22/08/25 Lab 3 : Study of Activation Functions and its role.

Aim: To study the activation functions and its role

Objective :

- To understand why activation functions are necessary in neural network.
- To study commonly used activation fn: sigmoid, Tanh, ReLu, Leaky ReLu, Softmax
- To compare their advantages, disadvantages and suitable application.

Pseudo code

1) Import required libraries. &
2) Define mathematical functions for sigmoid, tanh, relu, leaky rely, softmax.
3) Generate a range of input values (x).
4) Compute outputs of all activation functions
5) Plot Graph of each activation functions
6) Compare their behaviour and note observation.

Observation

Formula :

1) Sigmoid = $f(x) = \frac{1}{1+e^{-x}}$

2) Tanh : $f(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$

3) ReLU (Rectified Linear Unit)

$f(x) = \max(0, x)$

4) Leaky ReLU

$$f(x) = \begin{cases} a & x > 0 \\ ax & x \leq 0 \end{cases}$$

5) Softmax

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

Observation Table :

Activation fn	Output Range	Advantage	Use Case
1) Sigmoid	(0, 1)	Smooth, probabilistic output	Binary Classification
2) Tanh	(-1, 1)	Centered around 0, better than sigmoid	Hidden layer (old older network)
3) ReLU	[0, ∞)	Fast, reduce computation Time	Hidden layer (modern CNN/ ANN)
4) Leaky ReLU	(-∞, ∞)	Fixed ReLU (dying) issues	Deep Hidden layer
5) Softmax	(0, 1), sum = 1	Give probability distribution	Output layer for multi-class

* ReLU and Leaky ReLU are most effective in hidden layer.
* Sigmoid and Tanh are rarely used today due to vanishing gradient
* Softmax for multiclass classification problems

Result

Studied different activation function & their roles.

10.1.38.19/user/ra2311047010018/lab/tree/l5.ipynb

Verify it's you

Kernel Tabs Settings Help

l5.ipynb

Code

Notebook Python 3 (ipykernel)

Modified

onths ago
nutes ago
onths ago
onths ago
onths ago
days ago
days ago
days ago
nutes ago
conds ago
days ago
days ago

[2]:
```
import torch
import torch.nn.functional as F
```

```
Input: tensor([-2., -1.,  0.,  1.,  2.])
Sigmoid: tensor([0.1192, 0.2689, 0.5000, 0.7311, 0.8808])
Tanh: tensor([-0.9640, -0.7616,  0.0000,  0.7616,  0.9640])
ReLU: tensor([0., 0., 0., 1., 2.])
Leaky ReLU: tensor([-0.2000, -0.1000,  0.0000,  1.0000,  2.0000])
Softmax: tensor([0.0117, 0.0317, 0.0861, 0.2341, 0.6364])
```

```
x = torch.tensor([-2.0, -1.0, 0.0, 1.0, 2.0])
```

[6]:
```
sigmoid = torch.sigmoid(x)
tanh = torch.tanh(x)
relu = F.relu(x)
leaky_relu = F.leaky_relu(x, negative_slope=0.1)
softmax = F.softmax(x, dim=0)  # Softmax over the tensor elements
```

[7]:
```
print("Input:", x)
print("Sigmoid:", sigmoid)
print("Tanh:", tanh)
print("ReLU:", relu)
print("Leaky ReLU:", leaky_relu)
```

```
leaky_relu = F.leaky_relu(x, negative_slope=0.1)
softmax = F.softmax(x, dim=0)  # Softmax over the tensor elements
```

[7]:
```
print("Input:", x)
print("Sigmoid:", sigmoid)
print("Tanh:", tanh)
print("ReLU:", relu)
print("Leaky ReLU:", leaky_relu)
print("Softmax:", softmax)
```

```
Input: tensor([-2., -1.,  0.,  1.,  2.])
Sigmoid: tensor([0.1192, 0.2689, 0.5000, 0.7311, 0.8808])
Tanh: tensor([-0.9640, -0.7616,  0.0000,  0.7616,  0.9640])
ReLU: tensor([0., 0., 0., 1., 2.])
Leaky ReLU: tensor([-0.2000, -0.1000,  0.0000,  1.0000,  2.0000])
Softmax: tensor([0.0117, 0.0317, 0.0861, 0.2341, 0.6364])
```

[ ]:

odified
hs ago
es ago
hs ago
hs ago
hs ago
ys ago
ys ago
ys ago
es ago
te ago
ys ago
ys ago