# **ACTIVATION LAYER**

In artificial neural networks, an activation function is one that outputs a smaller value for tiny inputs and a higher value if its inputs are greater than a threshold. An activation function "fires" if the inputs are big enough; otherwise, nothing happens. An activation function, then, is a gate that verifies how an incoming value is higher than a threshold value.

Because they introduce non-linearities in neural networks and enable the neural networks can learn powerful operations, activation functions are helpful. A feedforward neural network might be refactored into a straightforward linear function or matrix transformation on to its input if indeed the activation functions were taken out.

Advertisement

By generating a weighted total and then including bias with it, the activation function determines whether a neuron should be turned on. The activation function seeks to boost a neuron's output's nonlinearity.

**Explanation**: As we are aware, neurons in neural networks operate in accordance with weight, bias, and their corresponding activation functions. Based on the mistake, the values of the neurons inside a neural network would be modified. This process is known as back-propagation. Back-propagation is made possible by activation functions since they provide the gradients and error required to change the biases and weights.

Need of Non-linear Activation Functions

An interconnected regression model without an activation function is all that a neural network is. Input is transformed nonlinearly by the activation function, allowing the system to learn and perform more challenging tasks.

It is merely a thing procedure that is used to obtain a node's output. It also goes by the name Transfer Function.

The mixture of two linear functions yields a linear function, so no matter how several hidden layers we add to a neural network, they all will behave in the same way. The neuron cannot learn if all it has is a linear model. It will be able to learn based on the difference with respect to error with a non-linear activation function.

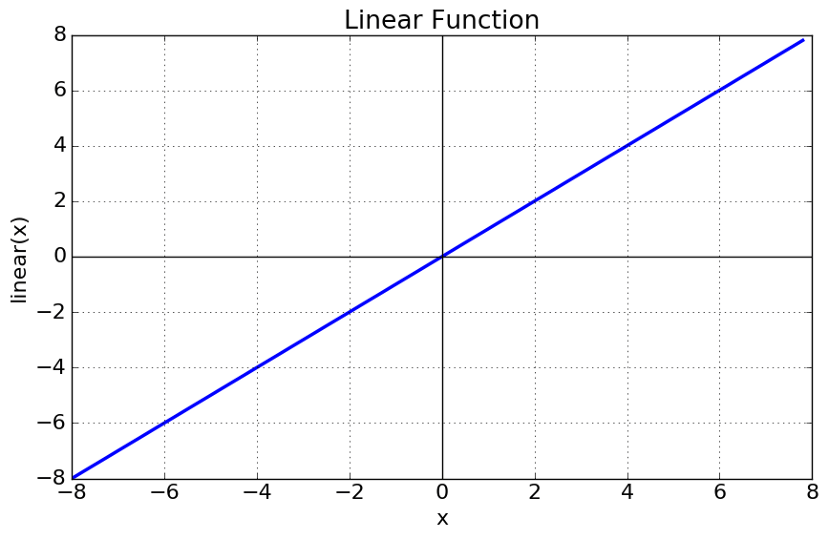
The mixture of two linear functions yields a linear function in itself, so no matter how several hidden layers we add to a neural network, they all will behave in the same way. The neuron cannot learn if all it has is a linear model.

The two main categories of activation functions are:

* Linear Activation Function
* Non-linear Activation Functions

Linear Activation Function

As can be observed, the functional is linear or linear. Therefore, no region will be employed to restrict the functions' output.



The normal data input to neural networks is unaffected by the complexity or other factors.

Non-linear Activation Function

The normal data input to neural networks is unaffected by the complexity or other factors.

Activation Function

* **Linear Function**

Equation: A linear function's equation, which is y = x, is similar to the eqn of a single direction.

The ultimate activation function of the last layer is nothing more than a linear function of input from the first layer, regardless of how many levels we have if they are all linear in nature. -inf to +inf is the range.

Uses: The output layer is the only location where the activation function's function is applied.

If we separate a linear function to add non-linearity, the outcome will no longer depend on the input "x," the function will become fixed, and our algorithm won't exhibit any novel behaviour.

A good example of a regression problem is determining the cost of a house. We can use linear activation at the output layer since the price of a house may have any huge or little value. The neural network's hidden layers must perform some sort of non-linear function even in this circumstance.

* **Sigmoid Function**

It is a functional that is graphed in a "S" shape.

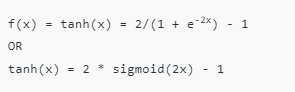
A is equal to 1/(1 + e-x).

Non-linear in nature. Observe that while Y values are fairly steep, X values range from -2 to 2. To put it another way, small changes in x also would cause significant shifts in the value of Y. spans from 0 to 1.

Uses: Sigmoid function is typically employed in the output nodes of a classification, where the result may only be either 0 or 1. Since the value for the sigmoid function only ranges from 0 to 1, the result can be easily anticipated to be 1 if the value is more than 0.5 and 0 if it is not.

* **Tanh Function**

The activation that consistently outperforms sigmoid function is known as tangent hyperbolic function. It's actually a sigmoid function that has been mathematically adjusted. Both are comparable to and derivable from one another.



Range of values: -1 to +1. non-linear nature

Uses: - Since its values typically range from -1 to 1, the mean again for hidden layer of a neural network will be 0 or very near to it. This helps to centre the data by getting the mean close to 0. This greatly facilitates learning for the following layer.

**Equation:**

max A(x) (0, x). If x is positive, it outputs x; if not, it outputs 0.

Value Interval: [0, inf]

Nature: non-linear, which allows us to simply backpropagate the mistakes and have the ReLU function activate many layers of neurons.

Uses: Because ReLU includes simpler mathematical processes than tanh and sigmoid, it requires less computer time to run. The system is sparse and efficient for computation since only a limited number of neurons are activated at any given time.

Simply said, RELU picks up information considerably more quickly than sigmoid and Tanh functions.

* **ReLU (Rectified Linear Unit) Activation Function**

Currently, the ReLU is the activation function that is employed the most globally. Since practically all convolutional neural networks and deep learning systems employ it.

The derivative and the function are both monotonic.

However, the problem is that all negative values instantly become zero, which reduces the model's capacity to effectively fit or learn from the data. This means that any negative input to a ReLU activation function immediately becomes zero in the graph, which has an impact on the final graph by improperly mapping the negative values.

* **Softmax Function**

Although it is a subclass of the sigmoid function, the softmax function comes in handy when dealing with multiclass classification issues.

Used frequently when managing several classes. In the output nodes of image classification issues, the softmax was typically present. The softmax function would split by the sum of the outputs and squeeze all outputs for each category between 0 and 1.

The output unit of the classifier, where we are actually attempting to obtain the probabilities to determine the class of each input, is where the softmax function is best applied.

The usual rule of thumb is to utilise RELU, which is a usual perceptron in hidden layers and is employed in the majority of cases these days, if we really are unsure of what encoder to apply.

A very logical choice for the output layer is the sigmoid function if your input is for binary classification. If our output involves multiple classes, Softmax can be quite helpful in predicting the odds for each class.

# TYPES OF LAYERS

**The Basic Layers in ANN**

**1. Input Layer**

The input layer is the first layer in an ANN and is responsible for receiving the raw input data. This layer's neurons correspond to the features in the input data. For example, in image processing, each neuron might represent a pixel value. The input layer doesn't perform any computations but passes the data to the next layer.

**2. Hidden Layers**

The hidden Layers are the intermediate layers between the input and output layers. They perform most of the computations required by the network. Hidden layers can vary in number and size, depending on the complexity of the task.

Each hidden layer applies a set of weights and biases to the input data, followed by an activation function to introduce non-linearity.

**3. Output Layer**

The Output Layer is the final layer in an ANN. It produces the output predictions. The number of neurons in this layer corresponds to the number of classes in a classification problem or the number of outputs in a regression problem. The activation function used in the output layer depends on the type of problem:

* **Softmax** for multi-class classification
* **Sigmoid** for binary classification
* **Linear** for regression

## Types of Hidden Layer :

**1. Dense (Fully Connected) Layers**

**Overview:**

Dense layers are the fundamental building blocks of neural networks. In these layers, every neuron is connected to every neuron in the preceding layer, allowing the network to learn complex relationships within the data.

**Key Characteristics:**

* **Complete Connectivity:** Each neuron receives input from all neurons of the previous layer, enabling comprehensive feature interactions.
* **Versatility:** Suitable for a wide range of problems, including classification and regression tasks.

**Applications:**

* **Image Classification (Final Layers):** After feature extraction, dense layers make the final predictions.
* **Tabular Data Processing:** Effective when dealing with structured data.

**2. Convolutional Layers**

**Overview:**

Convolutional layers are specialized for processing grid-like data structures, such as images. They apply convolution operations to extract local features.

**Key Characteristics:**

* **Local Connectivity:** Neurons connect to a small region of the input (receptive field), capturing spatial hierarchies.
* **Shared Weights (Filters/Kernels):** The same filter is applied across the entire input, detecting features regardless of their position.

**Applications:**

* **Image Recognition:** Identifying objects within images.
* **Medical Imaging:** Detecting anomalies in scans.
* **Self-Driving Cars:** Interpreting visual data from cameras.

**3. Recurrent Layers**

**Overview:**

Recurrent layers are adept at handling sequential data. They maintain a hidden state that captures information about previous inputs, making them ideal for time-dependent data.

**Key Characteristics:**

* **Temporal Dynamics:** Capture dependencies over time.
* **Memory Retention:** Can learn long-term dependencies (especially LSTMs and GRUs).

**Types of Recurrent Layers:**

* **Simple RNNs:** Basic form with short-term memory.
* **Long Short-Term Memory (LSTM):** Addresses the vanishing gradient problem, retaining information over longer periods.
* **Gated Recurrent Units (GRU):** Simplified LSTMs with fewer parameters.

**Applications:**

* **Language Translation:** Capturing context in sequences.
* **Speech Recognition:** Interpreting spoken words over time.
* **Stock Market Prediction:** Analyzing temporal trends.

**4. Pooling Layers**

**Overview:**

Pooling layers reduce the dimensions of feature maps, retaining essential information while minimizing computational load.

**Key Characteristics:**

* **Dimensionality Reduction:** Simplifies the output, making computations more efficient.
* **Translation Invariance:** Helps recognize objects regardless of their position in the input.

**Types of Pooling:**

* **Max Pooling:** Takes the maximum value within a pooling window.
* **Average Pooling:** Calculates the average within the window.
* **Global Pooling:** Reduces each feature map to a single value.

**Applications:**

* Used extensively in CNN architectures to progressively reduce spatial dimensions.

**5. Dropout Layers**

**Overview:**

Dropout layers improve network generalization by randomly dropping neurons during training.

**Key Characteristics:**

* **Regularization Technique:** Prevents over-reliance on specific neurons.
* **Promotes Redundancy:** Encourages the network to learn robust features.

**Applications:**

* Commonly used in dense layers of deep networks to mitigate overfitting.

**6. Normalization Layers**

**Overview:**

Normalization layers standardize inputs to stabilize learning.

**Key Characteristics:**

* **Reduces Internal Covariate Shift:** Makes training faster and more stable.
* **Enables Higher Learning Rates:** Facilitates quicker convergence.

**Types of Normalization:**

* **Batch Normalization:**
  + Normalizes inputs across the batch dimension.
  + Formula:
* **Layer Normalization:**
  + Normalizes inputs across the feature dimension for each sample.
* **Instance Normalization:**
  + Normalizes each sample independently, often used in style transfer.

**Applications:**

* Used in deep networks like ResNets to improve performance.

**7. Embedding Layers**

**Overview:**

Embedding layers map high-dimensional categorical variables into lower-dimensional continuous vectors.

**Key Characteristics:**

* **Captures Semantic Meaning:** Similar categories have similar embeddings.
* **Facilitates Machine Understanding:** Transforms discrete data into a form suitable for neural networks.

**Applications:**

* **Natural Language Processing:** Word embeddings like Word2Vec and GloVe.
* **Recommender Systems:** Representing users and items in a shared vector space.

**8. Advanced Layers and Concepts**

**a. Residual Layers (Residual Connections):**

* **Overview:** Allow gradients to bypass certain layers, enabling the training of very deep networks.
* **Key Benefit:** Mitigates the vanishing gradient problem.
* **Application:** Used in architectures like ResNet.

**b. Attention Mechanisms:**

* **Overview:** Allow the model to focus on specific parts of the input when generating each part of the output.
* **Key Benefit:** Improves performance on tasks requiring understanding of dependencies.
* **Application:** Central to Transformer models used in NLP.

**c. Capsule Networks:**

* **Overview:** Preserve hierarchical relationships between features.
* **Key Benefit:** More robust to spatial transformations.
* **Application:** Experimental, with potential in image recognition.

# TYPES OF OPTIMIZERS

**1. Stochastic Gradient Descent (SGD)**

**Overview:**

* **Fundamental Concept:** Updates the weights incrementally using a single sample or a small batch.
* **Mechanics:** At each step, weights are adjusted based on the gradient of the loss function with respect to the weights.

**Mathematical Update Rule:**

θt+1=θt−η⋅∇θL(θt)

* θ\theta: Model parameters (weights)
* η\eta: Learning rate
* LL: Loss function
* ∇θL(θt): Gradient at time
* **Characteristics:**
* **Simplicity:** The go-to optimizer for many applications.
* **Challenges:** Can be slow to converge and may get stuck in local minima.

**2. Momentum**

**Overview:**

* **Concept:** Accelerates SGD by adding a fraction of the previous update to the current update.
* **Analogy:** Like a heavy ball rolling down a slope, accumulating speed (momentum) over time.

**Benefits:**

* **Smoothest Updates:** Reduces oscillations in ravines of the error surface.
* **Faster Convergence:** Particularly useful in scenarios with high curvature.

**3. Nesterov Accelerated Gradient (NAG)**

**Overview:**

* **Concept:** An improvement over Momentum that anticipates the future position of the parameters.
* **Mechanics:** Calculates the gradient not at the current parameters but at the projected future parameters.

**Advantages:**

* **Lookahead Approach:** Provides more informed updates.
* **Improved Convergence:** Often faster than classic Momentum.

**4. Adagrad (Adaptive Gradient Algorithm)**

**Overview:**

* **Concept:** Adapts the learning rate for each parameter individually based on historical gradients.
* **Mechanics:** Parameters with larger gradients receive smaller updates, and vice versa.

**Benefits:**

* **Automatic Learning Rate Adjustment:** No need to manually tune the learning rate.
* **Good for Sparse Data:** Effective in NLP and computer vision tasks.

**Limitations:**

* **Learning Rate Decay:** The accumulated gradient can lead to excessively small learning rates over time.

**5. Adadelta**

**Overview:**

* **Concept:** Addresses Adagrad's diminishing learning rate problem by limiting the accumulation of past gradients.
* **Mechanics:** Uses a moving window of gradients instead of all past gradients.

**Advantages:**

* **No Manual Learning Rate:** Eliminates the need for an initial learning rate.
* **Adaptive Updates:** Adjusts updates based on past gradient magnitudes.

**6. RMSprop (Root Mean Square Propagation)**

**Overview:**

* **Concept:** Similar to Adadelta; designed to maintain a moving average of the squared gradients to normalize the gradient.
* **Developed By:** Geoffrey Hinton.

**Benefits:**

* **Effective for Non-Stationary Objectives:** Performs well with changing loss landscapes.
* **Widely Used in RNNs:** Handles the exploding gradient problem.

**7. Adam (Adaptive Moment Estimation)**

**Overview:**

* **Concept:** Combines ideas from Momentum and RMSprop.
* **Mechanics:** Keeps an exponentially decaying average of past gradients (first moment) and squared gradients (second moment).

**Advantages:**

* **Fast Convergence:** Efficient and reliable for large-scale problems.
* **Adaptive Learning Rates:** Adjusts learning rates individually for each parameter.

**Applications:**

* **Computer Vision**
* **Natural Language Processing**
* **Reinforcement Learning**

# TYPES OF LOSS

**1. Mean Squared Error (MSE)**

**Overview:**

* **Definition:** Measures the average squared difference between the predicted values and the actual values.

**Characteristics:**

* **Penalizes Larger Errors:** Squaring the errors emphasizes larger discrepancies.
* **Smooth Gradient:** Facilitates optimization using gradient descent algorithms.

**Applications:**

* **Regression Tasks:** Predicting continuous outcomes like house prices, temperatures, or stock prices.

**2. Mean Absolute Error (MAE)**

**Overview:**

* **Definition:** Computes the average of the absolute differences between predicted and actual values.

**Characteristics:**

* **Robust to Outliers:** Less sensitive to large errors compared to MSE.
* **Non-Differentiable at Zero:** The absolute function isn't differentiable at zero, which can be handled in practice.

**Applications:**

* **Regression Scenarios:** Where outliers are present and we want a more balanced error measure.

**3. Root Mean Squared Error (RMSE)**

**Overview:**

* **Definition:** The square root of MSE; provides error in the same units as the target variable.

**Characteristics:**

* **Interpretability:** Easier to interpret since it's on the same scale as the data.
* **Emphasizes Large Errors:** Like MSE, it penalizes larger errors more heavily.

**Applications:**

* **Regression Analysis:** Situations where understanding the error magnitude is crucial.

**4. Cross-Entropy Loss**

**Overview:**

* **Definition:** Measures the difference between two probability distributions—the true distribution and the predicted distribution.

**Types:**

* **Binary Cross-Entropy (Log Loss):**
* **Categorical Cross-Entropy:**

**Characteristics:**

* **Probabilistic Interpretation:** Ideal when outputs represent probabilities.
* **Sensitive to Confidence:** Penalizes incorrect confident predictions more severely.

**Applications:**

* **Classification Problems:** Image recognition, language processing, and any task requiring probabilistic outputs.

**5. Hinge Loss**

**Overview:**

* **Definition:** Used for maximum-margin classification, notably for Support Vector Machines (SVMs).

**Characteristics:**

* **Margin Maximization:** Encourages a decision boundary with the largest possible margin.
* **Zero Loss for Correctly Classified Samples Beyond Margin:** Only misclassified or borderline samples contribute to the loss.

**Applications:**

* **Binary Classification:** Especially when using linear classifiers that benefit from margin maximization.

**6. Kullback-Leibler Divergence Loss (KL Divergence)**

**Overview:**

* **Definition:** Measures how one probability distribution diverges from a second, expected probability distribution.

**Characteristics:**

* **Asymmetrical Measure:** Direction matters
* **Non-Negative:** Always yields a value greater than or equal to zero.

**Applications:**

* **Modelling Probability Distributions:** Used in variational autoencoders and other models where matching distributions is key.

**7. Huber Loss**

**Overview:**

* **Definition:** Combines MSE and MAE to be robust to outliers while maintaining smoothness.

**Characteristics:**

* **Smooth Transition:** Quadratic for small errors, linear for large errors.
* **Outlier Handling:** Reduces the influence of outliers compared to MSE.

**Applications:**

* **Regression with Outliers:** Ideal when data contains anomalies but a smooth loss is desired for optimization.

**8. Log-Cosh Loss**

**Overview:**

* **Definition:** The logarithm of the hyperbolic cosine of the prediction error.

**Characteristics:**

* **Combines MSE and MAE Advantages:** Approximates MSE for small losses and MAE for large losses.
* **Differentiable Everywhere:** Facilitates gradient-based optimization.

**Applications:**

* **Regression Tasks:** Where a balance between penalizing outliers and small errors is required.

**9. Poisson Loss**

**Overview:**

* **Definition:** Suitable for modelling count data that follows a Poisson distribution.

**Characteristics:**

* **Non-Negative Predictions:** Enforces the model to predict positive values.
* **Assumes Mean Equals Variance:** Reflecting the properties of Poisson-distributed data.

**Applications:**

* **modelling Counts:** Events per interval, like customer arrivals or failure occurrences.

**10. Quantile Loss**

**Overview:**

* **Definition:** Used to predict a particular quantile (e.g., median) and is asymmetric in penalizing overestimates and underestimates.

**Characteristics:**

* **Customizable Error Weights:** Adjusts the penalty for over-prediction and under-prediction.
* **Robust Estimation:** Useful when the distribution of errors is not symmetric.

**Applications:**

* **Forecasting:** Providing prediction intervals or risk assessments.

**11. Focal Loss**

**Overview:**

* **Definition:** Modifies cross-entropy loss to address class imbalance by focusing on hard-to-classify examples.

**Characteristics:**

* **Down-Weights Easy Examples:** Reduces loss contribution from well-classified samples.
* **Focuses on Hard Examples:** Encourages the model to learn from challenging cases.

**Applications:**

* **Object Detection:** Particularly in scenarios with a large class imbalance, like detecting rare objects.

**12. Dice Loss**

**Overview:**

* **Definition:** Based on the Dice coefficient, measuring overlap between predicted and true segments.

**Characteristics:**

* **Ideal for Segmentation Tasks:** Maximizes the overlap between predicted and actual segments.
* **Handles Class Imbalance:** Effective when foreground and background pixel proportions are unequal.

**Applications:**

* **Biomedical Imaging:** Segmenting tumours or organs where precise overlap is critical.

**13. Triplet Loss**

**Overview:**

* **Definition:** Encourages the model to bring similar items closer and push dissimilar items apart in embedding space.

**Characteristics:**

* **Relative Comparison:** Focuses on the relative distances between triplets rather than absolute positions.
* **Margin Enforcement:** Ensures a minimum separation between positive and negative pairs.

**Applications:**

* **Face Recognition:** Learning embeddings where images of the same person are close together.

**14. Wasserstein Loss**

**Overview:**

* **Definition:** Based on the Wasserstein distance (Earth Mover's Distance), measuring the minimum effort to transform one distribution into another.
* **Characteristics:**
  + **Stable GAN Training:** Used in Wasserstein GANs to improve training stability.
  + **Smooth Loss Landscape:** Provides meaningful gradients even when generator and discriminator distributions don't overlap.

**Applications:**

* **Generative Models:** Enhancing the quality and stability of GANs.

**15. Custom Loss Functions**

**Overview:**

* **Definition:** Tailor-made loss functions designed to address specific needs of a problem.
* **Characteristics:**
  + **Flexibility:** Can incorporate domain-specific knowledge or constraints.
  + **Optimization of Custom Metrics:** Directly optimize for metrics like F1 score, IoU, or others.

**Applications:**

* **Specialized Tasks:** Any scenario where standard loss functions don't capture the essence of the desired outcome.