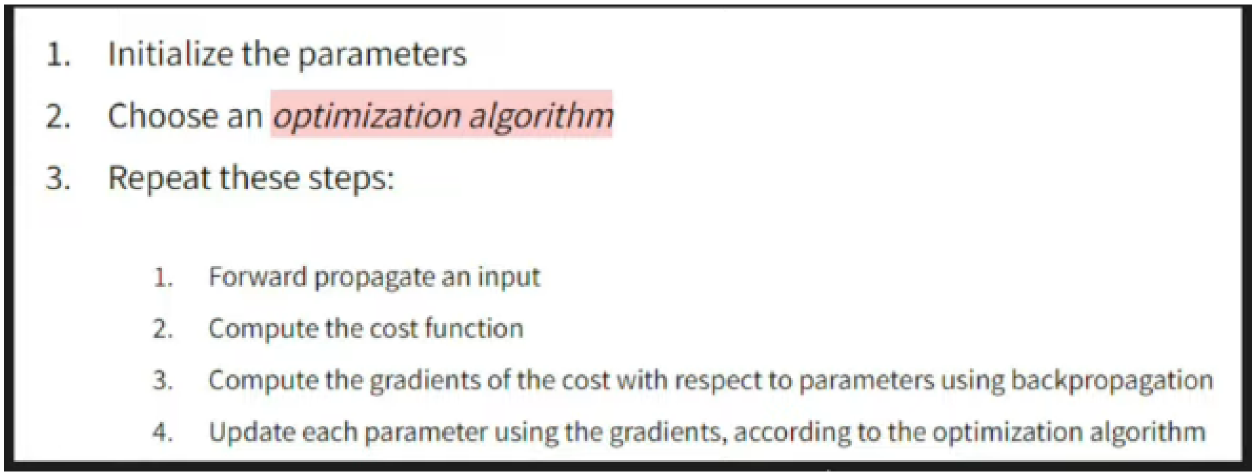
## ◾ How a Neural Network Works:-


1. Initialize the parameters
2. Choose an *optimization algorithm*
3. Repeat these steps:
    1. Forward propagate an input
    2. Compute the cost function
    3. Compute the gradients of the cost with respect to parameters using backpropagation
    4. Update each parameter using the gradients, according to the optimization algorithm

## ◾ Problems of not initializing the parameters properly:

- Vanishing Gradient Problem
- Exploding Gradient Problem
- Slow convergence
  (optimal solution maa dhila pugnu)

## ◾ What initialization techniques can't be used:-

### 1)Zero initialization :

(Interview question : kina weight haru 0 initialize garninna)

(Interview question: Why weights of NN are not initialized with 0)

1. Symmetry problem: When all weights are initialized to 0, all neurons in a given layer will produce the same output

during forward propagation. This symmetry in weights causes the neurons to learn identical features and essentially perform the same computations. As a result, the network will not be able to capture diverse or meaningful information, severely limiting its capacity to learn and generalize.

2. Vanishing gradients: During backpropagation, gradients are computed to update the weights and train the network. If all weights are initially set to 0, the gradients for each weight will also be 0, because the gradient is essentially a derivative that measures how a small change in a weight affects the network's error. With all gradients being 0, the network won't make meaningful weight updates, leading to extremely slow or no learning.

3. Stuck in local minima: Neural networks aim to find the global minimum of the loss function during training. When all weights are initialized to 0, every neuron computes the same output, and the network effectively behaves like a linear model. This can result in the network getting stuck in local minima and being unable to escape them to find better solutions.

4. Lack of expressiveness: Neural networks are designed to learn complex functions by adjusting their weights and biases. Initializing all weights to 0 limits the network's expressiveness, as it cannot represent diverse patterns and relationships within the data. Non-zero initial weights allow the network to explore a wider range of potential solutions.

➢(If hamle sigmoid function use gareko bhaye chai model ko each layer maa jati neuron vaye pani ,tesle eutai neuron jasto behave garxa.
Ani zero weight initialize garxa it acts as linear model.)

## 2)Non zero constant value initialization :

➢Symmetry problem: Initializing all weights to the same non-zero constant still leads to a symmetry problem. Neurons in a layer will produce identical outputs during forward propagation, causing them to learn the same features and perform similar computations. This restricts the network's ability to capture diverse and meaningful information.

➢ Slow convergence: While not as problematic as initializing weights to 0, constant non-zero initial values can still lead to slower convergence. It may take longer for the network to adapt its weights and learn optimal representations since all neurons start with similar values.

➢ Lack of adaptability: Neural networks are designed to adapt and adjust their weights based on the data they encounter during training. Initializing all weights to a constant value limits this adaptability, making it challenging for the network to effectively represent the complex patterns and relationships in the data.

## 3) Random Initialization:

➤ (Small numbers)
Yedi hamle np.random.random method use garera random initialization use garda ,vanishing gradient problem encounter garxa.
Tanh ra sigmoid activation function use garda vanishing gradient problem aauxa.

Tara relu use garda slow convergence hunxa.

➤(Large numbers)
Initializing weights with excessively large random values can lead to exploding gradients, causing numerical instability and failed training, while also slowing down learning due to aggressive weight updates.

Things not to do while initialization:

- ☑ ~~Zero value~~
- ☑ ~~Non zero constant value~~
- ☑ ~~Small random value~~
- ☑ ~~Large random value~~

## ◾ How to do initialization??

### 1)Glorot (Xavier) Initialization:-

➢ (tanh,linear,sigmoid,logistic,softmax activation function ko lagi perfectly kaam garxa)

Normal:-

➢ Formula : $\sqrt{\frac{1}{fan_{in}}}$ OR $\sqrt{\frac{2}{fan_{in}+fan_{out}}}$

where, fan_in = no. of inputs to a node

fan _out = no. of outputs

Uniform:-

➢ Formula: [-limit , limit]

Where, limit = $\sqrt{\frac{6}{fan_{in}+fan_{out}}}$

### 2)He Initialization:

(relu activation function sanga ramro kaam garxa)

Normal:

➢ Formula: $\sqrt{\frac{2}{fan_{in}}}$

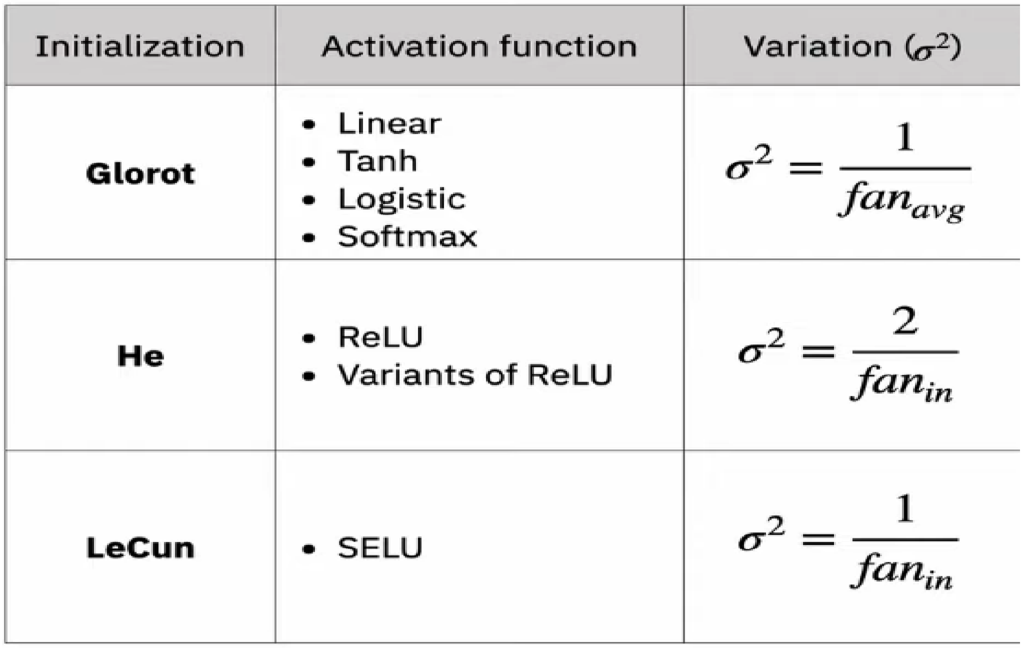
Uniform:

➤ Formula: [-limit , limit]

Where, limit = $\sqrt{\frac{6}{fan_{in}}}$

## 3)LeCun initialization:

➤ LeCun Initialization sets the initial weights of a neural network layer using a normal distribution with a standard deviation of `1 / sqrt(fan_in)`, where `fan_in` is the number of input units to the layer. It is designed to work well with activation functions like tanh and logistic sigmoid



| Initialization | Activation function | Variation ($\sigma^2$) |
|---|---|---|
| **Glorot** | • Linear<br>• Tanh<br>• Logistic<br>• Softmax | $\sigma^2 = \frac{1}{fan_{avg}}$ |
| **He** | • ReLU<br>• Variants of ReLU | $\sigma^2 = \frac{2}{fan_{in}}$ |
| **LeCun** | • SELU | $\sigma^2 = \frac{1}{fan_{in}}$ |

Sabai maa mean = 0

◾ **Code examples:**

➢ 1)

```
from keras.models import Sequential
from keras.layers import Dense
from keras.initializers import glorot_normal, glorot_uniform

model = Sequential()

model.add(Dense(units=64, activation='relu',
kernel_initializer=glorot_normal(), input_dim=input_shape))
```

➢ 2)

```
from keras.models import Sequential
from keras.layers import Dense
from keras.initializers import he_normal, he_uniform

model = Sequential()
model.add(Dense(units=64, activation='relu',
kernel_initializer=he_normal(), input_dim=input_shape))
```