UNIT 3

BACK PROPOGATION

Backpropagation is a fundamental algorithm for training artificial neural networks. It efficiently computes the gradient of the loss function with respect to the network's weights, enabling the network to learn and improve its performance.

1. Neural Network Basics

- **Structure:** A neural network comprises interconnected nodes (neurons) organized in layers: input, hidden, and output. Connections between neurons have associated weights.
- **Forward Pass:** Input data flows through the network, layer by layer. At each neuron, the weighted sum of inputs is calculated, and an activation function is applied. This process continues until the output layer produces a prediction.

2. The Optimization Problem

- **Loss Function:** A loss function measures the discrepancy between the network's prediction and the actual target value. The goal is to minimize this loss.
- **Gradient Descent:** Gradient descent is an optimization algorithm that iteratively adjusts the network's weights in the direction that reduces the loss. This direction is determined by the gradient of the loss function.

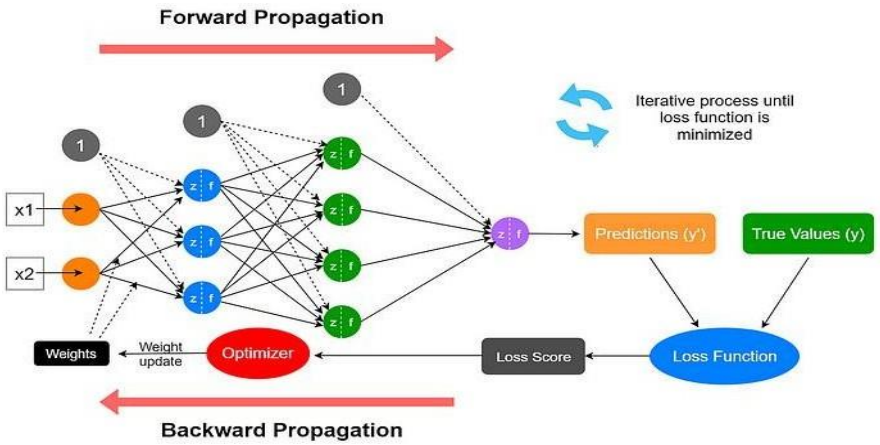
3. Backpropagation: Calculating Gradients

- **Chain Rule:** Backpropagation leverages the chain rule of calculus to efficiently compute the gradient of the loss function with respect to each weight.

Backward Pass:

1. **Error Calculation:** The error (difference between predicted and actual output) is calculated at the output layer.
2. **Error Propagation:** This error is propagated backward through the network, layer by layer.

3. **Gradient Calculation:** At each layer, the gradient of the loss function with respect to the weights connecting to that layer is calculated using the chain rule.
4. **Weight Update:** The weights are adjusted based on the calculated gradients and the learning rate.



- Forward Pass: Input data flows from left to right, through the input layer, hidden layers, and finally to the output layer.
- Backward Pass: The error signal propagates from right to left, starting from the output layer and moving back through the hidden layers to the input layer.

5. Mathematical Representation

Forward Pass:

- Weighted sum at a neuron: $z=\sum_i w_i x_i$
- Activation function output: $a=\sigma(z)$

Backward Pass:

- Error at output layer: $\delta_{output}=\partial a_{output}\partial L$
- Error at hidden layer: $\delta_{hidden}=\delta_{output}\cdot w_{hidden}\cdot\sigma'(z_{hidden})$

Weight Update:

- $w_{new}=w_{old}-\eta\cdot\delta\cdot x$
  - where η is the learning rate.

Hessian Matrix, Generalization, Cross Validation:

Hessian MatrixThe Hessian matrix is a square matrix of second-order partial derivatives of a scalar-valued function. It describes the local curvature of the function. In machine learning and optimization, it is used to understand the behavior of the cost or loss function near a point (e.g., the optimal solution in gradient-based optimization methods).

Definition

For a scalar function f(x)f(\mathbf{x})f(x) where x=(x1,x2,...,xn)\mathbf{x} = (x_1, x_2, ..., x_n)x=(x1,x2,...,xn) is a vector of variables, the Hessian matrix is defined as:

$$H(f) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix}$$

Example (for $f(x, y) = x^2 + y^2$):

$$\frac{\partial^2 f}{\partial x^2} = 2, \quad \frac{\partial^2 f}{\partial y^2} = 2, \quad \frac{\partial^2 f}{\partial x \partial y} = 0$$

So the Hessian matrix is:

$$H(f) = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$$

This indicates a positive definite matrix, suggesting that $f(x, y)$ has a **local minimum** at $(0, 0)$.

Generalization

Generalization refers to the ability of a machine learning model to perform well on unseen data, i.e., data that was not part of the training set. It measures how well the model learns the underlying patterns in the data rather than memorizing the training examples.

Bias-Variance Tradeoff:

- **Bias** refers to the error due to overly simplistic models that cannot capture the complexity of the data.

- **Variance** refers to the error due to models that are too complex and sensitive to noise in the training data.
- The **bias-variance tradeoff** suggests that as we reduce bias by making the model more complex, variance increases, and vice versa.

Generalization Error:

The generalization error (denoted as Egen\mathcal{E}_{gen}Egen) is the difference between the model's performance on the training set and on unseen data (test set). It can be written as:

$$\mathcal{E}_{gen} = \mathbb{E}[L(\hat{y}, y)]$$

Where:

- $\hat{y}$ is the predicted output by the model.
- y is the true output.
- $L(\hat{y}, y)$ is the loss function.

A good model is one that minimizes both bias and variance, hence achieving low generalization error.

Cross-Validation

Cross-validation is a statistical technique used to estimate the performance of a model on an independent dataset. It helps in assessing how the results of a model will generalize to an independent data set. The most common method is **k-fold cross-validation**.

k-Fold Cross-Validation:

In k-fold cross-validation, the dataset is divided into k subsets. The model is trained on k−1 subsets and tested on the remaining subset. This process is repeated k times, with each subset used as the test set exactly once. The final performance is averaged over the k tests.

Procedure:

1. Split the dataset into K equal-sized folds.

2. For each fold iii, use the data in the other k−1k-1k−1 folds for training and the fold iii for testing.
3. Compute the performance measure (e.g., accuracy, error rate) for each fold.
4. Average the performance across all folds.

Cross-Validation Error:

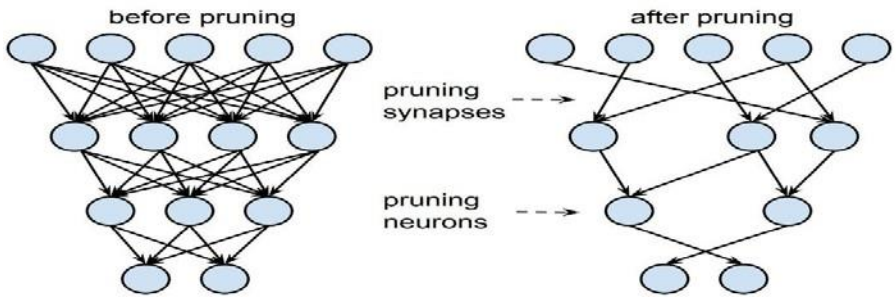
The cross-validation error (denoted as $\mathcal{E}_{cv}$) is the average of the errors over all k folds:

$$\mathcal{E}_{cv} = \frac{1}{k} \sum_{i=1}^{k} \mathcal{E}_i$$

Where:

- $\mathcal{E}_i$ is the error of the model on fold i.

Network Pruning Techniques:



- Neural network pruning involves removing redundant or less important connections (weights) or neurons from a neural network.
- The goal is to create a smaller, more efficient model that maintains acceptable accuracy.

Types of Pruning:

1. **Unstructured Pruning (Weight Pruning):**
   - This involves removing individual weights from the network.
   - It creates a sparse weight matrix, meaning many of the weights are zero.

Equations:

- A common method is magnitude-based pruning:
  - If |w| < threshold, then w = 0, where 'w' is a weight.
- Where the threshold is a value determined by the pruning algorithm.

Pros: High compression rates are possible.
•Cons: Requires specialized hardware or software to take full advantage of sparsity.
Can create irregular memory access patterns.

2. Structured Pruning (Filter/Neuron Pruning):

- This involves removing entire filters or neurons from the network.
- It leads to a more regular and hardware-friendly model.
- **Pros:** Results in actual reductions in computation and memory usage on standard hardware.Easier to implement.
- **Cons:** May not achieve the same compression rates as unstructured pruning.

Pruning Process:

•**Training:** Train the neural network to achieve the desired accuracy.
•**Pruning:** Evaluate the importance of weights or neurons.Remove the least important ones based on a chosen criterion (e.g., magnitude, gradient). **Fine-tuning (Retraining):**

- Retrain the pruned network to recover any lost accuracy.
- This step is crucial for maintaining performance.

•**Iteration:** Often pruning and fine tuning is done in an iterative process.

Applications:

- Mobile devices: Reduces model size and power consumption.
- Embedded systems: Enables deployment of deep learning models on resource-constrained devices.
- Cloud computing: Reduces storage and computational costs.

Virtues and Limitations of Back Propagation Learning, Accelerated Convergence, Supervised Learning:

- Backpropagation (backward propagation of errors) is an algorithm used to train feedforward neural networks.
- It calculates the gradient of the loss function with respect to the network's weights.
- This gradient is then used to update the weights, minimizing the loss and improving the network's accuracy.

Virtues of Backpropagation:

- **Simplicity and Efficiency:**
  - It's a relatively straightforward algorithm, making it easy to implement.
  - It's computationally efficient, especially compared to other optimization methods.
- **Wide Applicability:**
  - Backpropagation can be used to train a wide variety of neural network architectures.
  - It's a fundamental technique in deep learning and has been applied to numerous tasks.
- **Foundation for Deep Learning:**
  - Backpropagation enabled the development of deep neural networks by providing a way to efficiently train networks with many layers.

Limitations of Backpropagation:

- **Sensitivity to Initial Weights:**
  - The initial values of the weights can significantly affect the training process and the final accuracy of the network.
- **Vanishing and Exploding Gradients:**
  - In deep networks, gradients can become very small (vanishing) or very large (exploding) as they are propagated backward, hindering learning.
- **Local Minima:**
  - Backpropagation can get stuck in local minima of the loss function, preventing the network from reaching the global minimum.
- **Requires Labeled Data:**
  - It is a supervised learning algorithm, so it needs labeled data to function.
- **Computational Cost:**
  - Training very deep neural networks can be very computationally expensive.

Accelerated Convergence:

Several techniques have been developed to accelerate the convergence of backpropagation:

Momentum:

- Adds a fraction of the previous weight update to the current update, helping to overcome local minima and accelerate convergence.

- **Adaptive Learning Rate Methods:**
  - Algorithms like Adam, RMSprop, and AdaGrad adjust the learning rate for each weight during training, improving convergence.
- **Batch Normalization:**
  - Normalizes the activations of each layer, stabilizing training and allowing for higher learning rates.
- **Skip Connections (Residual Networks):**
  - These connections help to mitigate the vanishing gradient problem in deep networks.

Supervised Learning:

- Backpropagation is inherently a supervised learning algorithm.
- In supervised learning, the network is trained on a dataset of labeled examples, where each example consists of an input and its corresponding desired output.
- The network learns to map inputs to outputs by adjusting its weights to minimize the difference between its predicted outputs and the true outputs.

Key Concepts:

- **Loss Function:** A function that measures the difference between the predicted outputs and the true outputs.
- **Gradient Descent:** An optimization algorithm that iteratively adjusts the weights to minimize the loss function.

- **Learning Rate:** A parameter that controls the size of the weight updates.