Batch Normalization

`from keras.` `layers` `import` `BatchNormalization`

Batch Normalization

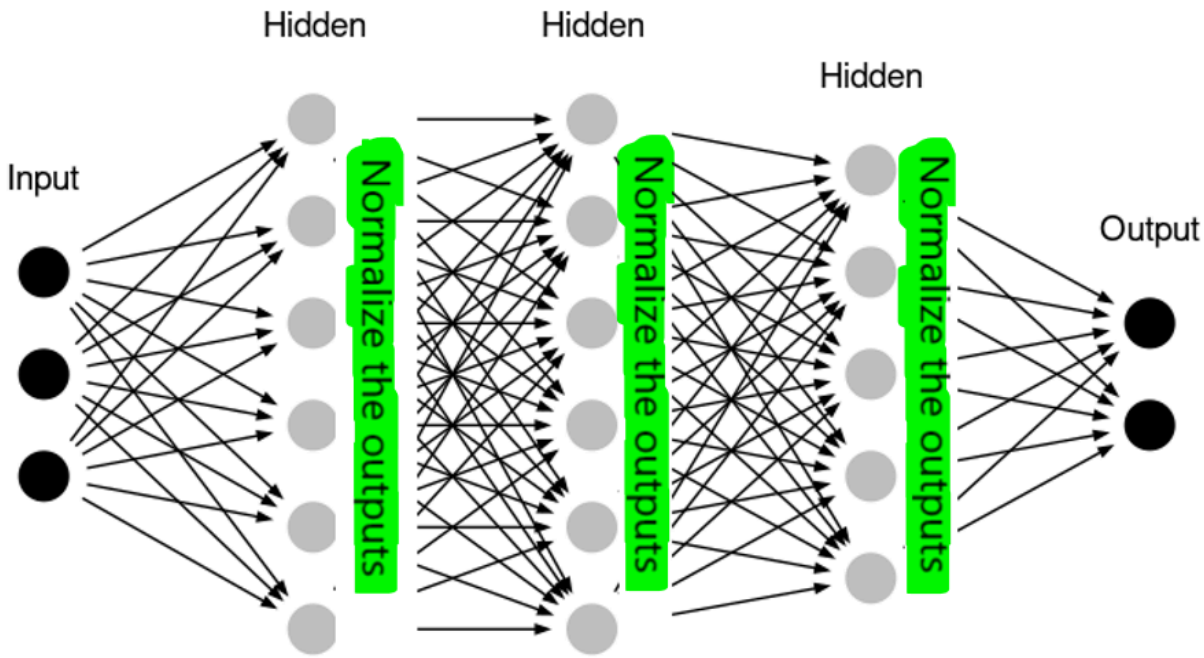
- Batch Normalization (BatchNorm) is a technique to **stabilize and accelerate** neural network training by normalizing layer inputs..
- **You need fewer epochs to reach to the optimal solution**

> 💡 Works only for Batch Gradient Descent.

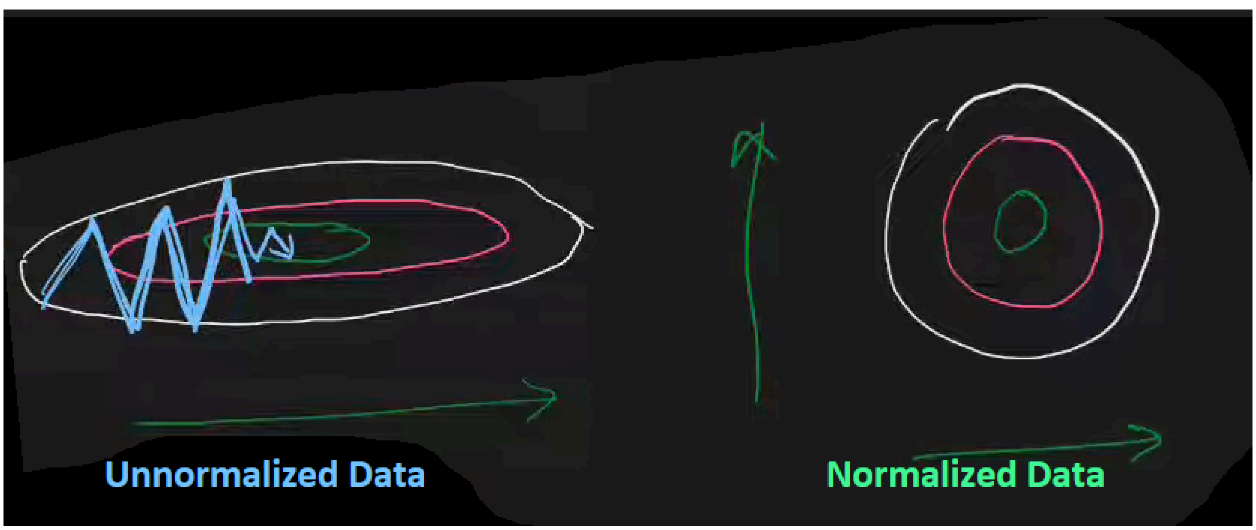
> **Mostly used with CNN. But can also be used with ANN.**

What BatchNorm Does?

- Normalizes the output of a layer **per batch** (mean=0, std=1).
- Adds two trainable parameters: **scale (γ)** and **shift (β)** to preserve model expressiveness.
- **Works best with smaller batch sizes (e.g., 32, 64).**



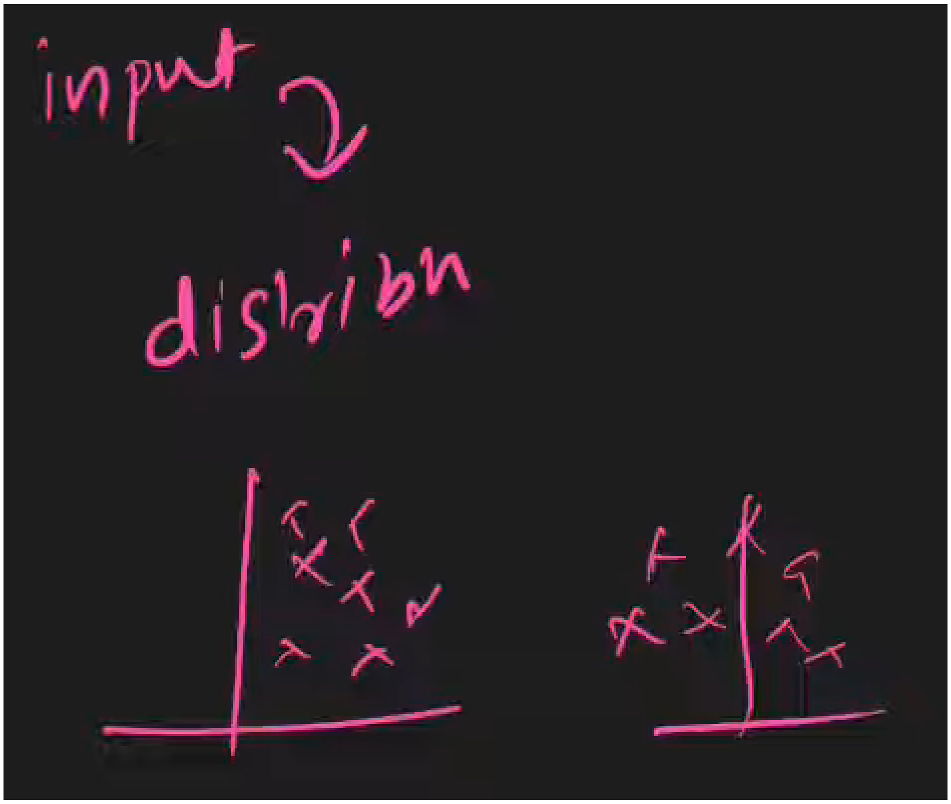
Contour Plot:



- Training becomes faster in case of normalized data as we don't need higher learning rate.

Internal Covariate Shift:

- When training deep neural networks, the **distribution of activations can change** as the model trains (this is called internal covariate shift).



- **When the distribution changes, model needs retraining.**

- 👆 BN **normalizes the outputs** of each layer so that they **maintain a consistent distribution**, which speeds up convergence and helps with training stability.

✨Why Use BatchNorm?

Benefit	Explanation
Faster Training	Reduces internal covariate shift, allowing higher learning rates.
Smoother Gradients	Prevents vanishing/exploding gradients in deep networks.

Benefit	Explanation
Regularization	Adds slight noise (due to batch statistics), acting like dropout.
Reduces Dependency on Initialization	Makes the network less sensitive to weight initialization.
Stable	We can set wider values of hyperparameters.
Reduces Weight initialization Impact	Reduces the impact of Weight initialization

⚙️ How Does It Work Internally?

Given:

- Input to a layer: `x = [x₁, x₂, ..., x_m]` (batch of `m` samples)

1. **Compute Mean:**

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x_i$$

2. **Compute Variance:**

$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu)^2$$

3. **Normalize:**

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

- `ε` is a small constant to avoid division by zero (default: `1e-5`)

4. **Scale and Shift:**

$$y_i = \gamma \hat{x}_i + \beta$$

- `γ` and `β` are learnable parameters (so the layer can still express anything)

✅ **Output:** Has **mean ≈ 0**, **variance ≈ 1**, but still **trainable**

🧠 Where is it Used?

- **Between Dense/Conv and Activation**

```
Dense → BatchNorm → Activation
```

- Helps with **any deep neural net** (MLP, CNN, RNN, etc.)

Python Code:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, BatchNormalization, Activation
from tensorflow.keras.optimizers import Adam

# Create a simple model with Batch Normalization
model = Sequential([
    Dense(64, input_dim=784),  # First dense layer
    BatchNormalization(),       # Apply Batch Normalization
    Activation('relu'),         # ReLU activation
    Dense(32),                  # Second dense layer
    BatchNormalization(),       # Apply Batch Normalization again
    Activation('relu'),         # ReLU activation
    Dense(10, activation='softmax')  # Output layer
])

# Compile the model
model.compile(optimizer=Adam(), loss='sparse_categorical_crossentropy', m
etrics=['accuracy'])

# Example data
import numpy as np
```

```
X_train = np.random.rand(1000, 784)  # 1000 samples, 784 features
y_train = np.random.randint(0, 10, 1000)  # 1000 labels (for classification)

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32) # ← Batch GD
```

💡 **Default batch size of 32**

```
model = Sequential()

model.add(Dense(3,activation='relu',input_dim=2))
model.add(BatchNormalization())
model.add(Dense(2,activation='relu'))
model.add(BatchNormalization())
model.add(Dense(1,activation='sigmoid'))
```

Where to Place Batch Normalization

- In practice, Batch Normalization is typically placed **after the linear transformation** (e.g., after the `Dense` layer) and **before the activation function**.

- **Common placement:** `Dense → BatchNormalization → Activation`.
- **Alternative placement:** `Dense → Activation → BatchNormalization` (less common).

When Not to Use BatchNorm❌

- **Very small batches** (e.g., < 8 samples) → Use **GroupNorm** or **LayerNorm**.
- **Recurrent networks (RNNs/LSTMs)** → Prefer **LayerNorm**.
- **Low-resource edge devices** → BatchNorm's runtime overhead may be prohibitive.

BatchNormalization Layer in Keras:

```
BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, ...)
```

Key Parameters:

Parameter	Meaning
`axis`	Which axis to normalize (default: `-1` → last axis, usually features)
`momentum`	Used to update running mean/variance (for inference)
`epsilon`	Small constant to prevent division by 0
`center`	If `True`, add β (default: `True`)
`scale`	If `True`, multiply by γ (default: `True`)

📈 Benefits:

Advantage	Description
✅ Faster convergence	Speeds up learning (fewer epochs)
✅ Higher learning rate	BN allows bigger learning rates
✅ Reduced overfitting	Acts like regularizer (like dropout)
✅ Works with any layer	Can be used in MLPs, CNNs, RNNs

⚠️ Limitations

Issue	Description
❌ Not good for very small batch sizes	Stats become noisy

Issue	Description
❌ Less effective in online/streaming data	Needs batch
❌ Tricky with variable sequence lengths	(use LayerNorm for RNNs instead)

Summary:

Objective	Solution
Stabilize and speed up training	Use `BatchNormalization()` layer
Add it **after Dense/Conv, before activation**	Best placement
Keep `batch_size` ≥ 16	To ensure stable statistics
Use `momentum` , `epsilon` , `axis` for tuning	Default values often work

> 💡 **If you get OOM (Out-of-Memory) errors, reduce `batch_size` .**