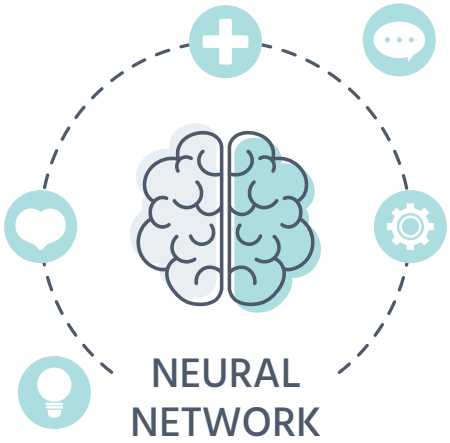
Cracking The Neural Computing



Unraveling the Neural Network Frontier: A Journey into Deep Learning

July 2023

Kanchon Gharami

Contents

Chapter 1

Introduction

1.1 Humans and Computers

Table 1.1: Comparison between Human Brain and Machines

Aspect	Human Brain	Machine/Artificial Intelligence
Processing Speed	Comparatively slow (neurons fire at millisecond intervals)	Extremely fast, processing data at lightning speed
Parallel/Serial Computing	Primarily serial processing	Highly capable of parallel processing
Computing Speed	Limited processing speed due to serial nature	High computing speed due to parallel processing
Pattern Recognition	Excellent pattern recognition capability	Requires vast amounts of labeled data for training

1.2 Graceful Degradation

Graceful degradation refers to the capability of the network to maintain performance and continue functioning, albeit at a reduced efficiency, even when certain parts of the network are damaged or some neurons are removed. In real-world applications, this means that even if some data is lost or corrupted, the overall performance of the network decreases slowly rather than failing entirely.

Key characteristics of graceful degradation include:

- **Fault Tolerance**: The network can handle faults or errors without completely failing.
- **Robustness**: The network maintains its performance even in the face of partial network

failures.

- **Redundancy**: Multiple neurons often store the same information, so the loss of a few neurons does not result in the loss of that information.

1.3 Effective Coupling for Information Transfer

Effective coupling refers to the effective connection and communication between neurons in a neural network. It determines the efficiency of information transfer in the network. The strength and nature of these connections (synapses) can significantly influence the network's learning capability and its capacity to solve complex problems.

Key characteristics of effective coupling include:

- **Connection Strength**: Strong connections allow for more efficient transfer of information.
- **Connection Pattern**: The structure and pattern of connections can influence the network's learning ability and functionality.
- **Dynamic Adaptability**: In many neural networks, connections can adapt and change their strength based on the learning process (synaptic plasticity).

Chapter 2

Pattern Recognition

2.1 Pattern

2.1.1 What Is Pattern?

A pattern in the context of neural computing refers to a regularity in the world or in a manmade design. It can also be thought of as a set of numerical or symbolic values that characterize a certain feature or object.

For instance, consider a sequence of numbers like 2, 4, 6, 8, 10. Here, the pattern can be described as "an increase of 2".

Here is an example of a pattern in a table:

Index	Value
1	2
2	4
3	6
4	8
5	10

Table 2.1: Example of a Numerical Pattern

The pattern in the above table is a sequence of even numbers where each number is increased by 2 from the previous one.

2.1.2 What Is Pattern Recognition?

Pattern recognition is a branch of machine learning that focuses on the detection and identification of patterns and regularities in data.

Steps of Pattern Recognition

Pattern recognition generally involves two key steps:

1. **Feature Extraction** Feature extraction is the process of transforming the raw data into a set of features, or "feature vectors", that can effectively represent the underlying patterns in a simpler way.
    - A measurement taken on the input pattern
2. **Classification** Classification is the task of assigning the object of interest to a predefined class or category based on its features.
    - Map these input features onto a classification state

It involves classifying input data into objects or classes based on key features, using either statistical(Numeric) or syntactic(Non-Numeric) approach.

- **Numeric Approach:** deterministic and statistical measures – made on the geometric pattern space
- **Non-Numeric Approach:** based on symbolic processing – i.e. – fuzzy set.

2.2 Feature Vectors

A feature vector is an n-dimensional vector of numerical features that represent some object.

These features are often measurements or descriptors that quantify the characteristics of the object in some meaningful way. For example, in image recognition, a feature vector might include values indicating color intensity, texture patterns, shapes present, etc.

2.3 Feature Space

Feature space refers to the n-dimensional space in which these feature vectors exist.

Each dimension represents a single feature, and the position of an object in this space is determined by its feature vector. The goal in pattern recognition and machine learning is often to find surfaces or boundaries in this space that separate objects of different classes or groups.

For instance, if we have a feature vector $F = [f_1, f_2, ..., f_n]$, its feature space is $\mathbb{R}^n$.

2.4 Discriminant Functions and Decision Boundaries

2.4.1 Discriminant Functions

Algebraic representation of a decision boundary is refer as discriminant functions.

In the context of pattern recognition, discriminant functions are used to determine to which category or class a new observation belongs. The decision is made based on a set of measurements, known as features, taken from the observation. Simply put, given an input (a feature vector), a discriminant function assigns it to one of several predefined categories.

2.4.2 Decision Boundaries

A decision boundary is a hypersurface that partitions the underlying input space into two or more regions. Each region corresponds to a class, and the decision boundary serves as the "line" that differentiates between these classes.

1. The simplest function that separate the two clusters is a straight line.
2. It represents a very widely used category of classifiers known as linear classifiers.

2.5 Methods of measurement based on distance

Distance Metrics

In the context of machine learning and pattern recognition, a distance metric, also known as a distance function, is a function that defines a distance between elements of a set. The output of this function, the distance, is a scalar measure of the dissimilarity or separation between two elements.

Common examples of distance metrics include Euclidean distance, Manhattan distance, and Hamming distance.

- **Euclidean Distance**: Also known as L^2 distance, it measures the "as-the-crow-flies" distance. Given two points $P = (p_1, p_2, ..., p_n)$ and $Q = (q_1, q_2, ..., q_n)$, the Euclidean distance is defined as:

$$d_{\text{euclidean}}(P, Q) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

- **Manhattan Distance**: Also known as L^1 distance or city-block distance, it is the distance a taxi would drive in a city (made up of a grid). For points P and Q, it's defined as:

$$d_{\text{manhattan}}(P, Q) = |q_1 - p_1| + |q_2 - p_2| + \cdots + |q_n - p_n|$$

- **Hamming Distance**: Used for binary vectors, it is the number of bit flips required to convert one binary vector into another. For binary vectors $P = (p_1, p_2, ..., p_n)$ and $Q = (q_1, q_2, ..., q_n)$, it's defined as:

$$d_{\text{hamming}}(P, Q) = \sum_{i=1}^{n} \delta(p_i, q_i), \text{ where } \delta(p, q) = \begin{cases} 1 & \text{if } p \neq q \\ 0 & \text{if } p = q \end{cases}$$

- **Squared Distance**: This is simply the square of the Euclidean distance. It avoids computing the square root and can be computationally more efficient. For points $P = (A_p, B_p, ..., C_p)$ and $Q = (A_q, B_q, ..., C_q)$, it's defined as:

$$d_{\text{squared}}(P, Q) = \max\left(|A_P - A_Q|, |B_P - B_Q|, |C_P - C_Q|\right)$$

Let's consider a small dataset with three data points:

- $A = (1, 2)$
- $B = (3, 4)$
- $C = (5, 6)$

We'll calculate the distance between all pairs of these points using various distance metrics.

2.5.1 Hamming Distance

Hamming distance isn't applicable to this dataset because it's used for binary data.

2.5.2 Euclidean Distance

Euclidean distance between A and B:

$$d_{\text{euclidean}}(A, B) = \sqrt{(3-1)^2 + (4-2)^2} = 2\sqrt{2} \tag{2.1}$$

Euclidean distance between B and C:

$$d_{\text{euclidean}}(B, C) = \sqrt{(5-3)^2 + (6-4)^2} = 2\sqrt{2} \tag{2.2}$$

2.5.3 City Block Distance (Manhattan Distance)

City block distance between A and B:

$$d_{\text{city block}}(A, B) = |3 - 1| + |4 - 2| = 2 + 2 = 4 \tag{2.3}$$

City block distance between B and C:

$$d_{\text{city block}}(B, C) = |5 - 3| + |6 - 4| = 2 + 2 = 4 \tag{2.4}$$

2.5.4 Squared Euclidean Distance

Squared Euclidean distance between A and B:

$$d_{\text{squared}}(A, B) = (3 - 1)^2 + (4 - 2)^2 = 4 + 4 = 8 \tag{2.5}$$

Squared Euclidean distance between B and C:

$$d_{\text{squared}}(B, C) = (5 - 3)^2 + (6 - 4)^2 = 4 + 4 = 8 \tag{2.6}$$

2.6 Linear Classifier

A linear classifier is a method used in machine learning for separating data points into different classes. It does this by drawing a linear decision boundary, or a hyperplane, in the feature space.

The decision boundary of a linear classifier is defined by a discriminating function $f(\mathbf{X})$ that is a weighted sum of the features, subtracting a threshold θ:

$$f(\mathbf{X}) = \sum_i w_i x_i \tag{2.7}$$

where $\mathbf{X} = [x_1, x_2, ..., x_n]$ is the feature vector, $\mathbf{W} = [w_1, w_2, ..., w_n]$ are the weights, and n is the number of features.

For a given input $\mathbf{X}$, the classifier assigns it to one class if $f(\mathbf{X})$ is greater than zero, and another class otherwise:

- If $f(\mathbf{X}) > 0$, then $\mathbf{X}$ is classified as Class 1.
- If $f(\mathbf{X}) \leq 0$, then $\mathbf{X}$ is classified as Class 2.

Using matrix algebra, the discriminant function can be written more compactly as:

$$f(\mathbf{X}) = \sum_i w_i x_i - \theta \tag{2.8}$$

2.7 Controlling the Decision Boundary

- The decision boundary's position in the pattern space is controlled by two parameters: the slope of the line and the y-axis intercept.
- The slope of the decision boundary is determined by the weight vector.
- When the classifier output is zero ($f(\mathbf{X}) = 0$), this condition is known as the crossover point or boundary condition.
- Comparing the decision function to the equation of a straight line, the ratio of weight values w_1 and w_2 controls the slope, while the bias value, θ, controls the y-axis intercept.

2.8 Finding the Weight Vector

- Determining the appropriate weight vector is a critical but non-trivial task.
- The weight vector is typically found through iterative trial and error methods that modify weights based on an error function.
- This error function measures the discrepancy between the classifier's output and the desired response.

2.9 k-Nearest Neighbors (k-NN) Algorithm

The k-NN algorithm is a type of instance-based learning method used in machine learning. Given a new, unseen observation, it operates by finding the 'k' samples in the training dataset that are closest (in terms of a distance metric) to the new sample, and outputs the most common output variable among these 'k' nearest neighbors as the prediction.

The steps of the k-NN algorithm are as follows:

1. **Define the parameter k**: The number of nearest neighbors to consider.
2. **Calculate Distances**: For a new input, compute its distance from all samples in the training set. The distance could be calculated using different distance measures like Euclidean, Manhattan etc.

$$D(x, x_i) = \sqrt{(x_1 - x_{i1})^2 + (x_2 - x_{i2})^2 + \cdots + (x_n - x_{in})^2}$$

3. **Find Nearest Neighbors**: Identify 'k' samples in the training data that are closest to the new point.
4. **Vote for Labels**: Each of the 'k' nearest neighbors votes for their class label.
5. **Make a prediction**: Assign the new point the class that has the most votes.

Chapter 3

The Basic Neuron

3.1 Modeling the Single Neuron

A neuron in a neural network is a computational unit that takes a set of inputs, applies a transformation to them, and produces an output. The neuron can be modeled as follows:

- **Inputs**: Each neuron receives multiple inputs, represented as a feature vector $\mathbf{X} = [x_1, x_2, ..., x_n]$. Each input is associated with a weight $\mathbf{W} = [w_1, w_2, ..., w_n]$.
- **Weighted Sum**: The inputs and weights are multiplied element-wise and then summed to produce a single value, which is then offset by a bias term θ. The output of this operation is the net input, represented as:

$$z = \sum_i w_i x_i - \theta \tag{3.1}$$

- **Activation Function**: The net input is then passed through an activation function f, which determines the neuron's output. Commonly used activation functions include the sigmoid, hyperbolic tangent, and ReLU functions. The output is calculated as:

$$y = f(z) \tag{3.2}$$

3.1.1 Threshold Function

The Threshold function is a type of activation function used in artificial neurons and neural networks, that produces a binary output, typically either 0 or 1, based on whether the input exceeds a certain threshold value.

$$f(x) = \begin{cases} 0 & \text{if } x < \text{threshold} \\ 1 & \text{if } x \geq \text{threshold} \end{cases} \tag{3.3}$$

Alternative names for the Threshold function include:

- Step Function
- Heaviside Function

3.1.2 Difference between Activation Function and Threshold Function

The activation function and the threshold function both play a role in determining the output of a neuron in a neural network, but they are used in different contexts and have different characteristics.

- **Activation Function**: An activation function is a general term for a function that takes the weighted sum of the inputs of a neuron and transforms it into an output signal. It can be linear or non-linear. Examples include the Sigmoid function, Hyperbolic Tangent (tanh) function, and Rectified Linear Unit (ReLU) function.
- **Threshold Function**: The threshold function is a specific type of activation function that produces binary output (usually 0 or 1) depending on whether the input exceeds a certain threshold. The threshold function is also referred to as a step function or a binary step function.

In summary, while the threshold function can serve as an activation function, not all activation functions are threshold functions due to the variety of forms activation functions can take.

3.2 Perceptron Learning Algorithm

The perceptron is a simple supervised learning algorithm for binary classifiers. It is a type of linear classifier that makes its predictions based on a linear predictor function.

The steps of the Perceptron Learning Algorithm are as follows:

1. Initialize the weight vector and the bias with zero or small random numbers.
2. For each example in our training set, perform the following steps:
    - Compute the output value using the current weights and bias.
    - Update the weights and the bias if an error is made.

3. Repeat step 2 until the algorithm converges (no errors) or a predetermined number of epochs are completed.

The weight update in step 2 can be done using the following rule:

$$w_i = w_i + \Delta w_i \tag{3.4}$$

where:

$$\Delta w_i = \eta(y^{(j)} - \hat{y}^{(j)})x_i^{(j)} \tag{3.5}$$

Here, $y^{(j)}$ is the true class label, $\hat{y}^{(j)}$ is the predicted class label, $x_i^{(j)}$ is the i^{th} feature value, and η is the learning rate.

3.3 Widrow-Hoff Rule for Adaptive Weights

The Widrow-Hoff rule, also known as the Least Mean Squares (LMS) rule or the Delta rule, is a method for learning the weights in a neural network. It's particularly used in the context of linear units, where the output is a weighted sum of the inputs.

The Widrow-Hoff rule updates the weights by minimizing the mean square error between the expected output and the actual output of the neuron. The update rule is as follows:

$$\Delta w_{ij} = \eta(t_j - y_j)x_i \tag{3.6}$$

Here:

- Δw_{ij} is the change of weight w_{ij}
- η is the learning rate
- t_j is the target output for the j^{th} output neuron
- y_j is the actual output of the j^{th} output neuron
- x_i is the i^{th} input

This rule adjusts the weights in the direction of steepest descent of the error surface, aiming to find the minimum error.

3.4 Limitations of Perceptrons

While perceptrons are foundational to neural networks and have their uses, they also have certain limitations. These include:

1. **Binary Outputs**: Perceptrons can only output binary classifications (0 or 1), which limits their utility in multi-class classification problems.
2. **Linearly Separable Data**: The perceptron learning algorithm is only guaranteed to converge if the classes are linearly separable. If the data is not linearly separable, the algorithm will never converge.
3. **Single Layer**: A single-layer perceptron can only learn linearly separable patterns, as it essentially draws a single line (or hyperplane in higher dimensions) to separate the data. More complex, non-linear patterns require multi-layer perceptrons (i.e., neural networks with hidden layers).
4. **No Probability Estimates**: Perceptrons do not provide probability estimates for the predictions, which are often useful in many machine learning tasks.
5. **Inflexible Threshold**: In a perceptron, the threshold is fixed (i.e., it cannot be adjusted), which can limit the model's ability to learn complex data patterns.

3.5 Credit Assignment Problem in Perceptrons

The "credit assignment problem" in the context of perceptrons, or more generally in neural networks, refers to the challenge of determining which neurons or connections in the network are responsible for any given error in the output.

When an error occurs, it can be challenging to determine which parts of the network should be adjusted to reduce the error. Should the weights of a particular connection be modified? Should the activation function of a particular neuron be adjusted? Or is some combination of these changes needed? These are the questions that the credit assignment problem seeks to answer.

This problem is especially difficult in networks with hidden layers, since the relationship between the input, the weights, and the output is not directly observable. The backpropagation algorithm is a commonly used method to solve the credit assignment problem in multi-layer perceptrons, as it effectively computes how much each neuron's weights contributed to the final error.

Chapter 4

The Multi-layer Perceptron

4.1 Backpropagation Algorithm for Multilayer Perceptron

Backpropagation is an efficient method used to calculate the gradient of the loss function in a neural network, which can then be used by a gradient descent algorithm to update the weights of the network. Here are the main steps of backpropagation for an MLP:

1. **Initialization**: Initialize the weights with small random numbers.
2. **Forward Pass**: For each input in the dataset:
   - Propagate the input forward through the network.
   - Calculate the output of each neuron from the input layer, through the hidden layers, to the output layer.

   The output of each neuron i in layer l can be calculated using:

$$y_i^l = f\left(\sum_j w_{ji}^l y_j^{l-1} + b_i^l\right) \tag{4.1}$$

   where f is the activation function, w_{ji}^l is the weight from neuron j in layer $l-1$ to neuron i in layer l, and b_i^l is the bias of neuron i in layer l.
3. **Backward Pass**: For each output neuron, calculate its error term and then propagate these errors back through the network to compute the error term for each hidden neuron. The error term δ_i^l of neuron i in layer l is calculated as follows:

- For the output layer ($l = L$):

$$\delta_i^L = (y_i^L - t_i) f'(h_i^L) \tag{4.2}$$

where t_i is the target output, and h_i^L is the weighted sum of inputs of the output neuron before activation.

- For the hidden layers ($l < L$):

$$\delta_i^l = \left(\sum_k w_{ik}^{l+1} \delta_k^{l+1} \right) f'(h_i^l) \tag{4.3}$$

4. **Weight Update**: Update each weight in the network using the error terms. The weights are updated according to the rule:

$$w_{ji}^l = w_{ji}^l - \eta \, y_j^{l-1} \delta_i^l \tag{4.4}$$

where η is the learning rate.

5. **Iteration**: Repeat the Forward Pass, Backward Pass, and Weight Update steps for the desired number of iterations or until the network performance meets the desired criteria.

4.2 Activation Functions

4.2.1 Rectified Linear Unit (ReLU)

Purpose: The ReLU function is used to introduce non-linearity into the network. It helps to mitigate the vanishing gradient problem, allowing the network to learn faster and perform better.

Equation:

$$f(x) = \max(0, x) \tag{4.5}$$

Advantages:

- Helps to alleviate the vanishing gradient problem.
- Computationally efficient.

Disadvantages:

- Neurons can "die" during training, i.e., they may stop outputting anything other than 0 if the weights are updated such that the weighted sum of the neuron's inputs is negative.

4.2.2 Sigmoid

Purpose: The sigmoid function is used to map the output of a neuron to a range between 0 and 1, making it useful for output neurons in networks intended for binary classification problems.

Equation:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{4.6}$$

Advantages:

- Maps output to range (0, 1), which can be useful for interpreting the output as a probability.
- Smooth gradient, which can be beneficial in gradient descent.

Disadvantages:

- Suffers from the vanishing gradient problem, which can slow down learning or cause it to stop altogether.
- Outputs are not zero-centered.
- Computationally expensive due to the exponential operation.

4.3 Equations in Multi-Layer Perceptron (MLP)

4.3.1 Input to Hidden Layer

In an MLP, the transformation from the input layer to the hidden layer is determined by a weight matrix, a bias vector, and an activation function. Given an input vector $\mathbf{x}$, the output $\mathbf{h}$ of the hidden layer can be calculated as:

$$\mathbf{h} = f(\mathbf{W}_{ih}\mathbf{x} + \mathbf{b}_h) \tag{4.7}$$

where $\mathbf{W}_{ih}$ is the weight matrix between the input and hidden layer, $\mathbf{b}_h$ is the bias vector of the hidden layer, and $f(\cdot)$ is the activation function.

4.3.2 Hidden to Output Layer

Similarly, the transformation from the hidden layer to the output layer is determined by another weight matrix, another bias vector, and typically another (or sometimes the same)

activation function. Given the output $\mathbf{h}$ of the hidden layer, the output $\mathbf{y}$ of the MLP can be calculated as:

$$\mathbf{y} = g(\mathbf{W}_{ho}\mathbf{h} + \mathbf{b}_o) \tag{4.8}$$

where $\mathbf{W}_{ho}$ is the weight matrix between the hidden and output layer, $\mathbf{b}_o$ is the bias vector of the output layer, and $g(\cdot)$ is the activation function.

4.4 Local Minima Problem and Solutions

4.4.1 Local Minima Problem

In the context of neural networks, the local minima problem refers to the training algorithm getting stuck in a local minimum of the loss function, instead of finding the global minimum. This can result in a model that is suboptimal.

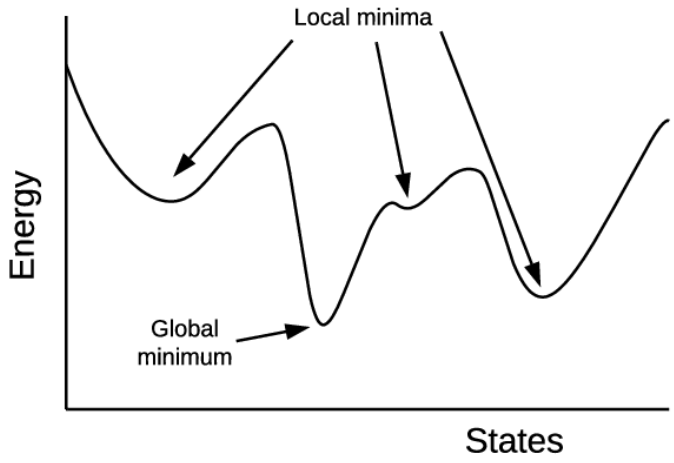


Figure 4.1: Illustration of Local vs Global Minima

4.4.2 Solutions to Local Minima Problem

1. **Random Initialization**: Initializing the weights to different values can result in the algorithm converging to different local minima.
2. **Momentum**: Adding a fraction of the update vector of the past time step to the current update vector can help the algorithm escape local minima.
3. **Learning Rate Schedule**: Adapting the learning rate over time can help the algorithm escape local minima.

4. **Using Advanced Optimizers**: Optimizers like Adam, RMSProp, etc. have mechanisms to avoid getting stuck in local minima.

5. **Adding Noise**: Injecting noise into either the input data or the gradient can help the model escape local minima.

4.5 Re-Learning in Multi-Layer Perceptrons (MLPs)

Re-learning is a process where an already trained MLP undergoes further training, typically to improve its performance based on new data, accommodate new classes, or adapt to changes in the data distribution.

The steps involved in re-learning are as follows:

1. **Retain the Learned Weights:** Keep the weights and biases from the previous learning phase.

2. **Introduce New Data:** Incorporate the new data into the training set. This could be new instances of existing classes, instances of new classes, or a combination of both.

3. **Training with New Data:** Use the same training algorithm (such as backpropagation) to train the network with the new data. During this process, the weights and biases are adjusted based on the errors the network makes when predicting the new data.

The main advantage of re-learning is that it allows the network to continuously adapt to changes or improvements in the data without having to undergo a complete retraining process.

4.6 Fault Tolerance in Multi-Layer Perceptrons (MLPs)

Fault tolerance refers to the ability of an MLP to retain acceptable performance levels in the presence of faults or errors. These could be due to erroneous input data or even malfunctioning neurons within the network (e.g., neurons with incorrect weights or biases).

Fault tolerance in MLPs can be achieved through a variety of methods:

1. **Redundancy:** Additional neurons or layers are introduced to the network. These extras can step in if certain neurons or layers fail, allowing the network to maintain performance. The training process needs to ensure the redundancy is effective.

2. **Robust Training Techniques:** Certain training techniques, like noise injection during training, can enhance the robustness of the network, making it more resistant to faults.

3. **Self-Repair Mechanisms:** Some networks implement mechanisms that allow them to detect and repair faults, such as adjusting the weights and biases of malfunctioning

neurons.

It is important to note that achieving high fault tolerance often requires a balance with other aspects of network performance and complexity. For instance, adding too much redundancy can make the network overly complex and could lead to problems such as overfitting.

4.7 Key Concepts in Model Training

Underfitting occurs when a model is too simple to capture the underlying structure of the data. The model has low variance but high bias. Mathematically, underfitting happens when the model error on the training set and the test set is high.

Overfitting happens when a model is excessively complex and starts to capture the noise in the data rather than the underlying structure. The model has low bias but high variance. Mathematically, overfitting occurs when the model error on the training set is low, but it's high on the test set.

Divergence in the context of training a neural network, refers to the situation where the error of the model on the training set keeps increasing during training instead of decreasing. This is often caused by an excessively high learning rate which results in the parameter updates overshooting the minimum of the loss function.

4.8 Vector Quantization

In the context of neural networks, Vector Quantization (VQ) is a process of mapping input vectors to a set of finite discrete values or classes represented by a unique "codebook" vector. These codebook vectors are learned during the training process.

A well-known type of neural network that utilizes VQ is the Self-Organizing Map (SOM), an unsupervised learning model.

The VQ process typically involves the following steps:

1. **Training**: During the training phase, the network learns a set of codebook vectors. These vectors serve as the reference points for quantization. In an SOM, the codebook vectors are the weights of the neurons.

2. **Quantization**: Each input vector is assigned to the closest codebook vector. The measure of "closeness" can be the Euclidean distance or any other distance measure.

Mathematically, the quantization process can be described as follows:

$$i = \arg\min_j \|\mathbf{x} - \mathbf{c}_j\| \tag{4.9}$$

where $\|\cdot\|$ denotes the distance measure (typically Euclidean distance), $\mathbf{x}$ is an input vector, $\mathbf{c}_j$ are the codebook vectors, and i is the index of the codebook vector that is closest to $\mathbf{x}$.

Chapter 5

Kohonen Self-Organising Network

5.1 Kohonen Self-Organizing Map Algorithm

The Kohonen Self-Organizing Map (SOM) is a type of neural network used for unsupervised learning. The algorithm is summarized as follows:

1. **Initialize** weights of neurons randomly.
2. **For** each training iteration:
   (a) **Present** a random input vector.
   (b) **Calculate** Euclidean distance from input vector to weight vector of each neuron.
   (c) **Identify** the Best Matching Unit (BMU), the neuron with the smallest distance to the input vector.
   (d) **Update** the weights of the BMU and its neighboring neurons to make them more similar to the input vector.
   (e) **Decrease** the learning rate and neighborhood radius over time.
3. **Repeat** the training iterations until convergence or the maximum number of iterations is reached.

This process maps the input vectors to a grid of neurons, preserving the topological properties of the input space in the grid.

Table 5.1: Comparison between Self-Organizing Neural Networks and Supervised Neural Networks

	Self-Organizing Networks	**Supervised Networks**
Learning Type	Unsupervised learning. No target output is provided.	Supervised learning. Target output is provided.
Training Data	Unlabeled data	Labeled data
Learning Process	The network learns the structure and patterns of the input data.	The network learns to map input data to given output.
Goal	Find hidden structure, clustering, or dimensionality reduction	Learn input-output mappings, classification or regression tasks
Example	Kohonen Self-Organizing Map (SOM)	Backpropagation, Support Vector Machine (SVM)
Adaptability	Adapts to find correlations and clusters in the data.	Adapts to minimize the difference between the actual and desired output.

5.2 Comparison between Self-Organizing Neural Networks and Supervised Neural Networks

5.3 Dimensionality Reduction

Dimensionality reduction refers to the process of converting data of very high dimensionality into data of much lower dimensionality such that each of the lower dimensions conveys much more information. This process is often necessary in applied machine learning, especially in situations where high-dimensional data causes problems (commonly referred to as the "curse of dimensionality"). Key points include:

Dimensionality Reduction:

- Simplifies high-dimensional data while retaining key information.
- Reduces computational cost and mitigates the risk of overfitting.
- Useful in visualizing high-dimensional data.

Self-Organizing Map (SOM) and Dimensionality Reduction:

- SOM is a type of artificial neural network that uses unsupervised learning to produce low-dimensional representation (typically two-dimensional) of high-dimensional data.
- It achieves dimensionality reduction by preserving the topological properties of the input data.
- SOMs can project high-dimensional patterns onto a grid of neurons, each representing

a region of the input space, thus effectively reducing the dimensionality.

- Aims to simplify the dataset without losing key information.
- Techniques include Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), and t-SNE.
- Reduces computational complexity.
- Helps to eliminate redundant features and noise.

5.4 Significance of Large Radius in Kohonen Networks

In a Kohonen Network, or Self-Organizing Map (SOM), the radius of the neighborhood function plays a significant role in determining the learning characteristics of the network.

- A large radius means that a large number of neurons are adjusted for each input vector, which leads to:
  - **Global order:** At the beginning of the training, a large radius ensures a global order among the neurons because many neurons are updated simultaneously, thereby capturing the general features of the input data.
  - **Smoothing effect:** A large radius produces a smoothing effect as adjacent neurons will have similar weights due to shared updates, leading to smoother transitions in the SOM.
- However, it is common to decrease the radius over time, which controls the algorithm's behavior:
  - **Fine-tuning:** Decreasing the radius during training allows for fine-tuning of the neurons to specific features of the input data, which enhances the map's precision.
  - **Avoidance of over-generalization:** A smaller radius towards the end of training prevents over-generalization by ensuring only the most similar neurons to the input data are updated.

5.5 Vector Quantization

Vector quantization is a process where input vectors are grouped into clusters, each represented by a prototype vector (the weight vector of a neuron), effectively discretizing the input space.

- **Vector Quantization:**

  - In a neural network, it is a process that assigns a "prototype" vector from the weight vectors of the neurons to each input vector.
  - Each neuron's weight vector in the network can be considered a "code" or "prototype". When an input vector is presented, the neuron with the most similar weight vector (the "winner") is selected.

- **Vector Quantization in Kohonen Networks:**
  - The Kohonen Network uses vector quantization as part of its competitive learning rule.
  - The input vectors are grouped based on the similarity of their features. Each group is represented by a prototype vector (weight vector of a neuron), effectively quantizing the input space.
  - This helps in identifying patterns, clustering, and dimensionality reduction in the input data.

Chapter 6

Fuzzy Logic

6.1 Fuzzy Logic

Fuzzy logic is a reasoning system that mimics human decision making with approximate, not exact, solutions.

- Allows partial membership values
- Mimics human reasoning

Advantages:

- Tolerant of imprecise data
- Flexibility in decision-making
- Handles uncertainty effectively

Disadvantages:

- Not exact, only approximate
- Requires extensive computing hardware
- Difficult to construct rules
- Not ideal for simple tasks

6.1.1 Fuzzy Logic System Architecture

Crisp Input/Output:

- Exact input/output

- Clear cut data
- No ambiguity
- Ex: temp 30 deg Celsius

Fuzzy set

- non exact value
- refer range
- Ex: temp is hot

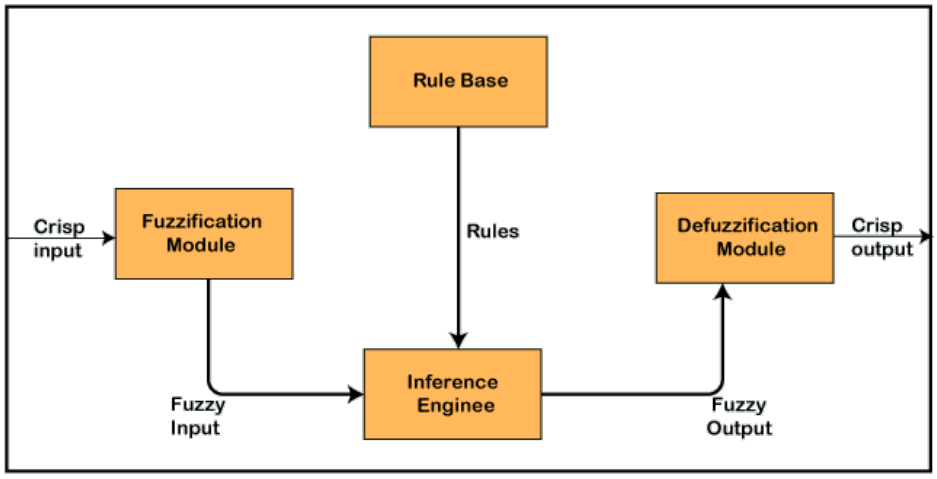


Figure 6.1: Illustration of Local vs Global Minima

6.2 Fuzzy Set Operations

- **Union:** Given two fuzzy sets A and B with membership functions $\mu_A(x)$ and $\mu_B(x)$ respectively, the union of A and B is a fuzzy set C with membership function $\mu_C(x) = \max\{\mu_A(x), \mu_B(x)\}$.
- **Intersection:** The intersection of A and B is a fuzzy set D with membership function $\mu_D(x) = \min\{\mu_A(x), \mu_B(x)\}$.
- **Complement:** The complement of a fuzzy set A is a fuzzy set E with membership function $\mu_E(x) = 1 - \mu_A(x)$.
- **Equality:** Two fuzzy sets A and B are considered equal if their membership functions are identical, i.e., $\mu_A(x) = \mu_B(x)$ for all x in the universe of discourse.

6.3 Fuzzy Logic Controller (FLC)

A Fuzzy Logic Controller (FLC) is a system that uses fuzzy logic to make decisions and control complex processes.

6.3.1 FLC Design: Scenireo

Consider a Fuzzy Logic Controller (FLC) for an air conditioning system that considers the temperature and humidity to adjust the cooling.

Inputs

- Temperature: Can be "Cold", "Comfortable", or "Hot"
- Humidity: Can be "Low", "Normal", or "High"

Output

- Air Conditioner Setting: Can be "Low", "Medium", or "High"

Rules

1. IF temperature is "Hot" AND humidity is "High" THEN air conditioner setting is "High"
2. IF temperature is "Comfortable" AND humidity is "Normal" THEN air conditioner setting is "Medium"
3. IF temperature is "Cold" AND humidity is "Low" THEN air conditioner setting is "Low"

Sample Dataset

Temperature	Humidity
Hot	High
Comfortable	Normal
Cold	Low

Table 6.1: Fuzzy Logic design scenario

With the given rules and dataset, the FLC will set the air conditioner to "High" when it's hot and humid, "Medium" when the temperature and humidity are comfortable and normal respectively, and "Low" when it's cold and the humidity is low.

6.3.2 FLC Design: Process

Step 1: Define Fuzzy Sets and Membership Functions
Here-

- T: Temperature
- H: Humidity
- AC: AC Setting

Variable	Set	Membership Function (Triangular)
T	Low	$\mu_{T_{Low}}(T) = \max(\min(1, 1 - T/50), 0)$
T	Medium	$\mu_{T_{Medium}}(T) = \begin{cases} \max(\min(1, (T-50)/25), 0) & \text{for } T \geq 50 \\ \max(\min(1, (100-T)/25), 0) & \text{for } T < 50 \end{cases}$
T	High	$\mu_{T_{High}}(T) = \max(\min(1, (T-50)/50), 0)$
H	Similar definitions as T	
AC	Similar definitions as T	

Step 2: Fuzzify Inputs

Assume T=80 and H=60, we get:

$$\mu_{T_{Low}}(80) = 0,$$
$$\mu_{T_{Medium}}(80) = 0.6,$$
$$\mu_{T_{High}}(80) = 1,$$
$$\mu_{H_{Low}}(60) = 0,$$
$$\mu_{H_{Medium}}(60) = 0.8,$$
$$\mu_{H_{High}}(60) = 0.2$$

Step 3: Inference Rules

Applying the rules to compute minima:

$$\text{Rule 1 (IF T High AND H High THEN A High)} : \min(1, 0.2) = 0.2$$
$$\text{Rule 2 (IF T Medium AND H Medium THEN A Medium)} : \min(0.6, 0.8) = 0.6$$
$$\text{Rule 3 (IF T Low AND H Low THEN A Low)} : \min(0, 0) = 0$$

Step 4: Defuzzification

Assume we are using the centroid method for defuzzification. When we combine the output sets and determine the centroid, let's say it's 80 (for simplicity). So, the FLC suggests AC setting to 80 (a high setting).