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

**Weight initialization**

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

While building and training neural networks, it is crucial to initialize the weights appropriately to ensure a model with high accuracy. If the weights are not correctly initialized, it may give rise to the Vanishing Gradient problem or the Exploding Gradient problem.

Hence, selecting an appropriate weight initialization strategy is critical when training DL models.

In **Deep Learning**, **weight initialization** is the process of setting the initial values of the weights in a neural network before training begins.

Proper initialization helps:

✅ Speed up convergence  
✅ Prevent vanishing/exploding gradients  
✅ Improve model performance

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

**Why is Weight Initialization Important?**

* Poor initialization can lead to vanishing gradients (if weights are too small) or exploding gradients (if weights are too large).
* If all weights are initialized to zero, the model won't learn effectively due to symmetry problems (all neurons in a layer will have the same updates).
* A good initialization helps the network propagate signals properly during forward and backward passes.

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

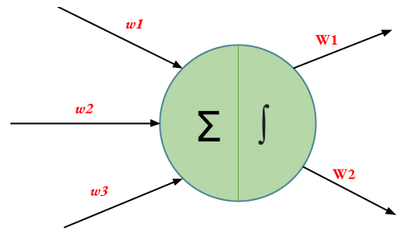
**Terminology or Notations**

Following notations must be kept in mind while understanding the Weight Initialization Techniques. These notations may vary at different publications. However, the ones used here are the most common, usually found in research papers.

**fan\_in =** Number of input paths towards the neuron

**fan\_out =** Number of output paths towards the neuron

**Example:** Consider the following neuron as a part of a Deep Neural Network.



fan\_in = 3 (Number of input paths towards the neuron)

fan\_out = 2 (Number of output paths towards the neuron)

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

**Common Weight Initialization Techniques**

**1. Zero Initialization (Bad Practice ❌)**

* All weights = 0, biases = 0.
* Leads to symmetry problem (all neurons behave the same).
* Does NOT work for deep networks.

🔹 Use Case: Not recommended for hidden layers but can be used for bias initialization.

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

**2. Random Initialization (Not recommended)**

* Assigns random small values to weights.
* Helps break symmetry but does not prevent vanishing/exploding gradients.

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

**3. Xavier (Glorot) Initialization**

* Best for: Sigmoid, Tanh activations.
* Xavier/Glorot Initialization often termed as Xavier Uniform Initialization
* Ensures that variance of activations remains the same across layers.
* In Xavier/Glorot weight initialization, the weights are assigned from values of a uniform distribution as follows:



*model = Sequential([*

*Dense(64, activation='tanh', kernel\_initializer=GlorotUniform(), input\_shape=(10,)),*

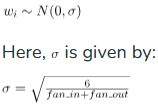
*Dense(1, activation='sigmoid')*

*])*

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

**4. Normalized Xavier/Glorot Initialization**

In Normalized Xavier/Glorot weight initialization, the weights are assigned from values of a normal distribution as follows:



Xavier/Glorot Initialization, too, is suitable for layers where the activation function used is **Sigmoid**.

*initializer = tf.keras.initializers.GlorotNormal()*

*layer = tf.keras.layers.Dense(3, kernel\_initializer=initializer)*

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

**5. He Uniform Initialization**

In He Uniform weight initialization, the weights are assigned from values of a uniform distribution as follows:



He Uniform Initialization is suitable for layers where **ReLU** activation function is used.

*initializer = tf.keras.initializers.HeUniform()*

*layer = tf.keras.layers.Dense(3, kernel\_initializer=initializer)*

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

**6. He Normal Initialization (for ReLU/Leaky ReLU)**

* Best for: ReLU and Leaky ReLU activations
* Modified version of Xavier, but works better for deep networks with ReLU.
* In He Normal weight initialization, the weights are assigned from values of a normal distribution as follows:



where **n/fan\_in** is the number of input neurons.

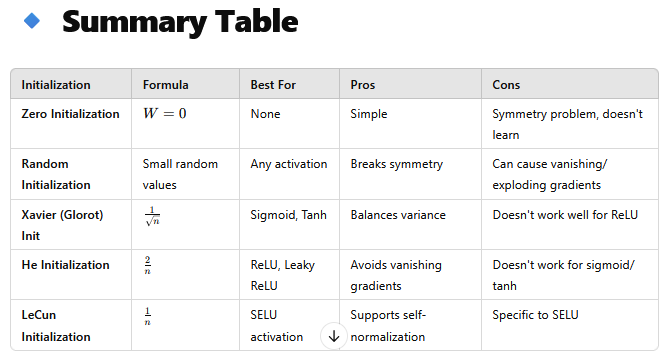
*model = Sequential([*

*Dense(64, activation='relu', kernel\_initializer=HeNormal(), input\_shape=(10,)),*

*Dense(1, activation='sigmoid')*

*])*

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



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

**Which Initialization Should You Use?**

* ReLU / Leaky ReLU → Use He Initialization
* Sigmoid / Tanh → Use Xavier Initialization
* SELU → Use LeCun Initialization

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