CHAPTER 4

4.1 Role of Deep Learning in Neural Network

Deep learning plays a central and transformative role in the field of neural networks. It is a specialized subset of machine learning that focuses on training deep neural networks with multiple hidden layers. Deep learning has been a game-changer for neural networks, unlocking their full potential and enabling them to tackle complex tasks with unprecedented accuracy.

4.2History of Neural Network

The history of neural networks spans several decades and involves contributions from various researchers and breakthroughs. Here's an overview of the key milestones in the history of neural networks:

1. McCulloch-Pitts Neuron (1943):

In 1943, Warren McCulloch and Walter Pitts proposed a mathematical model of a neuron, known as the McCulloch-Pitts (MCP) neuron. It was a binary threshold model that formed the foundation for early neural network research.

2. Perceptron (1957):

In 1957, Frank Rosenblatt introduced the perceptron, which was one of the earliest neural network models capable of learning from data. The perceptron was designed for binary classification tasks and was limited to linearly separable data.

3. Minsky-Papert Critique (1969):

In 1969, Marvin Minsky and Seymour Papert published the book "Perceptrons," in which they pointed out the limitations of single-layer perceptrons and their inability to solve problems that required nonlinear decision boundaries. This criticism led to a decline in interest and funding for neural networks research, known as the "AI winter."

4. Backpropagation Rediscovery (1986):

The backpropagation algorithm, a crucial technique for training neural networks with multiple layers, was independently rediscovered and extended by Geoffrey Hinton, David Rumelhart, and Ronald Williams in 1986. Backpropagation allowed neural networks to efficiently adjust their weights during training, overcoming the limitations of single-layer perceptrons.

5. Neural Network Renaissance (mid-1980s to late 1990s):

With the reinvigoration of interest in neural networks, researchers explored various network architectures, activation functions, and optimization techniques. This period saw the development of multi-layer neural networks and the application of neural networks in various domains, including pattern recognition and control systems.

6. Support Vector Machines Dominance (1990s to early 2000s):

Support Vector Machines (SVMs) gained popularity as a powerful alternative to neural networks for certain tasks, leading to reduced focus on neural networks during this time.

7. Deep Learning Resurgence (mid-2000s to present):

Around the mid-2000s, with the availability of large datasets and powerful GPUs, researchers began to unlock the potential of deep neural networks (neural networks with multiple hidden layers). Breakthroughs in deep learning, particularly in areas like image and speech recognition, occurred, leading to significant improvements in AI performance.

8. ImageNet Challenge (2012):

The ImageNet Large Scale Visual Recognition Challenge in 2012 marked a turning point for deep learning. A deep convolutional neural network called AlexNet, developed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, achieved a significant reduction in classification error compared to traditional methods. This victory demonstrated the power of deep learning and spurred its widespread adoption.

9. Continued Advancements:

Since the success of deep learning in the ImageNet Challenge, neural networks have continued to make significant strides in various domains, such as natural language processing, robotics, healthcare, and autonomous vehicles. Research in neural network architectures, regularization techniques, and optimization algorithms has been ongoing, leading to increasingly powerful and efficient models.

The history of neural networks is full of times of happiness and success, after periods of doubt and declining interest. However, with the advent of deep learning, neural networks have re-emerged as a major force in artificial intelligence and machine learning today.

4.3 Basics of Neural Network

1. Neurons and Layers:

A neural network is composed of interconnected nodes, known as neurons. Neurons are organized into layers, typically three types:

Input Layer: - Receives input data and passes it to the next layer.

Hidden Layers: - Intermediate layers between the input and output layers. They process information and learn patterns from the data.

Output Layer: - Produces the final output or prediction based on the information learned in the hidden layers.

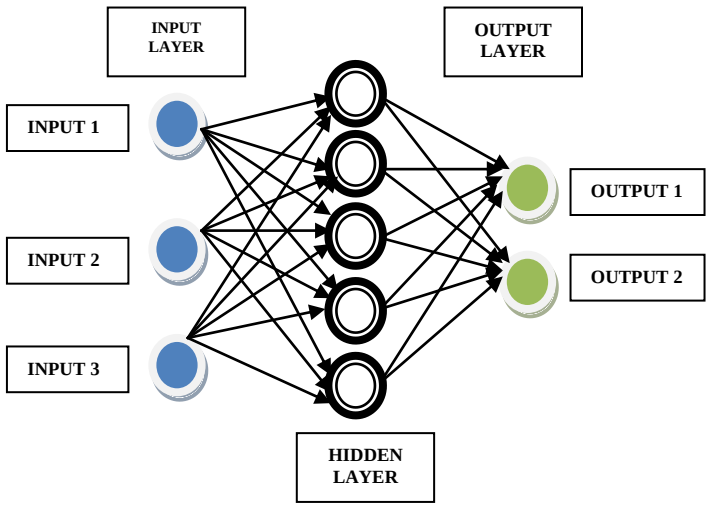


Figure 1: Structure of Neural Network

2. Weights and Biases:

Neurons in a layer are connected to neurons in the next layer through connections with associated weights and biases. The weights determine the strength of the connection between neurons, while biases adjust the output of each neuron.

Weights:

Weights are the parameters that control the strength of connections between neurons in different layers of a neural network. Each connection between two neurons has an associated weight that determines the influence of the input neuron on the output neuron. In a feedforward neural network, weights are learned during the training process to minimize the difference between the predicted output and the actual target output.

During the training process, the network adjusts these weights to improve its predictions. Weight updates are typically performed using optimization algorithms like gradient descent. The gradient of the loss function with respect to the weights indicates the direction and magnitude of weight adjustments needed to minimize the loss.

Biases:

Biases are the additional values added to the output of neurons before passing them through an activation function. They account for any shifts or offsets in the input data. While weights determine the strength of connections between neurons, biases control the neuron's activation threshold. Biases allow the neural network to model relationships that don't necessarily pass through the origin.

Biases are adjusted during training along with weights to improve the network's accuracy. The network learns the optimal bias values that help in fitting the data better and making accurate predictions.

Mathematical Representation:

In a single neuron, the output is calculated as the weighted sum of inputs plus a bias:

Output = Activation_Function(weighted_sum_of_inputs + bias)

In a multi-layer neural network, these calculations occur at each neuron in each layer.

Importance:

Weights and biases are the learnable parameters that enable neural networks to learn complex relationships and patterns in data. The proper adjustment of weights and biases through the training process is crucial for achieving accurate predictions and effective generalization to new data.

In summary, weights and biases are the core components that allow neural networks to adapt and learn from data. The learning process involves adjusting these parameters to minimize the discrepancy between predicted and actual outputs, enabling the network to make accurate predictions on unseen data.

3. Activation Function:

Each neuron has an activation function, which determines its output based on the weighted sum of its inputs plus the bias. Activation functions introduce non-linearity to the neural network, enabling it to learn complex relationships in the data.

An activation function is a key component in a neural network that introduces non-linearity into the model. It transforms the weighted sum of inputs and biases at a neuron into an output signal. Activation functions play a crucial role in enabling neural networks to learn and represent complex relationships within the data.

Some common activation functions used in neural networks:

I. **Sigmoid Function (Logistic Activation):** The sigmoid function maps input values to a range between 0 and 1. It's often used in the output layer of binary classification models because it can produce probabilities.

Formula: f(x) = 1 / (1 + e^(-x))

II. **Hyperbolic Tangent (Tanh) Function:** Similar to the sigmoid function, tanh also maps input values to a range between -1 and 1. It provides a stronger gradient for inputs compared to the sigmoid, making it helpful in training deep neural networks.

Formula: f(x) = (e^(x) - e^(-x)) / (e^(x) + e^(-x))

III. **Rectified Linear Unit (ReLU):** ReLU is widely used due to its simplicity and effectiveness. It outputs the input directly if it's positive, otherwise, it outputs zero. ReLU helps with addressing the vanishing gradient problem and speeds up the training process.

Formula: f(x) = max(0, x)

IV. **Leaky ReLU:** Leaky ReLU is a variation of ReLU that allows a small gradient when the input is negative, addressing the issue of "dying ReLU" where neurons can get stuck during training.

Formula: f(x) = x if x > 0, else ax (where 'a' is a small positive constant)

V. **Parametric ReLU (PReLU):** PReLU extends Leaky ReLU by allowing the coefficient 'a' to be learned during training. This introduces an extra degree of freedom for the model to adapt.

Formula: f(x) = x if x > 0, else ax

VI. **Exponential Linear Unit (ELU):** ELU is another alternative to ReLU that handles negative inputs by saturating them to a small negative value. ELU also addresses the vanishing gradient problem and can provide smoother gradients.

Formula: f(x) = x if x > 0, else a(e^x - 1) where 'a' is a positive constant

These activation functions enable neural networks to model complex relationships by introducing non-linearity. The choice of activation function depends on the specific problem, architecture of the network, and consideration of issues like vanishing gradients or dead neurons.

4. Feedforward and Backpropagation:

In feedforward neural networks, data flows through the network in one direction, from the input layer to the output layer. During training, the network adjusts the weights and biases using a

process called backpropagation, which involves computing gradients and updating parameters to minimize the error between predicted outputs and actual labels.

Feedforward and backpropagation are two fundamental concepts in training neural networks. They are key processes that enable a neural network to learn from data and improve its performance over time.

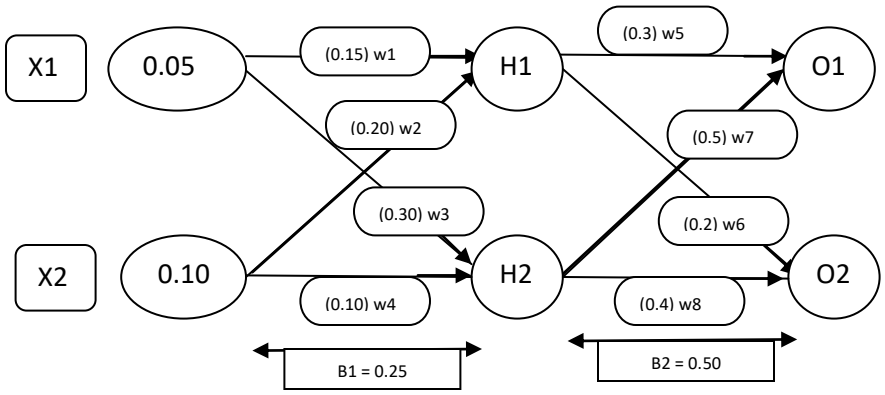
Feedforward:

Feedforward is the process of passing input data through the layers of a neural network to generate an output prediction. In a feedforward neural network (also known as a multilayer perceptron), data flows from the input layer through one or more hidden layers to the output layer. Each neuron in a layer calculates a weighted sum of its inputs, adds a bias, and passes the result through an activation function to produce an output.\

The feedforward process can be summarized as follows:

1. Input data is fed into the input layer.
2. Data is propagated through hidden layers by calculating weighted sums and applying activation functions.
3. The output layer produces the final prediction.

EXAMPLE Calculate two outputs for the feedforward neural network by using sigmoid function



Given Inputs X1: 0.05

X2: 0.10

There are total 8 Weights

Given Hidden Layer H1 and H2 and Two Output O1 and O2

First of all We have to calculate the Weighted term

The formula used is ∑(X1W1) +(X2W2)....+(XnWn)

Input for 1st hidden layer is dented by h1(in) = ∑ (X1)*(W1) +(X2)*(W2)+(B1)

Formula used for the weighted term ∑(0.05)*(0.15)+(0.10)*(0.20)+(0.25) = 0.2775

Output for hidden layer h1 is denoted by h1(out) so the formula used for calculating h1 (out) is = $1/1+e^{h1(in)}$ which is a sigmoid activation function

= $1/1+e^{0.2775}$

= 0.5689332093695

H2(in) = ∑(X1W3)+(X2W4)+(B1),

=(0.05)*(0.30)+(0.10)*(0.10)+(0.25)

=0.275

H2(out)= $1/1+e^{h2(in)}$

= $1/1+e^{0.275}$

=0.5683199

O1(in) = h1(out)*(w5)+h2(out)*(w7)+(b2)

=(0.5689332093695)*(0.3)+(0.5683199)*(0.5)+(0.50)

=0.95483985

$O1(out) = 1/1+e^{-O1(in)}$

$=1/1+e^{-0.95483985}$

=0.7220877

O2(in) = h1(out)*(w6)+h2(out)*(w8)+(b2)

=(0.5689332093695)*(0.2)+(0.5683199)*(0.4)+(0.50)

=0.90262156

$O2(out)= 1/1+e^{-O2(in)}$

= 0.7114879

Final Output: O1(out)+O2(out)

= 0.7220877+0.7114879

= 1.4335756

Backpropagation:

Backpropagation is the process of updating the weights and biases of a neural network based on the difference between the predicted output and the actual target output. It's a training algorithm that aims to minimize the error (or loss) between predictions and targets.

The backpropagation process involves the following steps:

1. Forward Pass: The input data is fed through the network to compute the predicted output.
2. Calculate Loss: The difference between the predicted output and the actual target output is calculated using a loss function.
3. Backward Pass: The gradient of the loss with respect to the weights and biases is calculated layer by layer, starting from the output layer and moving backward through the network.

4. Weight Update: The calculated gradients are used to update the weights and biases in the network. This step is performed using optimization algorithms like gradient descent or its variants.

Backpropagation iteratively refines the network's weights and biases to reduce the error between predictions and targets. This process continues for multiple iterations (epochs) until the network converges to a point where the loss is minimized and predictions are accurate.

5. Loss Function:

The loss function measures the difference between the predicted outputs and the true labels. The goal during training is to minimize this loss function, effectively improving the network's ability to make accurate predictions.

A loss function (also known as a cost function or objective function) is a mathematical function that measures the difference between the predicted output of a neural network and the actual target output (ground truth). The primary goal of a neural network is to minimize the value of the loss function, which indicates how well the network's predictions align with the true values.

Loss functions play a crucial role in training neural networks by providing a quantitative measure of the network's performance. During the training process, the network adjusts its weights and biases to minimize the loss function, resulting in better predictions over time.

Different types of problems (e.g., classification, regression) and network architectures might require different loss functions. Here are some common types of loss functions used in neural networks:

a. Mean Squared Error (MSE):

MSE is commonly used for regression problems, where the goal is to predict continuous values. It calculates the average of the squared differences between predicted and target values.

Formula: MSE = (1/n) Σ(y_pred - y_true)^2

b. Binary Cross-Entropy (Log Loss):

Binary cross-entropy is used for binary classification problems, where the network predicts one of two classes. It measures the dissimilarity between the predicted probability distribution and the true distribution.

Formula: BCE = - Σ(y_true * log(y_pred) + (1 - y_true) * log(1 - y_pred))

c. Categorical Cross-Entropy (Softmax Loss):

Categorical cross-entropy is used for multi-class classification problems. It calculates the dissimilarity between the predicted class probabilities and the true one-hot encoded class labels.

Formula: CCE = - Σ(y_true * log(y_pred))

d. Hinge Loss (SVM Loss):

Hinge loss is used in support vector machines (SVMs) and is also applicable to neural networks for classification problems. It encourages correct classification while penalizing predictions that are far from the true class.

Formula: Hinge Loss = max(0, 1 - y_true * y_pred)

e. Huber Loss:

Huber loss is a hybrid of the mean squared error and the mean absolute error. It's less sensitive to outliers compared to MSE.

Formula: Huber Loss = { 0.5 * (y_pred - y_true)^2, if |y_pred - y_true| <= delta; delta * |y_pred - y_true| - 0.5 * delta^2, otherwise }

These are just a few examples of loss functions, and there are others tailored for specific tasks and scenarios. The choice of the appropriate loss function depends on the nature of the problem and the desired behavior of the network during training.

6. Training:

The training process involves iteratively presenting training data to the network, computing predictions, calculating the loss, and updating the weights and biases through backpropagation.

Training continues until the network's performance reaches a satisfactory level or a predefined number of epochs.

Training a neural network involves the process of iteratively adjusting its weights and biases based on the provided input data and target outputs (ground truth). The goal is to minimize the difference between the network's predictions and the actual targets. This process is crucial for the network to learn meaningful patterns and relationships within the data.

Here are the key steps involved in training a neural network:

I. Initialization: Initialize the weights and biases of the network with small random values. Proper initialization can speed up convergence and prevent the network from getting stuck in poor solutions.

II. Forward Pass: Perform a forward pass by feeding the input data through the network. Calculate the predicted output for each input by applying the weights and biases and passing the results through activation functions.

III. Loss Calculation: Calculate the loss (error) between the predicted output and the actual target output using a chosen loss function. The loss represents the discrepancy between the network's predictions and the true values.

IV. Backpropagation: Perform backpropagation to compute the gradient of the loss with respect to the weights and biases. This involves calculating the derivative of the loss function with respect to each parameter in the network. Backpropagation efficiently distributes the error gradient back through the layers of the network.

V. Weight Update: Update the weights and biases using an optimization algorithm, such as gradient descent or one of its variants. The optimization algorithm adjusts the parameters in the direction that reduces the loss. The learning rate determines the step size of these updates.

VI. Iterative Process: Repeat the forward pass, loss calculation, backpropagation, and weight update for multiple iterations or epochs. Each iteration helps the network refine its predictions and minimize the loss.

VII. Validation: Monitor the network's performance on a validation dataset during training to prevent overfitting. Overfitting occurs when the network memorizes the training data but performs poorly on new data.

VIII. Early Stopping: If the validation loss starts to increase after a certain number of epochs, stop training to prevent overfitting. This is known as early stopping.

IX. Testing: Once the network has been trained, evaluate its performance on a separate test dataset that it has never seen before. This gives an estimate of how well the network generalizes to new, unseen data.

X. Fine-Tuning: Based on the performance on the test dataset, you might need to fine-tune hyperparameters or adjust the model architecture to improve results.

Training a neural network is an iterative and often computationally intensive process. The choice of hyperparameters, loss function, architecture, and optimization algorithm all influence the training process and the quality of the learned model. It's important to strike a balance between training for too long (overfitting) and stopping too soon (underfitting).

6. Overfitting and Regularization:

Neural networks can memorize training data, leading to poor generalization to new data, a phenomenon known as overfitting. Regularization techniques, such as L1 and L2 regularization or dropout, are used to prevent overfitting by adding penalties to the loss function based on the complexity of the model.

Overfitting:

Overfitting occurs when a neural network learns to perform extremely well on the training data but fails to generalize to new, unseen data. In other words, the network memorizes the training examples and noise in the data rather than learning the underlying patterns. As a result, it performs poorly on data it hasn't seen before. Overfitting is a common issue in machine learning and neural networks, and it can lead to poor performance and lack of robustness.

Causes of Overfitting:

1. Model Complexity: If the neural network is too complex relative to the size of the dataset, it can capture noise and outliers in the training data.

2. Insufficient Data: When the amount of training data is limited, the network may learn the data by heart rather than generalizing.

3. Training Duration: Training for too many epochs can lead to overfitting as the model fits the noise in the training data.

4. Noise in Data: Noisy data points can introduce randomness that the network may try to capture.

Regularization:

Regularization is a set of techniques used to prevent or mitigate overfitting by adding constraints or penalties to the training process. The goal of regularization is to encourage the network to learn only the most important patterns in the data, rather than fitting noise.

Common Regularization Techniques:

A. L1 and L2 Regularization: These techniques add a penalty term to the loss function based on the magnitudes of the weights. L1 regularization encourages sparsity by adding the sum of absolute values of weights, while L2 regularization encourages small weights by adding the sum of squared values of weights.

B. Dropout: Dropout randomly "drops out" (sets to zero) a fraction of neurons during each training iteration. This helps to prevent co-adaptation of neurons and encourages the network to rely on multiple paths to make predictions.

C. Early Stopping: As training progresses, the validation loss may start to increase due to overfitting. Early stopping involves monitoring the validation loss and stopping training when the loss begins to rise.

D. Data Augmentation: Increasing the size of the training dataset through techniques like rotating, cropping, or adding noise to images can help prevent overfitting.

E. Batch Normalization: Batch normalization normalizes the activations of neurons within a layer, making the network more stable during training and preventing overfitting.

F. Weight Constraints: Limiting the magnitude of weights can prevent the network from learning overly complex patterns.

Regularization techniques strike a balance between fitting the training data well and ensuring the network generalizes to new data. It's important to choose the appropriate regularization techniques based on the specific problem, dataset, and model architecture.

7. Hyperparameters:

Neural networks have various hyperparameters that need to be set before training, such as the number of hidden layers, the number of neurons in each layer, the learning rate, batch size, and more. The optimal values of these hyperparameters can significantly impact the network's performance.

Hyperparameters are parameters that are set before the training of a neural network begins. They are not learned from the data during the training process; rather, they influence how the training process takes place and how the network behaves. Choosing appropriate hyperparameters is crucial for achieving good performance and preventing issues like overfitting or slow convergence.

Here are some common hyperparameters in a neural network:

A. Learning Rate: The learning rate determines the step size at which the weights and biases are updated during each iteration of the optimization algorithm (e.g., gradient descent). A higher learning rate might cause the optimization to diverge, while a very low learning rate can lead to slow convergence.

B. Number of Epochs: The number of epochs specifies how many times the entire training dataset is passed through the neural network during training. Too few epochs might lead to underfitting, while too many epochs might lead to overfitting.

C. Batch Size: The batch size determines the number of training examples used in each iteration of the optimization algorithm. A larger batch size can speed up training, but too large a batch can consume excessive memory and slow down convergence.

D. Activation Functions: The choice of activation functions for hidden layers and the output layer can significantly impact the learning process. Common choices include ReLU, sigmoid, and tanh.

E. Network Architecture: The number of layers and the number of neurons in each layer are important architectural decisions. Deeper networks can capture more complex patterns, but they might require more data and computational resources.

F. Regularization Parameters: Parameters for regularization techniques like L1 or L2 regularization control the strength of the regularization effect.

G. Dropout Rate: In dropout, a fraction of neurons is randomly dropped out during each iteration. The dropout rate determines the probability of dropout.

H. Optimizer Choice: Different optimization algorithms (e.g., SGD, Adam, RMSProp) have different characteristics in terms of convergence speed and robustness.

I. Weight Initialization: The initial weights can affect the training process and convergence. Proper weight initialization can help prevent slow convergence or vanishing gradients.

J. Loss Function: The choice of loss function depends on the type of task (classification, regression) and the desired behaviour of the network.

K. Learning Rate Schedule: Instead of using a fixed learning rate, a schedule can be used to adjust the learning rate during training. Learning rate decay or adaptive learning rates can be used.

Hyperparameter tuning involves selecting the right values for these parameters to achieve optimal performance on the validation dataset. This process often requires experimentation, trial, and error. Techniques like grid search, random search, or more advanced methods like Bayesian optimization can be used to find the best hyperparameters efficiently.

Neural networks are incredibly versatile and powerful models, capable of learning complex patterns from data and performing various tasks, such as classification, regression, and sequence generation. However, designing an effective neural network requires careful consideration of the architecture, activation functions, and hyperparameters based on the specific problem at hand.

4.3.1 Here are some key roles of deep learning in neural networks:

1. **Representation Learning: -** Deep learning excels at automatically learning useful representations of data from raw input. In traditional neural networks with shallow architectures, the learned representations might be limited in their expressiveness. However, the depth of deep neural networks allows them to learn hierarchical representations, capturing abstract and complex patterns in the data.
2. **Solving Complex Problems: -** Deep learning has demonstrated remarkable success in solving complex problems that were previously challenging or infeasible for traditional methods. Tasks like image and speech recognition, natural language processing, autonomous driving, and game playing have seen significant advancements due to deep learning.
3. **Feature Engineering Automation: -** In traditional machine learning, feature engineering requires domain expertise to manually design relevant features from raw data. Deep learning automates this process to a large extent by automatically learning high-level features from the data, reducing the need for manual feature engineering.
4. **Handling Large-Scale Data: -** Deep learning algorithms are designed to handle large-scale datasets efficiently. Thanks to advancements in parallel computing and the availability of powerful GPUs and TPUs, deep neural networks can process massive amounts of data in a reasonable time frame.
5. **Transfer Learning: -** Deep learning models can be pre-trained on large, diverse datasets and then fine-tuned for specific tasks with limited data. This transfer learning approach allows models to benefit from knowledge learned in one domain to improve performance in related domains, saving time and resources.
6. **Architectural Innovations: -** Deep learning has brought about architectural innovations like convolution neural networks (CNNs) for image-related tasks and recurrent neural networks (RNNs) for sequential data. These specialized architectures leverage the hierarchical nature of deep networks to excel in specific tasks.
7. **Natural Language Processing Advancements: -** Deep learning has revolutionized natural language processing (NLP) by enabling the use of recurrent and transformer-based architectures like LSTM (Long Short-Term Memory) and BERT (Bidirectional Encoder Representations from Transformers). These models have greatly improved the

performance of tasks such as language translation, sentiment analysis, and question-answering.

8. **Generative Models: -** Deep learning has led to the development of generative models, such as Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs). These models are capable of generating new, realistic data samples, which have applications in art, data augmentation, and synthetic data generation.

Deep learning has revolutionized the capabilities of neural networks, making them the go-to approach for many AI tasks. Its ability to automatically learn hierarchical representations from data, handle large datasets, and solve complex problems has made deep learning a driving force behind many cutting-edge AI applications and research breakthroughs.

4.4 Role of ANN (Artificial Neural Network) in AI

The term "artificial neural network" is derived from a biological neural network that develops the structure of a human brain. Just as the human brain is made up of interconnected neurons, neural networks are made up of interconnected neurons in different layers of the network. These neurons are called nodes.

It is a class of algorithms originating from the functioning of the human brain and nervous system. Neural networks have proven to be very powerful and can be used for image and speech recognition, natural language processing, games, etc. It has achieved great success in many artificial intelligence applications.

Or a neural network is a model of artificial intelligence and machine learning inspired by the structure and function of the human brain. It is a computer made up of interconnected systems called as (neurons). Each neuron takes input, performs mathematical operations on it, and produces output. Neural networks can learn from data and adjust for inconsistencies (weights and biases) for accuracy or decision-making purposes for a variety of tasks such as image recognition, natural language processing, and pattern analysis. The term "neural" comes from the idea that these networks are loosely modeled after the biological ways neurons transmit and process information in the brain.

A simple example of a neural network for binary classification.

In this example, we'll build a neural network to classify whether a person is likely to have diabetes or not based on two input features: glucose level and body mass index (BMI).

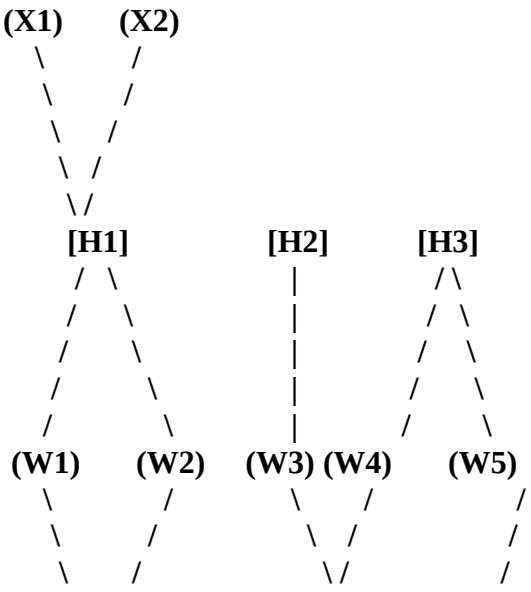
Dataset:

Suppose we have the following training dataset with labeled examples:

Glucose Level (X1)	BMI (X2)	Diabetes (Label)
120	25	0
150	30	1
100	22	0
180	35	1

4.4.1 Architecture of Neural Network:

We'll create a simple neural network with one hidden layer containing three neurons.



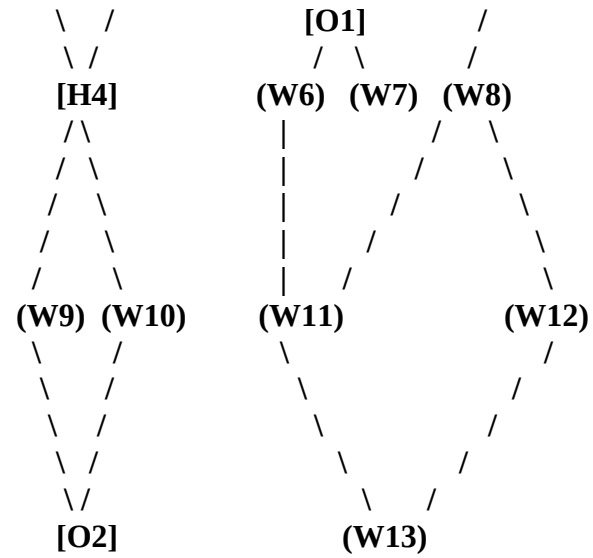


Figure: 2 Neural Network with one hidden layer

Equations:

Hidden Layer Neurons (H1, H2, H3):

$$H1 = \text{sigmoid}(W1 * X1 + W2 * X2)$$

$$H2 = \text{sigmoid}(W3 * X1 + W4 * X2)$$

$$H3 = \text{sigmoid}(W5 * X1 + W6 * X2)$$

Output Layer Neuron (O1):

$$O1 = \text{sigmoid}(W7 * H1 + W8 * H2 + W9 * H3)$$

Output Layer Neuron (O2):

$$O2 = \text{sigmoid}(W10 * H1 + W11 * H2 + W12 * H3)$$

In this example, the neural network takes input features (X1, X2), processes them through the hidden layer with three neurons (H1, H2, H3), and produces two outputs (O1, O2) after processing through the output layer. The sigmoid activation function squashes the values between 0 and 1, making it suitable for binary classification problems.

The neural network will be trained using backpropagation and an optimization algorithm (e.g., gradient descent) to minimize the prediction errors and classify new individuals as either having diabetes (1) or not having diabetes (0) based on their glucose levels and BMI.