======================================================================

**Batch normalization**

======================================================================

**What is Covariate Shift?**

Covariate shift means a change in the distribution of features (input variables) between the training and testing dataset in Machine Learning or Deep Learning models. In other words, it exists when the statistical properties of the input data change between the training and testing phases.

When a Machine Learning / Deep Learning model is trained on a dataset and then applied to another dataset with different statistical properties, the model might not perform well due to this ***Covariate Shift***.

Covariate shift can be misleading because the models assume that the training and testing data come from the same distribution. When this assumption is violated, the model performance might deteriorate because the training process is trying to generalize patterns from the training data while a different distribution is present in the test data.

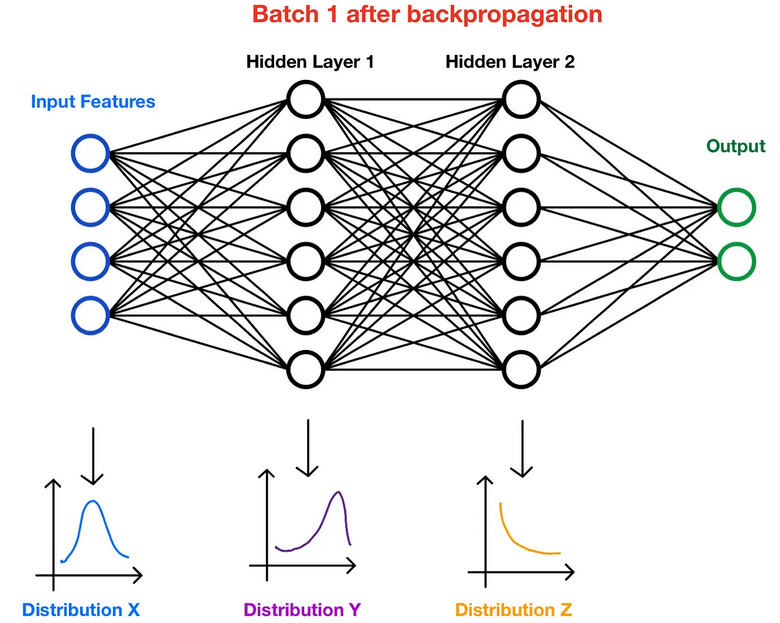
-------------------------------------------------------------------------------------------------------------

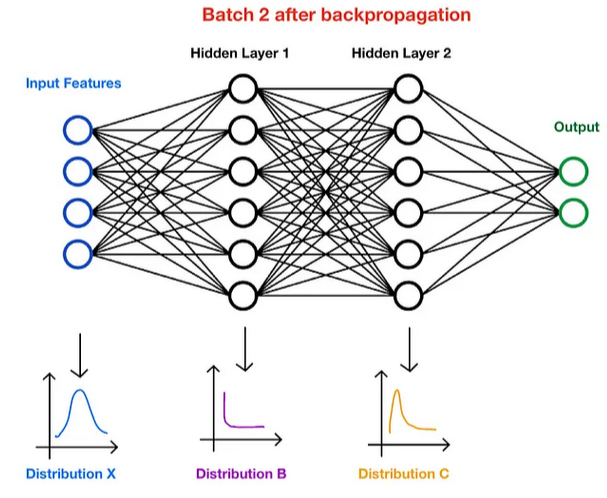
**What is Internal Covariate Shift?**

Internal covariate shift is a phenomenon specific to neural networks, where the distribution of each layer’s inputs changes as the parameters of the preceding layers are updated. In simpler terms, it refers to the changing distribution of the inputs to each layer within a neural network during the training process.

This phenomenon is termed “internal” covariate shift to distinguish it from the standard covariate shift that refers to changes in the input distribution between training and testing datasets.

As a neural network learns and updates its weights through backpropagation, the distribution of inputs to each layer can shift. This shift can make subsequent layers’ learning more challenging because they have to continually adapt to these changing distributions, slowing down the overall training process. Here is an illustration of ***Internal Covariate Shift***:





--------------------------------------------------------------------------------------------------------------

**What is Batch Normalization:**

Batch Normalization (BatchNorm) is a technique used in deep learning to normalize the inputs of each layer in a neural network. It helps improve training speed, stability, and generalization.

Introduced by Ioffe & Szegedy (2015), BatchNorm reduces internal covariate shift, ensuring that activations remain well-scaled throughout training.

--------------------------------------------------------------------------------------------------------------

**Why is Batch Normalization Important?**

✅ Reduces Internal Covariate Shift → Keeps activations stable, preventing drastic weight updates.  
✅ Faster Convergence → Allows higher learning rates by keeping inputs well-conditioned.  
✅ Prevents Vanishing/Exploding Gradients → Stabilizes gradients in deep networks.  
✅ Acts as Regularization → Reduces dependence on dropout.

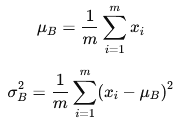
--------------------------------------------------------------------------------------------------------------

**How Does Batch Normalization Work?**

BatchNorm normalizes activations within a mini-batch before passing them through the activation function.

For each mini-batch:

1. Compute mean and variance for each feature:



1. Normalize the activations:



where **ε** is a small number to prevent division by zero.

1. Scale and shift using learnable parameters γ and β:



γ (scale) and β (shift) allow the model to learn optimal distributions. If needed, the model can learn to undo normalization by adjusting γ and β.

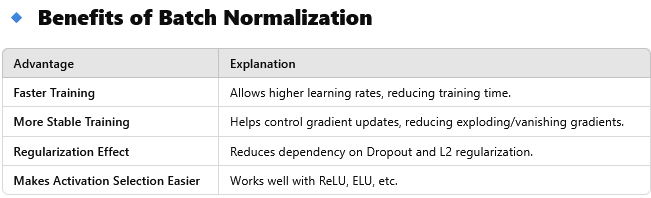
--------------------------------------------------------------------------------------------------------------

**Where Can We Apply Batch Normalization?**

BatchNorm can be applied:

* After fully connected (Dense) layers
* After Convolutional layers (before the activation function
* Before or after the activation function (Common practice: Before activation)

--------------------------------------------------------------------------------------------------------------



--------------------------------------------------------------------------------------------------------------

**Does Batch Normalization Work at Inference Time?**

At inference time, batch statistics (mean & variance) are replaced with running averages computed during training.

🔹 In TensorFlow/Keras, this is automatically handled during model.fit() and model.predict().

--------------------------------------------------------------------------------------------------------------

**When Should You Use Batch Normalization?**

✅ Use for deep networks to stabilize training.  
✅ Use with ReLU/Leaky ReLU for better performance.  
✅ Avoid if batch size is very small (e.g., <8) as variance estimates may be unstable.

--------------------------------------------------------------------------------------------------------------

*# Create a neural network with BatchNorm*

*model = Sequential([*

*Dense(128, input\_shape=(10,)), # Fully connected layer*

*BatchNormalization(), # Normalize activations*

*ReLU(), # Apply activation*

*Dense(64),*

*BatchNormalization(),*

*ReLU(),*

*Dense(1, activation='sigmoid') # Output layer*

*])*

*model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])*