Lecture 03 Notes [22/08/19]

[Sapna 15MI448]

[Nitish 15MI416]

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

Why do we need Non-linear activation functions?

The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks.

Activation Functions:

1.The Sigmoid Function:

Most often, we would want to predict our outcomes as YES/NO (1/0).

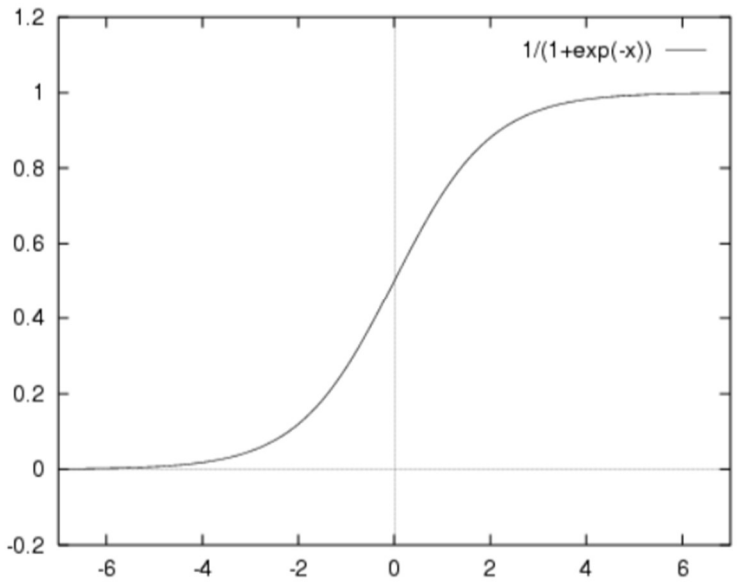
For example:

Is your favourite football team going to win the match today? — yes/no (0/1)

Does a student pass in exam? — yes/no (0/1)

Sigmoid function f(x):

f(x) = 1 / 1 + exp(-x)



The sigmoid curve

The sigmoid function gives an ‘S’ shaped curve.

This curve has a finite limit of:

‘0’ as x approaches $-\infty$

‘1’ as x approaches $+\infty$

The output of sigmoid function when x=0 is 0.5

Thus, if the output is more than 0.5, we can classify the outcome as 1 (or YES) and if it is less than 0.5, we can classify it as 0(or NO) .

For example: If the output is 0.65, we can say in terms of probability as:

“There is a 65 percent chance that your favourite football team is going to win today ” .

Thus the output of the sigmoid function can not be just used to classify YES/NO, it can also be used to determine the probability of YES/NO.

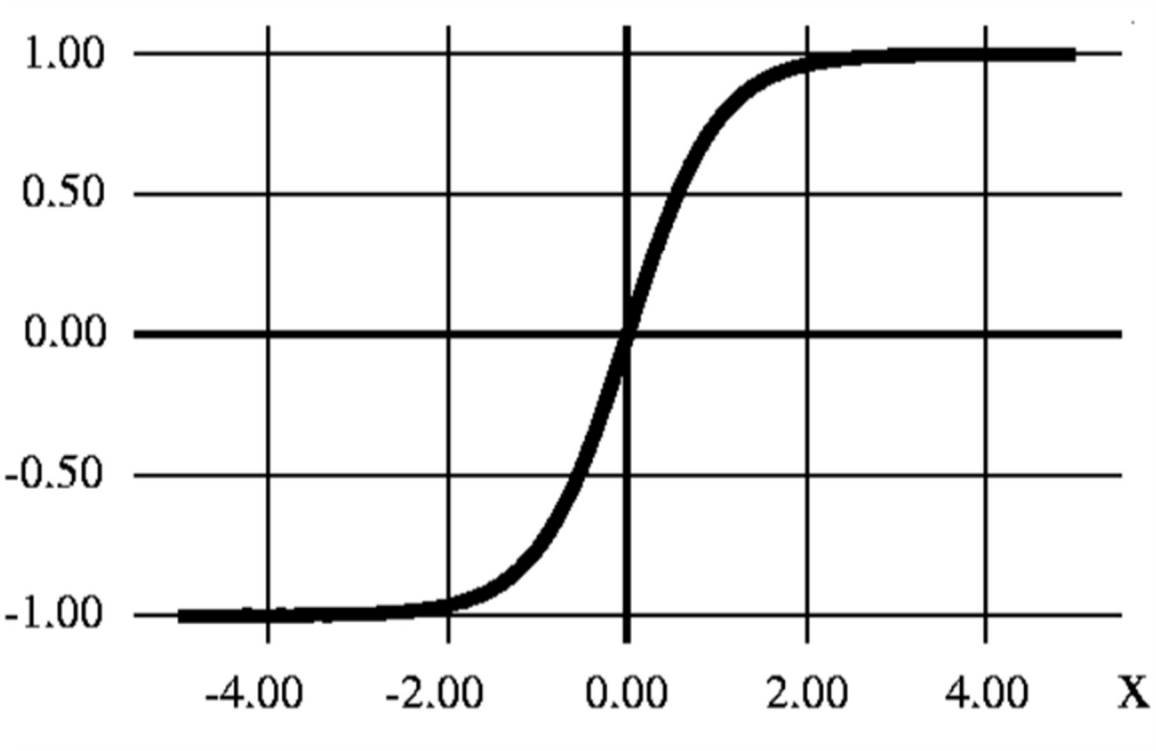
2.The Hyperbolic Tangent function- Tanh:

The tanh function is just another possible functions that can be used as a nonlinear activation function between layers of a neural network. It actually shares a few things in common with the sigmoid activation function. They both look very similar. But while a sigmoid function will map input values to be between 0 and 1, Tanh will map values to be between -1 and 1.

Mathematically:

$$f(x) = (1 - \exp(-2x)) / (1 + \exp(-2x))$$

The Tanh curve:



This curve has a finite limit of:

'-1' as x approaches $-\infty$

'1' as x approaches $+\infty$

Hence optimization is easier in this method hence in practice it is always preferred over Sigmoid function.

Usually used in hidden layers of a neural network as it's values lies between **-1 to 1** hence the mean for the hidden layer comes out be 0 or very close to it, hence helps in *centering the data* by bringing mean close to 0. This makes learning for the next layer much easier.

3. ReLu- RECTIFIED LINEAR UNIT

The Rectified Linear Unit or ReLU is considered the most commonly used activation function in deep learning models. The function simply outputs the value of 0 if it receives any negative input, but for any positive value z, it returns that value back like a linear function. So it can be written as f(z)=max(0,z)however, it should be noted that the ReLU function is still non-

linear so we are able to backpropagate the errors and have multiple layers of neurons.

This function was quickly adopted, as ReLU took care of several problems faced by the Sigmoid and the Tanh:

1.The ReLU function has a derivative of 0 over half of its range which spans across all the negative numbers. For positive inputs, the derivative is 1. So we have rectified the 'vanishing' gradient problem.

2.At a time only a few neurons are activated making the network sparse making it efficient.

3.It is computationally economical compared to Sigmoid and Tanh.

4.Cheap to compute as there is no complicated math and hence easier to optimize.

5.It converges faster. It accelerates the convergence of SGD compared to sigmoid and tanh (around 6 times).

6.Not have vanishing gradient problem like tanh or Sigmoid function.

7.It is capable of outputting a true zero value allowing the activation of hidden layers in neural networks to contain one or more true zero values called **Representational Sparsity.**

ReLU has its own set of limitations and disadvantages despite being a better activation function than its other non-linear alternatives:

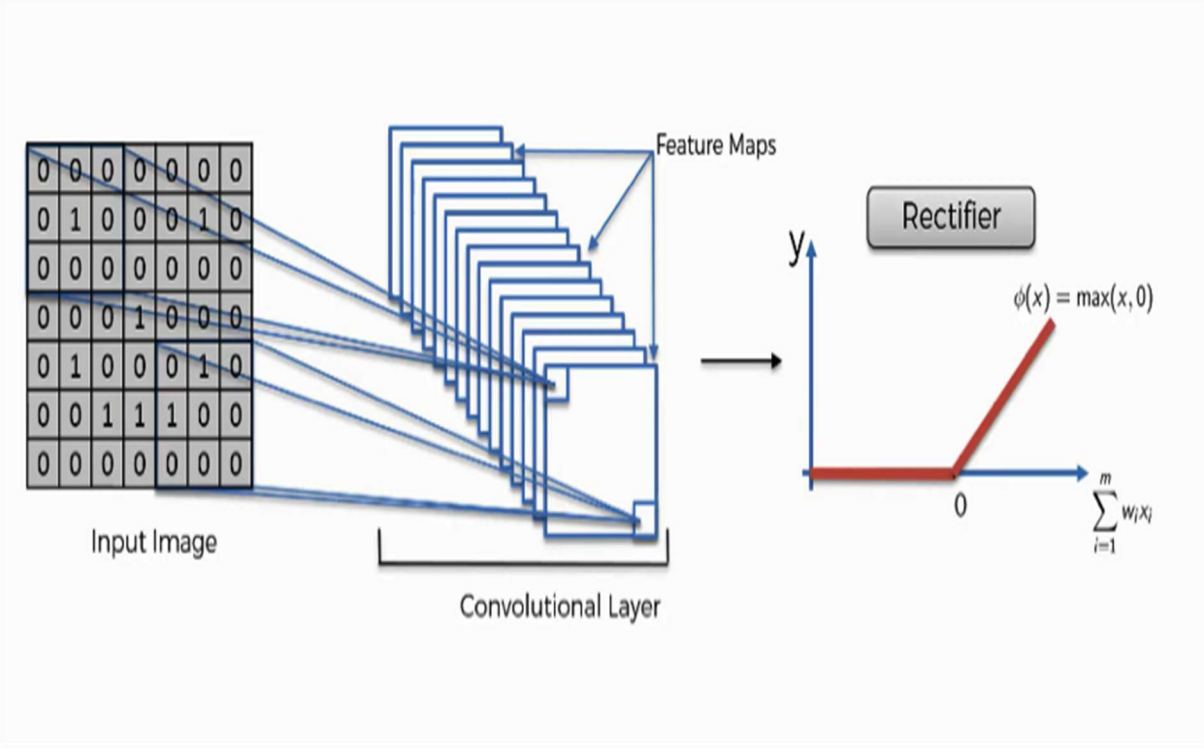
1.The function suffers from the dying ReLU problem — For activations correspondent to values of $z< 0$, the gradient will be 0 because of which the weights will not get adjusted during the gradient descent in backpropagation.

That means, such neurons will stop responding to variations in error/input, so the output network becomes passive due to added sparsity.

2.It is best used in between the input and output layers, more specifically within hidden layer.

APPLICATION OF ReLU:

The purpose of applying the rectifier function is to increase the non-linearity in our images. The reason we want to do that is that images are naturally non-linear.



THE INPUT IMAGE:

When you look at any image, you'll find it contains a lot of non-linear features (e.g. the transition between pixels, the borders, the colors, etc.).

The rectifier serves to break up the linearity even further in order to make up for the linearity that we might impose an image when we put it through the convolution operation. To see how that actually plays out, we can look at the following picture and see the changes that happen to it as it undergoes the convolution operation followed by rectification.

Feature detector:

By putting the image through the convolution process, or in other words, by applying to it a feature detector, the result is what you see in the following image.

As we see, the entire image is now composed of pixels that vary from white to black with many shades of gray in between.

Rectification

What the rectifier function does to an image like this is remove all the black elements from it, keeping only those carrying a positive value (the grey and white colors).The essential difference between the non-rectified version of the image and the rectified one is the progression of colors. If we look closely at the first one, we will find parts where a white streak is followed by a grey one and then a black one.

After we rectify the image, we will find the colors changing more abruptly. The gradual change is no longer there. That indicates that the linearity has been disposed of.

4. Leaky- ReLU

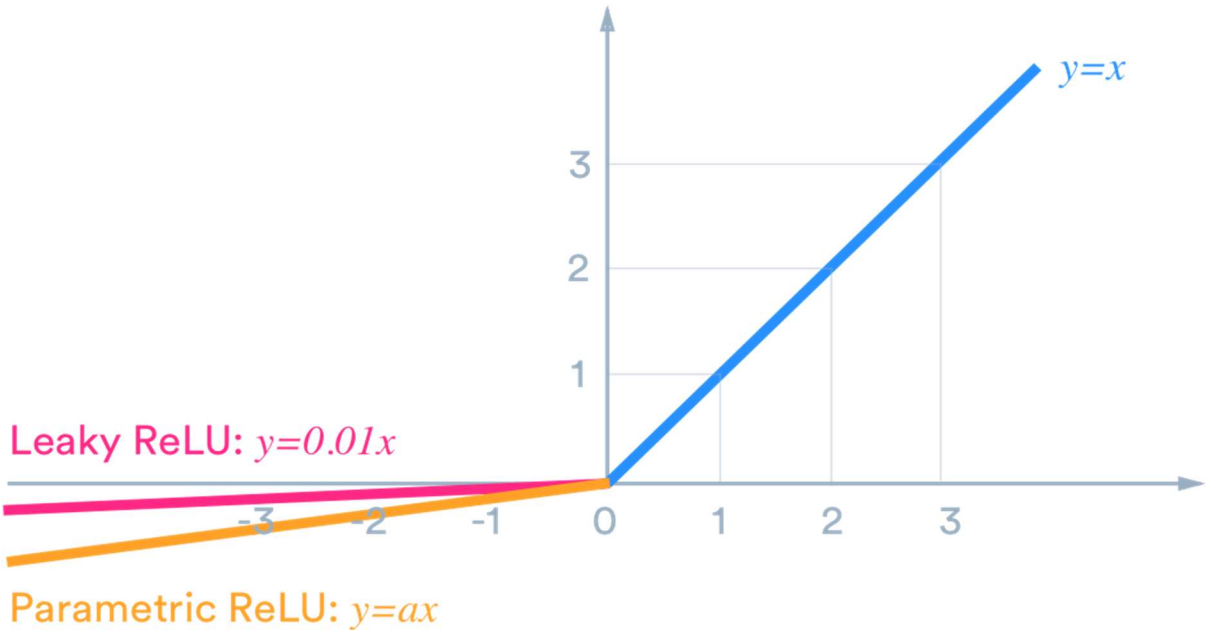
Leaky ReLU has a small slope for negative values, instead of altogether zero. For example, leaky ReLU may have $y = 0.01x$ when $x < 0$.

Leaky ReLU has two benefits:

1.It fixes the “dying ReLU” problem, as it doesn’t have zero-slope parts.

2.It speeds up training. There is evidence that having the “mean activation” be close to 0 makes training faster. (It helps keep off-diagonal entries of the Fisher information matrix small, but you can safely ignore this.) Unlike ReLU, leaky ReLU is more “balanced,” and may therefore learn faster.

The result is not always consistent. Leaky ReLU isn’t always superior to plain ReLU, and should be considered only as an alternative.



5. SOFTMAX

The softmax function is also a type of sigmoid function but is handy when we are trying to handle classification problems. The sigmoid function is able to handle just two classes. What shall we do when we are trying to handle multiple classes. Just classifying yes or no for a single class would not help then. The softmax function would squeeze the outputs for each class between 0 and 1 and would also divide by the sum of the outputs(as shown in the figure).

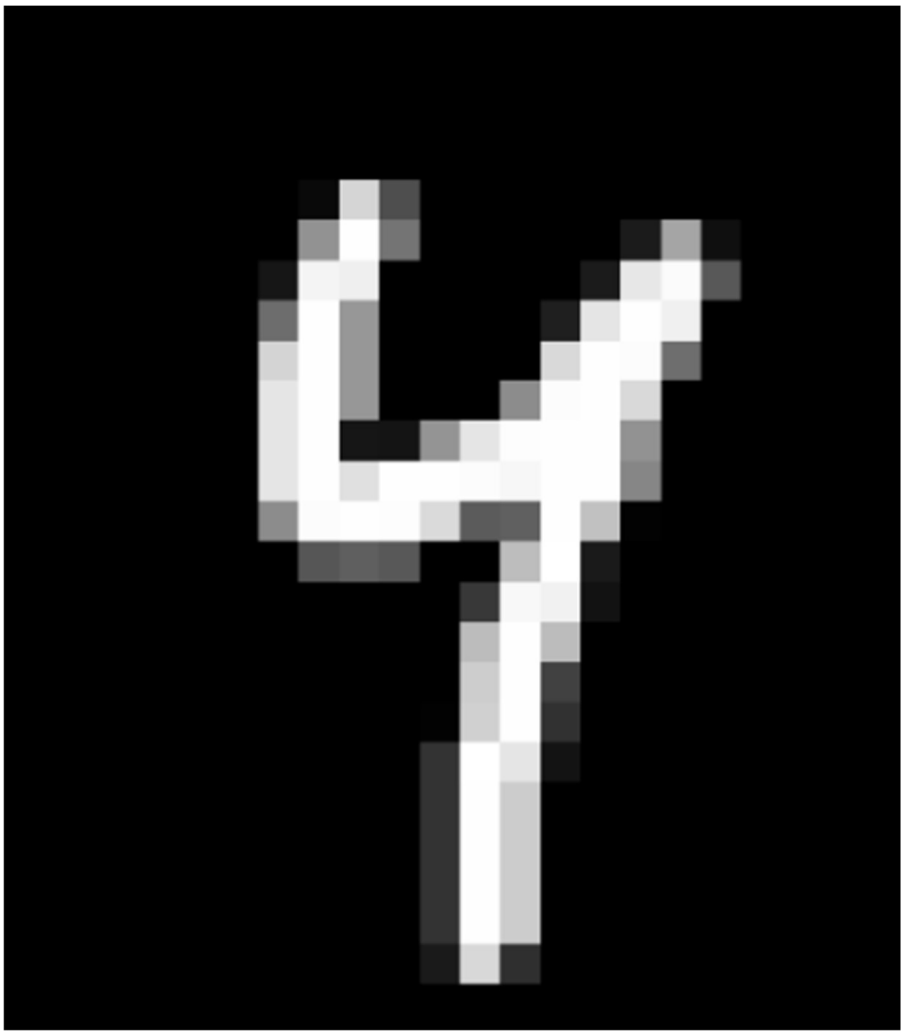
This essentially gives the probability of the input being in a particular class. It can be defined as –

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

EXAMPLE:

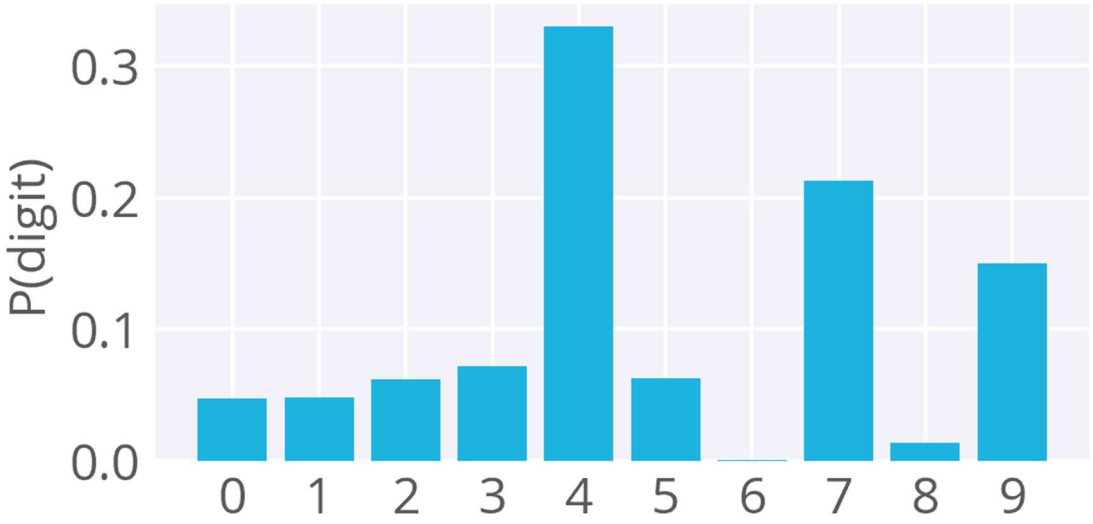
To understand this better, think about training a network to recognize and classify handwritten digits from images. The network would have ten output units, one for each digit 0 to 9. Then if you fed it an image of a number 4 (see

below), the output unit corresponding to the digit 4 would be activate



Building a network like this requires 10 output units, one for each digit. Each training image is labeled with the true digit and the goal of the network is to predict the correct label. So, if the input is an image of the digit 4, the output unit corresponding to 4 would be activated, and so on for the rest of the units.

For the example image above, the output of the softmax function might look like:



The image looks the most like the digit 4, so you get a lot of probability there. However, this digit also looks somewhat like a 7 and a little bit like a 9 without the loop completed. So, you get the most probability that it's a 4, but also some probability that it's a 7 or a 9.

The softmax can be used for any number of classes. It's also used for hundreds and thousands of classes, for example in object recognition problems where there are hundreds of different possible objects.
