Deep Learning

Tech Lead Data Science

Master en Data Science
2022-2023

Assembler
Institute of Technology

Index

1 Layers

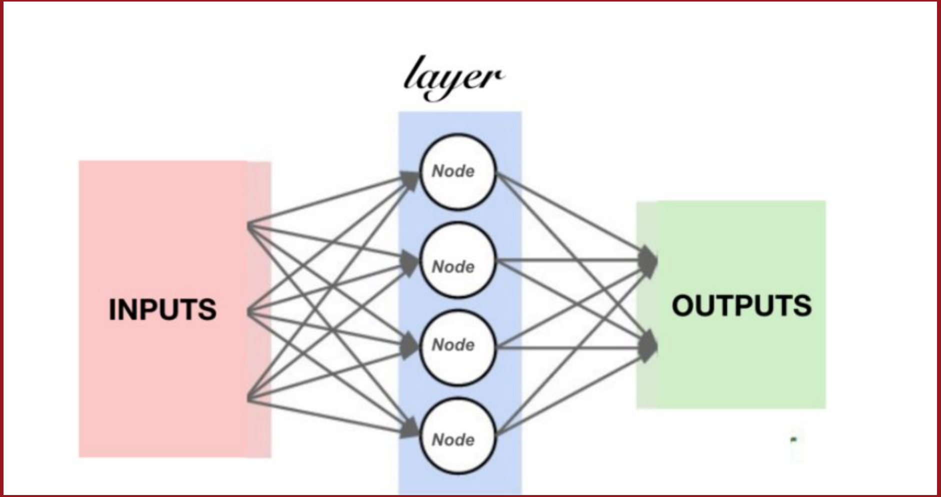
2 Optimizers

3 Activation Functions

LAYERS

LAYERS

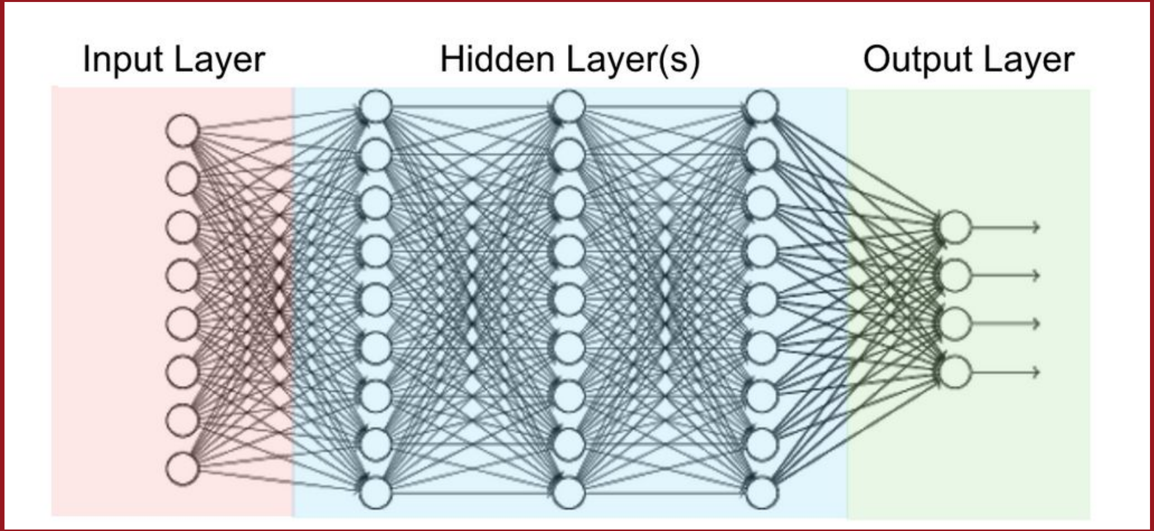
- In Deep Learning, a model is a set of one or more layers of neurons.
- Each layer contains several neurons that apply a transformation on each element of the input tensor.
- The main difference between the types of layers lies in the way the neurons behave.



Assembler Institute of Technology

LAYERS

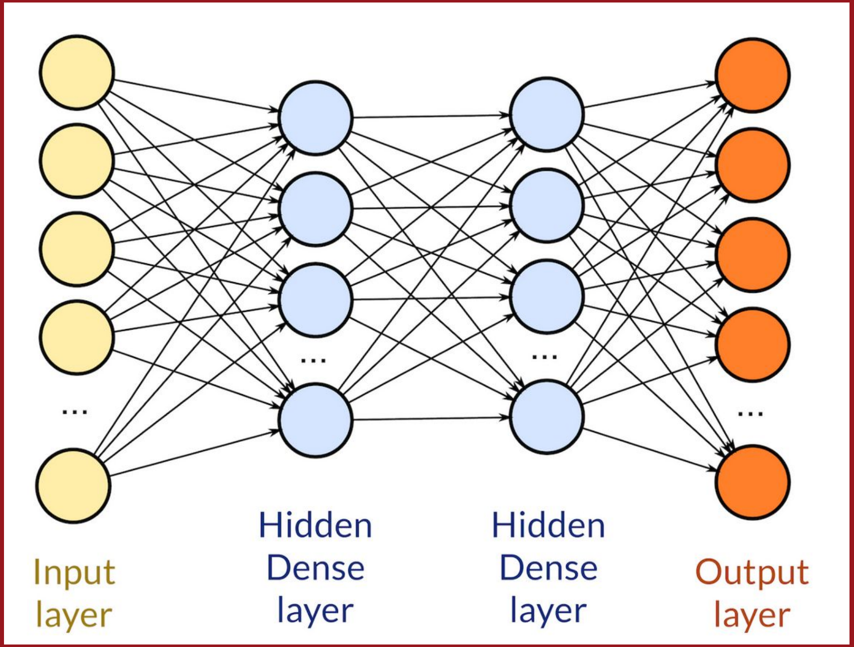
- **Input Layer:** This is the layer that receives data.
- **Output Layer:** This is the layer that produces results, devised by our neural network.
- **Hidden layer:** This is a layer in between input layers and output layers, where artificial neurons take in a set of weighted inputs and produce an output through an activation function.



Assembler
Institute of Technology

DENSE LAYER

- The dense layer neuron in a model receives output from every neuron of its preceding layer, where neurons of the dense layer perform matrix-vector multiplication.
- Matrix vector multiplication is a procedure where the row vector of the output from the preceding layers is equal to the column vector of the dense layer.

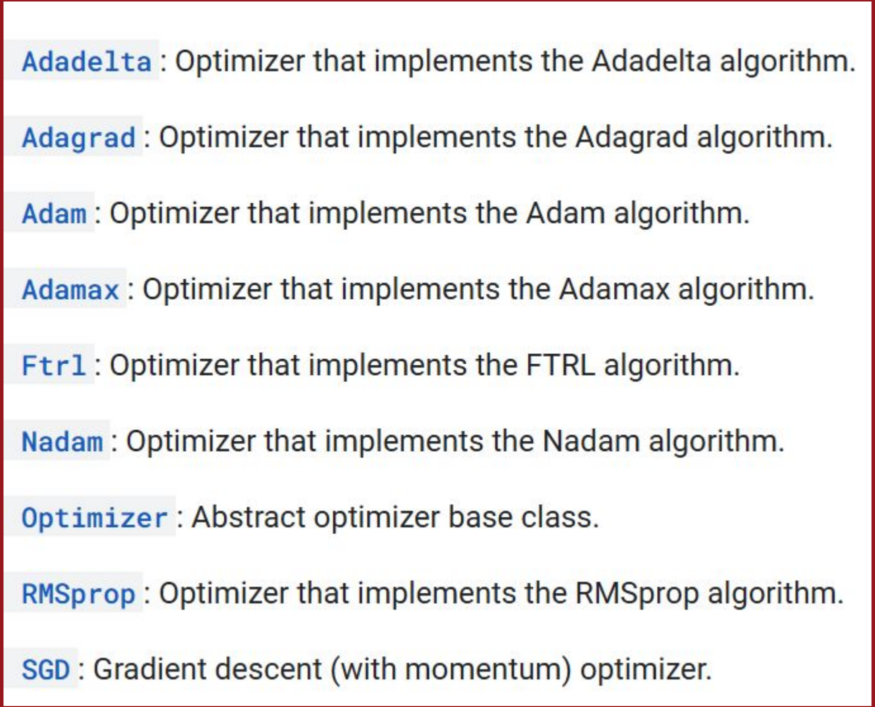


Assembler Institute of Technology

OPTIMIZERS

OPTIMIZERS

Optimization is an important process which optimize the input weights by comparing the prediction and the loss function. Keras provides quite a few optimizer as a module, *optimizers* and they are as follows:



`Adadelta`	: Optimizer that implements the Adadelta algorithm.
`Adagrad`	: Optimizer that implements the Adagrad algorithm.
`Adam`	: Optimizer that implements the Adam algorithm.
`Adamax`	: Optimizer that implements the Adamax algorithm.
`Ftrl`	: Optimizer that implements the FTRL algorithm.
`Nadam`	: Optimizer that implements the Nadam algorithm.
`Optimizer`	: Abstract optimizer base class.
`RMSprop`	: Optimizer that implements the RMSprop algorithm.
`SGD`	: Gradient descent (with momentum) optimizer.

Assembler
Institute of Technology

OPTIMIZERS

SGD – Stochastic gradient descent optimizer.

```
keras.optimizers.SGD(learning_rate = 0.01, momentum = 0.0, nesterov = False)
```

- It’s a variant of Gradient Descent.
- It tries to update the model’s parameters more frequently.
- In this, the model parameters are altered after computation of loss on each training example.
- If the dataset contains 1000 rows SGD will update the model parameters 1000 times in one cycle of dataset instead of one time as in Gradient Descent.
- $\theta=\theta-\alpha\cdot\nabla J(\theta;x(i);y(i))$ **, where {x(i) ,y(i)}** are the training examples.
- As the model parameters are frequently updated parameters have high variance and fluctuations in loss functions at different intensities.

Assembler
Institute of Technology

OPTIMIZERS

SGD – Stochastic gradient descent optimizer.

```
keras.optimizers.SGD(learning_rate = 0.01, momentum = 0.0, nesterov = False)
```

Advantages:

1. Frequent updates of model parameters hence, converges in less time.
2. Requires less memory as no need to store values of loss functions.
3. May get new minima's.

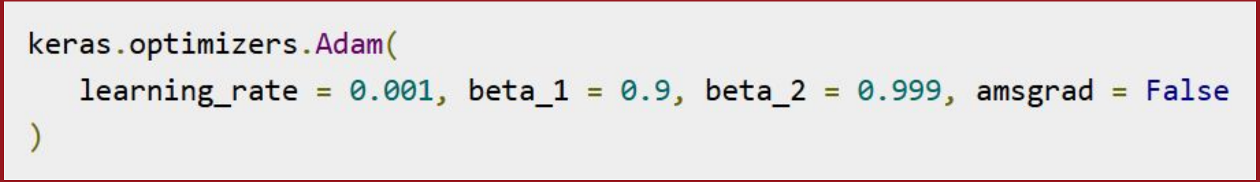
Disadvantages:

1. High variance in model parameters.
2. May shoot even after achieving global minima.
3. To get the same convergence as gradient descent needs to slowly reduce the value of learning rate.

Assembler
Institute of Technology

OPTIMIZERS

Adam – Adam optimizer



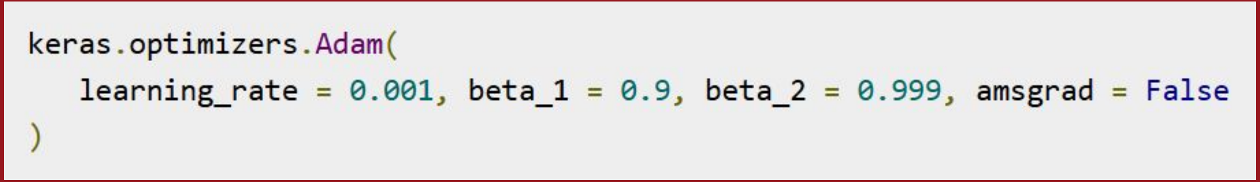
```
keras.optimizers.Adam(
    learning_rate = 0.001, beta_1 = 0.9, beta_2 = 0.999, amsgrad = False
)
```

- **Adam** (Adaptive Moment Estimation) works with momentums of first and second order.
- The intuition behind the Adam is that we don’t want to roll so fast just because we can jump over the minimum, we want to decrease the velocity a little bit for a careful search.
- In addition to storing an exponentially decaying average of past squared gradients like **AdaDelta**, ***Adam*** also keeps an exponentially decaying average of past gradients **M(t).**

Assembler
Institute of Technology

OPTIMIZERS

Adam – Adam optimizer



```
keras.optimizers.Adam(
    learning_rate = 0.001, beta_1 = 0.9, beta_2 = 0.999, amsgrad = False
)
```

Advantages:

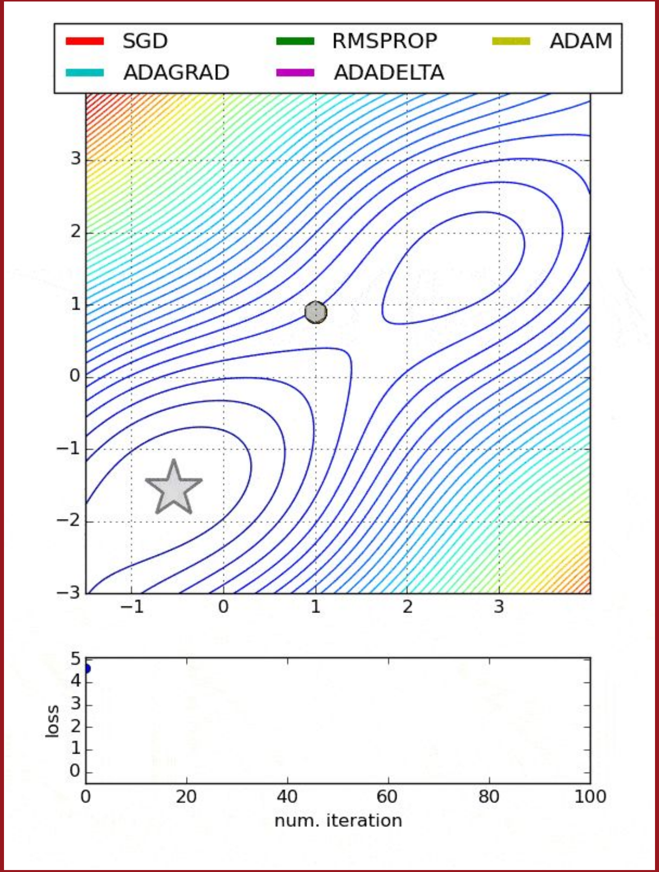
1. The method is too fast and converges rapidly.
2. Rectifies vanishing learning rate, high variance.

Disadvantages:

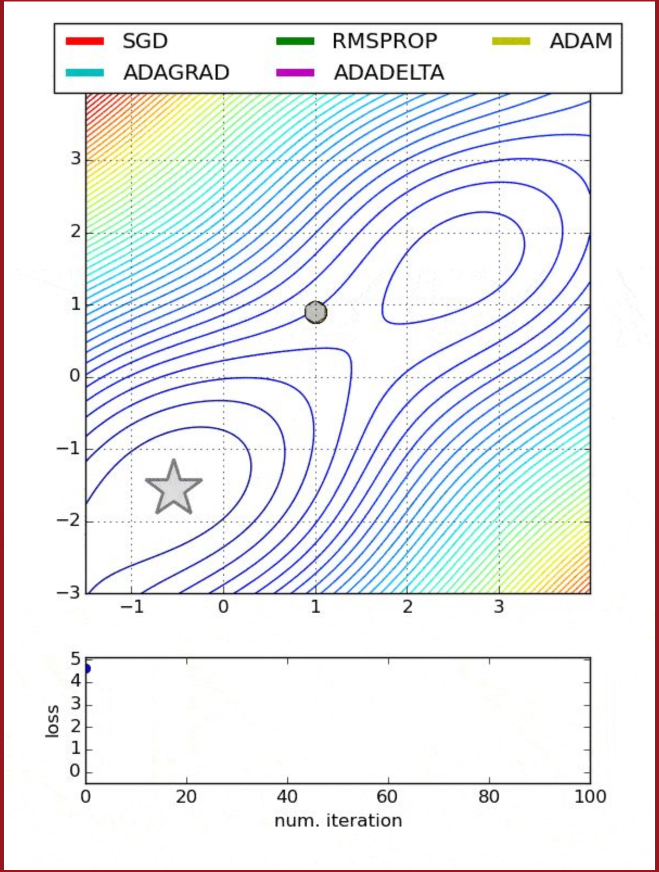
Computationally costly.

Assembler
Institute of Technology

OPTIMIZERS



Assembler Institute of Technology



How to choose optimizers?

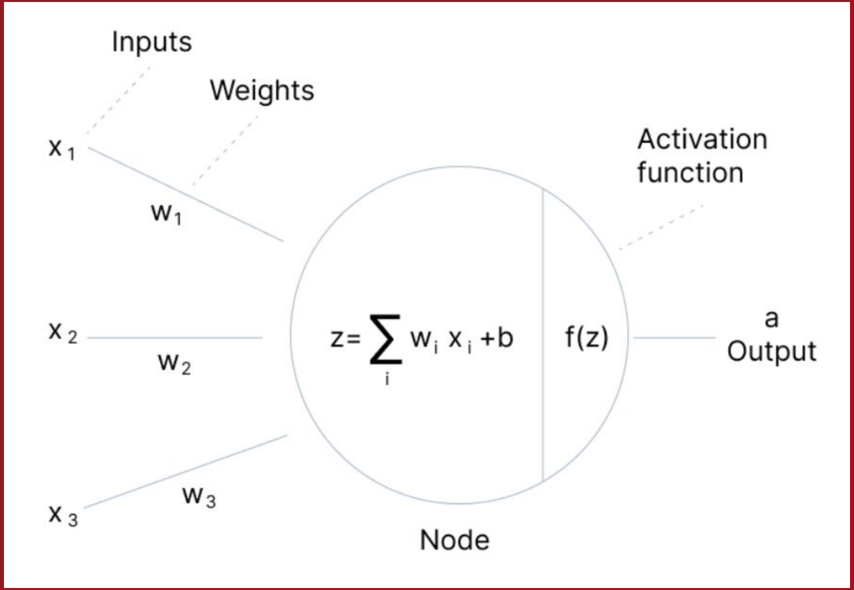
- If the data is sparse, use the self-applicable methods, namely Adagrad, Adadelta, RMSprop, Adam.
- RMSprop, Adadelta, Adam have similar effects in many cases.
- Adam just added bias-correction and momentum on the basis of RMSprop,
- As the gradient becomes sparse, Adam will perform better than RMSprop.

Assembler
Institute of Technology

ACTIVATION FUNCTIONS

ACTIVATION FUNCTIONS

- The activation function decides whether a neuron should be activated or not by calculating the weighted sum and further adding bias to it.
- The purpose of the activation function is to introduce non-linearity into the output of a neuron.



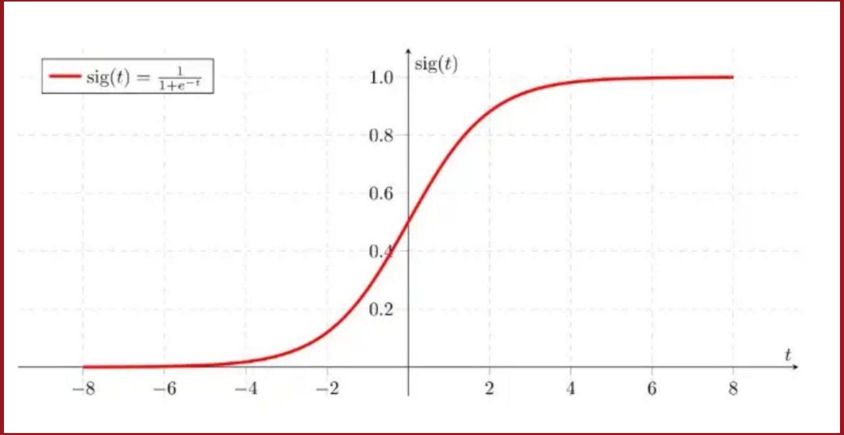
Assembler Institute of Technology

ACTIVATION FUNCTIONS

Sigmoid

- This function takes any real value as input and outputs values in the range of 0 to 1.
- The larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to 0.0.

- It is computationally expensive, causes vanishing gradient problem and not zero-centred.
- This method is generally used for binary classification problems.



Assembler Institute of Technology

ACTIVATION FUNCTIONS

Softmax

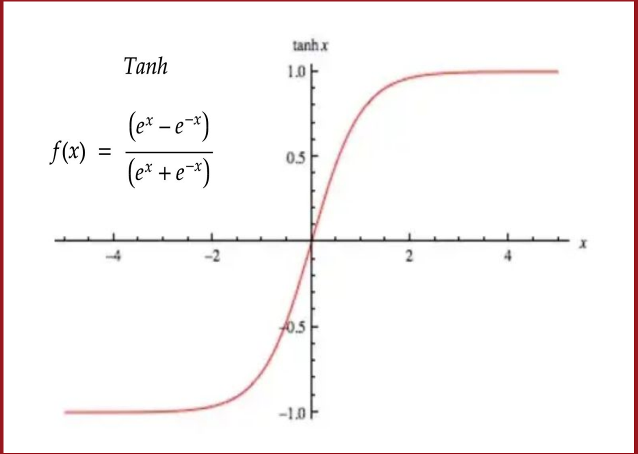
- The softmax is a more generalised form of the sigmoid. It is used in **multi-class classification problems**.
- Similar to sigmoid, it produces values in the range of 0–1 therefore it is used as the final layer in classification models.
- This function returns the probability for a datapoint belonging to each individual class.
- While building a network for a multiclass problem, the output layer would have as many neurons as the number of classes in the target.

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

ACTIVATION FUNCTIONS

Tanh

- The tanh function is very similar to the sigmoid function.
- The only difference is that it is symmetric around the origin.
- The range of values in this case is from -1 to 1.
- The inputs to the next layers will not always be of the same sign.
- Usually tanh is preferred over the sigmoid function since it is zero centered and the gradients are not restricted to move in a certain direction.
- The advantage is that the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero in the tanh graph.
- The function is differentiable.
- The function is monotonic while its derivative is not monotonic.
- The tanh function is mainly used classification between two classes.

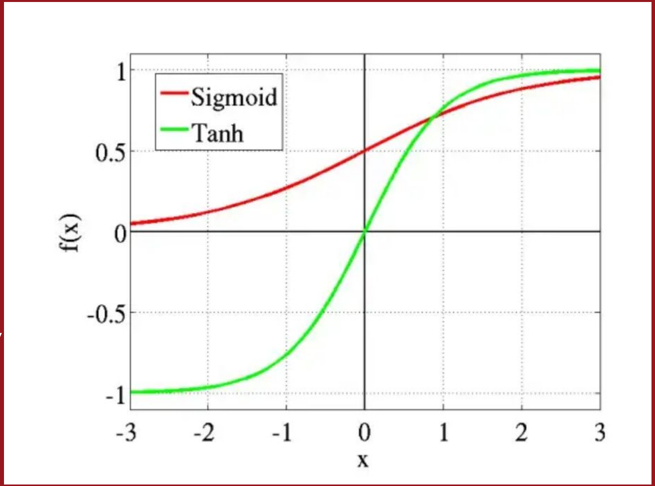


Assembler Institute of Technology

ACTIVATION FUNCTIONS

Tanh

- The tanh function is very similar to the sigmoid function.
- The only difference is that it is symmetric around the origin.
- The range of values in this case is from -1 to 1.
- The inputs to the next layers will not always be of the same sign.
- Usually tanh is preferred over the sigmoid function since it is zero centered and the gradients are not restricted to move in a certain direction.
- The advantage is that the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero in the tanh graph.
- The function is differentiable.
- The function is monotonic while its derivative is not monotonic.
- The tanh function is mainly used classification between two classes.

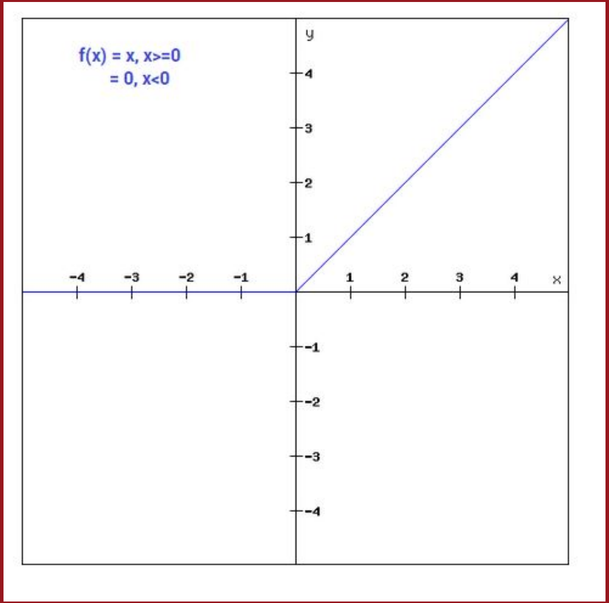


Assembler
Institute of Technology

ACTIVATION FUNCTIONS

ReLu

- ReLU (Rectified Linear Unit) is defined as f(x) = max(0,x).
- The main advantage of using the ReLU function over other activation functions is that it does not activate all the neurons at the same time.
- This means that the neurons will only be deactivated if the output of the linear transformation is less than 0.
- This is a widely used activation function, especially with Convolutional Neural networks.
- It is easy to compute and does not saturate and does not cause the Vanishing Gradient Problem.
- It has just one issue of not being zero centred.
- It suffers from “dying ReLU” problem: Since the output is zero for all negative inputs, it causes some nodes to completely die and not learn anything.



Assembler
Institute of Technology

How to use them in deep neural networks?

- **Tanh and sigmoid cause huge vanishing gradient problems**.
- **Start with ReLU in your network**. Activation layer is added after the weight layer (something like CNN, RNN, LSTM or linear dense layer). If you think the model has stopped learning, then you can replace it with a LeakyReLU to avoid the Dying ReLU problem. However, the Leaky ReLU will increase the computation time a little bit.
- **If you also have Batch-Norm layers in your network, that is added before the activation function making the order CNN-Batch Norm-*Act***.
- Activation functions work best in their default hyperparameters that are used in popular frameworks such as Tensorflow and Pytorch. However, one can fiddle with the negative slope in **LeakyReLU and set it to 0.02** to expedite learning.