Softmax

- Softmax is used as the activation function for multi-class classification tasks, usually the last layer.
- We talked about its role transforming numbers (aka logits) into probabilities that sum to one.
- Let's not forget it is also an activation function which means it helps our model achieve non-linearity. Linear combinations of linear combinations will always be linear but adding activation function helps gives our model ability to handle non-linear data.
- Output of other activation functions such as sigmoid does not necessarily sum to one. Having outputs summing to one makes softmax function great for probability analysis.
- The function is great for classification problems, especially if you're dealing with multi-class classification problems, as it will report back the "confidence score" for each class. Since we're dealing with probabilities here, the scores returned by the softmax function will add up to 1.

Mathematical representation

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Where:

σ = softmax

$\vec{z}$=input vector

e^{z_i} =standard exponential function for input vector

K = number of classes in the multi-class classifier

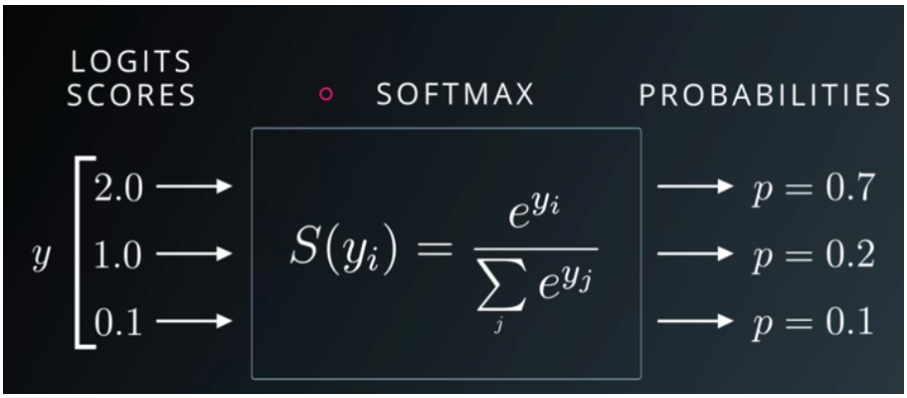
e^{z_j} =standard exponential function for output vector

- It states that we need to apply a standard exponential function to each element of the output layer, and then normalize these values by dividing by the sum of all the exponentials. Doing so ensures the sum of all exponentiated values adds up to 1.

Here are the steps For Softmax:

1. Exponentiate every element of the output layer and sum the results
2. Take each element of the output layer, exponentiate it and divide by the sum obtained in step 1

Example Implementation



To start, let's declare an array which imitates the output layer of a neural network:

In [1]:
```
## as we know softmax used in output layer so here i take a outputlayer value
import numpy as np
output_layer = np.array([2.0,1.0,0.1])
output_layer
```

Out[1]:
```
array([2. , 1. , 0.1])
```

By step 1 we need to exponentiate each of the elements of the output layer:

In [2]:
```
exponentiated = np.exp(output_layer)
exponentiated
```

Out[2]:
```
array([7.3890561 , 2.71828183, 1.10517092])
```

According to step 2 calculate probabilities! We can use Numpy to divide each element by exponentiated sum and store results in another array

In [3]:
```
probabilities = exponentiated / np.sum(exponentiated)
print(probabilities)
print(sum(probabilities))
print(np.argmax(probabilities))
```

```
[0.65900114 0.24243297 0.09856589]
1.0
0
```

Here see the output are formed. If we sum three then we get probability 1. after this you use argmax function which return highest value index number. See here return 0 as 0 index have 0.65 value which is highest among three value.

When you use softmax in your dataset you should use argmax function to predict output.

- From a probabilistic perspective, if the argmax() function returns 1 in the large value, it returns 0 for the other two array indexes.here it giving full weight to index 0 and no weight to index 1

and index 2 for the largest value in the list [0.65,0.24,0.09].

In the Keras deep learning library with a three-class classification task, use of softmax in the output layer may look as follows:

model.add(Dense(no.of output layer, activation='softmax'))

- It apply when you have multiclass problem aries

The Differences between Sigmoid and Softmax Activation Functions

In [4]:
```
import numpy as np
### sigmoid function
def sigmoid(x):
    s = 1 / (1 + np.exp(-x))
    return s

## softmax function
def softmax(x):
    exponentiated = np.exp(x)
    probabilities = exponentiated / np.sum(exponentiated)
    return probabilities
```

In [5]:
```
x = np.array([-0.5, 1.2, -0.1, 2.4])
a = sigmoid(x)
print("--- Sigmoid---")
print(a.round(2))
print(sum(a).round(2))

print(50*"*")

output_layer = np.array([-0.5, 1.2, -0.1, 2.4])
b = softmax(output_layer)
print("---Softmax---")
print(b.round(2))
print(sum(b))
```

```
--- Sigmoid---
[0.38 0.77 0.48 0.92]
2.54
**************************************************
---Softmax---
[0.04 0.21 0.06 0.7 ]
1.0
```

The key takeaway from this example is:

- **Sigmoid:** probabilities produced by a Sigmoid are independent. Furthermore, they are not constrained to sum to one: 0.38 + 0.77 + 0.48 + 0.92 = 2.54. The reason for this is because the Sigmoid looks at each raw output value separately.
- **Softmax:** the outputs are interrelated. The Softmax probabilities will always sum to one by design: 0.04 + 0.21 + 0.06 + 0.7 = 1.00. In this case, if we want to increase the likelihood of one class, the other has to decrease by an equal amount.

Summary..

Characteristics of a Sigmoid Activation Function:

1. Used for Binary Classification in the Logistic Regression model
2. The probabilities sum does not need to be 1
3. Used as an Activation Function while building a Neural Network

Characteristics of a Softmax Activation Function

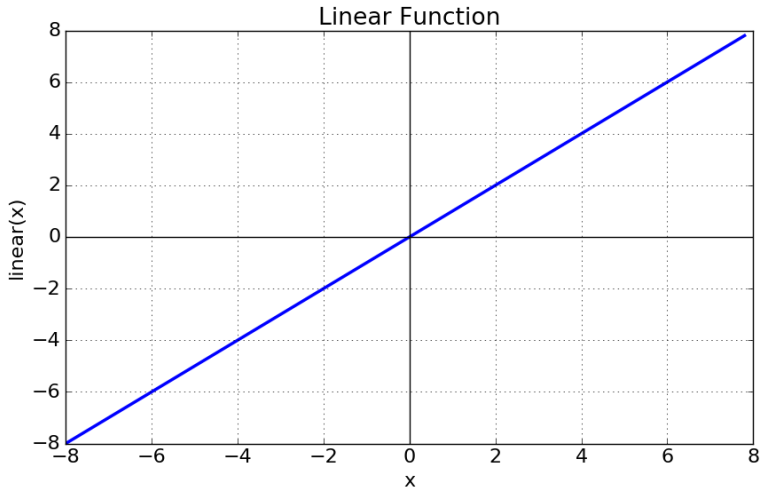
1. Used for Multi-classification in the Logistics Regression model
2. The probabilities sum will be 1
3. Used in the different layers of Neural Networks

Activation Function For Regression Problem

- Linear Activation Function:-

Equation : f(x) = x

Range : (-infinity to infinity)



- No matter how many layers we have, if all are linear in nature, the final activation function of last layer is nothing but just a linear function of the input of first layer.
- Linear activation function is used at just one place i.e. output layer.
- If we will differentiate linear function to bring non-linearity, result will no more depend on input “x” and function will become constant, it won’t introduce any ground-breaking behavior to our algorithm.

algorithm.

Uses: Calculation of price of a house is a regression problem. House price may have any big/small value, so we can apply linear activation at output layer. Even in this case neural net must have any non-linear function at hidden layers.