Artificial Neural Networks

1. Activation Functions in ANNs

Activation functions add the essential non-linearity to a neural network, allowing it to learn complex mappings between inputs and outputs. Without them, no matter how many layers the network has, the whole network would behave like a single linear transformation.

Common Activation Functions

- **Sigmoid (Logistic) Function**
- Maps real-valued inputs into the interval (0, 1), making it useful for binary classification at the output layer. However, its gradient may vanish for large absolute values, slowing training.
- *Example Formula:*
- $\sigma(x)=11+e-x|sigma(x) = |frac{1}{1 + e^{-x}}$
- **Tanh Function**

Similar in shape to the sigmoid but outputs values in the range (–1, 1), which makes the activations zero-centered. This sometimes leads to better performance in hidden layers.

Example Formula:

$tanh(x)=ex−e−xex+e−x|tanh(x) = |frac{e^x - e^{-x}}{e^x + e^{-x}}$

- **ReLU (Rectified Linear Unit)**

Has become the default for many hidden layers because it is computationally efficient and alleviates the vanishing gradient problem by keeping a constant gradient for positive inputs.

Example Formula:

$ReLU(x)=max(0,x)|text{ReLU}(x) = |max(0, x)$

- **Leaky ReLU and Parametric ReLU (PReLU)**

Variants of ReLU that allow a small, non-zero gradient for negative inputs, addressing the "dying ReLU" problem where neurons may become inactive.

Example Formula for Leaky ReLU:

$f(x)=\{xif\ x\geq 0\alpha xif\ x<0f(x) = \backslash begin\{cases\}\ x\ \&\ \backslash text\{if\ \}\ x\ \backslash ge\ 0\ \backslash\backslash\ \backslash alpha\ x\ \&\ \backslash text\{if\ \}\ x <\ 0\ \backslash end\{cases\}$ (with $\alpha \backslash alpha$ typically set around 0.01)

- **Softmax Function**

Typically used in the output layer for multiclass classification problems. It converts raw output scores into a probability distribution over classes such that the outputs sum to 1.

Example Formula:

$softmax(zi)=ezi\sum jezj \backslash text\{softmax\}(z\_i) = \backslash frac\{e\^{}\{z\_i\}\}\{\backslash sum\_j\ e\^{}\{z\_j\}\}$

2. Types of Layers in ANNs

A typical ANN is organized into layers, each serving a specific role in the flow of information:

- **Input Layer**

This is the first layer and is responsible for receiving the raw input data. Its neurons do not perform any computation other than passing the input along to subsequent layers.

- **Hidden Layers**

One or more layers that lie between the input and output layers. These layers perform the bulk of the computation and are responsible for extracting features and learning representations of the input data. In modern architectures, you might encounter:

    - **Fully Connected (Dense) Layers:** Every neuron in one layer connects to every neuron in the next layer.
    - **Convolutional Layers:** Used especially in image processing, they apply filters to local patches of the input to create feature maps.
    - **Pooling Layers:** Often follow convolutional layers to reduce spatial dimensions.

    - **Recurrent Layers:** Used in sequence processing (e.g., LSTM, GRU) where connections can span across time steps.
- **Output Layer**

The final layer that produces the network's prediction. Its activation function is typically chosen based on the type of problem:

    - For regression: often a linear activation.
    - For binary classification: typically a sigmoid.
    - For multiclass classification: often a softmax.

3. Types of Optimizers

During training, the goal is to minimize a loss function by adjusting the weights of the network. Optimizer algorithms use gradient-based methods to update weights. Common optimizers include:

- **Stochastic Gradient Descent (SGD):**

The basic optimizer that updates weights using the gradient computed from one (or a mini-batch) of training samples.

Update Rule:

$w \leftarrow w - \eta \nabla_w L$ |*leftarrow w - |eta |nabla_w L*

- **Momentum:**

This method builds up velocity by incorporating past gradients to help accelerate updates in consistent directions and dampen oscillations.

Key Idea:

$v_t = \gamma v_{t-1} + \eta \nabla_w L$ *v_t = |gamma v_{t-1} + |eta |nabla_w L*, then update $w \leftarrow w - v_t$ |*leftarrow w - v_t*

- **Nesterov Accelerated Gradient (NAG):**

An improved version of momentum that computes the gradient at a "look-ahead" position.

- **Adagrad:**

Adapts the learning rate for each parameter individually based on historical gradients. This is particularly useful for sparse data.

- **RMSProp:**

Modifies Adagrad by using a decaying average of past gradients to avoid the learning rate diminishing too quickly.

- **Adam (Adaptive Moment Estimation):**

Combines the benefits of RMSProp and momentum by keeping an exponentially decaying average of past gradients and squared gradients. Adam is one of the most popular optimizers for deep learning tasks.

4. Types of Loss Functions

The loss function (or cost function) measures how far the network's predictions are from the true values. The choice of loss function depends on the problem type:

For Regression Tasks

- **Mean Squared Error (MSE):**

Measures the average squared difference between the predicted and actual values.

Formula:

$$MSE=1N∑i=1N(yi−y^i)2|text{MSE} = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

- **Mean Absolute Error (MAE):**

Measures the average absolute difference between predictions and true values.

For Classification Tasks

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

Used for binary classification, it quantifies the difference between two probability distributions (the predicted probabilities and the actual binary outcomes).

Formula:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i) + (1-y_i) \log (1-\hat{y}_i) \right]$$

- **Categorical Cross-Entropy:**

Used for multiclass classification with one-hot encoded labels. It measures the difference between the predicted probability distribution and the true distribution.

- **Hinge Loss:**

Commonly used for “maximum-margin” classification such as in support vector machines (SVMs), it penalizes misclassifications.