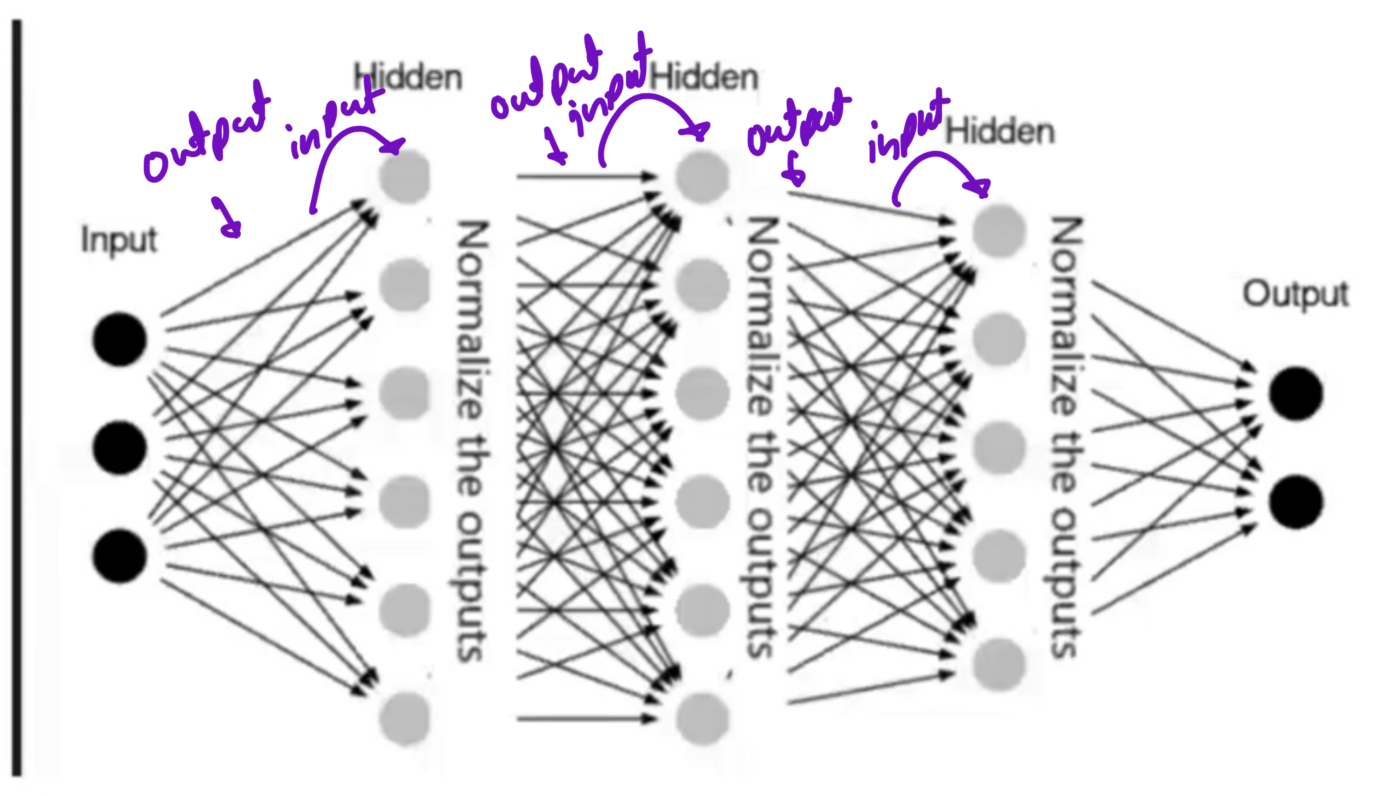
9.2 SOLUTION - BATCH NORMALIZATION

12 September 2025   02:07 AM   Avinash Yadav

Batch Normalization basically improves the training stability.



While training a neural network, we encounter a problem called as "Internal covariate Shift"
[ICS]

The phenomenon, where the distribution of layer activations changes during training as the network parameter's are updated, making it harder for subsequent layers to learn.

This instability slows down training (Difficult training), reduces performance and makes it difficult to train very deep network effectively (Slower Convergence)

So the idea is basically, during every mini-batch, whatever activation is coming from previous layer, we try to normalize them, and try to bring them in set given range by which the distribution of data across mini-batch remains same.

- **Applied to Hidden Layers:**
  - Typically applied to the hidden layers of a neural network, but not to the output layer.
- **Applied After Linear Layers and Before Activation Functions:**
  - Normalizes the output of the preceding layer (e.g., after nn.Linear) and is usually followed by an activation function (e.g., ReLU).
- **Normalizes Activations:**
  - Computes the mean and variance of the activations within a mini-batch and uses these statistics to normalize the activations.
- **Includes Learnable Parameters:**
  - Introduces two learnable parameters, gamma (scaling) and beta (shifting), which allow the network to adjust the normalized outputs.
- **Improves Training Stability:**
  - Reduces internal covariate shift, stabilizing the training process and allowing the use of higher learning rates.
- **Regularization Effect:**
  - Introduces some regularization because the statistics are computed over a mini-batch, adding noise to the training process.
- **Consistent During Evaluation:**
  - During evaluation, BatchNorm uses the running mean and variance accumulated during training, rather than recomputing them from the mini-batch.

* BEFORE APPLYING BATCH NORM :-

```python
class MyNN(nn.Module):
    def __init__(self, num_features):
        super().__init__()

        self.model = nn.Sequential(
            nn.Linear(num_features, 128),
            nn.ReLU(),
            nn.Dropout(p=0.3),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Dropout(p=0.3),
            nn.Linear(64, 10)
        )

    def forward(self, x):
        return self.model(x)
```

Python

* AFTER APPLYING BATCH NORM :-

```python
class MyNN(nn.Module):
    def __init__(self, num_features):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(num_features, 128),
            nn.BatchNorm1d(128),
            nn.ReLU(),
            nn.Dropout(p=0.3),
            nn.Linear(128, 64),
            nn.BatchNorm1d(64),
            nn.ReLU(),
            nn.Dropout(p=0.3),
            nn.Linear(64, 10)
        )

    def forward(self, x):
        return self.model(x)
```

Applying in hidden layer before Activation fxn

1D because our data is 1D

no. of activations we have in current layer

Python