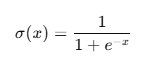
**Deep Learning Assignment (Part A)**

* **Explaining Activation Function**

Activation functions play a crucial role in neural networks by introducing non-linearity, enabling the network to learn complex patterns. Here’s an in-depth look at four popular activation functions: **Sigmoid, ReLU, Tanh and Leaky ReLU.**

1. **Sigmoid Activation Function:**

The sigmoid function is a special form of the logistic function and is usually denoted by sig(x). It is given by:



The graph of sigmoid function is an S-shaped curve as shown by the red line in the graph bellow.



Figure 1: Curve of a sigmoid function

**Use cases of Sigmoid Activation Function:**

* Often used in binary classification tasks.
* Suitable for probabilistic interpretations (output interpreted as probabilities).
* Input normalization by squashing them into a range of 0 to 1, which is useful in some preprocessing steps.
* Used in certain gates of Long Short-Term Memory (LSTM) units to control information flow.

**Limitations of Sigmoid Activation Function:**

* For very large or very small input values, the gradient becomes near zero, causing slow learning or stopping the training.
* Outputs are always positive, leading to inefficient gradient update in some cases.
* The sigmoid function contains the exponential operation, which requires hundreds of addition, subtraction, multiplication, and division instruction.
* This activation function faces saturation problem.

1. **ReLU (Rectified Linear Unit)**

The rectified linear unit (ReLU) or rectifier activation function introduces the property of nonlinearity to a deep learning model and solves the vanishing gradients issue. It interprets the positive part of its argument. It is one of the most popular activation functions in deep learning. ReLU is defined as:

****

It outputs the input directly if it is positive; otherwise, it outputs zero.

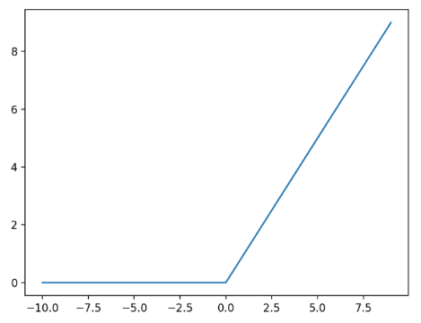


Figure 2: ReLU

The graph shows that the output is zero for negative inputs and linear for positive inputs.

**Use Cases of ReLU Activation Function:**

* Commonly used in hidden layers of deep neural networks.
* Effective for tasks where computational efficiency is critical.
* Used in transformed-based models and feed-forward layers in NLP tasks to capture semantic relationships.
* Help mitigating the vanishing gradient problem during machine learning model training and enabling neural networks to learn more complex relationships in data.

**Limitation of ReLU Activation function:**

* Neurons can become inactive (outputting zero) for all inputs if wrights are updated poorly, effectively “dying”.
* As it makes all the negative values become zero, which decreases the ability of the model to fit or train from the data properly.
* Unlike other activation functions like sigmoid or tanh, the ReLU activation is unbounded on the positive side, which can sometimes results in exploding gradients when training deep networks.
* The gradient of ReLU can be unstable during training, especially when weights are not properly initialized. In some cases, this can slow down learning or lead to poor performance.

1. **Tanh (Hyperbolic Tangent) Activation Function**

Tanh activation function is a mathematical function commonly used in ANN for their hidden layers. It transforms input values to produce output values between -1 and 1. It is expressed as the ratio of the difference between the exponential of the input value and the exponential of its negation to the sum of these exponential.



Like Sigmoid, it has an S-shaped curve but is zero-centered.

**Use Cases of Tanh Activation Function:**

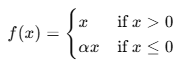
* Applied in hidden layers to help the network learn complex patterns while maintaining normalized activations.
* Used in LSTM cells to control input and output gates effectively.
* Often used in the bottleneck layer of auto encoders or in generating smooth outputs in GANs.
* Tasks like sentiment analysis or fraud detection, where an output represents opposing classes.
* Predicting bounded values like normalized scores or probabilities.

**Limitations of Tanh Activation function:**

* Suffers from the vanishing gradient problem, slowing down learning in deep networks.
* It is computationally expensive due to exponential calculations.
* Tanh is non-sparse, meaning all neurons stay active, which can hinder efficiency.
* Also sensitive to outliers, producing extreme outputs for large inputs.
* Its output range is limited to [-1,1], reducing flexibility in some tasks.

1. **Leaky ReLU (Rectified Linear Unit)**

Leaky ReLU is a variant of the ReLU activation function designed to address the dying ReLU problem, where neurons output zero for all inputs and stop learning. Unline ReLU, which outputs zero for negative inputs, Leaky ReLU allows a small, non-zero gradient for negative inputs.



Where *α* is a small constant that controls the slope for negative inputs.

**Use cases of Leaky ReLU:**

* Used in deep architectures to prevent dying neurons and improve gradient flow.
* Enhances feature extraction by maintain active neurons for all inputs.
* Helps stabilize training by preventing vanishing gradients.
* Improves reconstruction quality by allowing learning from negative inputs.

**Limitations of Leaky ReLU:**

* The small negative slope can lead to learning from noisy or irrelevant data when the slope is not properly tuned.
* The slope parameter must be chosen carefully. A wrong choice can negatively impact model performance or cause slower convergence.
* If the slope is too high, it can introduce unnecessary noise, leading to overfitting during training.
* Like ReLU, the output is not centered around zero, which can cause bias in gradients during training.
* **Discussion of Optimization Algorithm:**

1. **Comparison among SGD, Adam, and RMSprop:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Aspect** | **SGD** | **Adam** | **RMSProp** |
| **Basic Idea** | Updates weights using the gradient of the loss function with respect to parameters. | Combines momentum (moving average of gradients) and adaptive learning rates. | Uses adaptive learning rates by dividing the learning rate by a moving average of squared gradients. |
| **Updated Formula** |  |  |  |
| **Learning Rate** | Fixed or manually decayed over time. | Adaptive; adjusts for each parameter. | Adaptive; adjusts based on recent gradient history. |
| **Momentum** | Optional (classical SGD doesn’t have it ) | Uses both momentum (1st moment) and (2nd moment) | Uses momentum-like effect by averaging squared gradients. |
| **Convergence speed** | Slower, especially with noisy gradients. | Faster convergence in many cases due to adaptive steps. | Faster due to adaptive step sizes. |
| **Memory Usage** | Low (only stores current gradients) | High (stores moving averages of gradients and their squares). | Moderate (stores moving average of squared gradients). |
| **Pros** | -Simple and easy to implement.  - Works well with large batches. | -Adaptive learning rate.  - Works well with sparse gradients. | -Handle non-stationary objectives well.  -Reduces oscillations |
| **Variants** | SGD with momentum, Nesterov accelerated gradient. | AdamW (weight decay), AMSGrad. | RMSProp with momentum. |
|  |  |
|  |  |

1. **Explain how learning rate impacts model training and how it is addressed in modern optimizers.**

The learning rate is a critical hyperparameter in machine learning that controls the size of the steps taken in the parameter space during optimization. It significantly affects the convergence and performance of the model:

|  |  |
| --- | --- |
| **Learning Rate Value** | **Effect** |
| Too High | * Causes the model to overshoot the optimal solution. * May results in divergence or oscillation around the minimum. * Leads to unstable training. |
| Too Low | * Causes very slow convergence, increasing training time. * May get stuck in local minima or saddle points. * Reduces the effectiveness of training. |
| Optimal | * Enables steady convergence to a good solution. * Balances exploration and exploitation. |