Normalization in Neural Networks: A Deep Dive

Abdul Samad Khan

May 14, 2025

Introduction

In this notebook, we'll discuss the importance of normalization in neural networks (NN), particularly why we normalize both input data and activations. By the end of this section, you'll understand the reasons behind normalization and how it's applied both before and during the network's forward pass.

We can draw an analogy between normalization and quantum mechanics to help simplify the concept. Think of normalizing your input data as preparing the "initial state" of a quantum system, while applying Batch Normalization (BN) during training is like controlling the "evolution" of that state after every operation.

Preprocessing: Normalizing the Input Data

Before feeding the data into a neural network, it's essential to ensure that the features (input data) are on a similar scale. This is important because neural networks are sensitive to the range of input values, and large variations between features can cause unstable gradient updates during training.

In practice, there are two common methods for normalizing the features:

1. Standardization (Z-score Normalization)

Standardization transforms the data so that each feature has a mean of 0 and a standard deviation of 1. The formula for standardizing a feature x_i is:

$$\hat{x_i} = \frac{x_i - \mu_i}{\sigma_i}$$

where:

- x_i is the original feature value,
- μ_i is the mean of the feature,
- σ_i is the standard deviation of the feature.

This ensures that each feature has zero mean and unit variance, making the learning process more stable and efficient.

2. Min-Max Scaling

Alternatively, features can be scaled to a specific range, typically [0, 1], using Min-Max scaling. The formula is:

$$\hat{x_i} = \frac{x_i - \min(x_i)}{\max(x_i) - \min(x_i)}$$

Here:

- x_i is the original feature value,
- $\min(x_i)$ and $\max(x_i)$ are the minimum and maximum values of the feature, respectively.

This is useful when you need all features to be on the same scale, especially if your data has known bounds or if you're working with models that rely on distances (like k-nearest neighbors).

Batch Normalization During Training

After the input data is processed through the network layers, the activations at each layer can vary greatly in terms of their range and variance. To solve this issue, **Batch Normalization (BN)** is applied inside the network after each activation.

The core idea of BN is to standardize the activations in each mini-batch during training. The BN formula for a single activation h in layer l is:

$$\hat{h}_l = \frac{h_l - \mu_b}{\sqrt{\sigma_b^2 + \epsilon}}$$

where:

- h_l is the activation at layer l,
- μ_b and σ_b^2 are the mean and variance of the activations in the current mini-batch, respectively,
- ϵ is a small constant added for numerical stability.

The goal is to normalize the activations to have zero mean and unit variance for each mini-batch. By doing this, we stabilize the training process and reduce internal covariate shift, which occurs when the distribution of activations changes during training. This can significantly speed up training and lead to better generalization.

Additionally, BN introduces two learnable parameters:

$$\hat{h}_l = \gamma \cdot \frac{h_l - \mu_b}{\sqrt{\sigma_b^2 + \epsilon}} + \beta$$

where:

- γ is a scaling factor (learned during training),
- β is a shifting factor (also learned during training).

These parameters allow the network to retain the flexibility to adapt the normalized output to its needs during training.

Why Normalize?

The reasons for normalizing the data at these two stages are as follows:

1. Stabilizing Gradient Descent

In gradient-based optimization, the scale of the features and activations can influence the gradients' size, making them either too large or too small. This instability can slow down or even prevent convergence during training. Normalizing the input features ensures that the gradients update at a similar rate across all features, making the optimization process more stable.

2. Faster Convergence

By normalizing activations using Batch Normalization, the optimizer can more efficiently navigate the loss landscape, leading to faster convergence during training.

3. Preventing Exploding/Vanishing Gradients

In deep neural networks, especially those with many layers, the gradients can either become too large (exploding gradients) or too small (vanishing gradients). BN helps mitigate this issue by controlling the scale of activations and gradients, leading to more stable training.

Summary

To summarize:

- **Preprocessing Normalization** (Standardization or Min-Max scaling) is applied to input data *before feeding it into the model* to ensure that all features are on a similar scale, facilitating stable gradient updates.
- **Batch Normalization** is applied after the activations of each layer during training to stabilize the learning process, reduce internal covariate shift, and speed up convergence.

Both forms of normalization help maintain the stability of the neural network during training and enable the model to learn more efficiently. By combining these two techniques, the model can handle more complex data and achieve better performance.