1) Normalization:

1- Data Normalization:

Also refered to as "Standardization". It's a transformation applied to the input data. It is a calculation performed once, before the training starts. It transforms the data so that:

- mean = 0 , standard deviation = 1

→ the most common formula is:

$$x_{norm} = \frac{x - \mu}{\sigma}$$

x: original value, μ: mean, σ: std

Std = 1 means the average distance of data points from the mean is 1. Spread is moderate. Not too tight and not too wide.

Why do we do it?

→ Input features have different ranges (one in [0,1], another in [0, 1000], the model gives more importance to the large feature because it contributes more to the loss + causes the weights connected to that feature to dominate the training while the others are barely updated.

→ training unstable → Normalization fixes this by putting all features on a similar scale (mean=0 std=1) so they contribute equally to the learning process.

Example:

1) ImageNet and Image Normalization:

Most pretrained models (ViT..) are trained using the ImageNet normalization.

IMAGENET_MEAN = [0,485 , 0,456 , 0,406]
IMAGENET_STD = [0,229 , 0,224 , 0,225]

→ these values were computed over millions of images.

A pixel with RGB value:
$[R, G, B] = [120, 100, 90]$

Step 1: Scale to [0,1] (feature)
$R' = \frac{120}{255} = 0,47$; $G' = \frac{100}{255} = 0,392$; $B' = \frac{90}{255} = 0,353$

Step 2: Normalize using ImageNet std and mean
$R_{norm} = \frac{0,47 - 0,485}{0,229} = -0,065$
⋮

2) Let's say we have:
5 vectors (samples)
each vecta has 3 features (dimensions)
→ we'll do a per feature normalization

$$X = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \\ 4 & 7 & 10 \\ 5 & 8 & 11 \end{bmatrix}$$

(1 feature)

Step 1: Compute mean per feature = Column:

$$\mu = \begin{bmatrix} \frac{1+2+3+4+5}{5} \\ \frac{4+5+6+7+8}{5} \\ \vdots \end{bmatrix} = \begin{bmatrix} 3 \\ 6 \\ 9 \end{bmatrix}$$

Step 2: Compute Std per feature:

$$\sigma_j = \sqrt{\frac{1}{5}\sum_{i=1}^{5}(x_{ij} - \mu_j)^2} = \begin{bmatrix} 1,414 \\ 1,414 \\ 1,414 \end{bmatrix}$$

N — variance

Step 3: Normalize each value:

$$x_{ij\_norm} = \frac{x_{ij} - \mu_j}{\sigma_j}$$

3 Layer Normalization

→ It's applied to each input individually rather that to one feature across all inputs.

→ normalization per data point

→ It is a technique used to normalize the pre-activations (as it comes before the activation function) within a layer across the features dimensions.

Why is it used instead of BatchNorm?

RNNs & Sequence length varies → fixed to use batch stats.

→ for BatchNorm, we need shape = (B, D)
but in RNNs, shape = (B, T, D)
↓
Time step (e.g., words)

Suppose we have 3 sequences with different lengths

Sample	Seq Length
A	4
B	3
C	2

→ we need to pad them all to T = 4. Let's assume D = 1

↓

Input shape = (3, 4, 1)

→ X = [[1] [2] [3] [4] ; [4] [5] [6] [PAD] ; [1] [2] [PAD] [PAD]]

→ 0 (common practice)

Now, Let's apply BatchNorm per time step to have the desired shape = (B, D)
So at $t = 2$, we have

$x[:, 2, :] = \begin{bmatrix} 3 \\ 6 \\ 2 \end{bmatrix}$ a padding

→ So the calculus will be noisy

→ So, with LayerNorm, for a given sample
$x = [x_1, x_2 \dots x_d]$, we:

1- Compute the mean and std per sample

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

2- Normalize

$$\hat{x}_j = \frac{x_j - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$

3) Apply Learnable shift and scale

$$y_j = \gamma \hat{x}_j + \beta$$

Learnable parameters per feature ⚠

2) Batch Normalization

It is a regularisation or optimization technique. It is designed to normalize the activation of the hidden layers using statistics computed over a mini batch. By normalizing the output of a layer have a zero mean and unit std. BatchNorm allows to mitigate the problem of internal covariate Shift.

→ this is when the distribution of activations shifts during training.

BatchNorm is applied after the linear & convolutional layer and before the activation fonction (e.g., ReLU) which ensures that the inputs to the non-linearity are well conditioned.

Example:

1) Let's say we have:
- a batch of 3 samples
- each samples has 3 features

$$\rightarrow X = \begin{bmatrix} 3 & 1 \\ 4 & 2 \\ 5 & 3 \end{bmatrix} \text{ shape} = (3, 2)$$

→ we apply batch norm → normalize each column independantly across the batch.

Step 1:

f_1: $\mu_1 = \frac{1}{3}[3+4+5] = 4$; $\sigma_1^2 = \frac{1}{3}[(3-4)^2 + (4-4)^2 + (5-4)^2] = \frac{2}{3}$

f_2: $\mu_2 = 2$; $\sigma_2^2 = \frac{2}{3}$

Step 2: Normalize

- We add a small $\varepsilon = 1e{-}5 = 10^{-5}$ for stability
- $\sigma = \sqrt{\sigma^2} = 0,8165$
- Normalization formula:

$$\hat{x}_{ij} = \frac{x_{ij} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$ (column indice)

→ $\hat{X} = \begin{bmatrix} -1,225 & -1,225 \\ 0 & 0 \\ +1,225 & +1,225 \end{bmatrix}$

Step 3: Apply Learnable Scale and Shift

→ Let's assume $\gamma = [2, 0.5]$
$\beta = [1, 0]$

↓

$$y_{ij} = \gamma_j \hat{x}_{ij} + \beta_j$$

Linear/conv layer -> BatchNorm -> Activation fonction

Why use Scale and shift?

γ, β restore flexibility. They allow the model to "undo" the standardization if needed.

→ • Normalization always squashes everything between 0 and 1 but maybe some neurons should be more active (= features more emphasized)

- If the network wants the feature to behave like before we can learn $\gamma = \sigma$, $\beta = \mu$
- If the network wants to supress a feature $\gamma = 0$