Gradient Descent and Activation Functions Notes by Soham

Gradient Descent and Activation Functions: Detailed Notes with Examples

---

1. Gradient Descent

1.1. Introduction

Gradient Descent is an optimization algorithm used to minimize a loss function by iteratively adjusting the model parameters.

1.2. Mathematical Formulation

1. **Objective:** Minimize a loss function $L(\theta)$, where θ represents the model parameters.
2. **Gradient Computation:** Compute the gradient of the loss function with respect to each parameter:

$$\nabla_\theta L(\theta)$$

3. **Update Rule:** Update the parameters using the gradient and a learning rate η:

$$\theta_{\text{new}} = \theta_{\text{old}} - \eta \cdot \nabla_\theta L(\theta)$$

4. **Convergence:** Repeat the update process until changes in the loss function are below a predefined threshold.

1.3. Variants of Gradient Descent

1. **Batch Gradient Descent:**
   - Uses the entire dataset to compute the gradient.
   - Accurate gradient estimation.
   - Computationally expensive for large datasets.
2. **Stochastic Gradient Descent (SGD):**
   - Uses a single data point to compute the gradient and update parameters.
   - Faster updates, can escape local minima.
   - Noisy updates, convergence can be slow.
3. **Mini-Batch Gradient Descent:**
   - Uses a small batch of data to compute the gradient and update parameters.
   - Balances accuracy and computational efficiency.
   - Requires tuning of batch size.

1.4. Practical Considerations

1. **Learning Rate (η):**
   - Determines the step size for each update.
   - Too Large: May overshoot the minimum.
   - Too Small: Convergence can be slow.
2. **Momentum:**
   - Helps accelerate gradient vectors in the right direction.
   - Update Rule:

$$v_t = \beta v_{t-1} + (1 - \beta)\nabla_\theta L(\theta)$$

$$\theta_{\text{new}} = \theta_{\text{old}} - \eta \cdot v_t$$

   - β: Momentum term, typically set between 0.5 and 0.9.

3. **Adaptive Learning Rates:**
    - Algorithms like Adam, RMSprop, and AdaGrad adjust learning rates based on past gradients.

1.5. Example

Consider a quadratic function:

$$f(x) = x^2 - 4x + 4$$

1. **Compute Gradient:**
    - Differentiate $f(x)$ with respect to x:

$$\frac{d}{dx}(x^2 - 4x + 4) = 2x - 4$$

2. **Update Rule:**
    - Use the gradient to update x:

$$x_{\text{new}} = x_{\text{old}} - \eta \cdot (2x - 4)$$

3. **Iteration Steps:**
    - **Initialization:** $x_0 = 0, \eta = 0.1$
    - **Iteration 1:**

$$\text{Gradient} = 2 \cdot 0 - 4 = -4$$

$$x_1 = 0 - 0.1 \cdot (-4) = 0 + 0.4 = 0.4$$

    - **Iteration 2:**

$$\text{Gradient} = 2 \cdot 0.4 - 4 = -3.2$$

$$x_2 = 0.4 - 0.1 \cdot (-3.2) = 0.4 + 0.32 = 0.72$$

    - **Iteration 3:**

$$\text{Gradient} = 2 \cdot 0.72 - 4 = -2.56$$

$$x_3 = 0.72 - 0.1 \cdot (-2.56) = 0.72 + 0.256 = 0.976$$

    - Continue iterating until x converges to the minimum.

---

2. Activation Functions

2.1. Introduction

Activation functions introduce non-linearity into neural networks, allowing them to model complex patterns.

2.2. Common Activation Functions

1. **Sigmoid Function:**
    - **Formula:**

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

    - **Characteristics:**
        - Output range: (0, 1)
        - Used in binary classification.
    - **Drawbacks:**
        - Vanishing gradient problem for large $|x|$.
    - **Example:** In binary classification tasks, e.g., spam detection.

2. **Hyperbolic Tangent (tanh):**
   - **Formula:**

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

   - **Characteristics:**
     - Output range: (-1, 1)
     - Zero-centered.
   - **Drawbacks:**
     - Still suffers from vanishing gradient problem.
   - **Example:** Used in recurrent neural networks (RNNs) to ensure output values are centered around zero.
3. **Rectified Linear Unit (ReLU):**
   - **Formula:**

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

   - **Characteristics:**
     - Output range: [0, ∞)
     - Simple and efficient.
   - **Drawbacks:**
     - Dying ReLU problem (neurons can become inactive).
   - **Example:** Common in hidden layers of convolutional neural networks (CNNs).
4. **Leaky ReLU:**
   - **Formula:**

$$\text{Leaky ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases}$$

   - **Characteristics:**
     - Allows a small gradient when $x < 0$.
   - **Drawbacks:**
     - Introduces an additional hyperparameter α.
   - **Example:** Used to avoid the dying ReLU problem.
5. **Softmax:**
   - **Formula:**

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

   - **Characteristics:**
     - Converts logits into probabilities.
   - **Drawbacks:**
     - Can be computationally expensive for large output dimensions.
   - **Example:** Used in multi-class classification problems.

2.3. Practical Considerations

1. **Choosing Activation Functions:**
   - **Binary Classification:** Use Sigmoid.
   - **Multi-Class Classification:** Use Softmax.
   - **Hidden Layers:** Use ReLU or Leaky ReLU to introduce non-linearity and avoid vanishing gradients.
2. **Impact on Training:**
   - **Convergence Speed:** Different activation functions affect how quickly the network converges.
   - **Learning Capacity:** Non-linear functions allow the network to capture more complex patterns.