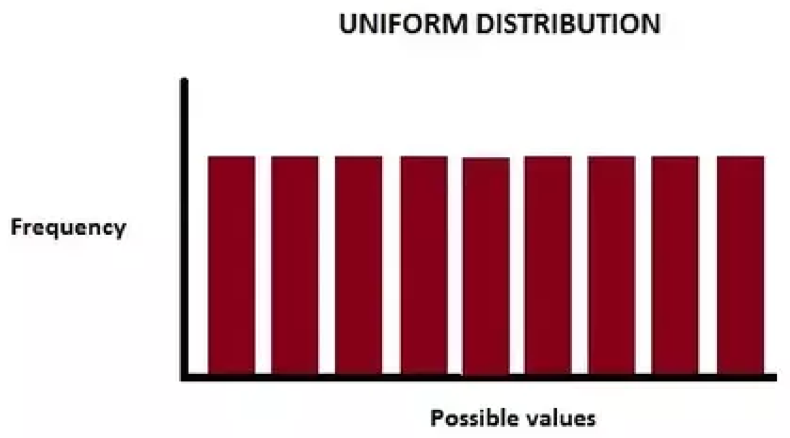
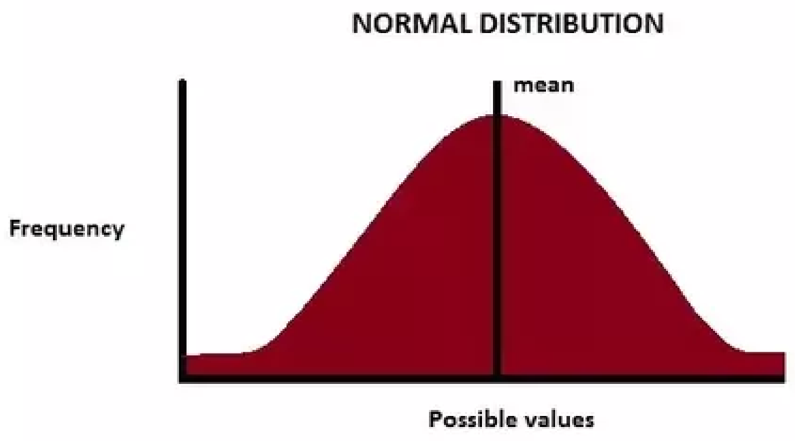
Weight Initialization Techniques [Habiba Shera]

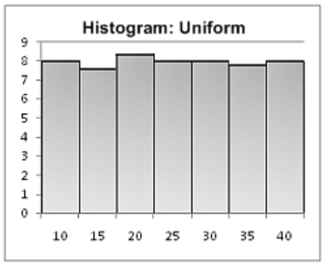
- **Zero Initialization**
    - all the weights are assigned to 0 as the initial value.
    - **highly ineffective** as neurons learn the same feature during each iteration.
    - **not preferred**

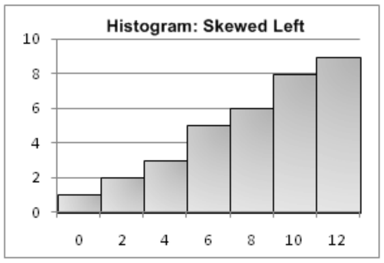
    ```
    # Zero Initialization
    from tensorflow.keras import layers
    from tensorflow.keras import initializers

    initializer = tf.keras.initializers.Zeros()
    layer = tf.keras.layers.Dense(3, kernel_initializer=initializer)
    ```

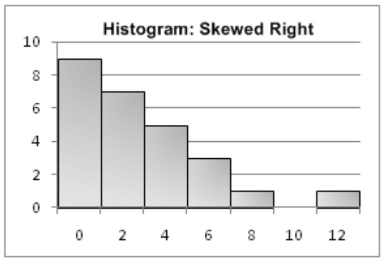
- **Random Initialization**
    - assigns random values **except for zeros** as weights to neurons
    - might causes problems such as **Overfitting**, **Vanishing Gradient Problem**, **Exploding Gradient Problem**
    - **It can be of two kinds:**
        - Random Normal : The weights are initialized from values in a **normal distribution**.
        - Random Uniform : The weights are initialized from values in a uniform distribution.

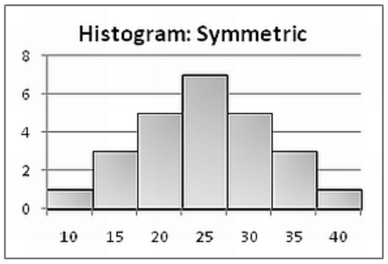

UNIFORM DISTRIBUTION
Frequency
Possible values

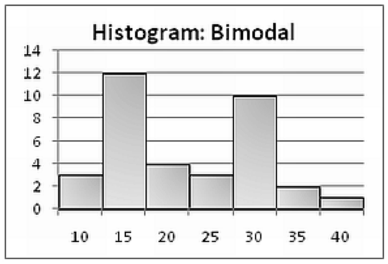

NORMAL DISTRIBUTION
mean
Frequency
Possible values


Histogram: Uniform


Histogram: Skewed Left


Histogram: Skewed Right


Histogram: Symmetric


Histogram: Bimodal

```
# Random Normal Distribution
from tensorflow.keras import layers
from tensorflow.keras import initializers

initializer = tf.keras.initializers.RandomNormal(mean=0., stddev=1.)
layer = tf.keras.layers.Dense(3, kernel_initializer=initializer)
```

```
# Random Uniform Initialization
from tensorflow.keras import layers
from tensorflow.keras import initializers

initializer = tf.keras.initializers.RandomUniform(minval=0.,maxval=1.)
layer = tf.keras.layers.Dense(3, kernel_initializer=initializer)
```

💡 **Vanishing Gradients :** If the weights initialized are very small then in case of deep networks, for any activation function, (dW) will get smaller and smaller as we go backwards with every layer during back propagation.

💡 **Exploding Gradients :** This is the exact opposite of the vanishing gradients problem. Consider you have non-negative and large weights and small activations (sigmoid(z)). When these weights are multiplied along the layers, they cause a large change in the cost. Thus, the gradients are also going to be large. This means that the changes in W, by W—α * dW, will be in huge steps.

- **Xavier/Glorot Initialization**
    - is suitable for layers where the activation function used is **Sigmoid or Tanh**

$$w_i \sim U[-\sqrt{\frac{\sigma}{fan\_in+fan\_out}}, \sqrt{\frac{\sigma}{fan\_in+fan\_out}}]$$

- the weights are assigned from values of a uniform distribution as follows :

💡 ***fan_in =** Number of input paths towards the neuron.*
***fan_out =** Number of output paths towards the neuron*

```
# Xavier/Glorot Uniform Initialization
from tensorflow.keras import layers
from tensorflow.keras import initializers

initializer = tf.keras.initializers.GlorotUniform()
layer = tf.keras.layers.Dense(3, kernel_initializer=initializer)
```

- **Normalized Xavier/Glorot Initialization**
    - is suitable for layers where the activation function used is **Sigmoid or Tanh**
    - the weights are assigned from values of a normal distribution as follows:

$$\sigma = \sqrt{\frac{6}{fan\_in + fan\_out}}$$

```
# Normailzed Xavier/Glorot Uniform Initialization
from tensorflow.keras import layers
from tensorflow.keras import initializers

initializer = tf.keras.initializers.GlorotNormal()
layer = tf.keras.layers.Dense(3, kernel_initializer=initializer)
```

- **He Uniform Initialization (called as Kaiming Initialization)**
  - suitable for layers where **ReLU** activation function is used.
  - the weights are assigned from values of a uniform distribution as follows:

$$w_i \sim U[-\sqrt{\frac{6}{fan\_in}}, \sqrt{\frac{6}{fan\_out}}]$$

```
# He Uniform Initialization
from tensorflow.keras import layers
from tensorflow.keras import initializers

initializer = tf.keras.initializers.HeUniform()
layer = tf.keras.layers.Dense(3, kernel_initializer=initializer)
```

```
tf.keras.layers.Dense(25, activation = "relu", kernel_initializer="he_uniform")
```

- **He Normal Initialization (called as Kaiming Initialization)**
  - is suitable for layers where **ReLU** activation function is used
  - the weights are assigned from values of a normal distribution as follows:

$$\sigma = \sqrt{\frac{2}{fan\_in}}$$

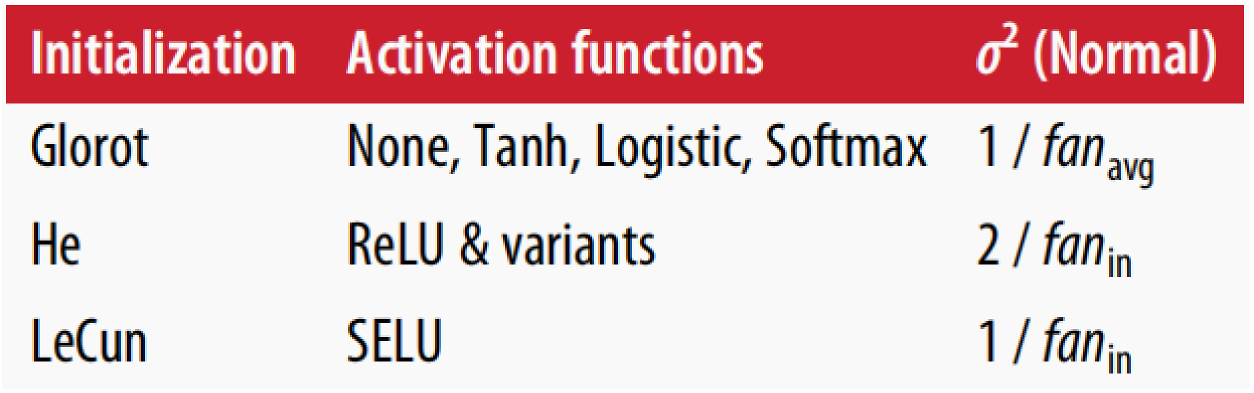
```
# He Normal Initialization
from tensorflow.keras import layers
from tensorflow.keras import initializers

initializer = tf.keras.initializers.HeNormal()
layer = tf.keras.layers.Dense(3, kernel_initializer=initializer)
```

```
tf.keras.layers.Dense(25, activation = "relu", kernel_initializer="he_normal")
```

💡 **ReLu** and **leaky ReLu** also **solves** the problem of **vanishing gradient.**

Conclusion :

- while using **Relu,** use **He uniform** or **He normalization techniques (weight initialization)**
- while using **Sigmoid**, **Tanh**, **softmax** use **Xavier/Glorot** or **normalized Xavier/Glorot techniques (weight inilialization)**



Initialization	Activation functions	σ^2 (Normal)
Glorot	None, Tanh, Logistic, Softmax	$1 / fan_{avg}$
He	ReLU & variants	$2 / fan_{in}$
LeCun	SELU	$1 / fan_{in}$

References

- Weight Initialization Techniques for Deep Neural Networks - GeeksforGeeks

- Weight Initialization Techniques in ANNs | NM DEV
- Weight Initialization Techniques in Neural Networks | by Saurabh Yadav | Towards Data Science