CHAPTER I

THE STUDY AND ITS PROBLEMS

A. BACKGROUND OF THE STUDY

Computer scientists have long been inspired by the human brain. In 1943, Warren S. McCulloch, a neuroscientist, and Walter Pitts, a logician, developed the first conceptual model of an artificial neural network. In their paper, "A logical calculus of the ideas imminent in nervous activity,” they describe the concept of a neuron, a single cell living in a network of cells that receives inputs, processes those inputs, and generates an output.

Their work, and the work of many scientists and researchers that followed, was not meant to accurately describe how the biological brain works. Rather, an artificial neural network (which we will now simply refer to as a “neural network”) was designed as a computational model based on the brain to solve certain kinds of problems.

It’s probably pretty obvious to you that there are problems that are incredibly simple for a computer to solve, but difficult for you. Take the square root of 964,324, for example. A quick line of code produces the value 982, a number Processing computed in less than a millisecond. There are, on the other hand, problems that are incredibly simple for you or me to solve, but not so easy for a computer. Show any toddler a picture of a kitten or puppy and they’ll be able to tell you very quickly which one is which. Say hello and shake my hand one morning and you should be able to pick me out of a crowd of people the next day. But need a machine to perform one of these tasks? Scientists have already spent entire careers researching and implementing complex solutions.

The most common application of neural networks in computing today is to perform one of these “easy-for-a-human, difficult-for-a-machine” tasks, often referred to as pattern recognition. Applications range from optical character recognition (turning printed or handwritten scans into digital text) to facial recognition.

A neural network is a “connectionist” computational system. The computational systems we write are procedural; a program starts at the first line of code, executes it, and goes on to the next, following instructions in a linear fashion. A true neural network does not follow a linear path. Rather, information is processed collectively, in parallel throughout a network of nodes (the nodes, in this case, being neurons).

One of the key elements of a neural network is its ability to learn. A neural network is not just a complex system, but a complex adaptive system, meaning it can change its internal structure based on the information flowing through it. Typically, this is achieved through the adjusting of weights. Each connection between two neurons indicates the pathway for the flow of information. Each connection has a weight, a number that controls the signal between the two neurons. If the network generates a “good” output (which we’ll define later), there is no need to adjust the weights. However, if the network generates a “poor” output—an error, so to speak—then the system adapts, altering the weights in order to improve subsequent results.

There are several strategies for learning, here are some of them:

Supervised Learning — Essentially, a strategy that involves a teacher that is smarter than the network itself. For example, take the facial recognition example. The teacher shows the network a bunch of faces, and the teacher already knows the name associated with each face. The network makes its guesses, then the teacher provides the network with the answers. The network can then compare its answers to the known “correct” ones and make adjustments according to its errors.

Unsupervised Learning — Required when there isn’t an example data set with known answers. Imagine searching for a hidden pattern in a data set. An application of this is clustering, i.e. dividing a set of elements into groups according to some unknown pattern.

Reinforcement Learning — A strategy built on observation. Think of a little mouse running through a maze. If it turns left, it gets a piece of cheese; if it turns right, it receives a little shock. (Don’t worry, this is just a pretend mouse.) Presumably, the mouse will learn over time to turn left. Its neural network makes a decision with an outcome (turn left or right) and observes its environment (yum or ouch). If the observation is negative, the network can adjust its weights in order to make a different decision the next time. Reinforcement learning is common in robotics.

This ability of a neural network to learn, to make adjustments to its structure over time, is what makes it so useful in the field of artificial intelligence.

Here are some standard uses of neural networks in software today:

Pattern Recognition — We've mentioned this several times already and it's probably the most common application. Examples are facial recognition, optical character recognition, etc.

Time Series Prediction — Neural networks can be used to make predictions. Will the stock rise or fall tomorrow? Will it rain or be sunny?

Signal Processing — Cochlear implants and hearing aids need to filter out unnecessary noise and amplify the important sounds. Neural networks can be trained to process an audio signal and filter it appropriately.

Control — Neural networks are often used to manage steering decisions of physical vehicles (or simulated ones).

Soft Sensors — A soft sensor refers to the process of analysing a collection of many measurements. A thermometer can tell you the temperature of the air, but what if you also knew the humidity, barometric pressure, dew point, air quality, air density, etc.? Neural networks can be employed to process the input data from many individual sensors and evaluate them as a whole.

Anomaly Detection —Because neural networks are so good at recognizing patterns, they can also be trained to generate an output when something occurs that doesn't fit the pattern. Think of a neural network monitoring your daily routine over a long period of time. After learning the patterns of your behaviour, it could alert you when something is amiss.

The computing world has a lot to gain from neural networks. Their ability to learn by example makes them very flexible and powerful. Furthermore, there is no need to devise an algorithm in order to perform a specific task; i.e. there is no need to understand the internal mechanisms of that task. They are also very well suited for real time systems because of their fast response and computational times which are due to their parallel architecture.

B. ALGORITHM

A short discussion about Deep Feedforward Neural Networks:

A feedforward neural network is a collection of “neurons” with “synapses” connecting them. The collection is organized into three main parts: the input layer, the hidden layer, and the output layer. Note that you can have n hidden layers, with the term “deep” learning implying multiple hidden layers. The term “deep” learning came from having many hidden layers. These layers are known as “hidden”, since they are not visible as a network output.

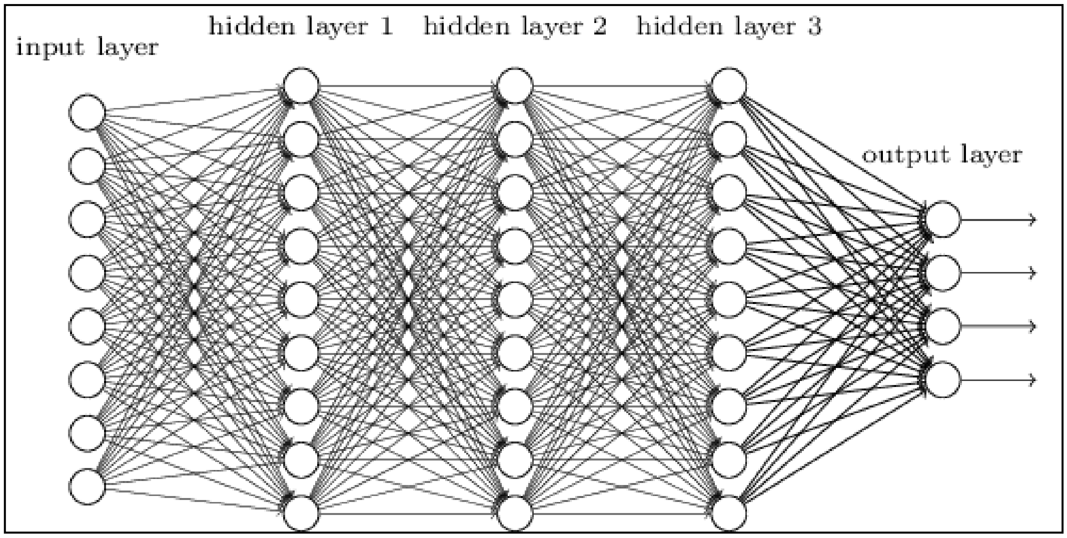


Figure 1: A Deep Neural Network

The circles represent neurons and lines represent synapses. Synapses take the input and multiply it by a “weight”. Neurons add the outputs from all synapses and apply an activation function. Training a neural network basically means calibrating all of the “weights” by repeating two key steps, *forward propagation* and *back propagation*.

Deep Feedforward Neural Networks Flowchart:

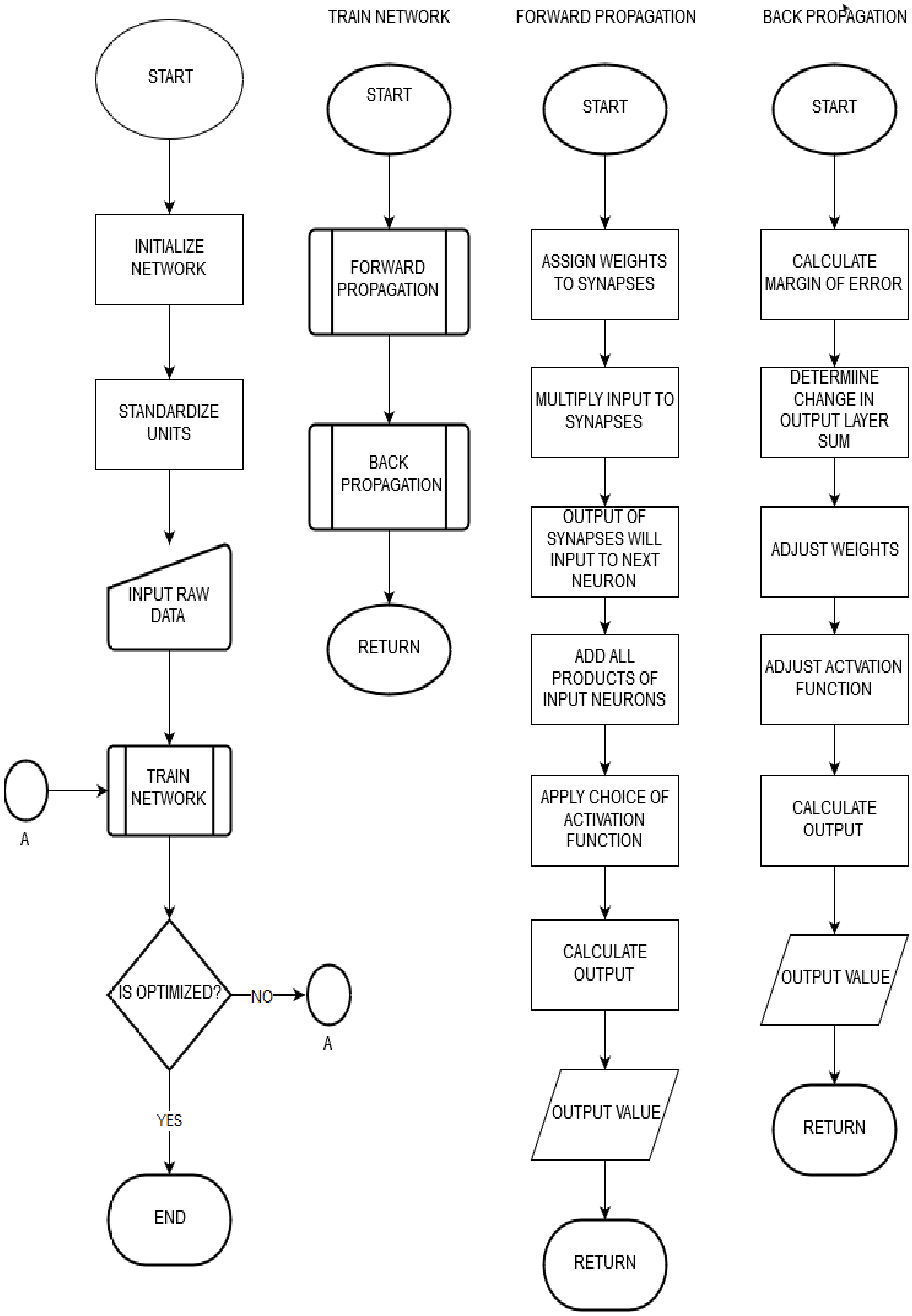


Figure 2:Deep Feedforward Neural Network Algorithm

Backpropagation Algorithm:

Phase 1: Propagation

Each propagation involves the following steps:

Step 1. Forward propagation of a training pattern's input through the neural network in order to generate the propagation's output activations.

Step 2. Backward propagation of the propagation's output activations through the neural network using the training pattern target in order to generate the deltas (the difference between the targeted and actual output values) of all output and hidden neurons.

Phase 2: Weight update

For each weight-synapse follow the following steps:

Step 1. Multiply its output delta and input activation to get the gradient of the weight.

Step 2. Denote m input values.

Step 3. Initialize each of the m inputs (synapses) with a weight w1, w2, ...,

Step 4. The input values are multiplied by their weights and summed.

Step 5. The output is some function y = f(v) of the weighted sum.

Step 6. The output of a neuron (y) is a function of the weighted sum.

Step 7. Apply activation function.

Step 8. Subtract a ratio (percentage) from the gradient of the weight.

The ratio (percentage) influences the speed and quality of learning; it is called the learning rate. The greater the ratio, the faster the neuron trains; the lower the ratio, the more accurate the training is. The sign of the gradient of a weight indicates where the error is increasing, this is why the weight must be updated in the opposite direction.

Repeat phase 1 and 2 until the performance of the network is satisfactory.

Pseudocode of Backpropagation Algorithm:

```
initialize network weights (often small random values)
  do
     forEach training example named ex
       prediction = neural-net-output(network, ex)
       actual = teacher-output(ex)
       compute error (prediction - actual) at the output units
       compute Delta w_h for all weights from hidden layer to output layer
       compute Delta w_i for all weights from input layer to hidden layer
       update network weights
  until all examples classified correctly or another stopping criterion satisfied
  return the network
```

The lines "compute Delta w_h for all weights from hidden layer to output layer" && "compute Delta w_i for all weights from input layer to hidden layer" can be implemented using the backpropagation algorithm, which calculates the gradient of the error of the network regarding the network's modifiable weights. Often the term "backpropagation" is used in a more general sense, to refer to the entire procedure encompassing both the calculation of the gradient and its use in stochastic gradient descent, but backpropagation proper can be used with any gradient-based optimizer, such as L-BFGS or truncated Newton.

Backpropagation networks are necessarily multilayer perceptrons (usually with one input, multiple hidden, and one output layer). In order for the hidden layer to serve any useful function, multilayer networks must have non-linear activation functions for the multiple layers: a multilayer network using only linear activation functions is equivalent to some single layer, linear network. Non-linear activation functions that are commonly used include the rectifier, logistic function, the softmax function, and the gaussian function.

C. STATEMENT OF PROBLEMS

1.) Problem 1: Algorithm consumes computing resources exponentially when calculating the local minimum cost value of a given set with a large number of elements.

Discussion:

When training a neural network in order to get accurate results trainers need to minimize the cost function. Given the fact that trainers have a very limited way to manipulate data directly, we need to rely on adjusting the weights and finding the optimized weight in a given set for a specific synapse manually. For example, we have one weight and we need to test it to get the smallest possible value in a set of one thousand. As expected, we can get the value almost instantly when simulated in a computer. But as weights increase in relative to the number of synapses of the network, so as the sets with its test elements, thus increasing the dimensions for optimizing the local minimum exponentially.

Location:

This problem resides on the Neural Network Training process, specifically inside the Back Propagation sub process. This issue can be clearly pinpointed when we adjust the weights from a previously accepted input from the output layer after calculating the margin of error and testing the accuracy of the neural network.

Algorithm:

Phase 1: Propagation

Each propagation involves the following steps:

Step 1. Forward propagation of a training pattern's input through the neural network in order to generate the propagation's output activations.

Step 2. Backward propagation of the propagation's output activations through the neural network using the training pattern target in order to generate the deltas (the difference between the targeted and actual output values) of all output and hidden neurons.

Phase 2: Weight update

For each weight-synapse follow the following steps:

Step 1. Multiply its output delta and input activation to get the gradient of the weight.

Step 2. Denote m input values.

Step 3. Initialize each of the m inputs (synapses) with a weight w1, w2, ...,

Step 4. The input values are multiplied by their weights and summed.

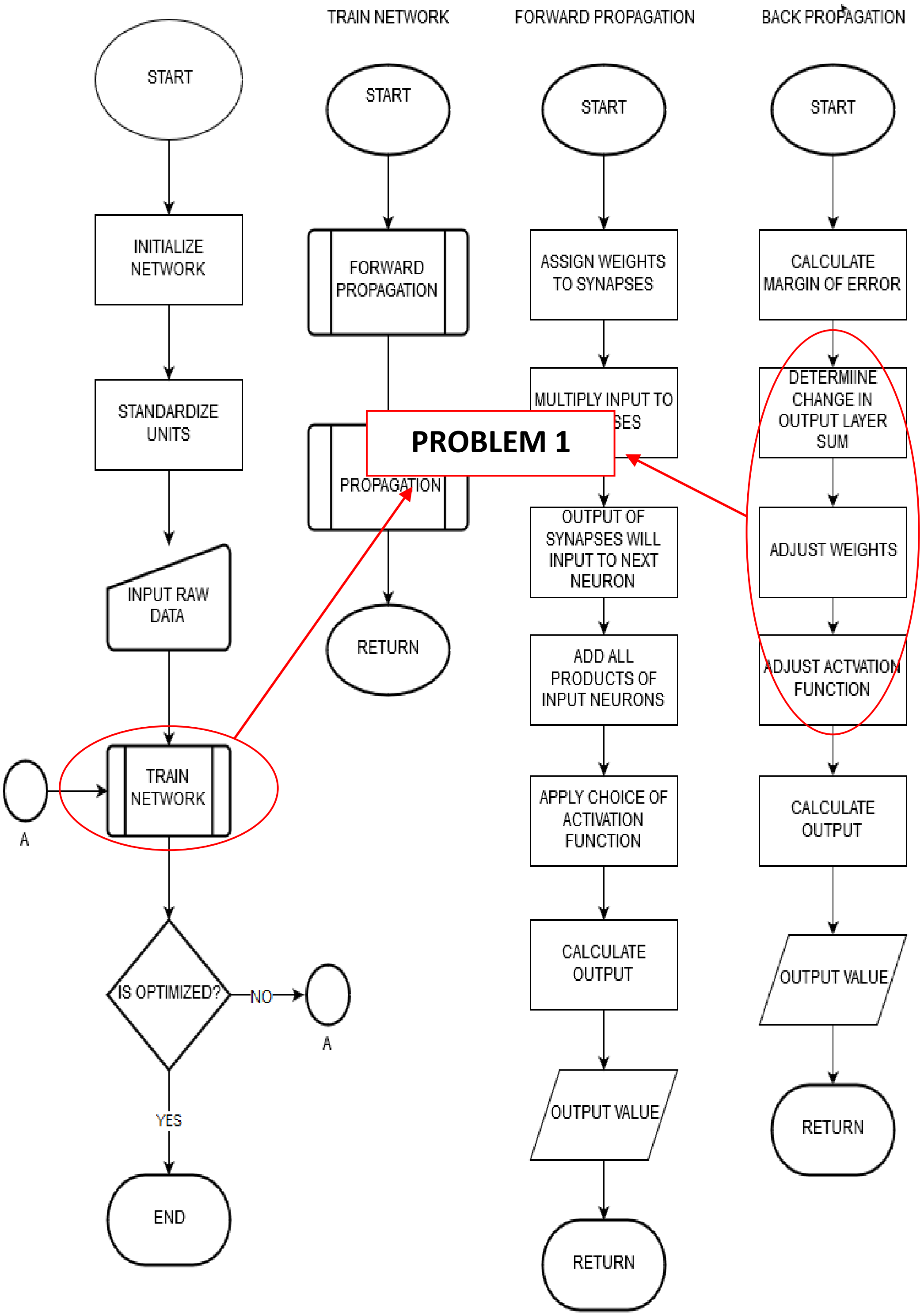
Step 5. The output is some function y = f(v) of the weighted sum.

Step 6. The output of a neuron (y) is a function of the weighted sum.

Step 7. Apply activation function.

Step 8. Subtract a ratio (percentage) from the gradient of the weight.

Illustrated Location:



Simulation:

Number of Weights	Dimensions	Number of Test Elements	Time (in seconds)
9	1	1000	0.041620999999
9	2	1000*1000	41.213241
9	3	1000*1000*1000	39600.43712

2.) Problem 2: Algorithm describes or classifies random error or "noise" when fitting a model to a set of training data resulting to poor generalization.

Discussion:

One of the problems that occur during neural network training is called overfitting. The error on the training set is driven to a very small value, but when new data is presented to the network the error is large. The network has memorized the training examples, but it has not learned to generalize to new situations.

Deep neural networks contain multiple non-linear hidden layers and this makes them very expressive models that can learn very complicated relationships between their inputs and outputs. With limited training data, however, many of these complicated relationships will be the result of sampling noise, so they will exist in the training set but not in real test data even if it is drawn from the same distribution which leads to overfitting.

Location:

This problem resides at the actual inputting a set of training data and fitting it to the network. This issue is observable when we compare network trained data to real world data and analysing the marginal error along with the non-linear or noise data that are not directly mapped into the training set.

Algorithm:

Phase 1: Propagation

Each propagation involves the following steps:

Step 1. Forward propagation of a training pattern's input through the neural network in order to generate the propagation's output activations.

Step 2. Backward propagation of the propagation's output activations through the neural network using the training pattern target in order to generate the deltas (the difference between the targeted and actual output values) of all output and hidden neurons.

Phase 2: Weight update

For each weight-synapse follow the following steps:

Step 1. Multiply its output delta and input activation to get the gradient of the weight.

Step 2. Denote m input values.

Step 3. Initialize each of the m inputs (synapses) with a weight.

Step 4. The input values are multiplied by their weights and summed.

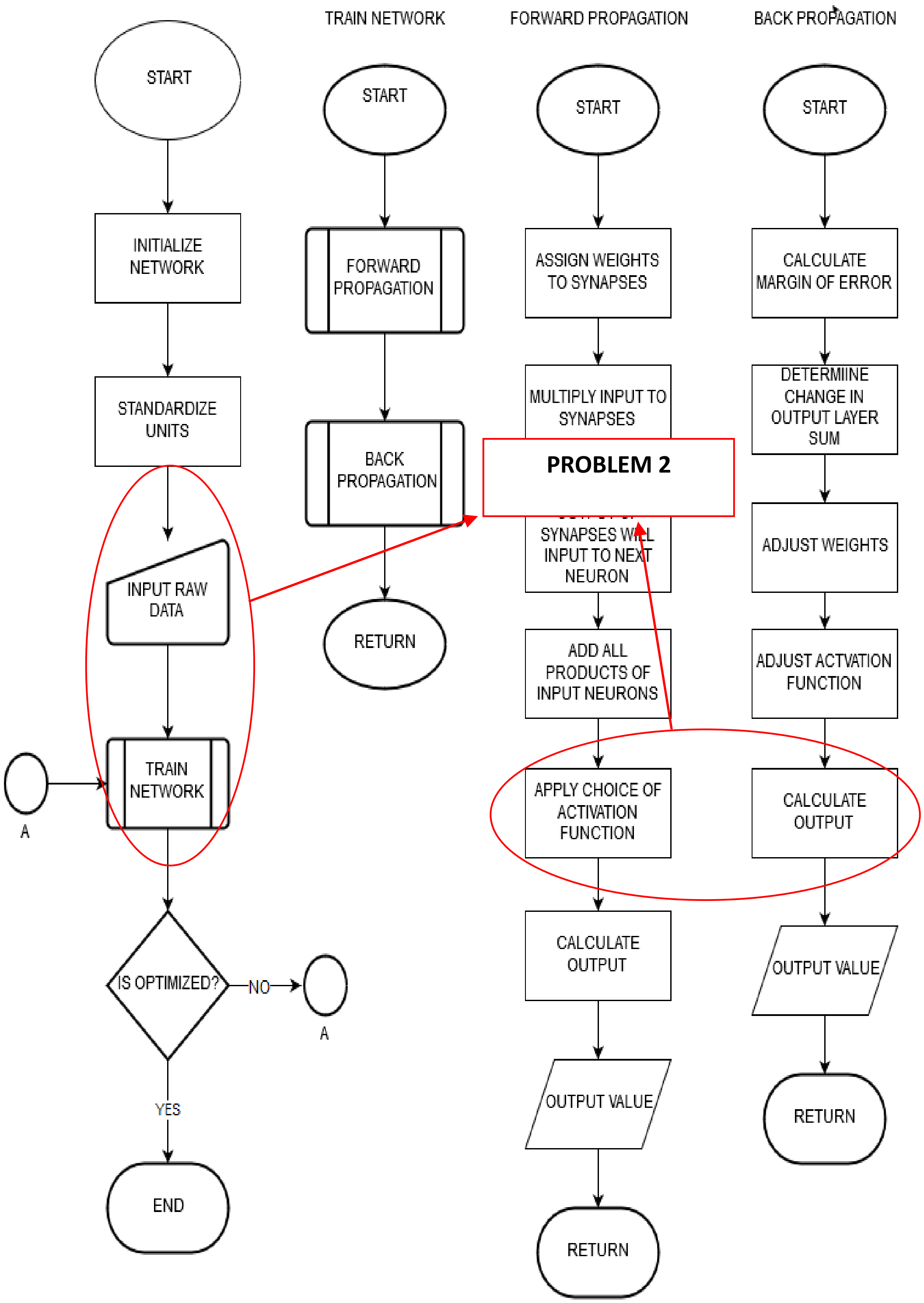
Step 5. The output is some function y = f(v) of the weighted sum.

Step 6. The output of a neuron (y) is a function of the weighted sum.

Step 7. Apply activation function.

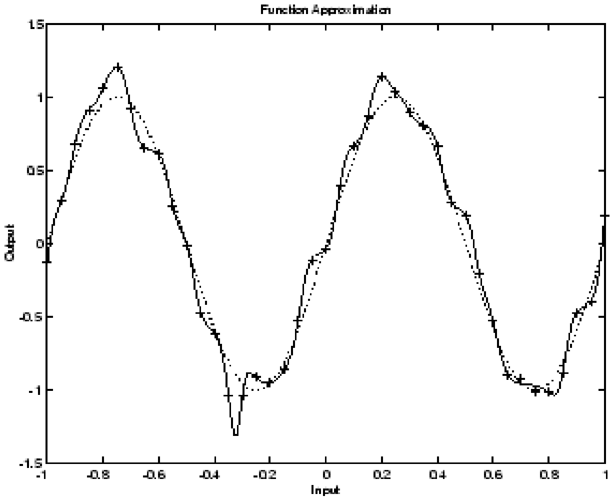
Step 8. Subtract a ratio (percentage) from the gradient of the weight.

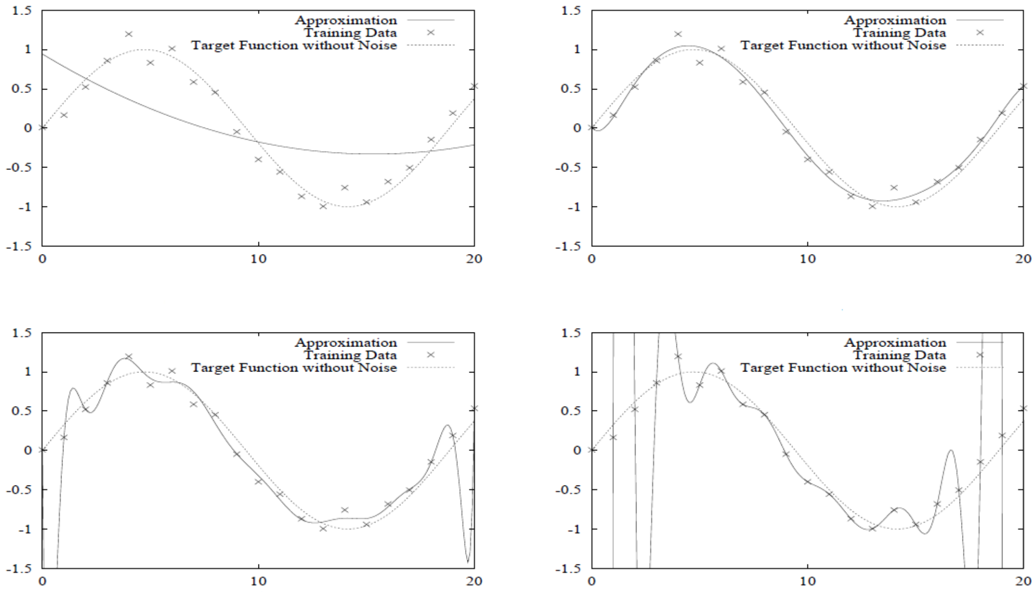
Illustrated Location:



Simulation:

The simulation below shows the response of a neural network that has been trained to approximate a noisy sine function. The underlying sine function is shown by the dotted line, the noisy measurements are given by the + symbols, and the neural network response is given by the solid line. Clearly this network has overfitted the data and will not generalize well.





3.) Problem 3: Algorithm gets "stuck" or starts to decrease in error rate correction during training rather quickly as the network learns.

Discussion:

The backpropagation algorithm is widely recognized as a powerful tool for training feedforward neural networks. However, since the algorithm employs the steepest descent technique to adjust the network weights, it suffers from a slow convergence rate and often produces suboptimal solutions, which is one of the major drawbacks of backpropagation. For example if you descend very fast at some steep valley, if you just use a first order gradient descent, you might get to the opposite slope and bounce back and forth all the time.

The problem of local minima is confronted with by the standard backpropagation. As one output of the modified training procedure, a bucket of all the possible solutions of weights matrices found during training is acquired, among which the best solution is chosen competitively based upon their performances on a validation dataset.

Location:

This problem in the algorithm resides on the adjustment of weights along with the adjustments of the activation function(s) when optimizing the descent and optimization of the global minimum to achieve the best possible weights on the synapses for the perceptrons.

Phase 1: Propagation

Each propagation involves the following steps:

Step 1. Forward propagation of a training pattern's input through the neural network in order to generate the propagation's output activations.

Step 2. Backward propagation of the propagation's output activations through the neural network using the training pattern target in order to generate the deltas (the difference between the targeted and actual output values) of all output and hidden neurons.

Phase 2: Weight update

For each weight-synapse follow the following steps:

Step 1. Multiply its output delta and input activation to get the gradient of the weight.

Step 2. Denote m input values.

Step 3. Initialize each of the m inputs (synapses) with a weight w1, w2, ...,

Step 4. The input values are multiplied by their weights and summed.

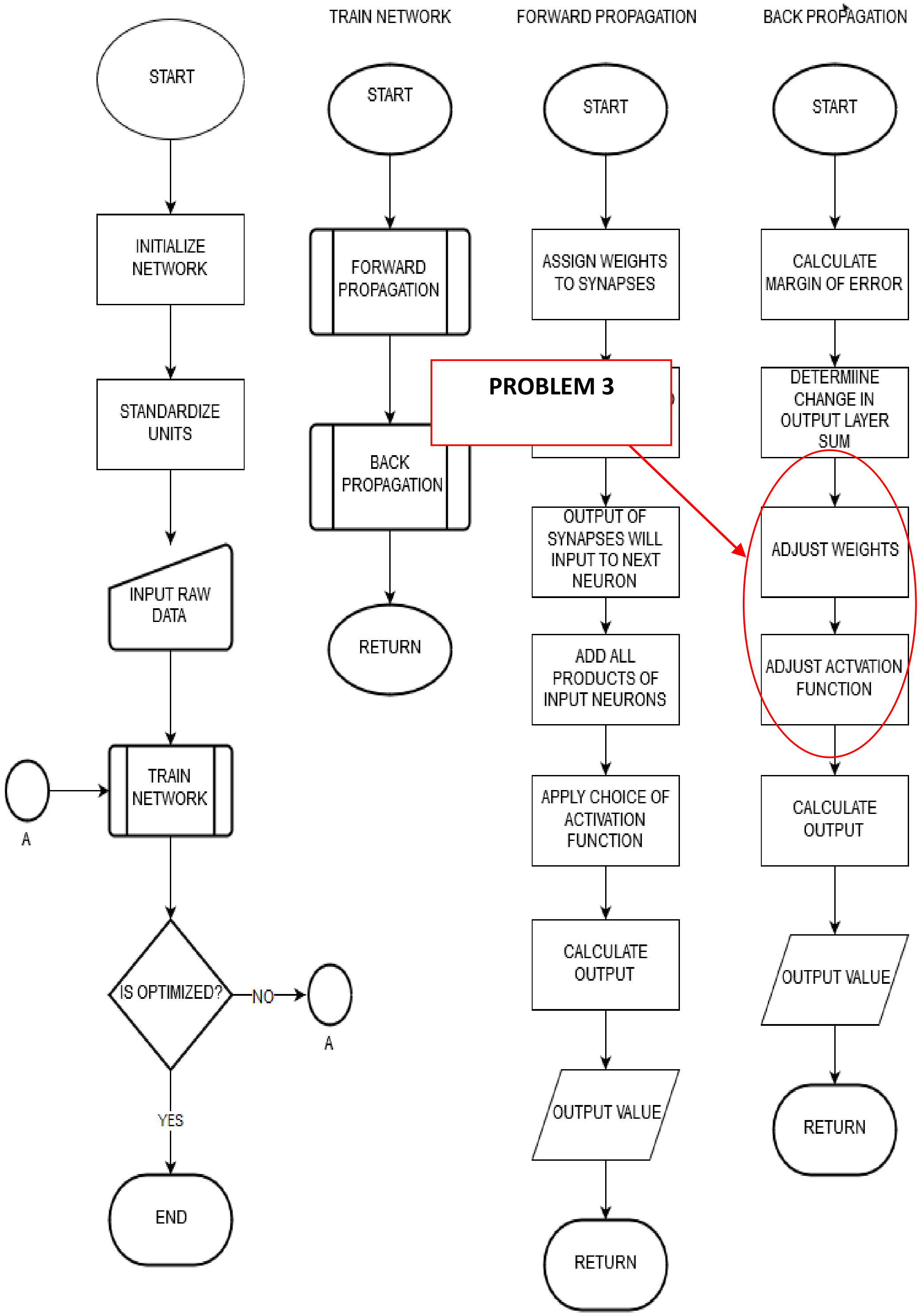
Step 5. The output is some function y = f(v) of the weighted sum.

Step 6. The output of a neuron (y) is a function of the weighted sum.

Step 7. Apply activation function.

Step 8. Subtract a ratio (percentage) from the gradient of the weight.

Illustrated Location


PROBLEM 3

Simulation:

During a successful training, the total error decreases rather quickly as the network learns, here is the output during simulated training:

```
...
...
Total error for this training set: 0.001000750550254843
Total error for this training set: 0.001000693973929822
Total error for this training set: 0.0010006374037948094
Total error for this training set: 0.0010005808398488103
Total error for this training set: 0.0010005242820908169
Total error for this training set: 0.0010004677305198344
Total error for this training set: 0.0010004111851348654
Total error for this training set: 0.0010003546459349181
Total error for this training set: 0.0010002981129189812
Total error for this training set: 0.0010002415860860656
Total error for this training set: 0.0010001850654351723
Total error for this training set: 0.001000128550965301
Total error for this training set: 0.0010000720426754587
Total error for this training set: 0.0010000155405646494
Total error for this training set: 9.99959044631871E-4


Testing trained XOR neural network
0 XOR 0: 0.023956746649767453
0 XOR 1: 0.9736079194769579
1 XOR 0: 0.9735670067093437
1 XOR 1: 0.045068688874314006
```

However, when it gets stuck, the total errors are decreasing, but it seems to be at a decreasing rate:

```
...
...
Total error for this training set: 0.1232548663141723
Total error for this training set: 0.12325486629199914
Total error for this training set: 0.12325486626982587
Total error for this training set: 0.1232548662476525
Total error for this training set: 0.12325486622547954
Total error for this training set: 0.12325486620330656
Total error for this training set: 0.12325486618113349
Total error for this training set: 0.12325486615896045
Total error for this training set: 0.12325486613678775
Total error for this training set: 0.12325486611461482
Total error for this training set: 0.1232548660924418
Total error for this training set: 0.12325486607026936
Total error for this training set: 0.12325486604809655
Total error for this training set: 0.12325486602592373
Total error for this training set: 0.12325486600375107
Total error for this training set: 0.12325486598157878
Total error for this training set: 0.12325486595940628
Total error for this training set: 0.1232548659372337
Total error for this training set: 0.12325486591506139
Total error for this training set: 0.12325486589288918
Total error for this training set: 0.12325486587071677
Total error for this training set: 0.12325486584854453
```

D. OBJECTIVES OF THE STUDY

As user demand scales for intelligent personal assistants (IPAs) such as Apple’s Siri, Google’s Google Now, and Microsoft’s Cortana, we are approaching an era automation. It is an open question how technology and developers alike should evolve to enable this emerging class of applications, and the lack of an Open Source IPA Development Platform workload is an obstacle in addressing this question.

In this paper, by enhancing and implementing deep learning with artificial neural networks, we present the design of Apiaro, an open end-to-end IPA Development Platform and API that can be used to build, code and extend a user’s own customized IPA.

Deep Neural Networks will be applied to features such as Speech to Text Processing, Natural Language Processing, Text to Speech, and Web Services Compatibility, the core of every artificial intelligent assistant.

We aim to give developers and enthusiast alike an algorithm. a model and a platform that will help them to build, customize and redistribute their own Intelligent Personal Assistant a whole lot easier and faster.

E. IMPORTANCE OF THE STUDY

Intelligent Personal Assistants is the future of computing and further development of technologies in this area will surely be the trend for the future. With the power of artificial neural networks, it became one of the strongest fields of development during 2015 and it will also be in 2016 and will certainly be one of the sectors generating most profits for years or decades to come.

1.) Intelligent machines can replace human beings in many areas of work. Painstaking activities, which have long been carried out by humans can be taken over by the computers.

2.) The ever growing Internet of Things must be easily coped with artificial intelligence and give people the comfort and ease of an automated lifestyle.

3.) Artificial intelligence can be utilized in carrying out repetitive and time-consuming tasks efficiently.

4.) Owing to the intelligence programmed in them, the machines can shoulder greater responsibilities and can be programmed to manage themselves.

F. SIGNIFICANCE OF THE STUDY

This study seeks to benefit the following groups of people:

1.) To the ***computer and smartphone users***, to make computing a more interesting and worthwhile experience through the use and utilization of Artificial Intelligence for automation and to ease them of the burden of executing certain computing tasks.

2.) To the ***programmers and developers,*** this study will encourage them to partake in the improvement of Artificial Intelligence and to advance and develop applications and system that utilizes this technology.

3.) To ***business owners and planners,*** A.I. is the current trend of today's technology. It exists now in almost every device. This study can help them to take their business ideas to the next level through the use of A.I. agents to cater customer's needs.

4.) Lastly, to the ***future researchers,*** that they will further this study through investing more time in finding out the effectiveness and relevance and to what extent can A.I. help in today's computing technologies.

G. SCOPE AND LIMITATIONS

This study focuses on the development of a Core Platform and API for building Intelligent Personal Assistants based from deep feedforward neural networks, an algorithm that is used constantly to machine learning, regression and classification solutions. With this platform, IPAs can be extended to include Machine Learning, Computer, Tasks, and Home Automation, Problem Solving and Queries.

Additionally, functions and features of this project only covers topic Artificial Intelligence, Computational Knowledge, Machine Learning and Cognitive Perceptions in the field of Computer Programming and Development and not topics directly related to Medicine, Engineering and other subject areas not mentioned or related above.

APIs and SDKs that will be used in the development of this project will be strictly Open Sourced Projects, Algorithms and Technologies with General Public License Version 3 and such, which allows developers to use, modify and redistribute code.

This study will not cover or in any way will involve proprietary technologies and features of similar Intelligent Personal Assistants such as Microsoft ‘s Cortana, Apple’s Siri, and Google’s Google Now, etc. Any similarity among these systems within this study is not intended and may be incidental.

H. DEFINITION OF TERMS

Acoustic Model is a set of words or sentences a Speech to Text Engine can understand.

Actuator is a Module or Code that executes a specific Intent.

Application Program Interface (A.P.I) is a set of routines, protocols, and tools for building software applications.

Artificial Intelligence (A.I.) is the simulation of human intelligence processes by machines.

Artificial Neural Network (A.N.N) is a network inspired by biological neural networks (the central nervous systems of animals, in particular the brain) which are used to estimate or approximate functions that can depend on a large number of inputs that are generally unknown.

Assistant is the term used for the Artificial Intelligence Entity, can be interchanged to Agent.

Back Propagation is a common method of training artificial neural networks used in conjunction with an optimization method such as gradient descent.

Belief is the knowledge base of the Agent about the world.

Deep Learning is a branch of machine learning based on a set of algorithms that attempt to model high-level abstractions in data by using a deep graph with multiple processing layers, composed of multiple linear and non-linear transformations.

Desire is the current goal of the Agent in relation to the Belief.

Environment means the real world where the Agent takes input from.

Feedforward Neural Network is an artificial neural network wherein connections between the units do not form a cycle.

Forward Propagation a method in where a neural network is initially trained by inputting random weight values.

Gradient Descent is a first-order iterative optimization algorithm.

Intelligent Personal Assistant (I.P.A.) is a software agent that can perform tasks or services for an individual.

Intent the code generated proposed "idea" of a natural language entry.

Intention is the method of action or the Agent will execute based from the Belief and Desire.

Internet of Things is a proposed development of the internet in which everyday objects have network connectivity, allowing them to send and receive data.

Local Minima also called a relative minimum, is a minimum within some neighbourhood that need not be (but may be) a global minimum.

Machine Learning is a subfield of computer science that evolved from the study of pattern recognition and computational learning theory in artificial intelligence.

Natural Language Processing (N.L.P.) is a field of computer science, artificial intelligence, and computational linguistics concerned with the interactions between computers and human.

Neurons is a mathematical function conceived as a model of biological neurons. Artificial neurons are the constitutive units in an artificial neural network.

Overfitting occurs when a model is excessively complex, such as having too many parameters relative to the number of observations.

Perceptrons is an algorithm for supervised learning of binary classifiers: functions that can decide whether an input (represented by a vector of numbers) belongs to one class or another.

Phoneme any of the perceptually distinct units of sound in a specified language that distinguish one word from another.

Phonetic Frame is one unit of phoneme in a given word (tokenized and syllabicated).

Raspberry Pi a brand of a programmable single computer board with ARM Architectures.

Speech Recognition is the process of translating speech into human readable text.

Speech Synthesis the process of transcribing text into audible computer generated voice

Synaptic Weight refers to the strength or amplitude of a connection between two nodes, corresponding in biology to the amount of influence the firing of one neuron has on another.