NAME : SYED RIAZ ALI

EMAIL : syedriazali1997@gmal.com

Whatsapp : 923002502513

DAY - 24 : ACTIVATION FUNCTION ASSIGNMENT

1- INTRODUCTION TO ACTIVATION FUNCTION

- The Internet provides access to plethora of information today. Whatever we need is just a Google (search) away. However, when we have so much information, the challenge is to segregate between relevant and irrelevant information.
- When our brain is fed with a lot of information simultaneously, it tries hard to understand and classify the information into "useful" and "not-so-useful" information. We need a similar mechanism for classifying incoming information as "useful" or "less-useful" in case of Neural Networks.
- This is important in the way a network learns because not all the information is equally useful. Some of it is just noise. This is where activation functions come into picture. The activation functions help the network use the important information and suppress the irrelevant data points.

2 - TYPES OF ACTIVATION FUNCTION

1. Binary Step
2. Linear
3. Sigmoid
4. Tanh
5. ReLU
6. Leaky ReLU
7. Parameterised ReLU
8. Exponential Linear Unit
9. Swish
10. Softmax

We understand that using an activation function introduces an additional step at each layer during the forward propagation. Now the question is if the activation function increases the complexity so much, can we do without an activation function?

Imagine a neural network without the activation functions. In that case, every neuron will only be performing a linear transformation on the inputs using the weights and biases. Although linear transformations make the neural network simpler, but this network would be less powerful and will not be able to learn the complex patterns from the data.

A neural network without an activation function is essentially just a linear regression model.

Thus we use a non linear transformation to the inputs of the neuron and this non-linearity in the network is introduced by an activation function.

Without activation function, weight and bias would only have a linear transformation, or neural network is just a linear regression model, a linear equation is polynomial of one degree only which is simple to solve but limited in terms of ability to solve complex problems or higher degree polynomials.

But opposite to that, the addition of activation function to neural network executes the non-linear transformation to input and make it capable to solve complex problems such as language translations and image classifications.

In addition to that, Activation functions are differentiable due to which they can easily implement back propagations, optimized strategy while performing backpropagations to measure gradient loss functions in the neural networks.

1. Binary Step Function

The first thing that comes to our mind when we have an activation function would be a threshold based classifier i.e. whether or not the neuron should be activated based on the value from the linear transformation.

In other words, if the input to the activation function is greater than a threshold, then the neuron is activated, else it is deactivated, i.e. its output is not considered for the next hidden layer.

This is the simplest activation function, which can be implemented with a single if-else condition in python

```
def binary_step(x):
    if x<0:
        return 0
    else:
        return 1
```

The binary step function can be used as an activation function while creating a binary classifier. As you can imagine, this function will not be useful when there are multiple classes in the target variable. That is one of the limitations of binary step function.

Moreover, the gradient of the step function is zero which causes a hindrance in the back propagation process. That is, if you calculate the derivative of f(x) with respect to x, it comes out to be 0.

Gradients are calculated to update the weights and biases during the backpropogation process. Since the gradient of the function is zero, the weights and biases don't update.

2. Linear Function

We saw the problem with the step function, the gradient of the function became zero. This is because there is no component of x in the binary step function. Instead of a binary function, we can use a linear function. We can define the function as

```
def linear_function(x):
    return 4*x
```

Although the gradient does not become zero, but it is a constant which does not depend upon the input value x at all. This implies that the weights and biases will be updated during the backpropagation process but the updating factor would be the same.

In this scenario, the neural network will not really improve the error since the gradient is the same for every iteration. The network will not be able to train well and capture the complex patterns from the data. Hence, linear function might be ideal for simple tasks where interpretability is highly desired.

3. Sigmoid Activation Function / S - Function

The next activation function that we are going to look at is the Sigmoid function. It is one of the most widely used non-linear activation function. Sigmoid transforms the values between the range 0 and 1. Here is the mathematical expression for sigmoid

```
f(x) = 1/(1+e^-x)
```

A noteworthy point here is that unlike the binary step and linear functions, sigmoid is a non-linear function. This essentially means when I have multiple neurons having sigmoid function as their activation function,the output is non linear as well. Here is the python code for defining the function in python

```
import numpy as np
def sigmoid_function(x):
    z = (1/(1 + np.exp(-x)))
    return z
```

The sigmoid function causes a problem mainly termed as vanishing gradient problem which occurs because we convert large input in between the range of 0 to 1 and therefore their derivatives become much smaller which does not give satisfactory output. To solve this problem another activation function such as ReLU is used where we do not have a small derivative problem.

4. Tanh Function (Hyperbolic Tangent Activation Function)

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. Thus the inputs to the next layers will not always be of the same sign. The tanh function is defined as

```
tanh(x)=2sigmoid(2x)-1

def tanh_function(x):
    z = (2/(1 + np.exp(-2*x))) -1
    return z
```

The gradient of the tanh function is steeper as compared to the sigmoid function. You might be wondering, how will we decide which activation function to choose? 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.

5. ReLU (Rectified Linear unit) Activation Function

The ReLU function is another non-linear activation function that has gained popularity in the deep learning domain. ReLU stands for Rectified Linear Unit. 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.

```
def relu_function(x):
    if x<0:
        return 0
    else:
        return x
```

For the negative input values, the result is zero, that means the neuron does not get activated. Since only a certain number of neurons are activated, the ReLU function is far more computationally efficient when compared to the sigmoid and tanh function.

If you look at the negative side of the graph, you will notice that the gradient value is zero. Due to this reason, during the backpropogation process, the weights and biases for some neurons are not updated. This can create dead neurons which never get activated. This is taken care of by the 'Leaky' ReLU function.

6. Leaky ReLU

Leaky ReLU function is nothing but an improved version of the ReLU function. As we saw that for the ReLU function, the gradient is 0 for x<0, which would deactivate the neurons in that region.

Leaky ReLU is defined to address this problem. Instead of defining the Relu function as 0 for negative values of x, we define it as an extremely small linear component of x. Here is the mathematical expression

```
f(x)= 0.01x, x<0
    =   x, x>=0
```

By making this small modification, the gradient of the left side of the graph comes out to be a non zero value. Hence we would no longer encounter dead neurons in that region

Since Leaky ReLU is a variant of ReLU, the python code can be implemented with a small modification

```
def leaky_relu_function(x):
    if x<0:
        return 0.01*x
    else:
        return x
```

Apart from Leaky ReLU, there are a few other variants of ReLU, the two most popular are – Parameterised ReLU function and Exponential ReLU.

7. Parameterised ReLU

This is another variant of ReLU that aims to solve the problem of gradient's becoming zero for the left half of the axis. The parameterised ReLU, as the name suggests, introduces a new parameter as a slope of the negative part of the function. Here's how the ReLU function is modified to incorporate the slope parameter

```
f(x) = x, x>=0
     = ax, x<0
```


However, in case of a parameterised ReLU function, 'a' is also a trainable parameter. The network also learns the value of 'a' for faster and more optimum convergence.

The parameterized ReLU function is used when the leaky ReLU function still fails to solve the problem of dead neurons and the relevant information is not successfully passed to the next layer.

8. Exponential Linear Unit

Exponential Linear Unit or ELU for short is also a variant of Rectified Linear Unit (ReLU) that modifies the slope of the negative part of the function. Unlike the leaky ReLU and parametric ReLU functions, instead of a straight line, ELU uses a log curve for defining the negatice values. It is defined as

```
f(x) = x,   x>=0
     = a(e^x-1), x<0

def elu_function(x, a):
    if x<0:
        return a*(np.exp(x)-1)
    else:
        return x
```

9. Swish

Swish is a lesser known activation function which was discovered by researchers at Google. Swish is as computationally efficient as ReLU and shows better performance than ReLU on deeper models. The values for swish ranges from negative infinity to infinity. The function is defined as

```
f(x) = x*sigmoid(x)
f(x) = x/(1-e^-x)
```

As you can see, the curve of the function is smooth and the function is differentiable at all points. This is helpful during the model optimization process and is considered to be one of the reasons that swish out performs ReLU.

10. Softmax

Softmax function is often described as a combination of multiple sigmoids. We know that sigmoid returns values between 0 and 1, which can be treated as probabilities of a data point belonging to a particular class. Thus sigmoid is widely used for binary classification problems.

The softmax function can be used for multiclass classification problems. 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. For instance if you have three classes, there would be three neurons in the output layer. Suppose you got the output from the neurons as [1.2 , 0.9 , 0.75].

Applying the softmax function over these values, you will get the following result [0.42 , 0.31, 0.27]. These represent the probability for the data point belonging to each class.

```
def softmax_function(x):
    z = np.exp(x)
    z_ = z/z.sum()
    return z_
```

3 - Choosing the right Activation Function

Now that we have seen so many activation functions, we need some logic / heuristics to know which activation function should be used in which situation. Good or bad – there is no rule of thumb.

However depending upon the properties of the problem we might be able to make a better choice for easy and quicker convergence of the network.

- Sigmoid functions and their combinations generally work better in the case of classifiers
- Sigmoids and tanh functions are sometimes avoided due to the vanishing gradient problem
- ReLU function is a general activation function and is used in most cases these days
- If we encounter a case of dead neurons in our networks the leaky ReLU function is the best choice
- Always keep in mind that ReLU function should only be used in the hidden layers
- As a rule of thumb, you can begin with using ReLU function and then move over to other activation functions in case ReLU doesn't provide with optimum results