On-Device AI 실습
Quantization for LLM

Youngmin Jeon

Overview

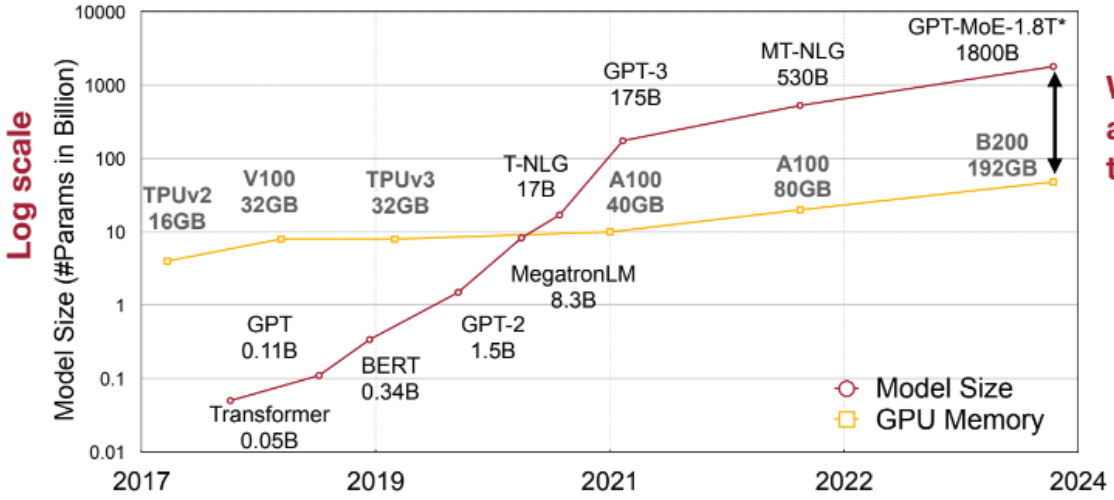
1. **Weight-only Quantization**
    1. AWQ

2. **Weight and Activation Quantization**
    1. SmoothQuant
    2. QuaRot/SpinQuant

Challenge for LLM deployment

Despite being powerful, LLMs are hard to serve on the edge

- LLM sizes and computation are increasing exponentially.
- Domain-specific accelerator alone is not enough.
  - We need model compression techniques and system support to bridge the gap.



We need efficient algorithms and systems to bridge the gap.

AWQ: Activation-aware Weight Quantization

Observation: Weights are not equally important; 0.1% salient weights

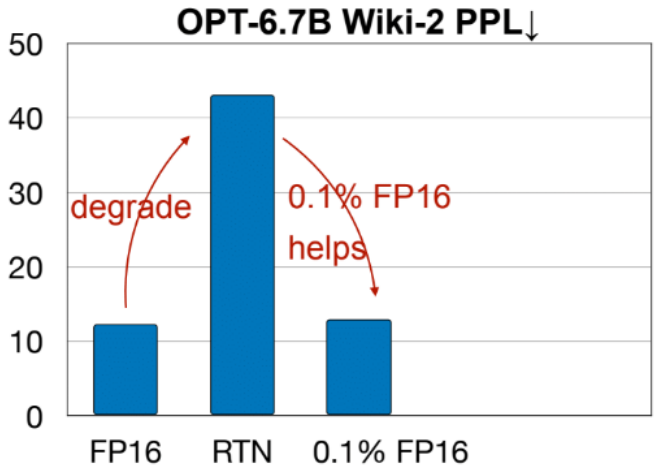
W_{FP16}

+1.2	−0.2	−2.4	−3.4
−2.5	−3.5	+1.9	+1.4
−0.9	+1.6	−2.5	−1.9
−3.5	+1.5	+0.5	−0.1
+1.8	−1.6	−3.2	−3.4
+2.4	−3.5	−2.8	−3.9
+0.1	−3.8	+2.4	+3.4
+0.9	+3.3	−1.9	−2.3

RTN →

$Q(W)_{MixPrec}$

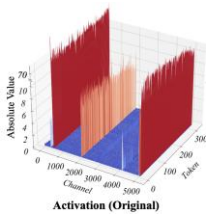
+1	+0	−2	−3	
−2.5	−3.5	+1.9	+1.4	FP16 channel
−1	+2	−3	−2	
−4	+2	+1	+0	
+2	−2	−3	−3	
+2	−4	−3	−4	
+0	−4	+2	+3	
+1	+3	−2	−2	



- We find that weights are not equally important, keeping **only 0.1%** of salient weight channels in FP16 can greatly improve perplexity
- But how do we select salient channels? Should we select based on weight magnitude?

Protect 1% salient channels

Salient weights are determined by activation distribution, not weight



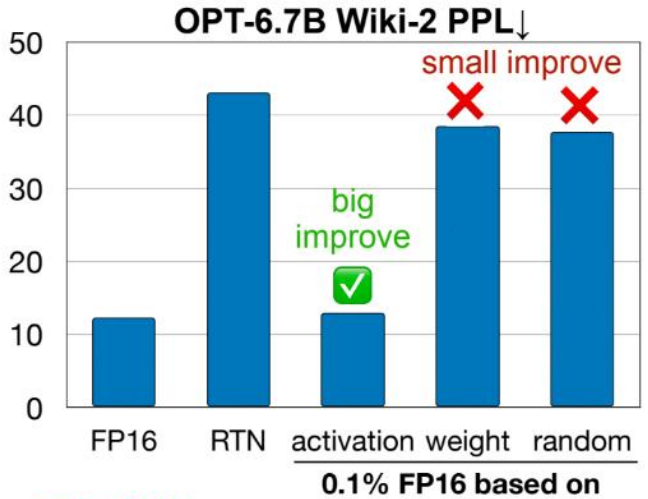
determine the salient weights by activation

X

Q(W)MixPrec

+1	+0	−2	−3
−2.5	−3.5	+1.9	+1.4
−1	+2	−3	−2
−4	+2	+1	+0
+2	−2	−3	−3
+2	−4	−3	−4
+0	−4	+2	+3
+1	+3	−2	−2

FP16 channel



- We find that weights are not equally important, keeping **only 0.1%** of salient weight channels in FP16 can greatly improve perplexity
- But how do we select salient channels? Should we select based on weight magnitude?
- No! We should look for **activation distribution, but not weight!**

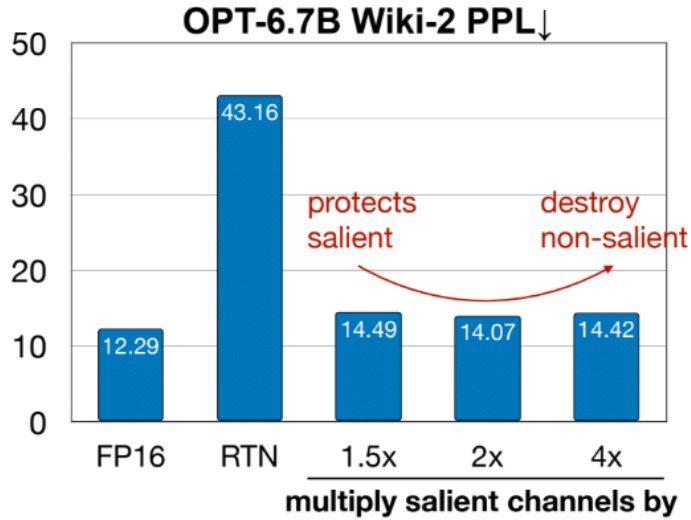
Scale 0.1% salient channels

Protecting salient weights by scaling (no mixed prec.)

$$Q\left(\begin{array}{c|cccc} & \mathbf{W} & & & \\ \hline & +1.2 & -0.2 & -2.4 & -3.4 \\ & -2.5 & -3.5 & +1.9 & +1.4 \\ & -0.9 & +1.6 & -2.5 & -1.9 \\ & -3.5 & +1.5 & +0.5 & -0.1 \\ & +1.8 & -1.6 & -3.2 & -3.4 \\ & +2.4 & -3.5 & -2.8 & -3.9 \\ & +0.1 & -3.8 & +2.4 & +3.4 \\ & +0.9 & +3.3 & -1.9 & -2.3 \end{array} \begin{array}{c} \\ \times 1 \\ \times 2 \\ \times 1 \\ \times 1 \\ \times 1 \\ \times 1 \\ \times 1 \\ \times 1 \end{array}\right)$$

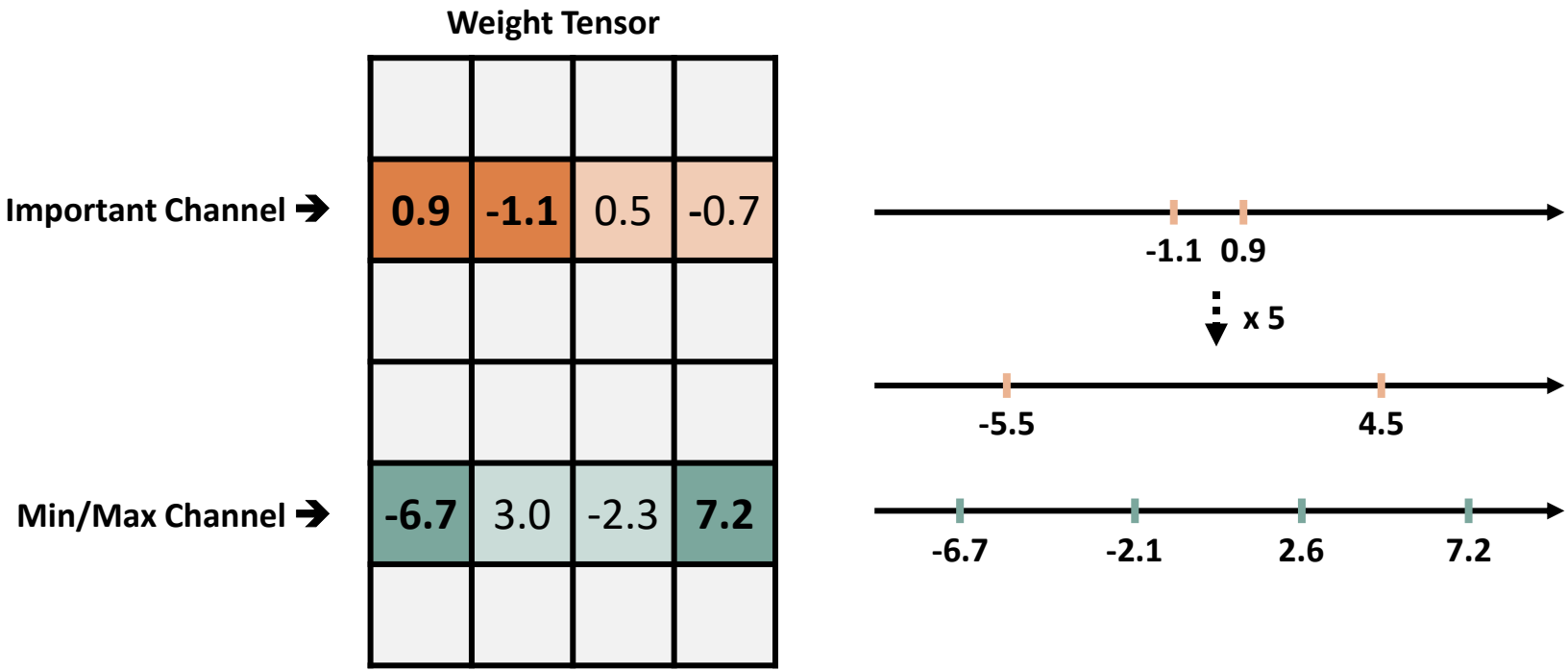
fuse to previous op

$$\mathbf{WX} \longrightarrow Q(\mathbf{W} \cdot \mathbf{s})(\mathbf{s}^{-1} \cdot \mathbf{X})$$



- Multiplying the salient channels with $s > 1$ reduces its quantization error
- Why?

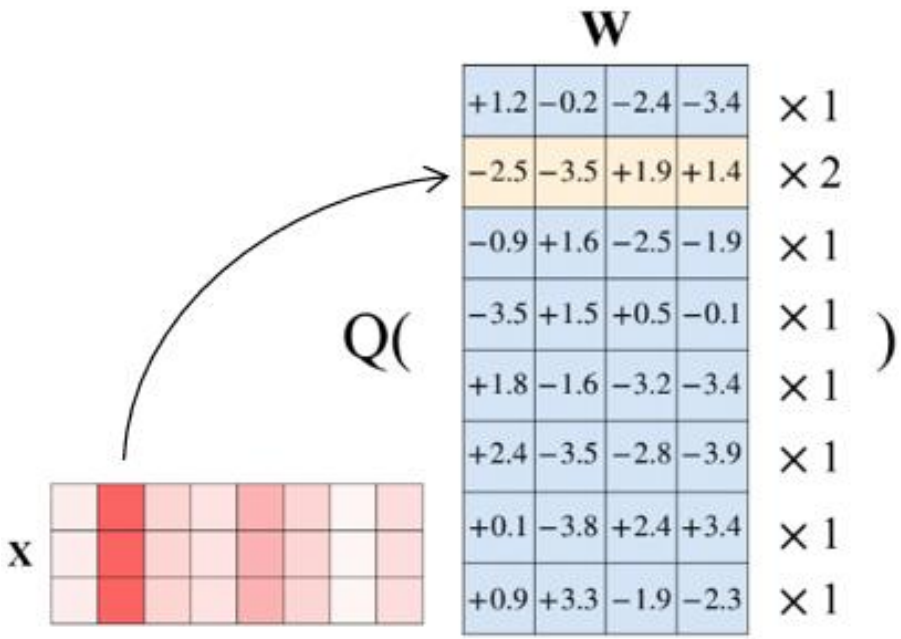
Scale salient channels



Scale 1% salient channels

- **Protecting salient weights by scaling**
  - $y = \mathbf{W}x$ ➔ $\widetilde{\mathbf{W}} = \Delta \cdot \mathrm{Round}\left(\frac{\mathbf{W}}{\Delta}\right), \Delta = \frac{\max(\mathbf{W})}{2^{N-1}}$
    - $\mathrm{Error} = \Delta \cdot \mathrm{RoundError}$
  - $y = (\mathbf{W} \cdot s)(x/s)$ ➔ $\widetilde{\mathbf{W} \cdot s} = \Delta' \cdot \mathrm{Round}\left(\frac{\mathbf{W} \cdot s}{\Delta'}\right)$
    - $\mathrm{Error}' = \Delta' \cdot \mathrm{RoundError} \cdot \frac{1}{s} = \Delta \cdot \mathrm{RoundError} \cdot \frac{1}{s} = \mathrm{Error} \cdot \frac{1}{s}$
    - $\max(\mathbf{W}) = \max(\mathbf{W} \cdot s)$ ➔ $\Delta' = \Delta$

[실습1] Scale 1% salient channels



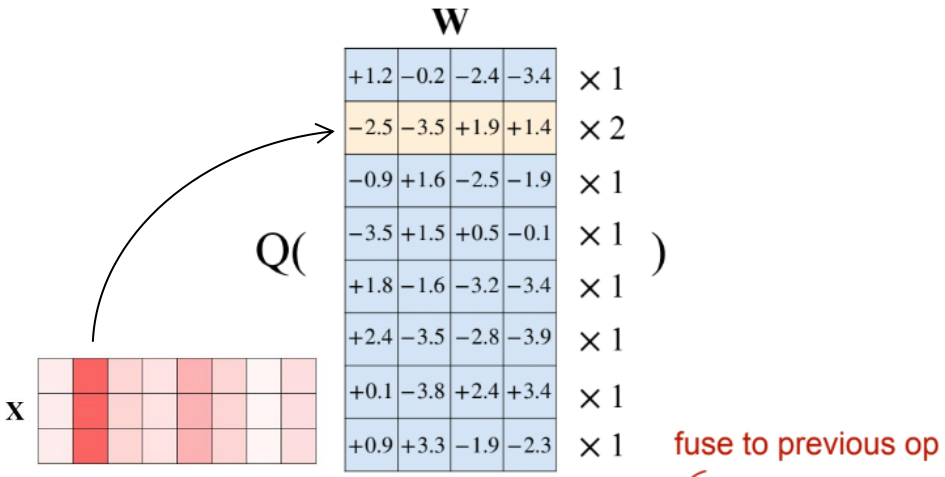
$$\mathbf{WX} \longrightarrow Q(\mathbf{W} \cdot \mathbf{s})(\mathbf{s}^{-1} \cdot \mathbf{X})$$

Answer


```
################ YOUR CODE STARTS HERE ################

# Step 1: importance를 기준으로 1%의 중요한 채널을 찾으세요  (hint: use torch.topk())
# hint : torch.topk() 함수를 사용하세요. torch.topk() 함수는 PyTorch에서 텐서의 값 중 상위 k개의 값과 그들의 인덱스를 반환하는 함수입니다.
outlier_mask = torch.topk(importance, int(len(importance) * 0.01))[1]
assert outlier_mask.dim() == 1

################ YOUR CODE ENDS HERE ##################

# 스케일 팩터를 적용하는 것을 시뮬레이션하기 위해, 양자화 전에 스케일 팩터를 곱하고, 양자화 후에 스케일 팩터로 나눕니다.
# scale_factor를 이용해 중요한 가중치 채널의 값을 확대합니다.
m.weight.data[:, outlier_mask] *= scale_factor

m.weight.data = pseudo_quantize_tensor(m.weight.data, n_bit=w_bit, q_group_size=q_group_size)

################ YOUR CODE STARTS HERE ################

# Step 2: scale_factor를 이용해 중요한 가중치 채널의 값을 다시 축소하세요.
m.weight.data[:, outlier_mask] /= scale_factor

################ YOUR CODE ENDS HERE ##################
```

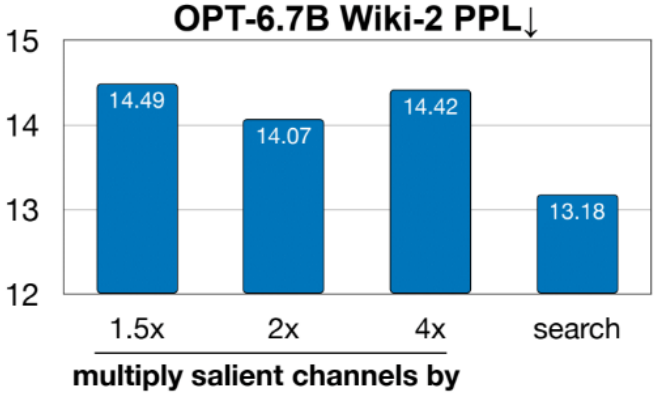
Scale Factor Search

Protecting salient weights by scaling (no mixed prec.)



fuse to previous op

$$\mathbf{WX} \longrightarrow Q(\mathbf{W} \cdot \mathbf{s})(\mathbf{s}^{-1} \cdot \mathbf{X})$$



$$\mathcal{L}(\mathbf{s}) = \|Q(\mathbf{W} \cdot \mathbf{s})(\mathbf{s}^{-1} \cdot \mathbf{X}) - \mathbf{WX}\|$$

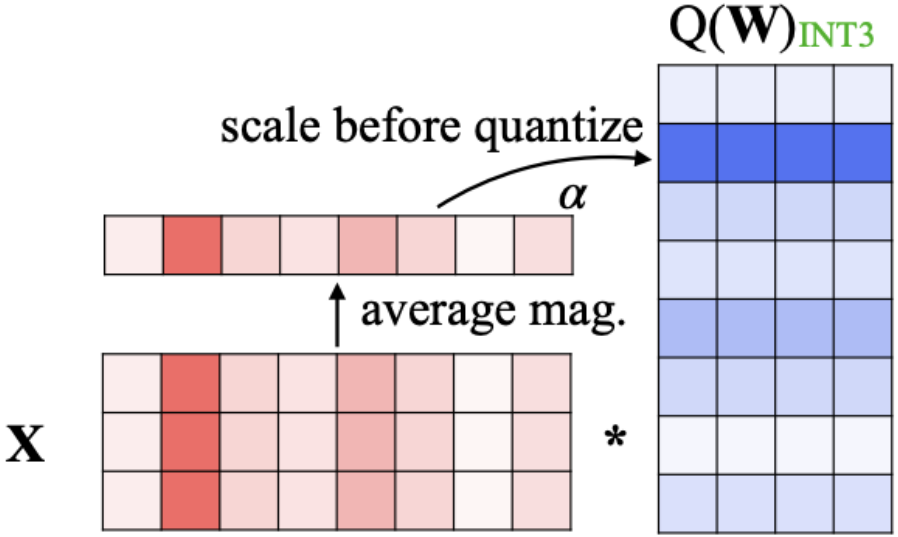
$$\mathbf{s} = \mathbf{s_X}^{\alpha}$$

Activation-awareness is important, but not weight-awareness

- Multiplying the salient channels with $s > 1$ reduces its quantization error
- Take a data-driven approach with a fast **grid search**

L1 norm of Activation

[실습2] Scale Factor Search



$$\mathbf{s}_{\mathbf{X}^{\alpha}} = \|\boldsymbol{X}\|_{\mathbf{1}}, \qquad \|\boldsymbol{X}\|_{1} = \boldsymbol{\Sigma}\,|\boldsymbol{X}_{\mathrm{i}}|$$

$$\mathbf{s} = \mathbf{s}_{\mathbf{X}}{}^{\alpha}$$

Activation-awareness is important, but not weight-awareness

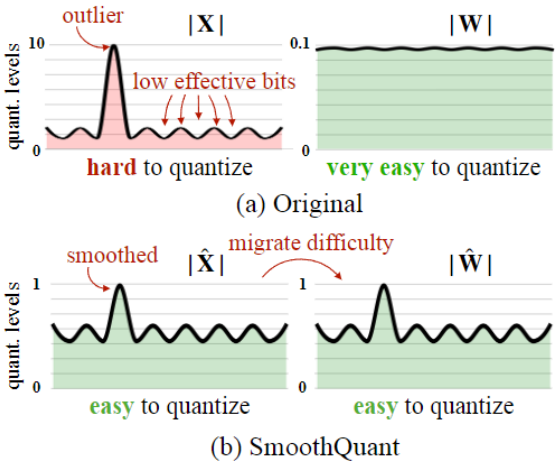
- Multiplying the salient channels with $s > 1$ reduces its quantization error
- Take a data-driven approach with a fast **grid search**

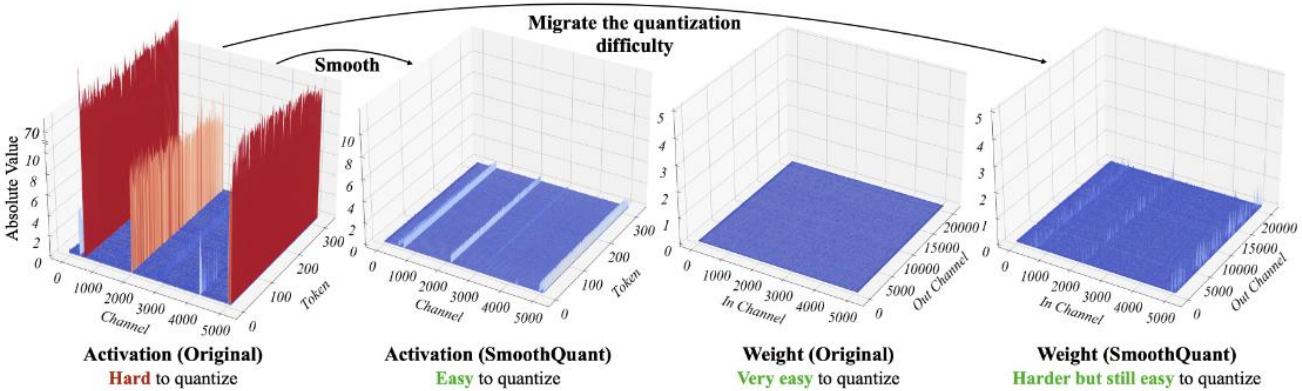
Answer


```
############### YOUR CODE STARTS HERE ###############

# Step 2: 공식에 따라 스케일 계산: scales = s_x^ratio
scales = s_x ** ratio # must clip the s_x, otherwise will get nan later

assert scales.shape == s_x.shape

############### YOUR CODE ENDS HERE #################

scales = scales / (scales.max() * scales.min()).sqrt().view(1, -1)

for fc in linears2scale:

    scales = scales.to(fc.weight.device)

    # scale_factor를 이용해 중요한 가중치 채널의 값을 확대합니다.
    fc.weight.mul_(scales)

    fc.weight.data = pseudo_quantize_tensor(fc.weight.data, w_bit, q_group_size)

    ############### YOUR CODE STARTS HERE ###############

    # Step 3: scale_factor를 이용해 중요한 가중치 채널의 값을 다시 축소하세요.
    fc.weight.data /= scales

    ############### YOUR CODE ENDS HERE #################
```

SmoothQuant

- **SmoothQuant's intuition**
  - Migrate **scale** from activations to weights W before quantization





SmoothQuant

- **Main idea of SmoothQuant**

$$\mathbf{Y} = \mathbf{X} \cdot \mathbf{W}, \mathbf{Y} \in \mathbb{R}^{T \times C_o}, \mathbf{X} \in \mathbb{R}^{T \times C_i}, \mathbf{W} \in \mathbb{R}^{C_i \times C_o},$$

$$\mathbf{s}_j = \max(|\mathbf{X}_j|)^{\alpha} / \max(|\mathbf{W}_j|)^{1-\alpha}$$

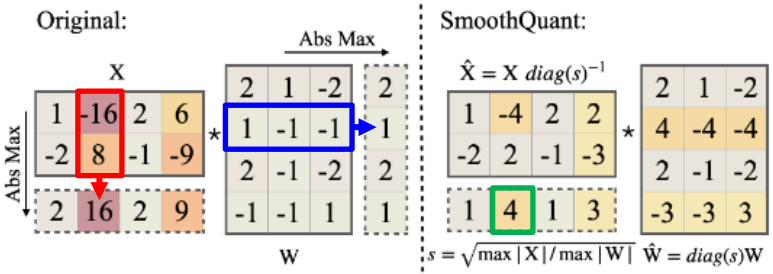


Figure 5: Main idea of SmoothQuant when α is 0.5. The smoothing factor s is obtained on calibration samples and the entire transformation is performed offline. At runtime, the activations are smooth without scaling.

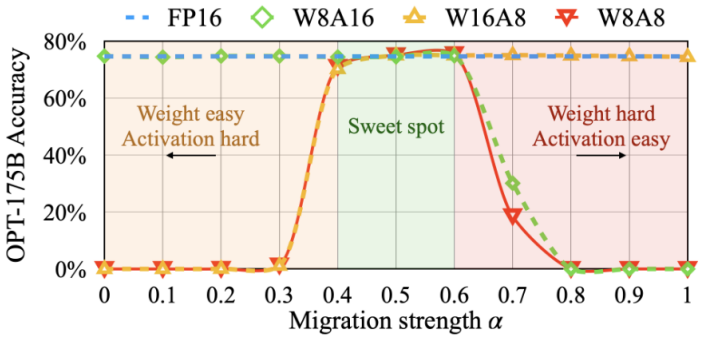
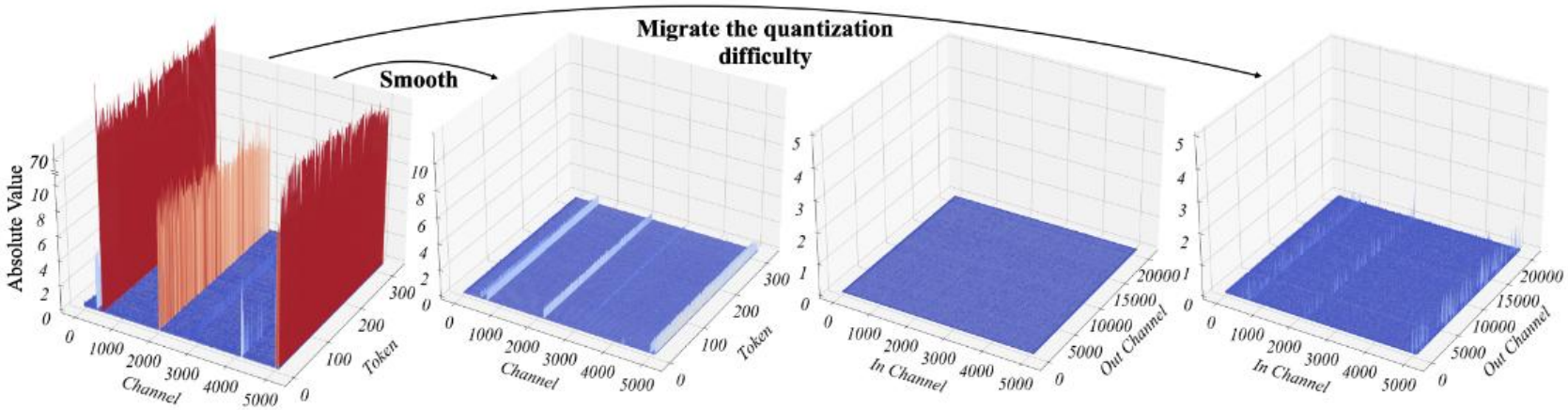


Figure 10: A suitable migration strength α (sweet spot) makes both activations and weights easy to quantize. If the α is too large, weights will be hard to quantize; if too small, activations will be hard to quantize.

[실습3] Quantization Difficulty Migration



Answer


```python
def smooth_ln_fcs_by_scale(ln, fcs, scale):
    if not isinstance(fcs, list):
        fcs = [fcs]
    assert isinstance(ln, nn.LayerNorm)
    for fc in fcs:
        assert isinstance(fc, nn.Linear)
    ############### YOUR CODE STARTS HERE ###############
    # Step 1: layernorm의 weight와 bias를 scale로 나누어주세요. (hint: div_()함수를 통해 tensor 전체를 특정한 값으로 나누어 줄 수 있습니다.)
    ln.weight.div_(scale)
    ln.bias.div_(scale)
    ################ YOUR CODE ENDS HERE ################

    for fc in fcs:
        ############### YOUR CODE STARTS HERE ###############
        # Step 2: fc의 weight에 scale을 곱해주세요. (hint: mul_()함수를 통해 tensor 전체에 특정한 값을 곱해 줄 수 있습니다.)
        fc.weight.mul_(scale)
        ################ YOUR CODE ENDS HERE ################
```

[실습4] Scale Factor Search

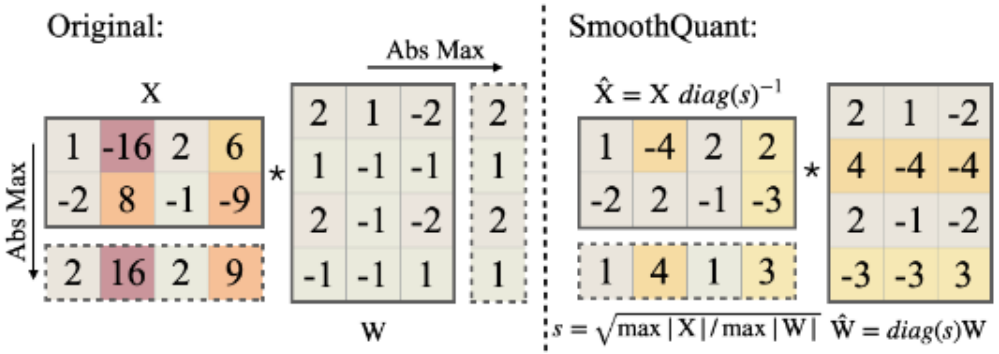


Figure 5: Main idea of SmoothQuant when α is 0.5. The smoothing factor s is obtained on calibration samples and the entire transformation is performed offline. At runtime, the activations are smooth without scaling.

Answer

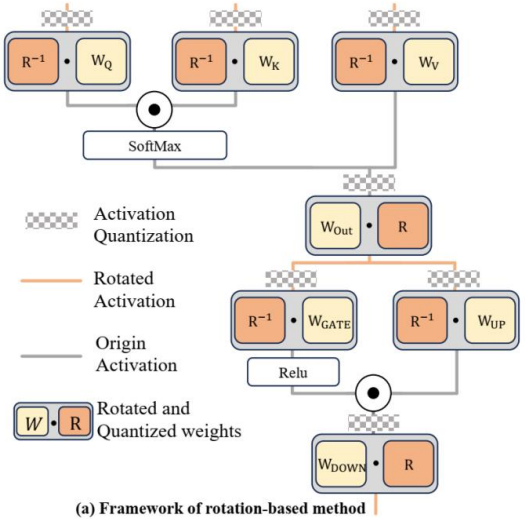

```
scales = (
    ############### YOUR CODE STARTS HERE ###############
    #Activation Scales 값과 Weight Scales 값에 alpha를 적절히 거듭제곱해주어야 합니다.
    #Hint: pow()함수를 통해서 거듭제곱을 사용할 수 있습니다.
    (act_scales.pow(alpha) / weight_scales.pow(1 - alpha))
    ############### YOUR CODE ENDS HERE ################
)
```

Ratation based Quantization

$$\mathbf{Y} = (\mathbf{XR})\left(\mathbf{R^{-1}W^{-1}}\right) = \mathbf{XW^{T}}$$

A rotation matrix is an orthogonal matrix $\mathbf{R}$ satisfied $\mathbf{RR^{T}} = 1$ and $|\mathbf{R}| = 1$

QuaRot



(a) Framework of rotation-based method

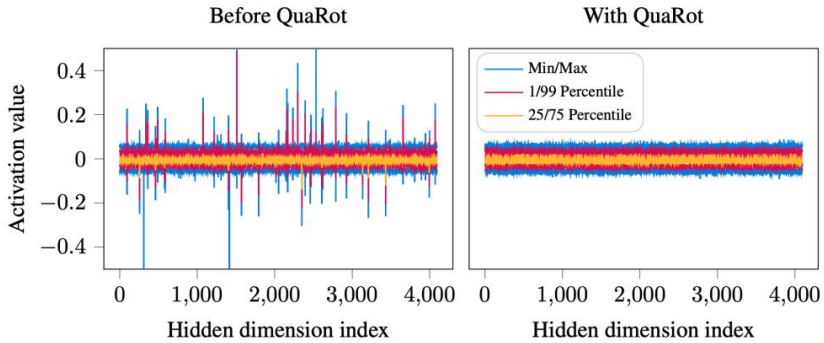
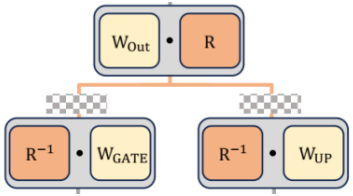


Figure 1: The distributions of activations at the input to the FFN block in LLAMA2-7B model, in the tenth layer. Left: using the default configuration as downloaded from Hugging Face. Right: after processing using QuaRot. The processed distribution has no outliers, leading to superior quantization.

Layernorm ⇔ Linear Fusion

- $\boldsymbol{y = x \cdot W_{out} \cdot R \cdot R^{-1} \cdot W_{GATE} = x \cdot W_{out} \cdot W_{GATE}}$

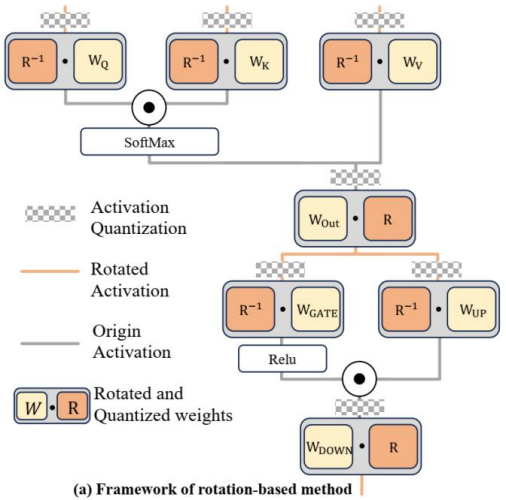


- **In real case,** $\boldsymbol{y = LN(x \cdot W_{Out} \cdot R) \cdot R^{-1} \cdot W_{GATE} \neq x \cdot W_{out} \cdot W_{GATE}}$
  - $LN(x) = x \cdot \gamma + \beta$
  - $LN(x) \cdot W + b = (x \cdot \gamma + \beta) \cdot W + b$
    $= x \cdot \gamma W + \beta W + b = x \cdot \widetilde{W} + \tilde{b}$
  - $y = LN(x \cdot W_{Out} \cdot R) \cdot R_{-1} \cdot \widetilde{W_{GATE}} = x \cdot W_{out} \cdot \widetilde{W_{GATE}}$

[실습5] Rotate Matrix 적용

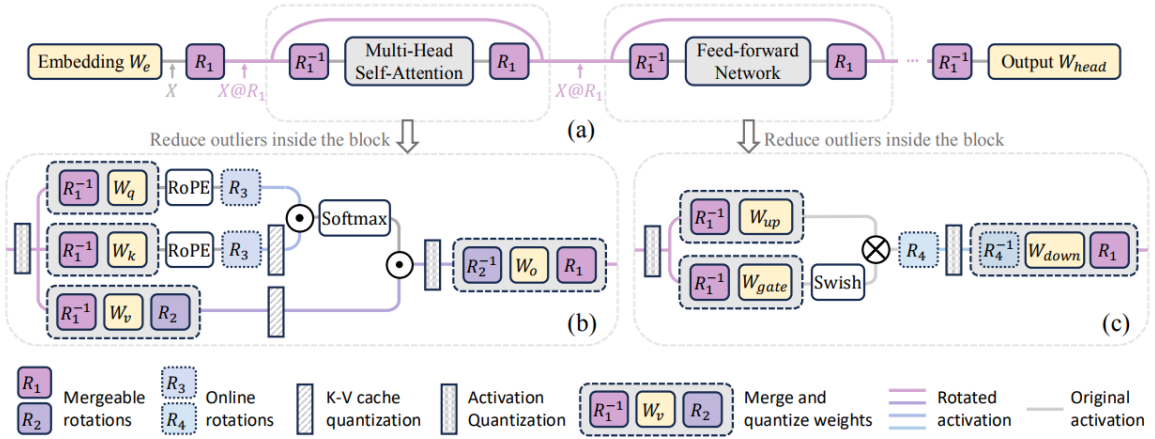
- nn.Linear 연산은 W^T 형태로 저장
  - $W^T \cdot R$ ➔ 연산시에 $R^T \cdot W$
- Embedding Parameter Shape :
  (Num_Tokens, Hidden_dim)
- Linear Parameter Shape :
  (Output_Channel, Input_Channel)
- Roation Matrix Shape :
  (Hidden_dim, Hidden_dim)

QuaRot



(a) Framework of rotation-based method

SpinQuant



Answer


```
############### YOUR CODE STARTS HERE ###############
# Pytorch에서 @ 연산이 Dot Product 임을 사용하시기 바랍니다.
# nn.Linear 연산의 Parameter는 W^T 형태로 저장되어 있다는 것을 유의하시기 바랍니다.
# Embedding Parameter Shape : (Num_Tokens, Hidden_dim)
# Linear Parameter Shape : (Output_Channel, Input_Channel)
# Roation Matrix Shape : (Hidden_dim, Hidden_dim)

if isinstance(m, nn.Embedding):
  W_ = m.weight.data
  m.weight.data = W_ @ R1

if isinstance(m, nn.Linear):
  if "out_proj" in n or "fc2" in n:
    # Att Out Proj, FFN Down Proj
    W_ = m.weight.data
    m.weight.data = R1.T @ W_      ⇒ x · W · R

  else:
    # QKV Proj, FFN Up Proj, FFN Gate Proj
    W_ = m.weight.data
    m.weight.data = W_ @ R1        ⇒ x · R^T · W

############### YOUR CODE ENDS HERE #################
```