Hadamard Transform for Post-Training Quantization (PTQ)

1. What is a Hadamard Matrix?

- A Hadamard matrix $H \in \{\pm 1\}^{n \times n}$ satisfies

$$HH^\top = nI_n.$$

- Up to a scale, H is an **orthogonal transform**: it is invertible, energy-preserving, and does not lose information.
- Intuition: it **spreads the energy** of a vector across dimensions, instead of letting it concentrate on a few coordinates.

2. Why Hadamard Helps Quantization?

- In deep models, activations and weights often contain **rare but very large outliers**:
  - These outliers are usually **semantically important** (e.g., gating certain tokens, making spaces or special symbols effectively ignored).
  - In post-training quantization (PTQ), they **force a very wide quantization range**, so most "normal" values are represented poorly.
- Hadamard transform:
  - Rotates the representation so that outlier energy is **distributed across channels**.
  - Reduces the **maximum absolute value per channel**, enabling a smaller quantization step size.
  - Does **not change the underlying function** of the layer if we transform consistently and invert properly.

3. Zero-Overhead Trick: Folding Hadamard into Weights

Consider a linear layer (or a convolution reshaped as a matrix)

$$y = xW,$$

where x is the activation and W is the weight.

- Choose a Hadamard matrix $H \in \mathbb{R}^{C \times C}$ satisfying

$$H^{-1} = \frac{1}{C} H^\top.$$

- Define transformed activation and weight:

$$\tilde{x} = xH, \qquad \tilde{W} = H^{-1}WH.$$

- Then

$$\tilde{y} = \tilde{x}\tilde{W} = xHH^{-1}WH = xWH.$$

Key idea:

- The transformed output $\tilde{y}$ is just the original output in a **rotated basis** (post-multiplied by H).
- We can fold H and H^{-1} into the weights offline, so at inference time:
  - Only the very first input may need a Hadamard transform (or this can even be folded into preprocessing).
  - All later layers remain standard convolutions / matrix multiplications: **no extra runtime cost**.

4. Quantization in Hadamard Space

4.1 Standard PTQ (Baseline)

Symmetric quantization of activations (example):

$$x_{\text{int}} = \text{round}\left(\frac{x}{s_x}\right), \qquad x_q = s_x \cdot x_{\text{int}},$$

and symmetric quantization of weights:

$$W_{\text{int}} = \text{round}\left(\frac{W}{s_w}\right), \qquad W_q = s_w \cdot W_{\text{int}}.$$

4.2 Hadamard-Aware PTQ

1. Apply Hadamard transform:

$$\tilde{x} = xH, \qquad \tilde{W} = H^{-1}WH.$$

2. Quantize in the transformed domain:

$$\tilde{x}_q = \mathcal{Q}(\tilde{x}), \qquad \tilde{W}_q = \mathcal{Q}(\tilde{W}),$$

   where $\mathcal{Q}(\cdot)$ denotes symmetric quantization.

3. Use $(\tilde{x}_q, \tilde{W}_q)$ in place of (x, W).

Benefits:

- Because Hadamard "smooths out" outliers, $\max|\tilde{x}|$ and $\max|\tilde{W}|$ are often smaller.
- This allows tighter quantization ranges and lower MSE between floating-point and quantized outputs.

5. Weight–BN Fusion and α-Scaling (Weight Combine)

To further reduce quantization error, we combine:

5.1 Conv + BN Fusion

For a convolution followed by batch normalization, we fuse into an effective weight and bias:

$$W_{\text{fused}} = \gamma \frac{W}{\sqrt{\sigma^2 + \epsilon}}, \qquad b_{\text{fused}} = \beta - \gamma \frac{\mu}{\sqrt{\sigma^2 + \epsilon}},$$

where (μ, σ^2) are the BN running mean/variance and (γ, β) are the BN scale/shift.

5.2 Introducing a Scalar α (Weight Combine)

We introduce a scalar factor α to further shrink the dynamic range before quantization:

$$W_{\text{fused}} \leftarrow \alpha \cdot W_{\text{fused}}.$$

In our code:

- We fuse Conv+BN for the chosen 8×8 target layer.
- Apply Hadamard transform and α-scaling on the fused weight.
- Then compare **MSE of the quantized output**:

$$\text{Baseline PTQ} \quad \text{vs.} \quad \text{Hadamard} + \alpha \text{ PTQ}.$$

6. Experimental Setup (ResNet-20 on CIFAR-10)

- **Model:** ResNet-20 trained on CIFAR-10 (test accuracy $> 90\%$).
- **Layer:** We pick a specific convolution layer and compute:
  - Full-precision reference output y_{ref}.
  - Baseline 4-bit PTQ on (x, W) to obtain y_{base}.
  - Hadamard+α: transform (x, W), then apply 4-bit PTQ to obtain y_{had}.
- **Metric:** Mean Squared Error (MSE) between quantized output and full-precision output:

$$\text{MSE}_{\text{base}} = \mathbb{E}\big[\|y_{\text{base}} - y_{\text{ref}}\|^2\big], \qquad \text{MSE}_{\text{had}} = \mathbb{E}\big[\|y_{\text{had}} - y_{\text{ref}}\|^2\big].$$

6.1 Observation on ResNet-20

- On the chosen ResNet layer, the Hadamard+α setting reduces MSE to about

$$\text{MSE}_{\text{had}} \approx 0.8 \times \text{MSE}_{\text{base}}$$

  (roughly a 20% reduction on average).
- However, the improvement is **not very stable**:
  - Different input batches and random seeds can change the ratio noticeably.
  - Sometimes Hadamard is clearly better; sometimes it is close to or slightly worse than the baseline.
- Likely reason: ResNet-20 is a small, relatively well-behaved network with weaker outlier problems, so the benefit of Hadamard smoothing is less consistent than what we expect on large models.

7. Key Takeaways (Poster Bullets)

- **Hadamard is an orthogonal, lossless transform** that redistributes activation/weight energy.
- By smoothing outliers, it can **reduce quantization range and error** in PTQ.
- With proper folding into weights, it introduces **zero extra runtime cost**.
- Combining **Conv+BN fusion** with a scalar α (weight combine) further refines dynamic range before quantization.
- On small networks (ResNet-20), improvements are modest and somewhat unstable, but the technique is **promising for large models and LLMs** where outliers dominate quantization behavior.

Optional Figure Caption

Fig: Hadamard transforms the activation/weight space to flatten out outliers before quantization, then is folded back into standard convolution layers, adding no inference-time overhead.