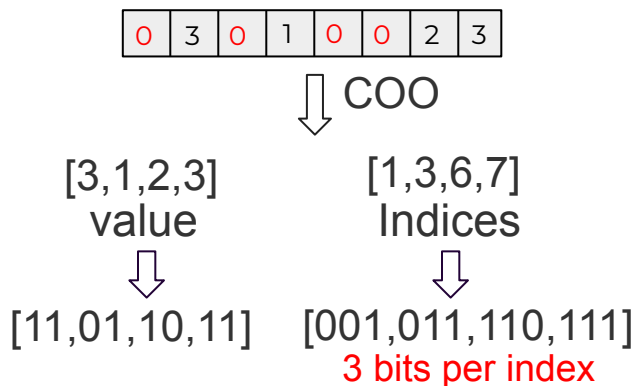
Run Length Encoding (RLC)

- Record the values and length of zero runs between the values.
- Assume 2 bits are used to encode the length of zero runs (0-3).
- Each value requires 2 bits.



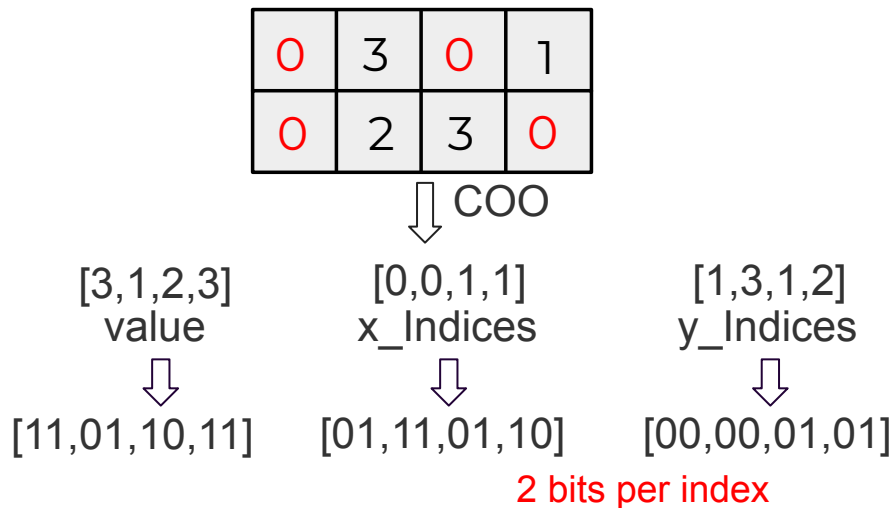
- RLC can reduce storage requirement when sparsity is moderate.
- Hard to formulate it analytically.

Coordinate Format (COO)



- COO is efficient with the sparsity level is extremely high.
- The storage cost (in bits) is: $(1-p) \times L \times n + (1-p) \times L \times \text{Ceil}(\log_2 L)$
- n : number bits per value
- L : number of elements
- p : sparsity (%)

Coordinate Format (COO)



- COO is efficient with the sparsity level is extremely high.

Compressed Sparse Row/Column (CSR/CSC)

$$\begin{pmatrix} 10 & 20 & 0 & 0 & 0 & 0 \\ 0 & 30 & 0 & 40 & 0 & 0 \\ 0 & 0 & 50 & 60 & 70 & 0 \\ 0 & 0 & 0 & 0 & 0 & 80 \end{pmatrix}$$

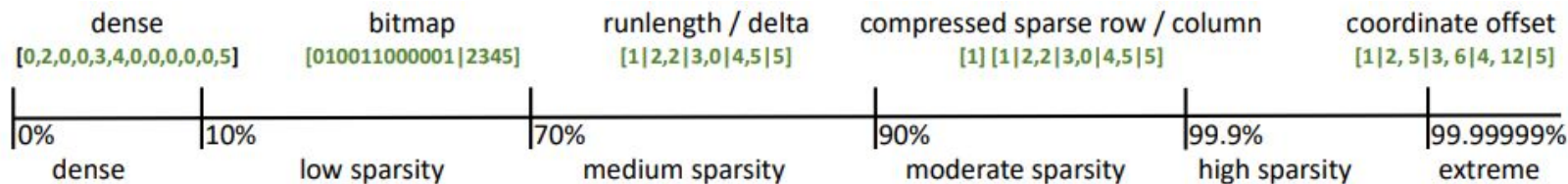
Sparse matrix

$$\begin{aligned} V &= [10 \ 20 \ 30 \ 40 \ 50 \ 60 \ 70 \ 80] \\ \text{COL_INDEX} &= [0 \ 1 \ 1 \ 3 \ 2 \ 3 \ 4 \ 5] \\ \text{ROW_INDEX} &= [0 \ 2 \ 4 \ 7 \ 8] \end{aligned}$$

Encoded form

- CSR/CSC is also suitable for matrices with high sparsity.
- The row index specifies the amount of cumulative nonzero values across each row.

Encoding Approaches Tailored to Different Sparsity Levels



- Different encoding scheme can be applied for different sparsity levels.

Topics

- Why pruning?
 - Reduce running cost
 - Reduce storage
- General pruning techniques
- Transformer pruning
- Large model pruning

Pruning Criteria: Magnitude Pruning

- We can prune the weights using the importance score:
 - Magnitude
 - Gradient
 - Hessian
 - ...

Magnitude
pruning

0.1	3	0.2	1	0.4	0.2	-1	3	0.4	0.6
-----	---	-----	---	-----	-----	----	---	-----	-----

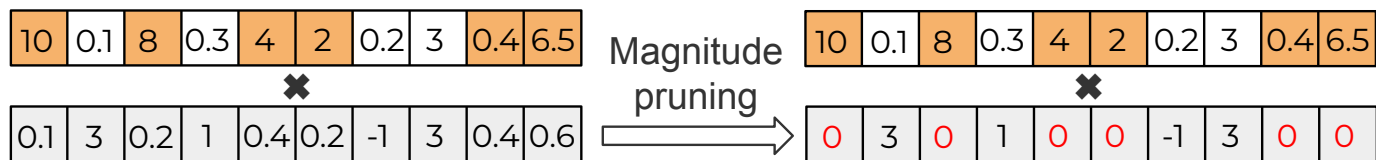
↓

0	3	0	1	0	0	-1	3	0	0
---	---	---	---	---	---	----	---	---	---

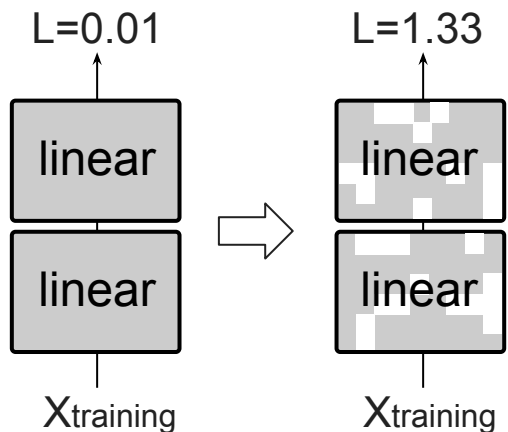
$$m_i = \begin{cases} 1 & \text{if } \|w_i\|_1 \geq a \\ 0 & \text{if } \|w_i\|_1 < a \end{cases}$$

Drawbacks of Magnitude Pruning

- The major drawback of magnitude based pruning is that it does not consider the impact of the input when making the pruning decision.



Pruning Criteria: Training Loss Change



$$\mathcal{L}(w) = \sum_{(x,y) \in D_{\text{training}}} l(y, F_w(x))$$

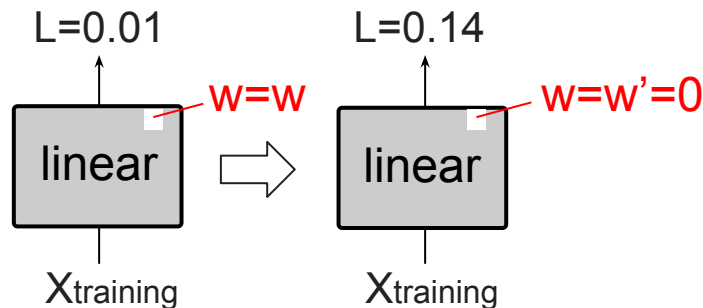
D_{training} : is the training dataset

$F_w(\cdot)$: neural network function with parameter w .

$L(\cdot)$: Training loss function

- Another pruning principle is to minimize the impact on training loss as much as possible.
- For a trained DNN, $L(\cdot)$ remains low.
- Training loss is typically computed over either a subset or the entire training dataset.

Pruning Criteria: Training Loss Change



$$\mathcal{L}(w) = \sum_{(x,y) \in D_{\text{training}}} l(y, F_w(x))$$

D_{training} : is the training dataset

$F_w(\cdot)$: neural network function with parameter w .

$l(\cdot)$: loss function

- Another pruning principle is to minimize the impact on training loss as much as possible.
- For a trained DNN, $L(\cdot)$ remains low.

Pruning Criteria: Training Loss Change

- Pruning criteria:
 - Keep the change on training loss as small as possible
 - Let $L(\cdot)$ denote the training loss
 - For trained DNN, $L(\cdot)$ will be low.

$$\mathcal{L}(w') = \mathcal{L}(w) + \nabla \mathcal{L}(w)^T (w - w')$$

- If w is pruned, then we have $w'=0$:

$$\mathcal{L}(0) - \mathcal{L}(w) = \frac{d\mathcal{L}}{dw} w$$

- We can use it as the pruning criteria
 - Sort the weight based on the product of gradient and its value.

Pruning Criteria: Training Loss Change

$$\mathcal{L}(w') = \mathcal{L}(w) + \nabla \mathcal{L}(w)^T (w - w') + \frac{1}{2} (w - w')^T \nabla^2 \mathcal{L}(w) (w - w')$$

- When reflecting on each individual value, the pruning criteria becomes:

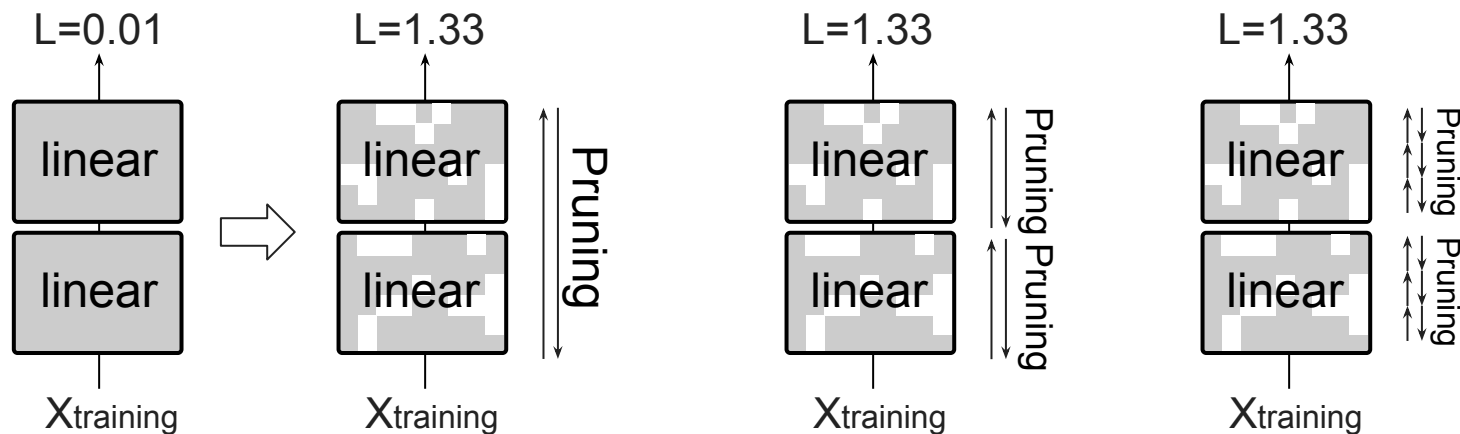
$$\mathcal{L}(0) - \mathcal{L}(w) = \frac{d\mathcal{L}(w)}{dw} w + \frac{1}{2} \frac{d^2 \mathcal{L}(w)}{dw^2} w^2$$

The gradient is usually estimated to zero.

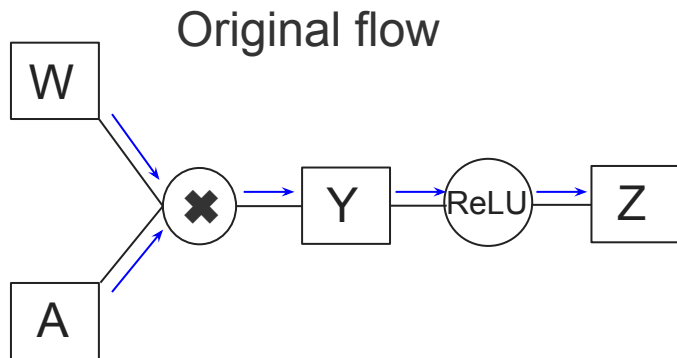
- Multiple approaches have been proposed to estimate the Hessian:
 - Empirical Fisher (Outer Product of Gradients)

$$H(\theta) \approx \frac{1}{N} \sum_{i=1}^N g_i g_i^\top, \quad g_i = \nabla_{\theta} \log p(x_i | \theta)$$

Granularity of Pruning

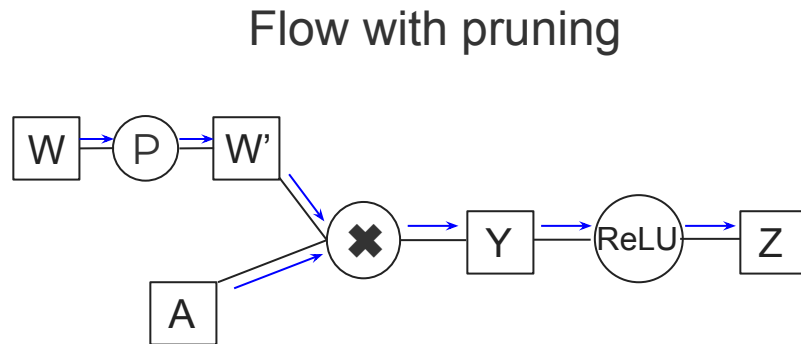


Computational Flow of Pruning



$$Y = WA, Z = \text{ReLU}(Y)$$

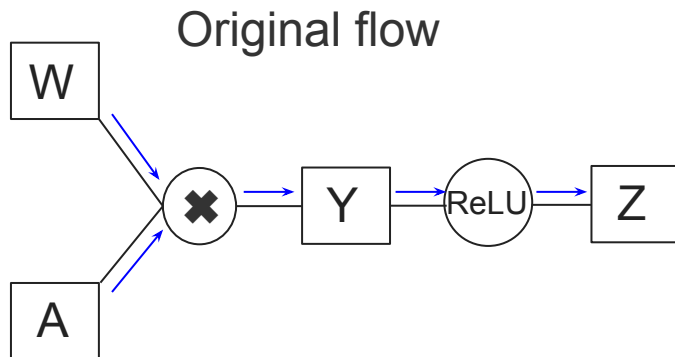
$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial Z} \frac{\partial Z}{\partial Y} \frac{\partial Y}{\partial W}$$



$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial Z} \frac{\partial Z}{\partial Y} \frac{\partial Y}{\partial W'} \frac{\partial W'}{\partial W}$$

$$\frac{\partial W'}{\partial W} = 0 \text{ if } W \text{ is pruned, otherwise } = 1$$

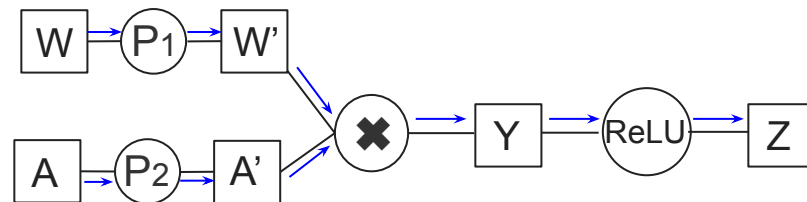
Computational Flow of Pruning



$$Y = WA, Z = \text{ReLU}(Y)$$

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial Z} \frac{\partial Z}{\partial Y} \frac{\partial Y}{\partial W}$$

Flow with pruning

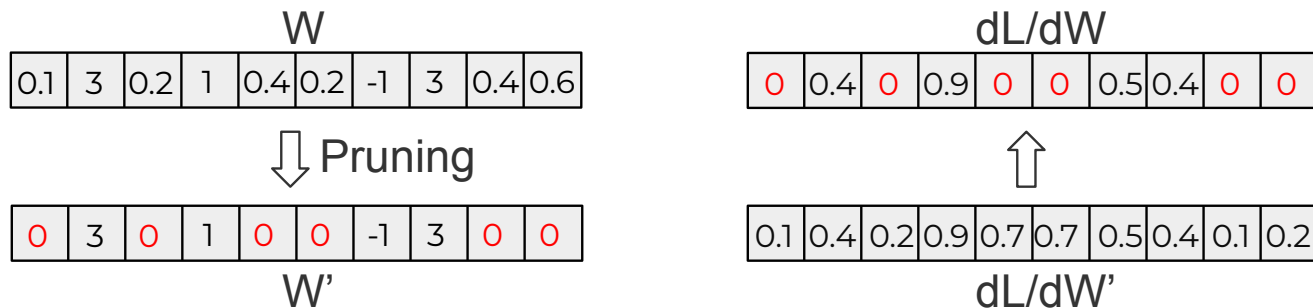


$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial Z} \frac{\partial Z}{\partial Y} \frac{\partial Y}{\partial W'} \frac{\partial W'}{\partial W}$$

$$\frac{\partial W'}{\partial W} = 0 \text{ if } W \text{ is pruned, otherwise } = 1$$

$$\frac{dL}{dA} = \frac{dL}{dZ} \frac{dZ}{dY} \frac{dY}{dA} \frac{dA'}{dA}$$

Backward Pass of Pruning Operation



- During backward pass, the gradients of the pruned elements are masked.

Pytorch Implementation of Iterative Pruning

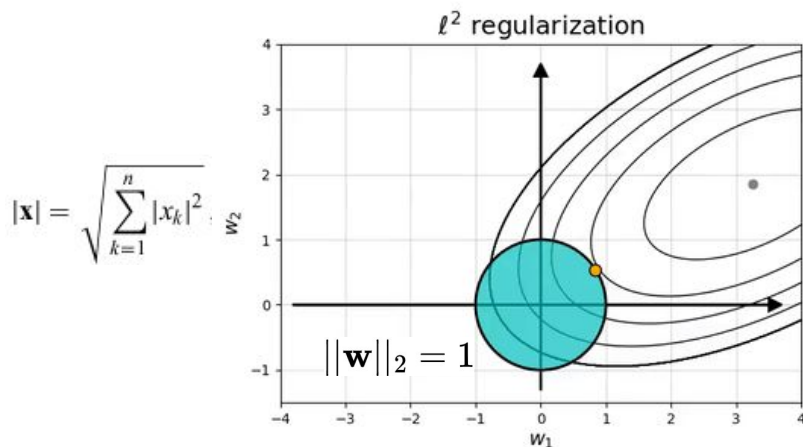
```
def forward(self, x):  
    y = F.conv2d(self.w, x)  
    return y
```

Fake pruning to stimulate the
impact of sparse weight on
the model accuracy!

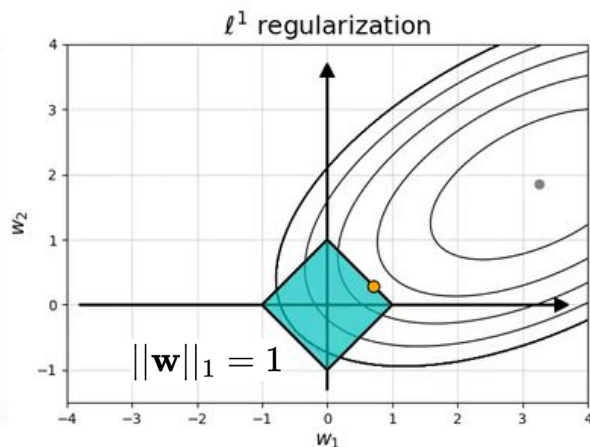
```
mask = nn.Parameter(... requires_grad=False)  
For w in each layer:  
    mask = mask_Prune(w, mask, percent)  
    w = w * mask  
...  
def forward(self, x):  
    y = F.conv2d(self.w, x)  
    return y  
...
```

Regularization-based Pruning

ℓ^1 induces sparse solutions for least squares



$$\min \mathcal{L}(w) + \lambda \|\mathbf{w}\|_2$$



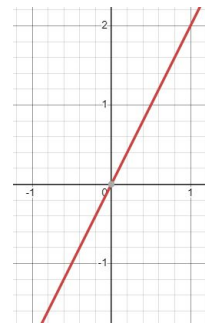
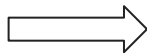
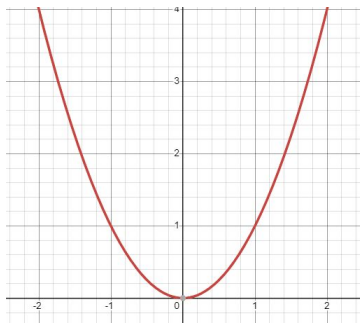
$$\min \mathcal{L}(w) + \lambda \|\mathbf{w}\|_1 \quad \text{Lasso}$$

$$|\mathbf{x}|_1 = \sum_{r=1}^n |x_r|$$

- Add this term can make DNN naturally select the unimportant weight during the training process.

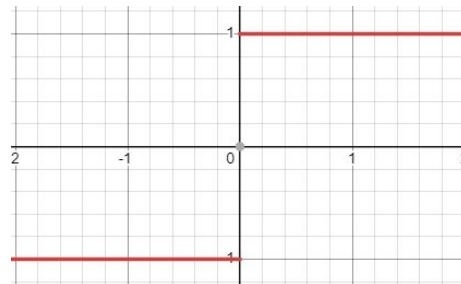
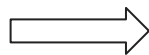
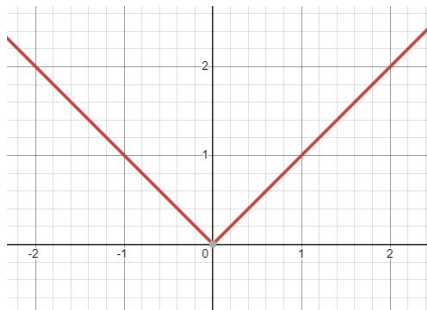
Regularization-based Pruning

$$y = x^2$$



$$y = 2x$$

$$y = |x|$$

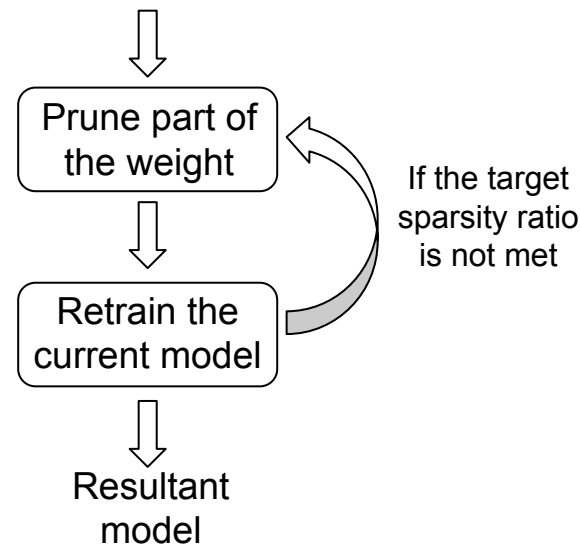
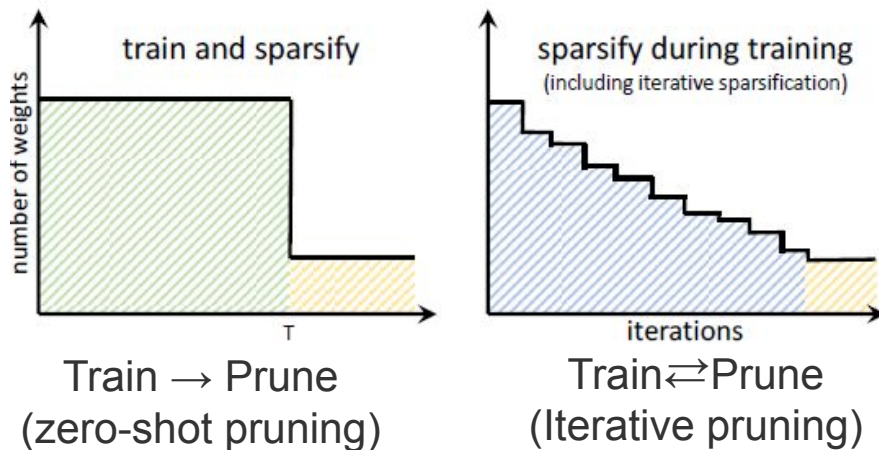


$$y = \text{sign}(x)$$

Taxonomy of Pruning

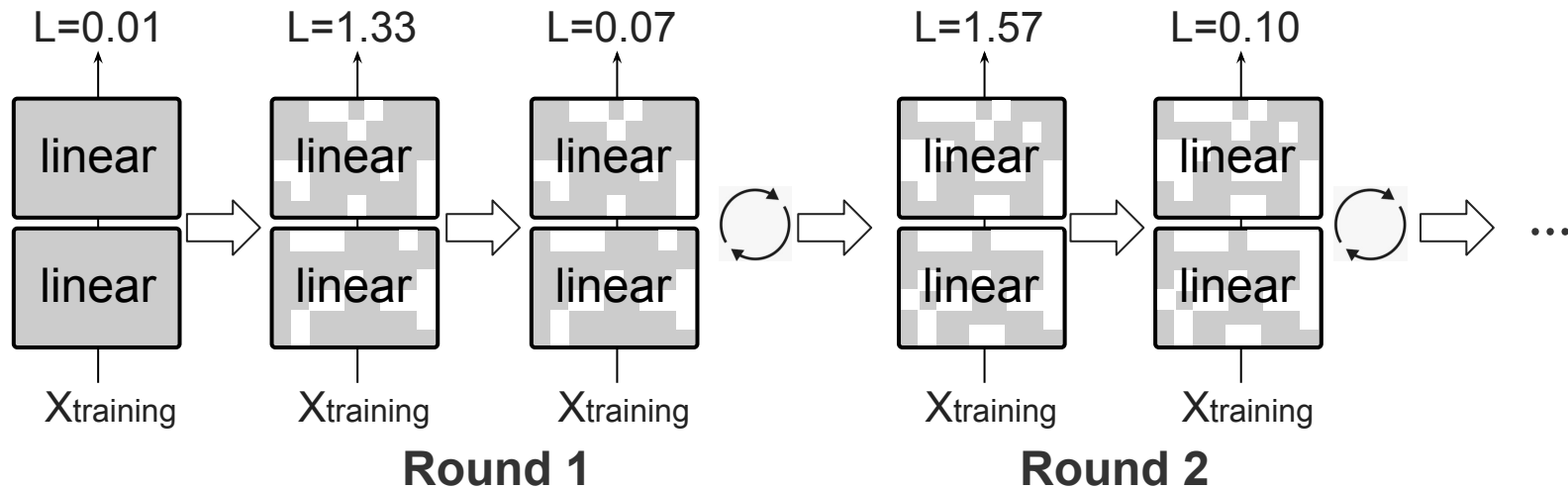
- Pruning techniques can be classified from different perspectives
 - Iterative pruning, zero-shot pruning
 - Structured pruning, unstructured pruning, N:M pruning
 - Weight pruning, activation pruning
 - Static pruning and dynamic pruning
 - Pruning for inference, pruning for training

When to Prune?



- Usually interactive pruning has the best accuracy performance, however, it also requires multiple rounds of training and computational cost.
- Zero-shot pruning also termed post-training pruning.

Iterative Pruning



Lottery Ticket Hypothesis

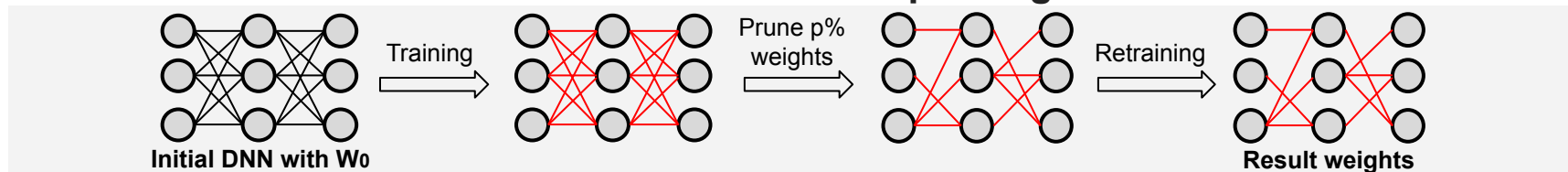
“A randomly-initialized, dense neural network contains a subnetwork that is initialized such that—when trained in isolation—it can match the test accuracy of the original network after training for at most the same number of iterations.”

How to Find the Winning Tickets?

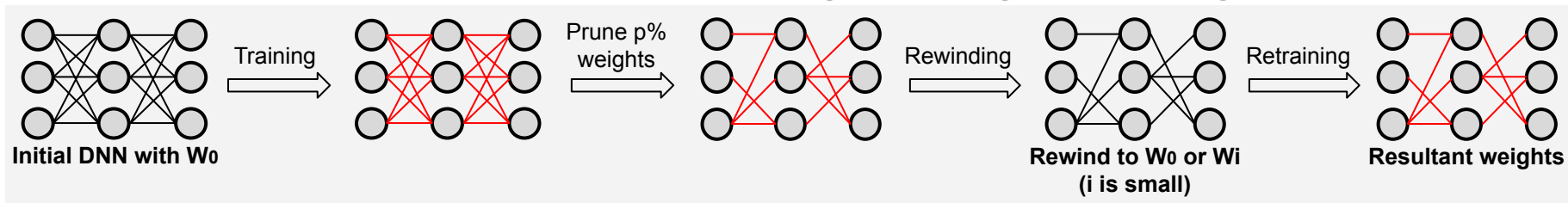
- **Iterative Magnitude Pruning (IMP):**
 - Initialized DNN with random weights w_0 .
 - While the sparsity level has not reached:
 - Train the DNN with k epochs until convergence
 - prune $p\%$ of the nonzero weights.
 - Reinitialize the remaining weights using the values in w_0 , finetune the remaining weights for k epochs (**Rewind**).
 - Return the weights.
- Later work has shown that rewind to w_i (i is small) works better for larger networks.

Weight Rewinding

Conventional iterative pruning



Conventional iterative pruning with weight rewinding

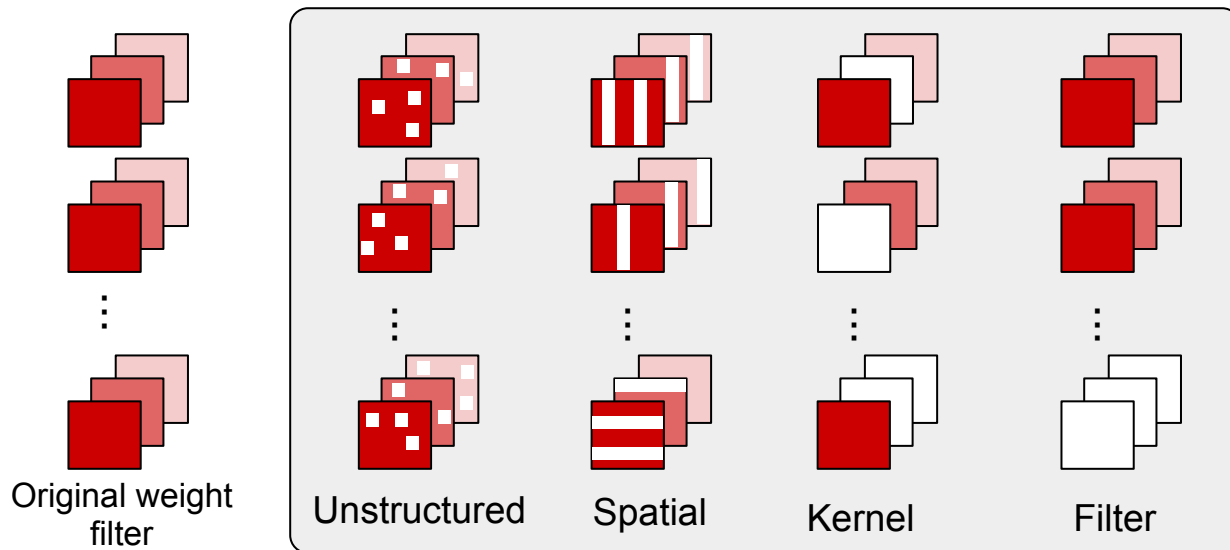


- The pruned architecture itself, rather than a set of inherited “important” weights, is more crucial to the accuracy in the final model, which suggests that in some cases pruning can be useful as an architecture search paradigm.

Taxonomy of Pruning

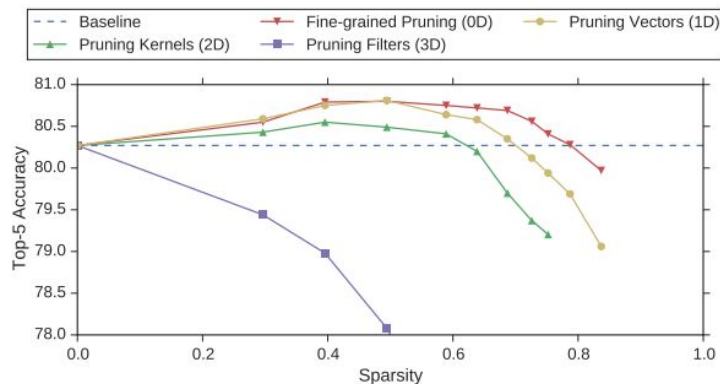
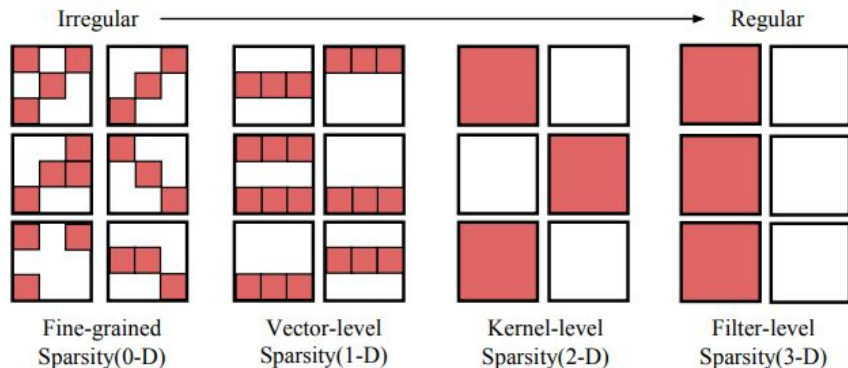
- Pruning techniques can be classified from different perspectives
 - Iterative pruning, zero-shot pruning
 - Structured pruning, unstructured pruning, N:M pruning
 - Weight pruning, activation pruning
 - Static pruning and dynamic pruning
 - Pruning for inference, pruning for training

Unstructured/Structured Pruning



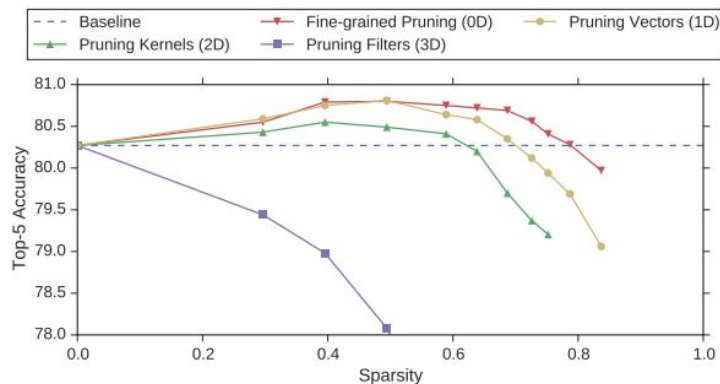
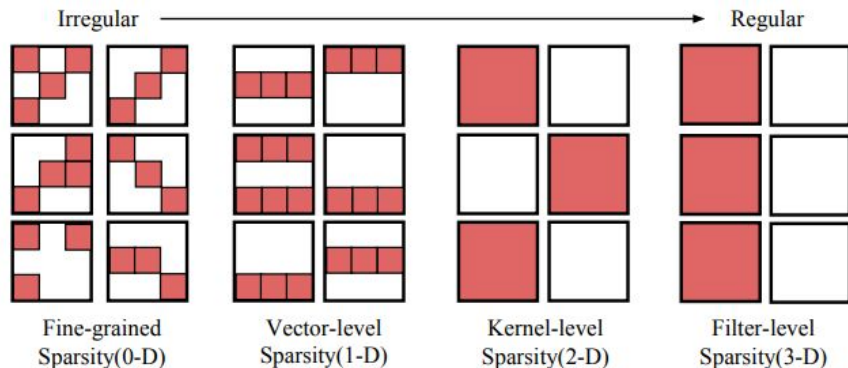
- Structured pruning is amenable to hardware performance, due to the regular sparsity distribution.

Unstructured/Structured Pruning



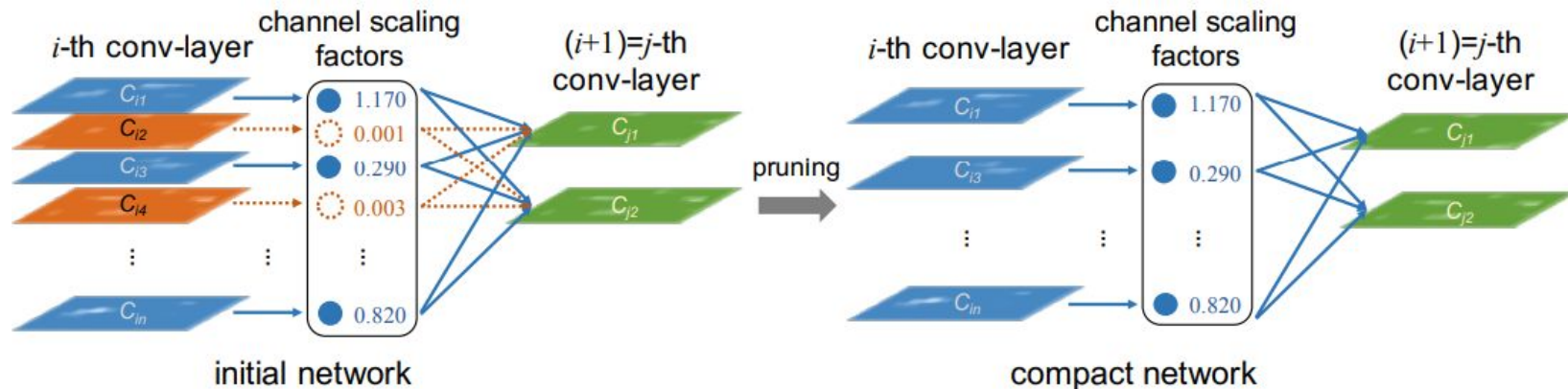
- Unstructured sparsity has a better accuracy than structured sparsity.
- We can apply the same method as the unstructured pruning to prune a group of parameters.

Unstructured/Structured Pruning



- In magnitude-based weight filter pruning, we first compute the sum of absolute weight values within each filter, then rank the filters and prune those with the smallest sums.
- Similarly, structured pruning can be performed by assessing the importance of a weight group through the sum of their importance scores.

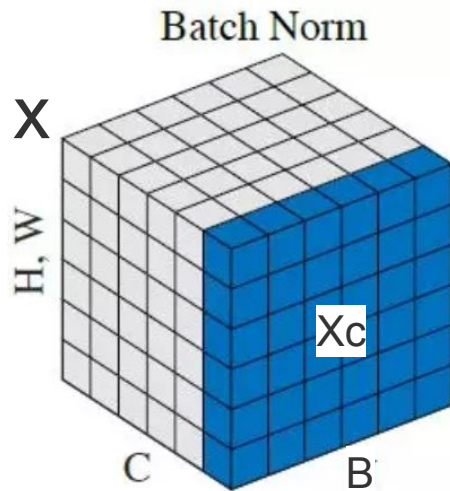
Network Slimming



- We associate a scaling factor (from a batch normalization layer) with each filter in convolutional layers. Sparsity regularization is imposed on these scaling factors during training to automatically identify unimportant filters.

$$L = \sum_{(x,y)} l(f(x, W), y) + \lambda \sum_i |p_i|$$

Batch Normalization



$X: HW \times B \times C$

$$Y_c = \alpha_c \frac{X_c - \mu_c}{\sigma_c} + \beta_c \quad \text{For each } c \in C$$

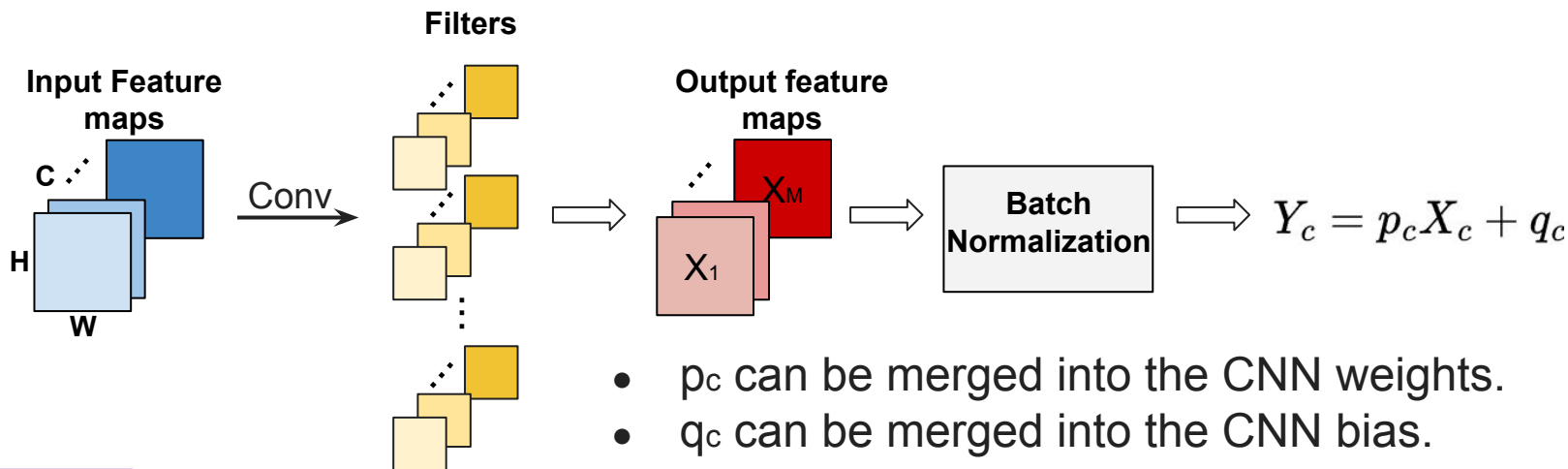
$$\alpha = \{\alpha_c\}, \beta = \{\beta_c\}, \mu = \{\mu_c\}, \sigma = \{\sigma_c\}$$

- For each channel c , we have:
 - X_c : ($HW \times B$)
 - μ_c and σ_c are the mean and standard deviation of X_c .
 - α_c and β_c are learnable parameters
 - $\alpha_c, \beta_c, \mu_c, \sigma_c$ are scalars
- Overall, we have:
 - μ, σ, α and β all have a length of C
 - μ, σ, α and β are all fixed during the inference
 - μ, σ are statistics based on the training dataset

Batch Normalization: During Inference

- Given all the parameters are fixed, for each channel c , we have:

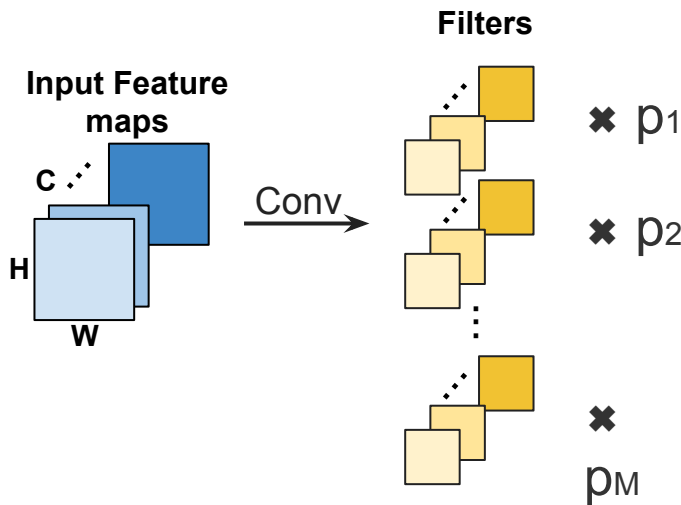
$$Y_c = \alpha_c \frac{X_c - \mu_c}{\sigma_c} + \beta_c = \frac{\alpha_c}{\sigma_c} X_c + \left(\beta_c - \frac{\alpha_c \mu_c}{\sigma_c} \right) \implies Y_c = p_c X_c + q_c$$



Batch Normalization

- For each channel c , we have:

$$Y_c = \alpha_c \frac{X_c - \mu_c}{\sigma_c} + \beta_c = \frac{\alpha_c}{\sigma_c} X_c + \left(\beta_c - \frac{\alpha_c \mu_c}{\sigma_c} \right) \Rightarrow Y_c = p_c X_c + q_c$$



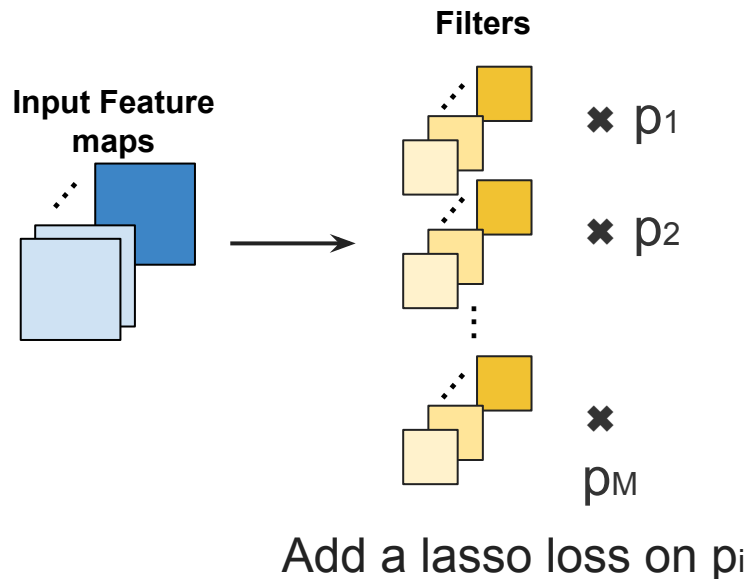
- We can fold in the p and q to the weights and bias of convolutional layer during inference and reduce the online computational cost.

Network Slimming

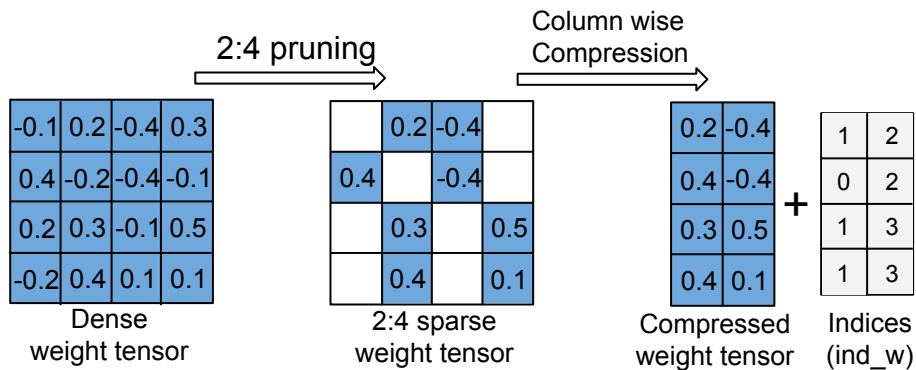
- Lasso regularization is imposed on the scaling factors of batch normalization during training to automatically identify unimportant weight filters.
- $g(\cdot)$ is the lasso l1-norm $g(\cdot) = \sum |p_i|$

$$L = \sum_{(x,y)} l(f(x, W), y) + \lambda \sum_i |p_i|$$

- The unimportant weight filters are naturally eliminated during the training process.



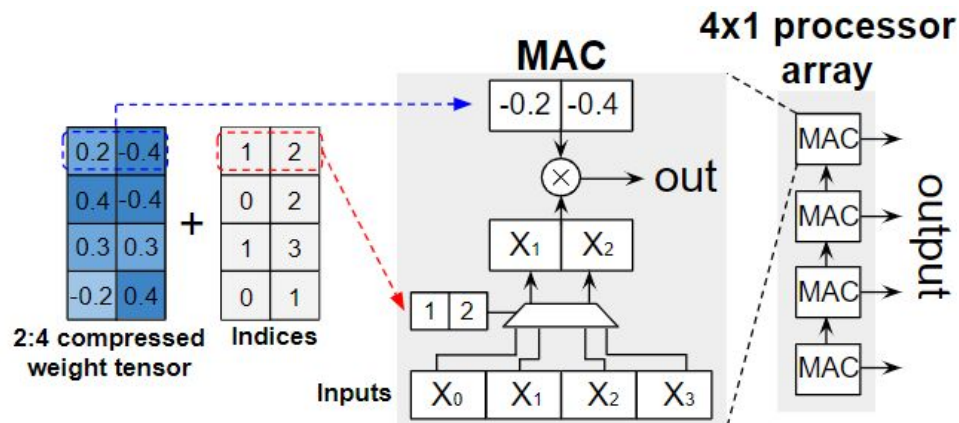
N:M Sparsity



- DNN with structured sparsity can be easily adopted for acceleration, but incur low accuracy.
- On the other hand, DNN with unstructured sparsity is hard to accelerate.

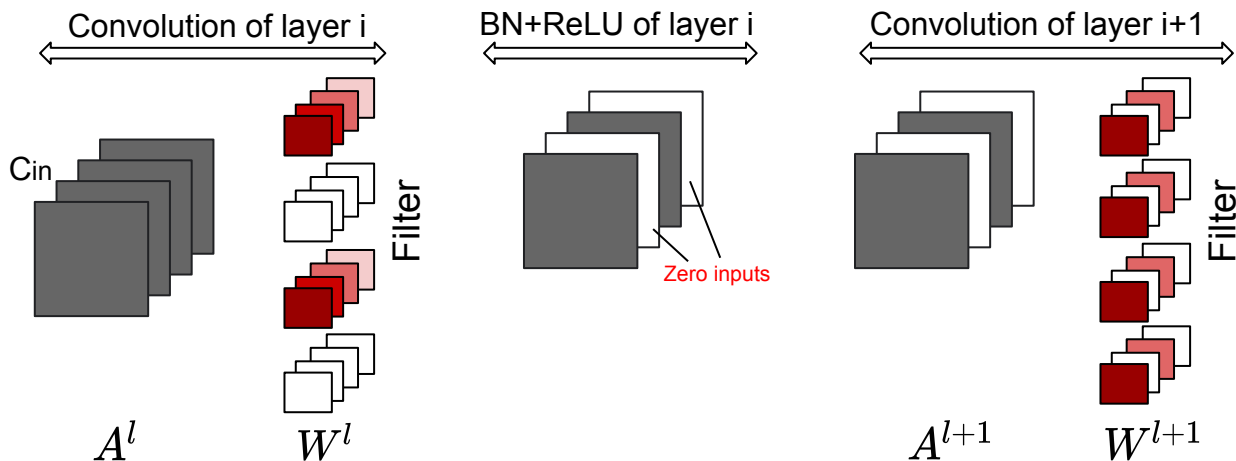
- N:M sparsity is proposed as a middleground between structured and unstructured sparsity.
- 2:4 sparsity is supported in Nvidia V100 GPUs.

N:M Sparsity



- N:M sparsity is proposed as a middleground between structured and unstructured sparsity.
- 2:4 sparsity is supported in Nvidia V100 GPUs.

Cascade Effect of Filterwise Pruning in CNN



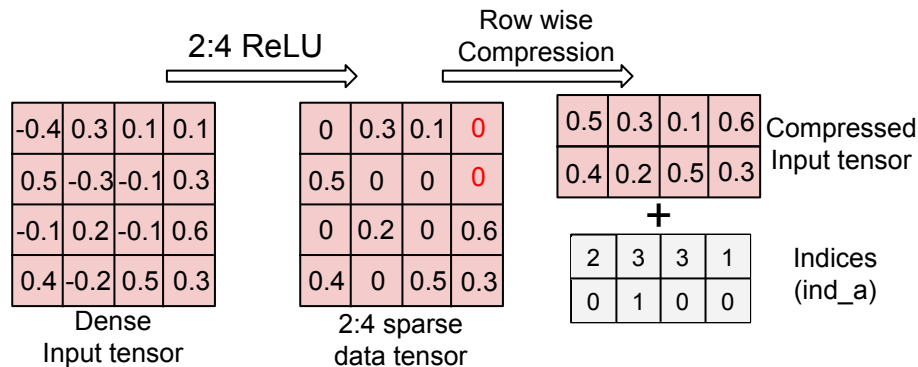
- Assume the bias of the batch normalization is zero.
- Filter pruning at layer l can also result in weight and input sparsity in layer $l+1$.
- When the bias is not zero, the feature maps of layer $i+1$ will contain a uniform constant value.

Taxonomy of Pruning

- Pruning techniques can be classified from different perspectives
 - Iterative pruning, zero-shot pruning
 - Structured pruning, unstructured pruning, N:M pruning
 - **Weight pruning, activation pruning**
 - Static pruning and dynamic pruning
 - Pruning for inference, pruning for training

Pruning on Input Activation

- Why pruning can not be applied to input activation?
 - Large computing cost to determine the importance scores.
 - Due to the usage of ReLU, activation in CNN are 50% sparse, but with irregular sparsity distributions.

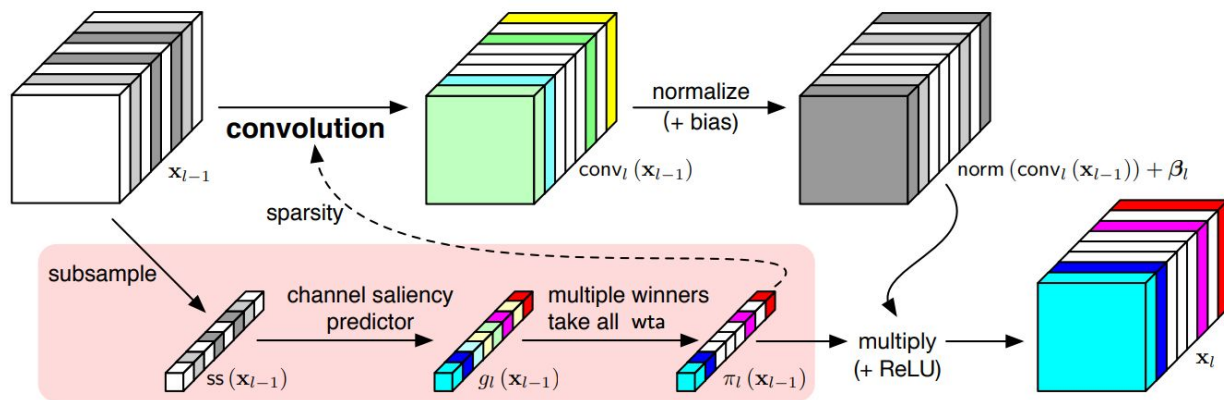


Taxonomy of Pruning

- Pruning techniques can be classified from different perspectives
 - Iterative pruning, zero-shot pruning
 - Structured pruning, unstructured pruning, N:M pruning
 - Weight pruning, activation pruning
 - Static pruning and dynamic pruning
 - Pruning for inference, pruning for training

Static vs Dynamic Pruning

- Conventional pruning adopts static pruning criteria and permanently removes components.
- Dynamic pruning exploits input-specific characteristic pruning criteria and preserves the entire network structures and accelerates the networks by dynamically skipping unimportant components.

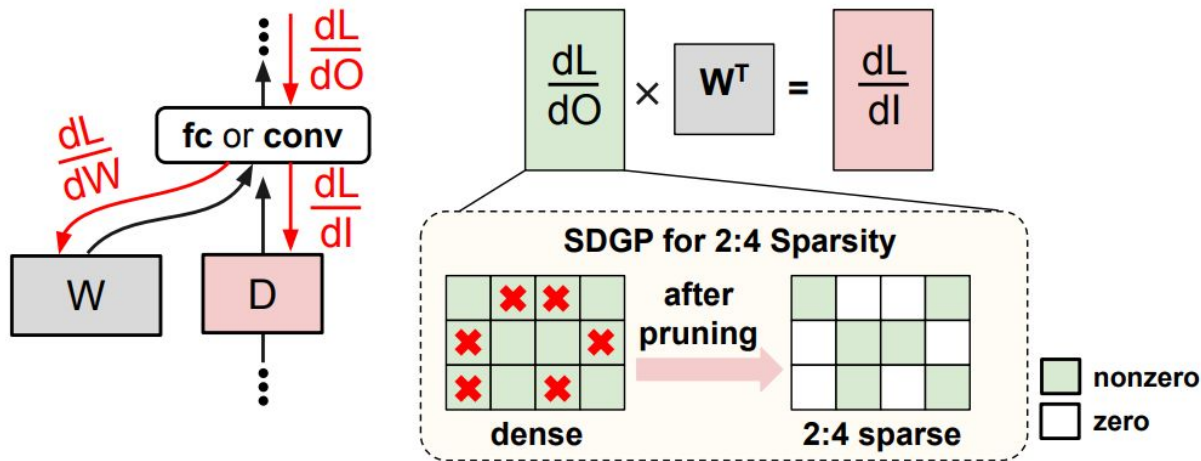


- a channel-wise importance measure is generated.

Taxonomy of Pruning

- Pruning techniques can be classified from different perspectives
 - Iterative pruning, zero-shot pruning
 - Structured pruning, unstructured pruning, N:M pruning
 - Weight pruning, activation pruning
 - Static pruning and dynamic pruning
 - Pruning for inference, pruning for training

Pruning during DNN Training

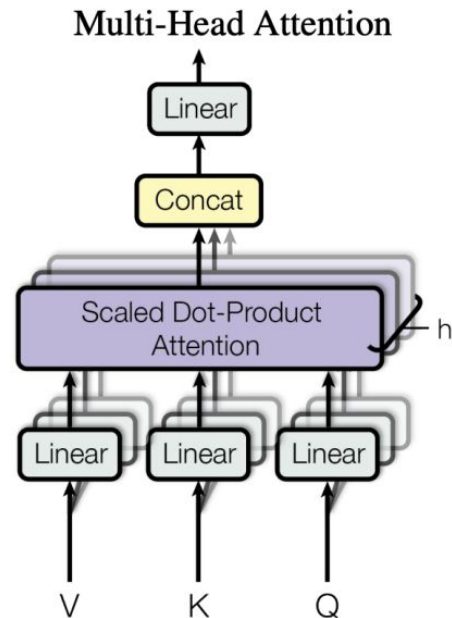


Topics

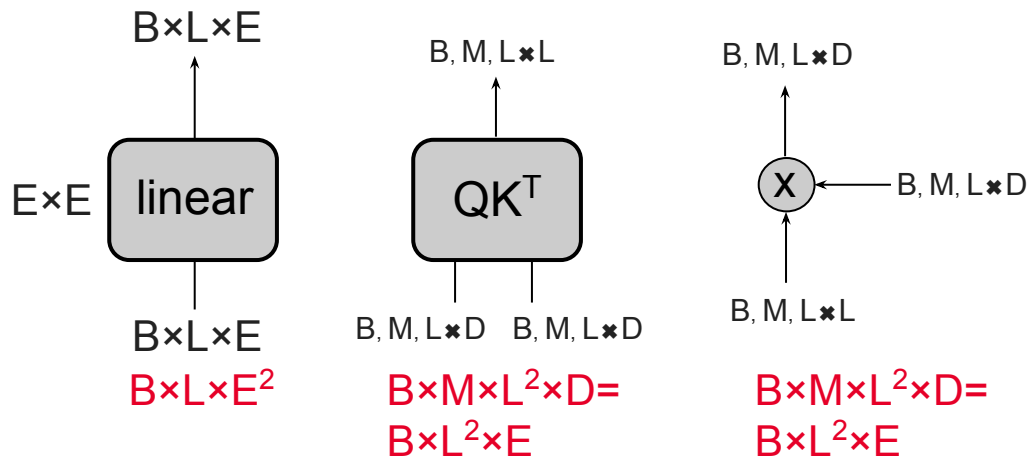
- Why pruning?
 - Reduce running cost
 - Reduce storage
- General pruning techniques
- Transformer pruning
- Large model pruning

Multi-headed Attention

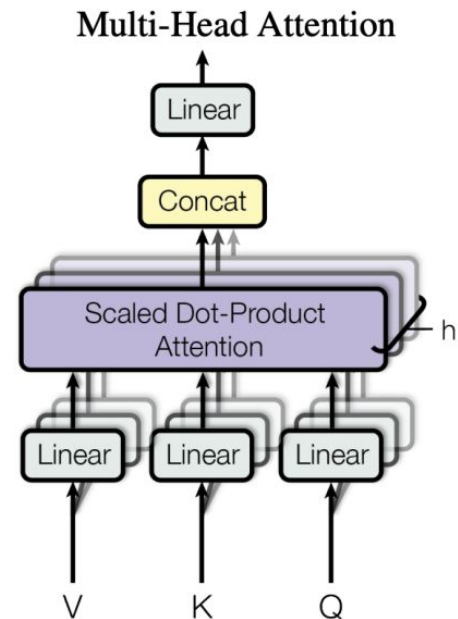
- Q, K, V tensors are broken into multiple components along the embedding dimension.
 - $(B, L, E) \times (E \times E) \rightarrow (B \times L \times E)$
 - $(B, L, E) \rightarrow (B, M, L, E/M) \rightarrow (B, M, L, D)$, where $D=E/M$
- All the following operations can be performed independently over each head M .
 - $QK^T \rightarrow (B, M, L \times D) \times (B, M, D \times L) \rightarrow (B, M, L \times L)$
 - $\text{Softmax}(QK^T) \rightarrow (B, M, L \times L)$
 - $\text{Softmax}(QK^T) \times V \rightarrow (B, M, L \times L) \times (B, M, L \times D) \rightarrow (B, M, L \times D) \rightarrow (B \times L \times E)$



Multi-Head Attention

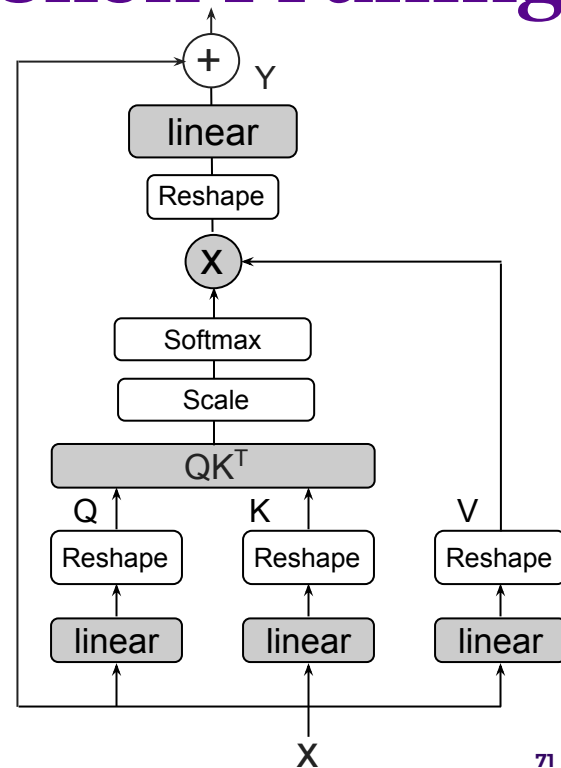


- The introduction of multiple heads do not change the computational cost of the transformer.



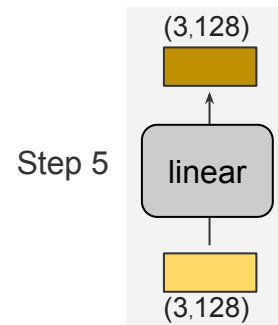
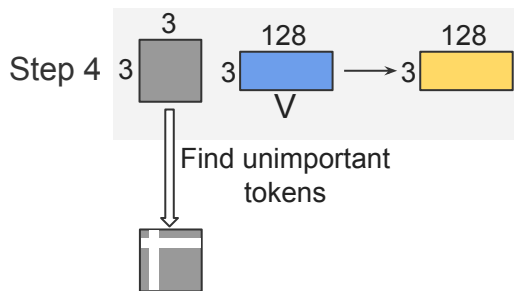
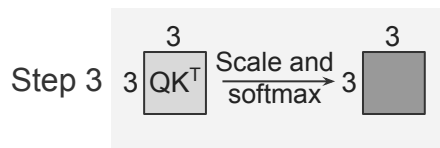
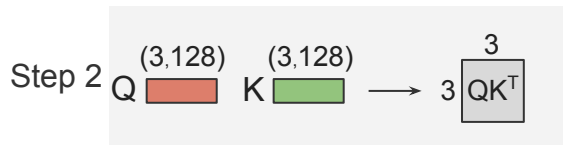
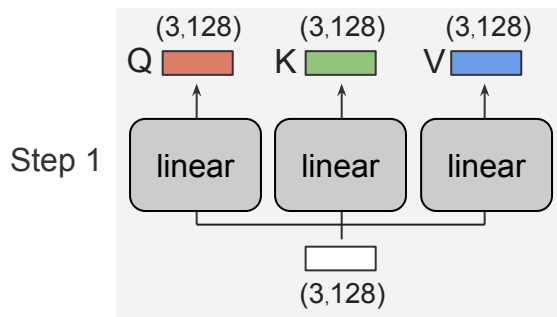
Pruning on Transformers: Token Pruning

- Given input x , the first step in calculating self-attention is to create three vectors from each of the input x , denoted as: Query (Q), Key (K), Value (V).
 - $(B, L, E) * (E * E) \rightarrow (B * L * E)$
- The second step in calculating self-attention. This will compute the attention score between each pair of input tokens.
 - $QK^T \rightarrow (B, L * E) * (B, E * L) \rightarrow (B, L * L)$
- Scale and normalize the score using softmax.
 - $\text{Softmax}(QK^T) \rightarrow (B, L * L)$
- Multiply each value vector by the softmax score.
 - $\text{Softmax}(QK^T) * V$
 - $(B, L * L) * (B, L * E) \rightarrow (B, L * E)$
- Pass the result to the linear layer, sum with the input.

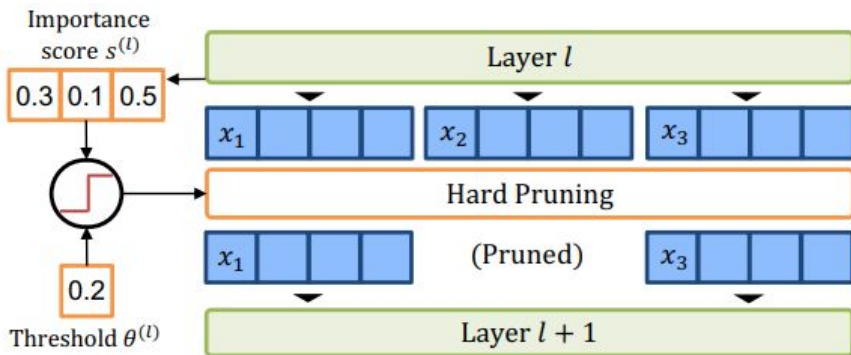


Pruning on Transformers: Token Pruning

“I love AI” $\longrightarrow$ 3×128



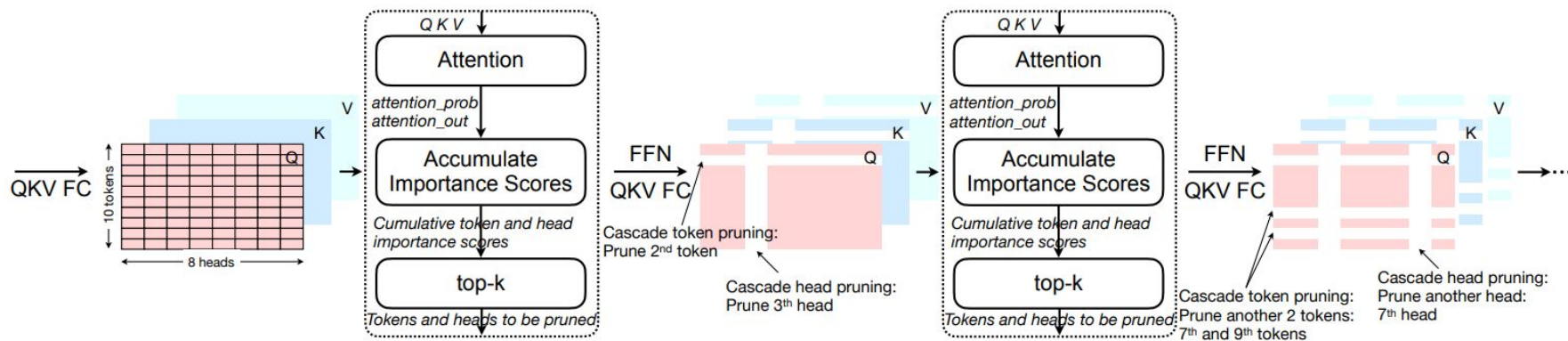
Pruning on Transformers: Token Pruning



$$s^{(l)}(x_i) = \frac{1}{N_h} \frac{1}{n} \sum_{h=1}^{N_h} \sum_{j=1}^n A^{(h,l)}(x_i, x_j)$$

- One simple approach involves computing the importance score of each token, and remove the tokens whose importance score is lower than a predefined threshold.

Pruning on Transformers: Token Pruning



- Tokens and heads can be pruned jointly, the removed tokens and heads will result in a much reduced computation for all the following layers.

Pruning on Transformers: Token Pruning



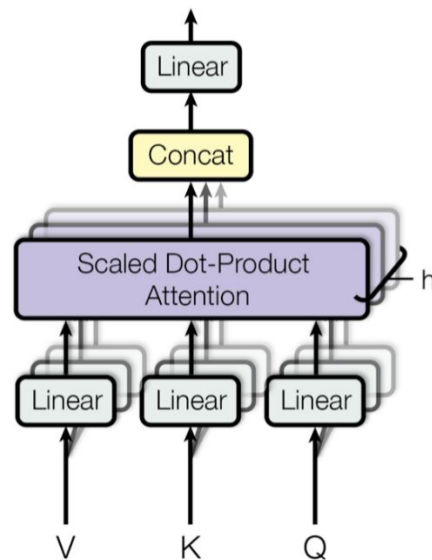
- Not all the tokens are necessary to generate the final results.
- Unimportant tokens can be removed progressively as an input sequence passes through transformer layers.

New Pruning Dimension: Head

Pruning

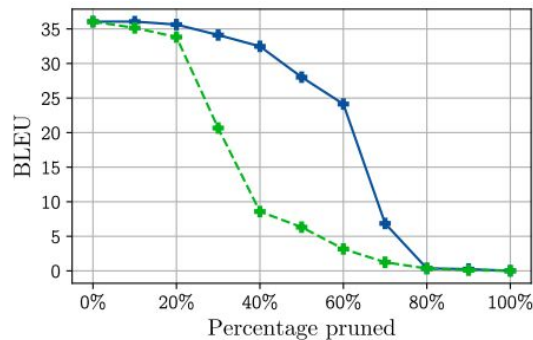
- In addition to the valewise and channelwise pruning, transformer allows for a new type of pruning: multi-head pruning.

$(1, 197, 768) \rightarrow (1, 12, 197, 64) \rightarrow (1, 4, 197, 64)$
Input Input with 12 heads Input with 4 heads

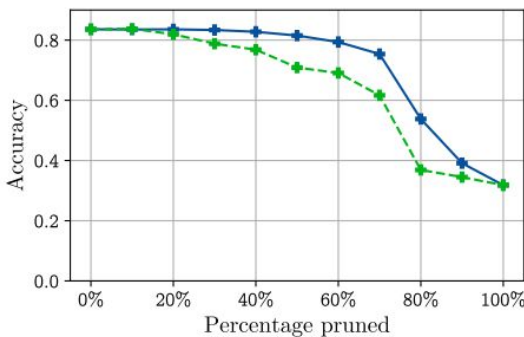


Multi-headed Attention

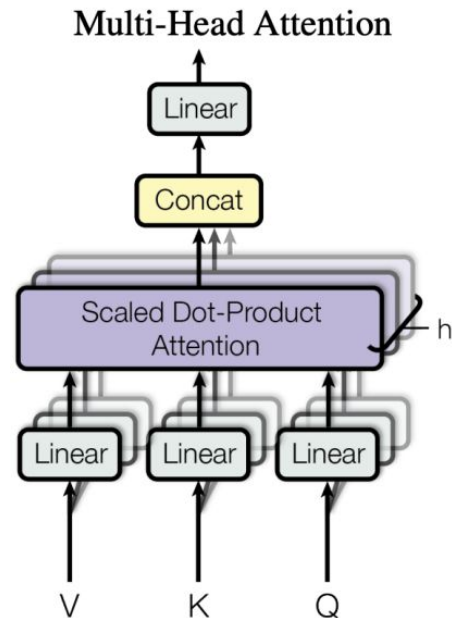
- We observe that the majority of attention heads can be removed without deviating too much from the original score. Surprisingly, in some cases removing an attention head results in an increase in BLEU/accuracy.



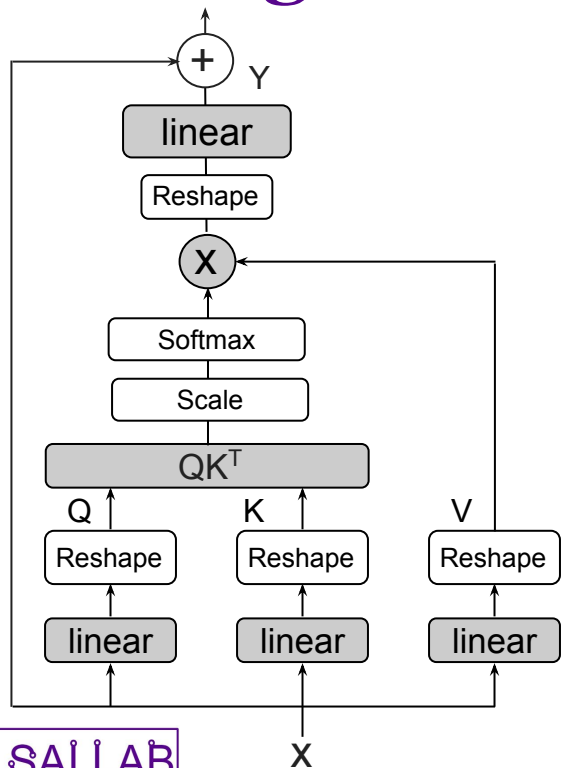
(a) Evolution of BLEU score on newstest2013 when heads are pruned from WMT.



(b) Evolution of accuracy on the MultiNLI-matched validation set when heads are pruned from BERT.



Pruning on Transformers: Head Pruning



- X_i is an embedded vector of i th token.
- There are in total n tokens.
- The output vector of q th token can be expressed as:

$$\text{Att}_{W_k, W_q, W_v, W_o}(\mathbf{x}, q) = W_o \sum_{i=1}^n \alpha_i W_v x_i$$

$$\text{where } \alpha_i = \text{softmax} \left(\frac{q^T W_q^T W_k x_i}{\sqrt{d}} \right)$$

- If we further expressed with multi-head attention, the output vector can be expressed as:

$$\text{MHAtt}(\mathbf{x}, q) = \sum_{h=1}^{N_h} \xi_h \text{Att}_{W_k^h, W_q^h, W_v^h, W_o^h}(\mathbf{x}, q)$$

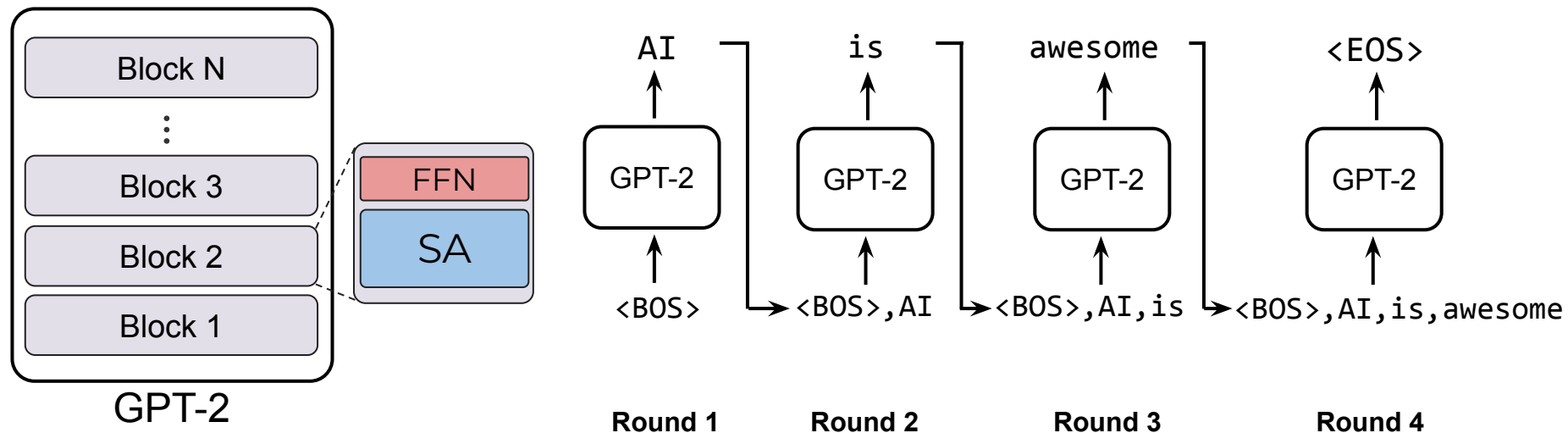
- Where the ξ_h are mask variables with values in $\{0, 1\}$.

$$I_h = \mathbb{E}_{x \sim X} \left| \frac{\partial \mathcal{L}(x)}{\partial \xi_h} \right|$$

Topics

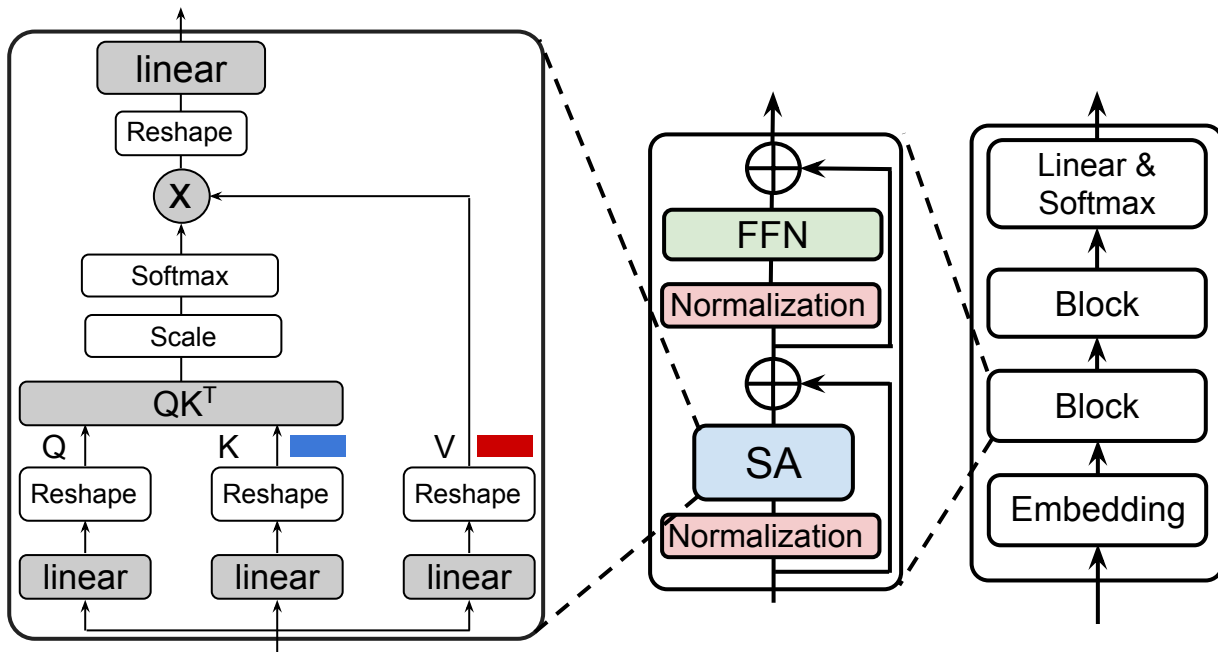
- Pruning in CNN and transformers
 - Reduce running cost
 - Reduce storage
- Sparsity encoding
- General pruning techniques
- Transformer pruning
- Large model pruning

Pruning on Large Models: KV cache Pruning



- Each token is generated in an autoregressive manner.

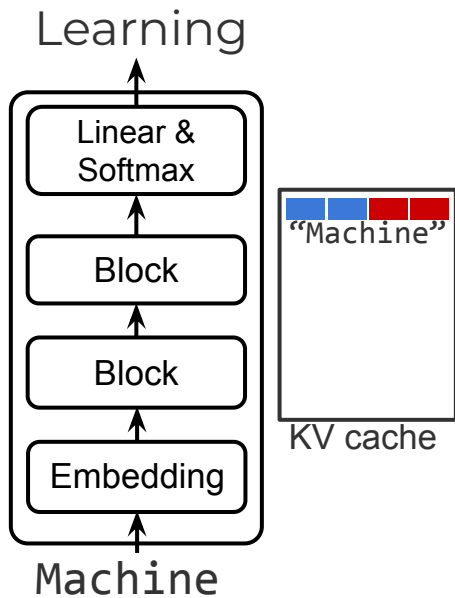
Pruning on Large Models: KV cache Pruning



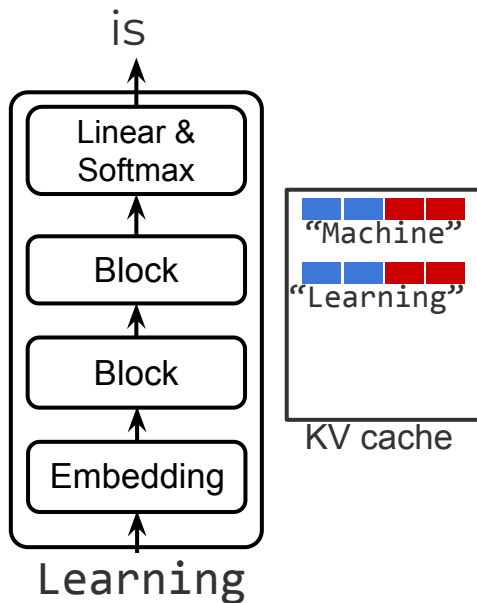
- We need to buffer the v and k for later usage.

Pruning on Large Models: KV cache Pruning

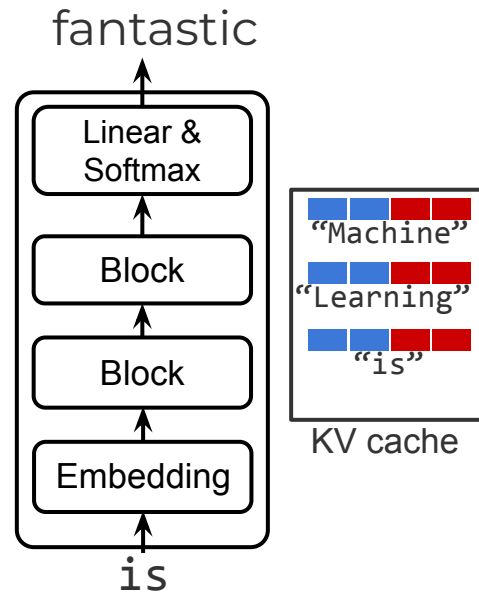
■ Key vectors ■ Value vectors



Round 1

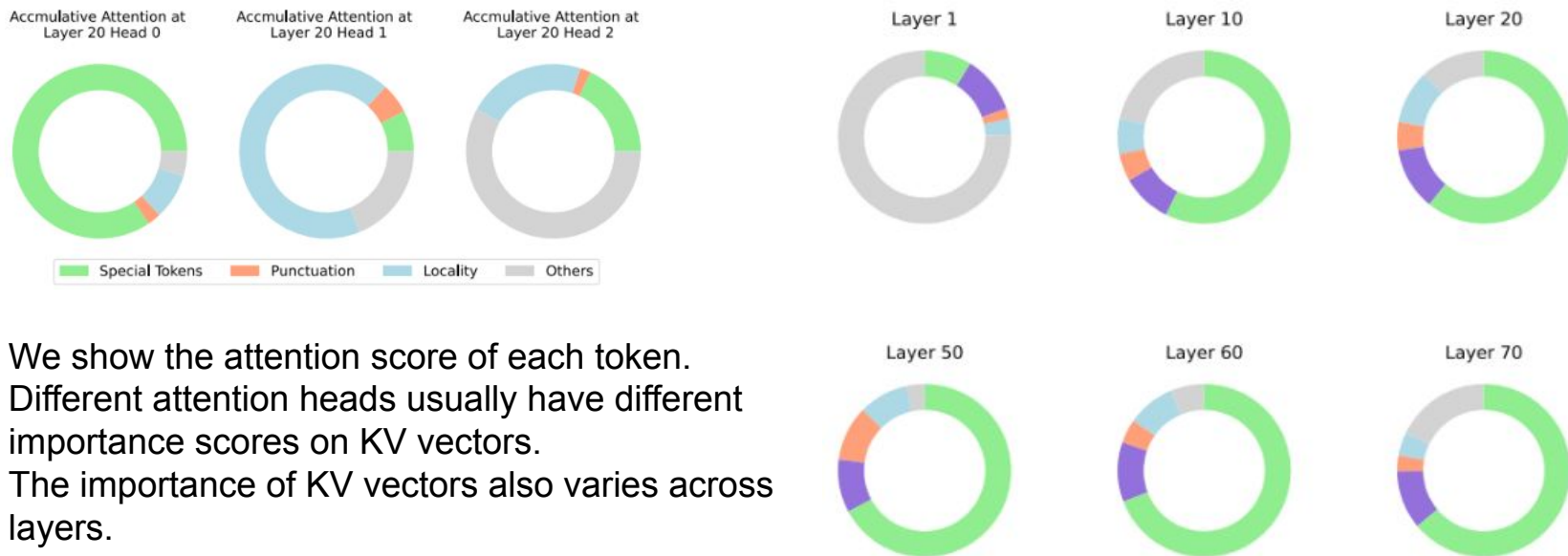


Round 2



Round 3

Pruning on Large Models: KV Cache Pruning

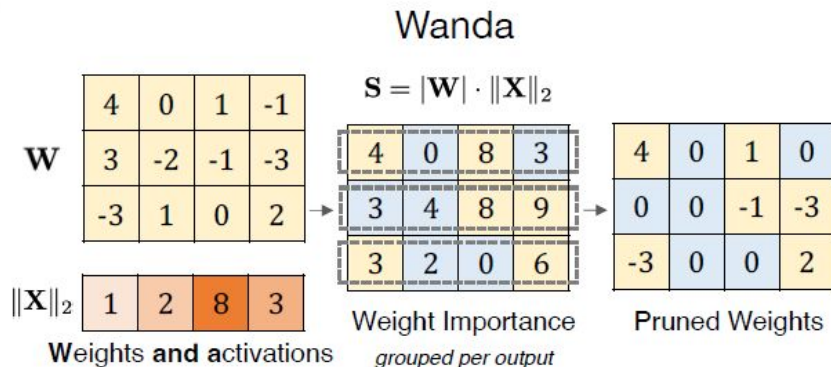
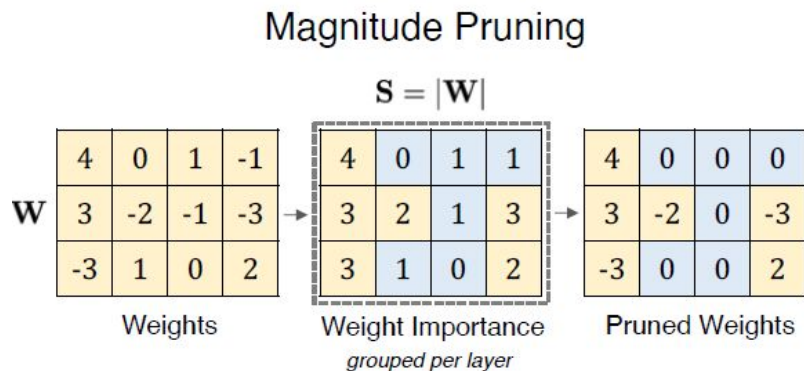


- We show the attention score of each token.
- Different attention heads usually have different importance scores on KV vectors.
- The importance of KV vectors also varies across layers.

Ge, Suyu, et al. "Model tells you what to discard: Adaptive kv cache compression for llms." *arXiv preprint arXiv:2310.01801* (2023).

Tianhua Xia, Sai Qian Zhang, "Kelle: Co-design KV Caching and eDRAM for Efficient LLM Serving in Edge Computing", 58th IEEE/ACM International Symposium on Microarchitecture (MICRO-58), 2025.

LLM Pruning: Wanda



- A zero-shot pruning method.
- Prune the weights by considering the input statistics.
- For each weight, if the corresponding input's magnitude is large, the output will also be large.
- Need some training samples for calibration.