SALALE UNIVERSITY

COLLEGE OF NATURAL SCIENCES

DEPARTMENT OF COMPUTER SCIENCE

INTRODUCTION TO MACHINE LEARNING ASSIGNMENT

Prepared by:

S.No.	Name	ID.No.
1	Girum Eshetu	WU 0026/13
2	Beshana Eshete	WU 0009/13
3	Wudinesh Nugusie	WU 0245/13
4	Mekete Hailu	WU 0035/13
5	Tolera Girma	WU 0059/13

SUBMITTED TO: **Ins. Deribew**, CS. Department, Salale University
Date Submitted: Apr –, 2025
Fiche, Oromia, Ethiopia

Table of Contents

Neural Network

1.1 Introduction

Neural networks are an essential component of artificial intelligence and machine learning. They have revolutionized various fields, including image processing, speech recognition, and autonomous decision-making. Neural networks are designed to simulate the functionality of the human brain, enabling them to learn from data and make predictions. This chapter explores the foundation of neural networks, beginning with an understanding of the biological brain and extending into the computational models that mimic its behavior.

1.1.1 Understanding the Brain

The human brain consists of billions of neurons, which communicate through electrical and chemical signals. Each neuron has dendrites that receive signals, a cell body that processes them, and an axon that transmits signals to other neurons. The brain's ability to learn and adapt is due to synaptic plasticity, where connections between neurons strengthen or weaken based on experience. This biological mechanism serves as inspiration for artificial neural networks (ANNs), where artificial neurons are interconnected in a similar manner to process and analyze information.

Neurons in the brain function collectively to recognize patterns, solve problems, and make decisions. For instance, when recognizing a face, different regions of the brain process visual inputs, identify features such as eyes and mouth, and then combine this information to recognize a person. Neural networks attempt to replicate this distributed processing to perform complex tasks in machine learning.

1.1.2 Neural Networks as a Paradigm for Parallel Processing

One of the key advantages of neural networks is their ability to perform parallel processing. Unlike traditional computing, which follows a sequential execution model, neural networks can process multiple inputs simultaneously. This parallelism is what makes them efficient for tasks such as image recognition, where thousands of pixels are analyzed at once.

Parallel processing in neural networks also contributes to their robustness. Even if certain neurons or connections fail, the network can still function due to redundancy in the connections. This characteristic is analogous to the human brain, where damage to certain neurons does not necessarily result in complete cognitive failure.

Another critical aspect of neural networks is their adaptability. Unlike conventional algorithms that require explicit programming, neural networks learn from data by adjusting their weights and biases. This learning process allows them to generalize from examples and improve performance over time, making them suitable for applications such as natural language processing and medical diagnosis.

1.2 The Perceptron

The perceptron is one of the earliest and simplest models of an artificial neural network. It was introduced by Frank Rosenblatt in 1958 as a binary classifier that can distinguish between two different classes based on input features. A perceptron consists of input nodes, weights, a bias, an activation function, and an output node.

Mathematically, a perceptron computes the weighted sum of inputs and passes the result through an activation function. The equation for a perceptron's output is given by:

$$y = f(\sum_{i=1}^{n} w_i x_i + b)$$

where w_i are the weights, x_i are the input features, b is the bias, and f is the activation function.

The perceptron follows a simple rule for weight adjustment: if the output is incorrect, the weights are updated to reduce the error. This learning process continues until the perceptron correctly classifies all training examples.

Limitations of the Perceptron

While the perceptron is useful for linearly separable problems, it struggles with non-linearly separable data, such as the XOR problem. To address this limitation, researchers developed multilayer perceptrons (MLPs), which incorporate hidden layers and nonlinear activation functions, allowing them to solve complex classification problems.

1.3 Training a Perceptron

Training a perceptron involves adjusting its weights using a learning algorithm. The most commonly used method is the perceptron learning rule, which updates the weights based on the error between predicted and actual outputs. The algorithm follows these steps:

1. Initialize weights randomly or set them to small values.
2. For each training example, compute the output using the activation function.
3. Compare the output with the expected result and calculate the error.
4. Update the weights using the rule:

   $$w_i = w_i + \eta(d - y)x_i$$

   where η is the learning rate, d is the desired output, and y is the actual output.
5. Repeat the process until the error is minimized or a predefined number of iterations is reached.

By iteratively adjusting the weights, the perceptron learns to classify data accurately. However, as mentioned earlier, it is limited to solving problems where the data is linearly separable. More complex models, such as multilayer perceptrons, overcome these limitations by introducing hidden layers and advanced learning techniques.

This concludes the first section. Let me know if you would like me to continue with the next sections on artificial neural networks and multilayer perceptrons.

1.4 Artificial Neural Network

An Artificial Neural Network (ANN) is a computational model inspired by the human brain's neural structure. It consists of layers of interconnected artificial neurons that process information in a hierarchical manner. These networks are used in various applications, including image and speech recognition, medical diagnosis, and financial forecasting.

ANNs can be classified into different types based on their architecture and functionality. Some common types include:

- **Feedforward Neural Networks (FNNs)**: Information moves in one direction, from input to output, without cycles.
- **Recurrent Neural Networks (RNNs)**: These networks have connections that allow information to be passed from one step to the next, making them suitable for sequence prediction tasks such as language modeling.
- **Convolutional Neural Networks (CNNs)**: Specialized for image and video processing by using filters to capture spatial features.
- **Radial Basis Function Networks (RBFNs)**: Used for classification and function approximation by measuring the distance of inputs from a central point.

1.4.1 Components of an ANN

An artificial neural network consists of three main layers:

1. **Input Layer**: Receives raw data and passes it to the next layer.
2. **Hidden Layer(s)**: Processes inputs using weighted connections and activation functions to extract features and patterns.
3. **Output Layer**: Produces the final result based on computations from previous layers.

Each artificial neuron in an ANN processes inputs by applying a weighted sum followed by an activation function. Common activation functions include:

- **Sigmoid**: $f(x) = \frac{1}{1+e^{-x}}$, used in binary classification problems.
- **Tanh**: $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$, which outputs values between -1 and 1.
- **ReLU (Rectified Linear Unit)**: $f(x) = max(0, x)$, widely used in deep learning for efficient gradient propagation.

1.5 Multilayer Perceptron

A Multilayer Perceptron (MLP) is an extension of the perceptron model that includes multiple hidden layers, allowing it to learn complex patterns in data. Unlike the simple perceptron, which can only handle linearly separable problems, MLPs can model non-linear relationships using activation functions and backpropagation for training.

1.5.1 Structure of an MLP

An MLP typically consists of three or more layers:

1. **Input Layer**: Accepts numerical features as input.
2. **One or More Hidden Layers**: Each neuron in a hidden layer applies a transformation to the inputs, allowing the model to learn abstract features.
3. **Output Layer**: Generates the final prediction, which could be a class label (classification) or a continuous value (regression).

Each connection between neurons is associated with a weight, which determines the influence of an input on the next layer. Learning in MLPs involves adjusting these weights using training algorithms like gradient descent.

1.6 Backpropagation Algorithm

The backpropagation algorithm is a supervised learning method used to train MLPs by minimizing error. It involves two phases:

1. **Forward Propagation**: Inputs are passed through the network layer by layer, and the output is computed.

2. **Backward Propagation**: Errors are calculated by comparing predicted outputs with actual values, and adjustments are made to weights using gradient descent.

The weight update rule follows: $w = w - \eta \frac{\partial E}{\partial w}$ where η is the learning rate, and $\frac{\partial E}{\partial w}$ is the gradient of the error function.

1.6.1 Nonlinear Regression

Backpropagation enables ANNs to perform nonlinear regression tasks, where the relationship between input and output is not a simple linear function. By using activation functions like ReLU and sigmoid, MLPs can learn complex mappings from data, making them useful in financial forecasting and medical diagnosis.

1.6.2 Two-Class Discrimination

Two-class discrimination refers to the problem of classifying data into two distinct categories. Neural networks, particularly MLPs, are commonly used for binary classification tasks such as spam detection (spam vs. non-spam) and medical diagnosis (disease vs. no disease).

A neural network designed for two-class discrimination consists of:

- An input layer representing feature vectors.
- One or more hidden layers for feature extraction.
- An output layer with a single neuron, which uses the sigmoid activation function to map the output to a probability between 0 and 1.

The learning process involves minimizing a loss function such as binary cross-entropy: $L = -\frac{1}{N} \sum_{i=1}^{N} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$ where y_i is the true label and $\hat{y}_i$ is the predicted probability.

1.6.3 Multiclass Discrimination

Multiclass discrimination extends two-class discrimination to more than two categories. Instead of a single output neuron, the output layer contains multiple neurons corresponding to different classes.

For example, in digit classification (0-9), the neural network outputs 10 probability values, each representing the likelihood of a digit. The softmax activation function is commonly used in the output layer: $P(y = j|x) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$ where z_j is the network's raw score for class j, and K is the total number of classes.

The loss function used for multiclass classification is categorical cross-entropy: $L = -\sum_{i=1}^{N} \sum_{j=1}^{K} y_{ij} \log(\hat{y}_{ij})$ where y_{ij} is 1 if the sample belongs to class j and 0 otherwise.

1.6.4 Multiple Hidden Layers

Adding multiple hidden layers enhances the learning capability of neural networks. Deeper architectures can capture hierarchical features and learn complex representations. However, deeper networks require:

- **More computational power** for training and inference.
- **Regularization techniques** (such as dropout and weight decay) to prevent overfitting.
- **Optimized initialization** methods to prevent vanishing/exploding gradients.

Deep neural networks with multiple hidden layers are the foundation of modern deep learning applications, including speech recognition, image classification, and natural language processing.

Would you like me to continue with training procedures, improving convergence, overtraining, and structuring the network?

1.7 Training Procedures

Training a neural network involves optimizing the network's parameters (weights and biases) to minimize the difference between predicted and actual values. This process is typically performed using a set of training data and a chosen optimization algorithm.

1.7.1 Improving Convergence

Convergence in neural network training refers to the point where further training does not significantly improve performance. Various techniques can help improve convergence speed and stability:

Learning Rate Adjustment

The learning rate determines the step size during weight updates. If the learning rate is too high, the training process may overshoot the optimal solution. If it is too low, convergence can be slow. Methods for adjusting the learning rate include:

- **Adaptive Learning Rate Methods**: Techniques such as AdaGrad, RMSprop, and Adam dynamically adjust the learning rate during training to improve convergence.
- **Learning Rate Scheduling**: The learning rate is reduced over time based on a predefined schedule, allowing finer weight adjustments as training progresses.

Batch Normalization

Batch normalization normalizes inputs to each layer, reducing internal covariate shifts and leading to faster training and improved stability. This technique is particularly useful in deep networks.

Momentum and Optimizers

Using momentum in gradient descent helps accelerate convergence by reducing oscillations. Popular optimization algorithms that incorporate momentum include:

- **SGD with Momentum**: Enhances standard stochastic gradient descent by accumulating previous gradients.
- **Adam (Adaptive Moment Estimation)**: Combines momentum and adaptive learning rate methods for efficient convergence.
- **Nesterov Accelerated Gradient (NAG)**: Improves momentum-based updates by anticipating the next step in parameter adjustments.

1.7.2 Overtraining

Overtraining, or overfitting, occurs when a neural network learns the noise in the training data instead of the actual patterns. This results in high accuracy on training data but poor generalization to new data.

Techniques to Prevent Overtraining

- **Regularization**: Methods such as L1 (Lasso) and L2 (Ridge) regularization add penalty terms to the loss function to discourage overly complex models.
- **Dropout**: Randomly deactivates neurons during training, preventing the network from relying too much on specific features.
- **Early Stopping**: Stops training when validation performance starts to degrade, preventing unnecessary parameter adjustments.
- **Data Augmentation**: Expands the training dataset by applying transformations such as rotation, scaling, or flipping, improving generalization.

1.7.3 Structuring the Network

The architecture of a neural network significantly affects its performance. Key considerations include:

- **Number of Layers**: More layers enable learning of hierarchical features but require more data and computational resources.
- **Number of Neurons per Layer**: Too few neurons can underfit, while too many can lead to overfitting.
- **Activation Functions**: Choosing appropriate activation functions (ReLU, Sigmoid, Tanh) for different layers is crucial for effective learning.
- **Connectivity Patterns**: Fully connected layers provide more expressiveness, but sparse connections (e.g., in convolutional networks) reduce computational complexity.

Properly structuring a neural network involves balancing complexity and generalization. Techniques such as hyperparameter tuning, cross-validation, and architecture search help design optimal networks for specific tasks.

1.8 Tuning the Network Size

Choosing the appropriate size for a neural network is crucial for balancing computational efficiency and performance. The network size is determined by factors such as the number of layers, neurons per layer, and overall model complexity.

1.8.1 Determining the Optimal Network Size

The optimal network size varies depending on the problem domain, dataset complexity, and computational resources. Key factors include:

- **Dataset Size**: A larger dataset allows for more complex models without overfitting.
- **Feature Complexity**: If the data has highly non-linear relationships, a deeper network may be required.
- **Computational Resources**: Larger networks demand more memory and processing power.

1.8.2 Techniques for Finding the Right Network Size

Pruning

Pruning removes unnecessary neurons or connections in a network to reduce model size and computational cost while maintaining accuracy. Common pruning techniques include:

- **Weight Pruning**: Eliminates weights that contribute the least to the network's performance.
- **Neuron Pruning**: Removes entire neurons based on their importance to the learning process.

Dropout and Regularization

Applying dropout during training helps determine whether a network has redundant parameters. Regularization techniques such as L1 and L2 penalties prevent excessive network growth and help find an optimal size.

Grid Search and Random Search

Hyperparameter tuning methods such as grid search and random search explore different network sizes to identify the best configuration.

Neural Architecture Search (NAS)

NAS is an advanced technique that automates the process of finding optimal neural network architectures using machine learning algorithms.

By carefully tuning the network size, developers can achieve efficient models that generalize well while minimizing computational costs.