**Activation Functions**

Brief insight into **Neural** **Network:**

|  |  |
| --- | --- |
| **Single-Layer Perceptron:**   1. **Inputs** are fed into the perceptron 2. **Weights** are multiplied to each input 3. **Summation** and then add **bias** 4. **Activation function** is applied. 5. **Output** is either triggered as 1, or not, as 0. Note we use ***y hat***to label output produced by our perceptron model   Note: This is just a very basic model. More complex has evolved as time passed. I have considered the diagram for the sake of simplicity…. as the functionality remains the same. |  |
| **Multi-Layer Perceptron:**  6. Backpropagation, a procedure to repeatedly adjust the weights so as to minimize the difference between actual output and desired output  7. Hidden Layers, which are neuron nodes stacked in between inputs and outputs, allowing neural networks to learn more complicated features (such as XOR logic) |  |

Crisp way of describing the basic process carried out by a neuron in a neural network is:

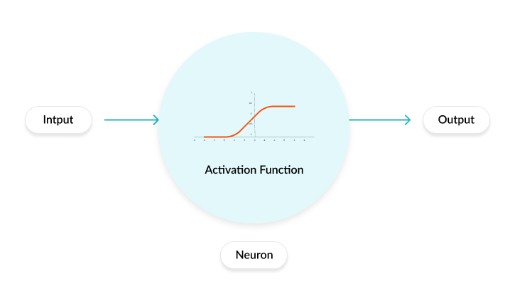


**Activation functions:**

Activation functions determine the output of a deep learning model, its accuracy, and also the computational efficiency of training a model—which can make or break a large scale neural network. Activation functions also have a major effect on the neural network’s ability to converge and the convergence speed, or in some cases, activation functions might prevent neural networks from converging in the first place.

**Role of the Activation Function in a Neural Network Model**

In a neural network, numeric data points, called inputs, are fed into the neurons in the input layer. Each neuron has a weight, and multiplying the input number with the weight gives the output of the neuron, which is transferred to the next layer.

The activation function is a mathematical “gate” in between the input feeding the current neuron and its output going to the next layer. It can be as simple as a step function that turns the neuron output on and off, depending on a rule or threshold. Or it can be a transformation that maps the input signals into output signals that are needed for the neural network to function.

Increasingly, neural networks use non-linear activation functions, which can help the network learn complex data, compute and learn almost any function representing a question, and provide accurate predictions.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Function** | **Definition** | **Equation** | **Range** | **Diagram** | **Advantages** | **Disadvantages** |
| **Sigmoid** | It is a function which is plotted as **‘S’** shaped graph. |  | 0 OR 1 |  | * Smooth gradient, preventing “jumps” in output values. * Usually used in output layer of a binary classification, where result is either 0 or 1. | * Cannot be used for multi class problems (>2) * It is exponential operation, hence heavy in terms of computation. * Vanishing gradient problem exists. * Outputs not zero centred |
| **Tanh** | It is a Hyperbolic Tangent | Or | -1 to 1 |  | * Outputs are zero centred and hence used for normalizing the data | * It is exponential operation, hence heavy in terms of computation. * Vanishing gradient problem exists. |
| **ReLU** | This function returns 0 if it receives any negative input, but for any positive value it returns that value back. |  | 0 to ∞ |  | * Vanishing Gradient problem do not exist * It is less complex and faster computation than Sigmoid and Tanh functions | * **Dying ReLU problem**—when inputs approach zero, or are negative, the gradient of the function becomes zero, the network cannot perform backpropagation and cannot learn. * Outputs not zero centred |
| **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. |  |  |  | * Prevents dying ReLU problem—this variation of ReLU has a small positive slope in the negative area, so it does enable backpropagation, even for negative input values | * Results not consistent—It does not provide consistent predictions for negative input values. |
| **Parametric ReLU** | Instead of alpha value as 0.01, it is a learnable parameter taking any value. | Note:  If alpha value is 0.01 then it is Leaky ReLU  If alpha value is 1 then it is ReLU |  |  | Allows the negative slope to be learned—unlike leaky ReLU, this function provides the slope of the negative part of the function as an argument. It is, therefore, possible to perform backpropagation and learn the most appropriate value of alpha.  Otherwise like ReLU | May perform differently for different problems |
| **Exponential Linear Unit** | X is represented as exponential of x |  |  |  | No Gradient Dipping  Average of the output is close to zero. Hence can be used for normalizing the data | For negative dataset it is computational expensive |
| **Softmax** | It is a function that takes input as real numbers and normalizes it into probability distribution values |  | 0 to 1 |  | Used for handling multiple classes problems (probability values are calculated) |  |
| **Maxout** | It is a generalised version of ReLU and Leaky ReLU | Here, the weights are calculated and then the max of these weights is considered |  |  | It gives benefits of both linear and ReLU functions  Unlike other functions Maxout DONOT depend on predefined situation |  |
| **SWIS ()** | Swish is a new, self-gated activation function discovered by researchers at Google. |  |  |  | it performs better than ReLU with a similar level of computational efficiency.  It is simple and it looks promising  Always we get a smooth curve  NO vanishing gradient |  |
| **Softplus or SmoothReLU** | SoftPlus function’s  graph looks like smoothened ReLU | The derivative of this function is logistic/sigmoid function!! |  |  |  |  |

**LOSS Functions**

|  |  |  |  |
| --- | --- | --- | --- |
| **Loss Function** | **Description** | **Equation** | **Pros and Cons** |
| **L1 Loss function** | L1 Loss function stands for Least Absolute Deviations. Also known as LAD.  L1 Loss Function is used to minimize the error which is the sum of the all the absolute differences between the true value and the predicted value. |  | L1 loss is small.  The first derivative is zero. Hence, It cannot be used when we have to derive the equation. |
| **L2 Loss function** | L2 Loss function stands for Least Square Errors. Also known as LS.  L2 Loss Function is used to minimize the error which is the sum of the all the squared differences between the true value and the predicted value. |  | L2 penalizes the error by squaring it.  It is not good to use it when outliers exist |
| **Huber Loss** | In L2 if outliers are present then the error is going to increase. To overcome this Huber loss was formulated.  A threshold **δ** is set initially. If we get a value greater than or equal to **δ** then it is set as outlier.  The bottom blue line is the Huber loss |  | Is less sensitive to outliers in data than the squared error loss.  It combines good properties from both MSE and MAE. |
| **Pseudo Huber Loss** | Pseudo-huber loss is a variant of the Huber loss function.  It can be used as smooth approximation of Huber Loss  Derivatives are continuous for all degrees | Where 𝛿 is the set parameter, the larger the value, the steeper the linear part on both sides. |  |
| **Hinge Loss** | The hinge loss is used for "maximum-margin" classification, most notably for support vector machines (SVMs).  It penalizes the predictions y<1 similar to SVM | For an intended output *t* = ±1 and a classifier score *y*, the hinge loss of the prediction *y* is defined as | Plot of hinge loss (blue, measured vertically - hypotenuse) vs. zero-one loss (measured vertically; misclassification, green: y < 0) for t = 1 and variable y (measured horizontally). Note that the hinge loss penalizes predictions y < 1, corresponding to the notion of a margin in a support vector machine. |
| **Cross-entropy** | Cross-entropy loss, or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1. Cross-entropy loss increases as the predicted probability diverges from the actual label. So predicting a probability of .012 when the actual observation label is 1 would be bad and result in a high loss value. A perfect model would have a log loss of 0.  It generally calculating the difference between two probability distributions. |  | Cross-entropy can be used as a loss function when optimizing classification models like logistic regression and artificial neural networks. |
| **Sigmoid Cross Entropy** | Replace all the log with Sigmoid function. This loss requires that the predicted value is a probability. Generally, we calculate 𝑠𝑐𝑜𝑟𝑒𝑠=𝑥∗𝑤+𝑏 . Entering this value into the sigmoid function can compress the value range to (0,1). |  |  |
| **Softmax Cross Entropy** | It uses a softmax function to convert the score vector into a probability vector |  |  |