Course Materials for GEN-AI

Northeastern University

Instructor: Ramin Mohammadi
`r.mohammadi@northeastern.edu`

Thank you for your understanding and collaboration.

Regularization in CNNs

Previously in Regularization

- We learned about batch-norm, we saw that first they thought that the problem in terms of optimization is internal covariate shift, and they wanted to normalize each layer to take care of it. Then we learned that the intuition is not correct; the network works though. But batch norm doesn't fix covariate shift. We also learned that covariate shift actually doesn't hurt.
- What batch-norm does actually is to reduce the lippchitzness of the loss function & makes it L-smooth.
- One problem with batch-norm is that if batch size is small, the mean & variance are not reliable.

Technically what we do is finding mean, variance & scale the data then we add some shift to the rescaled.

Ideas to Get Around the Problem

Tensor Representation

- **Tensor:** $\mathbf{T} = (b, i_1, i_2, \ldots, i_N, c)$
  - b: batch size
  - c: number of channels
  - $i_1, i_2, \ldots, i_N$: input height, width, depth, etc.

Comparison of Normalization Methods

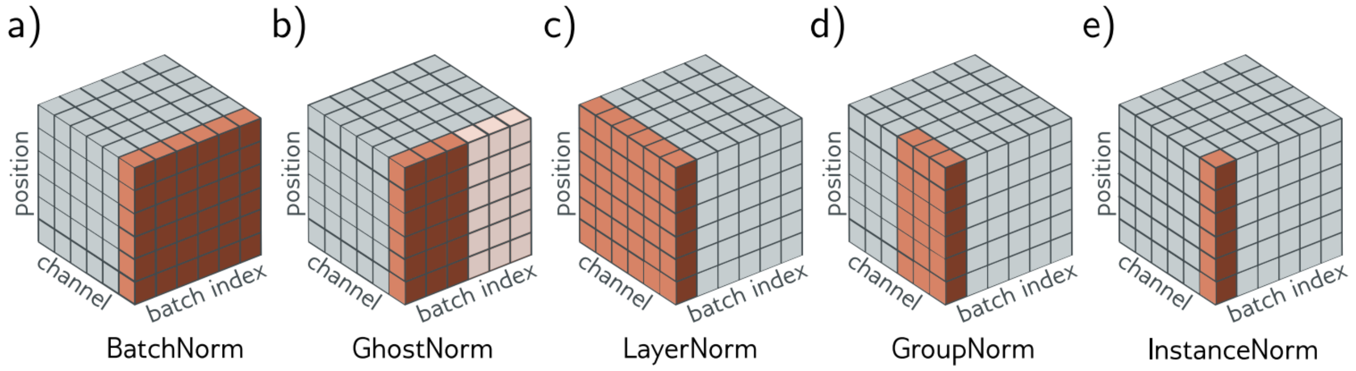


Figure 1: Normalization schemes. Colored pixels are normalized with the same mean and variance. BatchNorm modifies each channel separately but adjusts each batch member in the same way based on statistics gathered across the batch and spatial position. Ghost BatchNorm computes these statistics from only part of the batch to make them more variable. LayerNorm computes statistics for each batch member separately, based on statistics gathered across the channels and spatial position. It retains a separate learned scaling factor for each channel. GroupNorm normalizes within each group of channels and also retains a separate scale and offset parameter for each channel. InstanceNorm normalizes within each channel separately, computing the statistics only across spatial position. Adapted from Wu and He (2018).

Layer Normalization Formulas

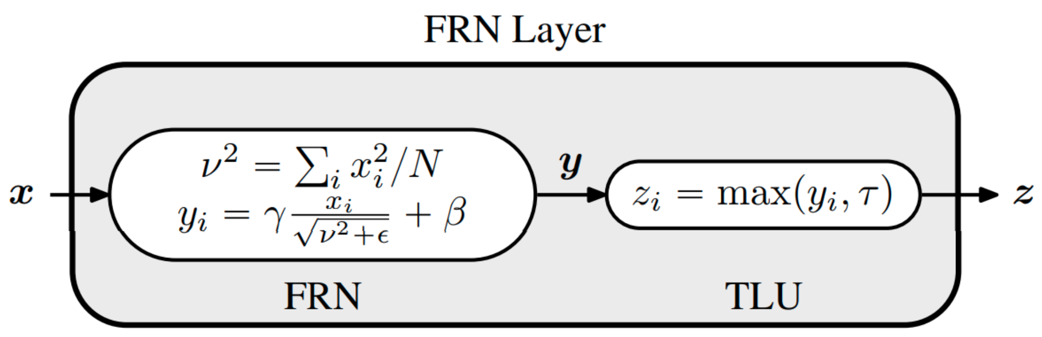
- Layer norm computes:

$$\mu_i = \frac{1}{|S_i|} \sum_{k \in S_i} z_k$$

$$\sigma_i^2 = \sqrt{\frac{1}{|S_i|} \sum_{k \in S_i} (z_k - \mu_i)^2 + \epsilon}$$

  - S_i: Set of elements pooled over/by i.
  - z_i: Individual element in the tensor.
  - ϵ: Small constant for numerical stability.

Filter Response Normalization (FRN)



FRN is a normalization technique designed for feed-forward convolutional neural networks (CNNs). It operates on the activation maps (filter responses) produced by convolutional layers. These activation maps are represented as a 4D tensor $\mathbf{X}$ with shape $[B, W, H, C]$, where:

- B: Mini-batch size.
- W, H: Spatial dimensions of the feature map.
- C: Number of output channels (filters used in convolution).

Normalization Process

1. **Filter Responses:** For each channel c in the b-th batch, the filter response vector is:

$$\mathbf{x} = \mathbf{X}_{b,:,:,c} \in \mathbb{R}^N, \quad \text{where } N = W \times H.$$

2. **Mean Squared Norm:** Compute the mean squared norm (ν^2) of $\mathbf{x}$:

$$\nu^2 = \frac{\sum_{i=1}^N x_i^2}{N}.$$

3. **Normalization:** Normalize $\mathbf{x}$ using the root mean square (RMS):

$$\hat{\mathbf{x}} = \frac{\mathbf{x}}{\sqrt{\nu^2 + \epsilon}},$$

   where $\epsilon > 0$ is a small constant added for numerical stability.

4. **Affine Transformation:** After normalization, apply a learnable affine transformation:

$$\mathbf{y} = \gamma \hat{\mathbf{x}} + \beta,$$

   where γ and β are learnable scaling and shifting parameters, respectively.

Thresholded Linear Unit (TLU)

To address the potential bias introduced by the lack of mean-centering in FRN, TLU modifies the standard ReLU activation function by introducing a learned threshold parameter τ.

Activation Function

The TLU is defined as:

$$z_i = \max(y_i, \tau),$$

where:

- y_i: The output of FRN after affine transformation.
- τ: A learnable parameter enabling the threshold to adapt during training.

Properties

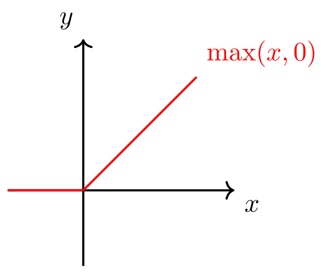
- **ReLU Relationship:** TLU can be expressed as:

  $$z_i = \max(y_i - \tau, 0) + \tau = \text{ReLU}(y_i - \tau) + \tau.$$

  This formulation ensures that the threshold does not overly penalize activations below the threshold while maintaining optimization stability.
- **Advantages:**
  - Mitigates issues with dead units and poor learning caused by biases in ReLU.
  - Improves performance by making the activation function more robust to initialization and training dynamics.

Effect on ReLU

- If your activation function is ReLU, you might encounter the **Dying ReLU** problem.
- Illustration of the problem:



- If the mean of the data is not zero, a shift introduced by TLU might move the positive values into the negative zone, causing their ReLU activations to become zero (Dying ReLU). The same issue occurs if all negatives become positive, turning ReLU into a linear function.

Key Insight

- In TLU, we introduce:

  $$y = \max(x, \tau)$$

- We learn τ (decision point) via backpropagation. This helps us understand where we should apply the shift.

Benefits of FRN

- **Stability:** Provides stable activation and gradients.
- **Robust Training:** Ensures robust training across various batch sizes and network architectures.

Cons

- Similar to batch-norm, they are trying to fix internal covariate shift. The method works, but it has been proven later that internal covariate shift is not the problem.
- This tells us that we really don't understand the optimization of neural networks.

Normalizer-Free Networks

- A technique that trains models **without any normalization**.
- This approach uses a method called **Gradient Clipping**.

Gradient Clipping

- **Purpose:**
  - Prevents gradients from becoming too large.
  - Ensures that gradients do not become excessively large while maintaining their direction.
- The formula for gradient clipping is:

$$\hat{g} = \min\left(1, \frac{c}{\|g\|_2}\right) g$$

- Where c represents the *gradient clipping parameter*.
- If $\|g\|_2 \gg c$, then $\hat{g} \approx \min\left(1, \frac{c}{\|g\|_2}\right) \to c$, meaning the gradient is clipped to prevent large values.

Dynamic Gradient Clipping in Normalizer-Free Networks

- Dynamic gradient clipping is used in normalizer-free neural networks.
- The clipping threshold ϵ_t is defined as:

$$\epsilon_t = \alpha \cdot \text{mean}(\|\mathbf{g}_{t-1}\|_2) + \beta$$

  where:

  - α: A hyperparameter that controls the *adaptation speed* of the clipping threshold.
  - β: Controls the base level of ϵ_t.
  - $\|\mathbf{g}_{t-1}\|_2$: The average norm of gradients across mini-batches previous time-step.

Application Example

- ResNet was trained using this approach **without normalization** and just by managing large gradients using dynamic gradient clipping.

References

[1] Simon J.D. Prince. *Understanding Deep Learning*. MIT Press, 2023.

[2] Filter Response Normalization Layer - arXiv:1911.09737v2