BATCH NORMALIZATION

Batch Normalization is used to reduce the problem of internal covariate shift in neural networks. It works by normalizing the data within each mini-batch. This means it calculates the mean and variance of data in a batch and then adjusts the values so that they have similar range. After that it scales and shifts the values so that model learn effectively.

In traditional neural networks as the input data propagates through the network, the distribution of each layer's inputs changes. This phenomenon is known as internal covariate shift and it can slow down training process. Batch Normalization aims to reduce this issue by normalizing the inputs of each layer.

This process keeps the inputs to each layer of the network in a stable range even if the outputs of earlier layers change during training. As a result training becomes faster and more stable.

Need of Batch Normalization

Batch Normalization makes sure outputs of each layer stay steady as model learns. This helps model train faster and learn more effectively.

- Solves the problem of internal covariate shift.
- Makes training faster and more stable.
- Allows use of higher learning rates.
- Helps avoid vanishing or exploding gradients.
- Can act like a regularizer sometimes reduce the need for dropout.

Fundamentals of Batch Normalization

In this section we are going to discuss the steps taken to perform batch normalization.

Step 1: Compute the Mean and Variance of Mini-Batches

For mini-batch of activations $x_1, x_2, \ldots, x_m$, the mean μ_B and variance σ_B^2 of the mini-batch are computed.

Step 2: Normalization

Each activation $x_i$is normalized using the computed mean and variance of the mini-batch. The normalization process subtracts the mean μ_B from each activation and divides by the square root of the variance σ_B^2, ensuring that the normalized activations have a zero mean and unit variance.

Additionally a small constant ϵ is added to the denominator for numerical stability, particularly to prevent division by zero.

$$\widehat{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

Step 3: Scale and Shift the Normalized Activations

The normalized activations x^i are then scaled by a learnable parameter γ and shifted by another learnable parameter β. These parameters allow the model to learn the optimal scaling and shifting of the normalized activations giving the network additional flexibility.

$$y_i = \gamma \widehat{x_i} + \beta$$

Batch Normalization in TensorFlow

In the code below we built a simple neural network using TensorFlow. We added Batch Normalization layer using tf.keras.layers.BatchNormalization(). This layer helps normalize the output or activations from the previous layer

Batch Normalization in PyTorch

In the following code we have build a simple neural network with batch normalization using PyTorch. We have define a subclass of 'nn.Module' and added the 'nn.BatchNorm1D' after the first fully connected layer to normalize the activations.

We have used 'nn.BatchNorm1D' as the input data is one-dimensional but for two-dimensional data like Convolutional Neural Networks 'BatchNorm2D' is used.

Benefits of Batch Normalization

- **Faster Convergence:** Batch Normalization reduces internal covariate shift, allowing for faster convergence during training.
- **Higher Learning Rates:** With Batch Normalization, higher learning rates can be used without the risk of divergence.
- **Regularization Effect:** Batch Normalization introduces a slight regularization effect that reduces the need for adding regularization techniques like dropout.
- By normalizing the inputs to each layer Batch Normalization helps stabilize the learning process and allows for faster convergence, making training more effective and reducing the need for careful initialization and learning rate tuning.