CS 599: Foundations of Deep Learning
Assignment #3

Janhvi Narayan Mishra
Northern Arizona University
jm5747@nau.edu

Dataset Summary

For this project, I modified the Fashion-MNIST dataset to meet the CNN input requirements. The loaded dataset generated the following shapes:

- **Training set shape:** (60000, 28, 28, 1)
- **Test set shape:** (10000, 28, 28, 1)

Each image is 28×28 grayscale with a single channel.

Baseline Model (Without Normalization)

The baseline was a simple CNN model that has not been trained using Batch Normalization, Layer Normalization, or Weight Normalization. As specified in the assignment, the model was trained using the tf.GradientTape() technique for backpropagation.

The model failed to converge due to its basic architecture and lack of stability.

- **Baseline Test Accuracy:** 0.0718

This accuracy is similar to random guessing in a 10-class classification problem.

Custom Normalization Implementations

Three normalization layers were developed manually using only TensorFlow operations and the mathematical methods provided.

Batch Normalization

Given a mini-batch $\{x_1, \ldots, x_N\}$, batch mean and variance are computed as:

$$\mu_{MB} = \frac{1}{N}\sum_{i=1}^{N} x_i, \qquad \sigma^2_{MB} = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_{MB})^2$$

Normalization and rescaling:

$$\hat{x}_i = \frac{x_i - \mu_{MB}}{\sqrt{\sigma^2_{MB} + \epsilon}}, \qquad z_i = \gamma\hat{x}_i + \beta$$

Layer Normalization

LayerNorm normalizes across features within a single sample:

$$\mu = \frac{1}{H}\sum_{j=1}^{H} x_j, \qquad \sigma^2 = \frac{1}{H}\sum_{j=1}^{H}(x_j - \mu)^2$$

$$\hat{x}_j = \frac{x_j - \mu}{\sqrt{\sigma^2 + \epsilon}}, \qquad z_j = \gamma\hat{x}_j + \beta$$

Weight Normalization

Weight vectors were reparameterized as:

$$w = \frac{g}{\|v\|}v$$

This isolates weight magnitude and direction, resulting in better optimum conditioning.

Comparison With TensorFlow Built-In Normalization Layers

Each custom normalization layer was compared to its equivalent TensorFlow implementation. The mean absolute difference (MAD) between outputs was calculated as:

$$\text{MAD} = 4.4569155 \times 10^{-8}$$

Because this value is extremely small, it confirms:

- The custom implementations are numerically consistent with TensorFlow,
- Any differences are only from floating-point rounding,
- The backward pass through the custom operations is implemented correctly.

Model Performance With Normalization

BatchNorm, LayerNorm, and WeightNorm were individually added to the model and retrained.

For every normalization method:

- **Test Accuracy**: 0.0718

This corresponds to both the baseline model and the TensorFlow normalization model.

The results show that normalization alone is insufficient to overcome the underfitting induced by the CNN architecture's simplicity. However, the assignment objective—correct implementation of normalizing layers—was effectively met.

Analysis and Discussion

Why Accuracy Did Not Improve

The presented model architecture is purposefully simplistic, and it lacks the ability to successfully learn Fashion-MNIST patterns. As a result, all models, with or without normalization, stayed at about random performance ($\sim 7\%$).

Normalization layers help to stabilize training but do not make up for insufficient model depth or representational capacity.

BatchNorm vs LayerNorm vs WeightNorm

BatchNorm:

- Normalizes across batches.
- Highly is quite effective in deep CNNs.
- Performance degrades with small batch sizes.

LayerNorm:

- Normalizes across features within a single sample.
- Independent of batch size.
- More stable for RNNs, Transformers, and variable batch inputs.

WeightNorm:

- Reparameterizes weights rather than activations.
- Helpful for optimization, but does not stabilize activations directly.

Why LayerNorm is Often Better

LayerNorm avoids relying on batch statistics and thus:

- Works consistently for small batch sizes,
- Is steady during training and,
- Is commonly employed in modern architecture, particularly in transformers.

Final Findings

- My homemade BatchNorm, LayerNorm, and WeightNorm implementations match TensorFlow's results with just floating-point differences.
- The mean absolute difference of 4.4569155×10^{-8} confirms the correctness of both forward and backward passes.
- All models (baseline and normalized variations) achieved the same accuracy: **0.0718**.
- LayerNorm is theoretically superior in scenarios with small or variable batch sizes.

Code Repository

All source code for this assignment is available at: **https://github.com/jm5747-ux**