**3.1 Methodology**

This research started with a review of existing works in predicting network traffic. A framework that aims to solve some of the limitations of the existing systems is then proposed. The research methodology aimed to predict the network traffic using deep learning models and a Genetic algorithm to optimize the solution. The research methodology is in six steps to achieve the research objectives. The steps are:

1. Data Collection
2. Data Preprocess
3. Algorithm selection and model development
4. Training/Testing
5. Prediction
6. Optimization.
7. Performance evaluation metric

**3.1.1 Data collection**

This first stage of the system architecture (Data Collection) is very necessary because it is the quality of data captured that will determine the overall performance of the system. Good capturing tools will be installed to effectively capture the university network traffic at strategic locations within the University.

**3.1.2 Data Preprocess**

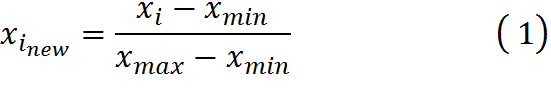
When it comes to creating a deep learning model, data pre-processing is the first step marking the initiation of the process. Real-world data is normally incomplete, inconsistent, inaccurate, and contains errors or outliers.

**3.1.2.1 Data Cleaning**

* Data cleaning is one of the processes in data preprocessing. Data cleaning involves detecting, correcting, removing, and handling corrupt or inaccurate records from a dataset.
* It involves handling missing data or missing values, noisy data, removing duplicates, handling outliers, etc. It is very important to do data cleaning because if model(s) learned from incorrect data without proper cleaning, it might lead to inaccurate predictions.
* A well-cleaned dataset can lead to more accurate and reliable deep learning models, while a poorly cleaned dataset can lead to misleading results or wrong predictions and conclusions

**3.1.2.2 Data Normalization**

* Normalization is a method where the values are adjusted to a range, which is usually the range between 0 and 1.

 3.1

* Equation (1) shows the formula to normalize all the features of *x* and transforms to a range between *min* new and *max* new
  + 𝑥𝑖𝑛𝑒𝑤 = normalized value
  + 𝑥*i* = process value
  + x𝑚𝑖𝑛 = minimum value in the dataset
  + 𝑥𝑚𝑎𝑥 = maximum value in the dataset

The set of attribute with the highest accuracy is returned and selected *Kemal and Baha, 2018*

**3.1.2.3 Data Reduction**

* The data reduction method was used due to the large volume of the dataset.
* It is used to reduce the original size of the dataset while still preserving the most and original important information or maintaining the integrity of the original data

**3.2 Algorithm/model selection**

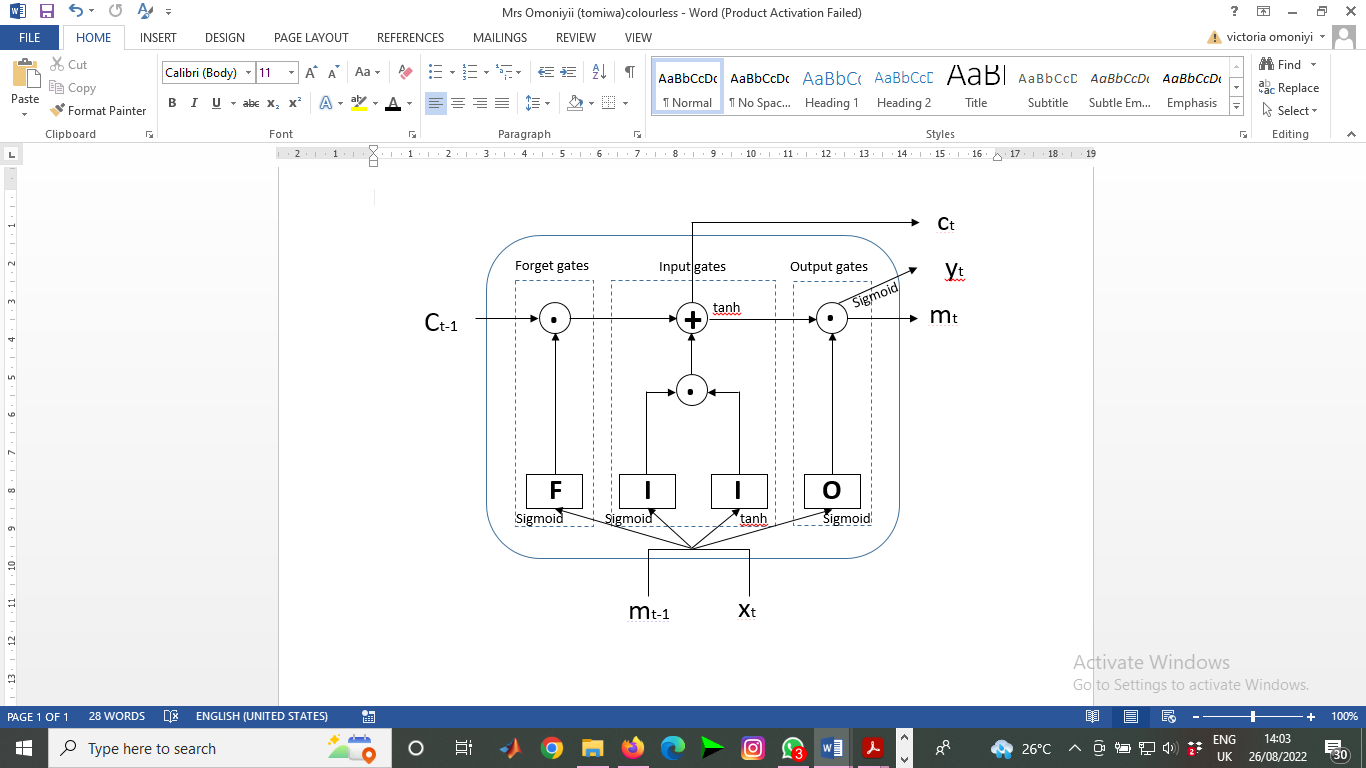
The prediction models proposed for this research:

I long short-term memory with Genetic Algorithm

ii Restricted Boltzmann machine with Genetic Algorithm

iii An optimized Gated Recurrent Unit (OGRU) model with Genetic Algorithm

**3.2.1 Long short-term memory (LSTM)**

****

**Figure 1 Department structure within LSTM. *Jihong* and *Xiaoyuan (2022)***

Long Short Term Memory (LSTM) is a type of recurrent neural network (RNN) architecture designed to overcome the limitation of traditional RNNs in learning long term dependencies. LSTM achieves this through a complex structure composed of gates and memory cells that control the flow of information.

**3.2.1.1 Components of LSTM**

An LSTM unit consists of the following components

i. **Cell State (Ct)** Represent the memory of the network. It carries information across different time steps

ii. **Hidden State(ht):** The output of the LSTM unit at each time step, which is also passed to the next unit.

iii. **Gates**: There are three main gates that control the flow of information.

(a) **forget Gate (f):** Decides what information to discard from the cell state.

(b) **Input Gate (i):** Decides what new information to add to the cell state

(c) **Output Gates (o):** Decides what part of the cell state to output and pass to the next time step

**3.2.1.1.1 FORGET GATE**

The forget gate plays a crucial role in deciding which information from the previous cell state should be retained or discarded. This gate is essential for the LSTM’s ability to manage long term dependencies and mitigate issues like the vanishing gradient problem commonly encountered in traditional recurrent neural networks (RNNs). Forget gate helps the LSTM decide which information to retain and which to discard.

The forget gate output is a value between 0 and 1. A value closer to 0 implies that the information is mostly discarded while a value closer to 1 signifies that the information is primarily retained in other words. The process can be expressed as:

3.2

Where:

F is the forget gate vector at time step t, it contains values between 0 and 1, indicating the degree to which each component of the previous cell state should be retained or forgotten.

σ is the sigmoid function

is the weight matrix associated with the forget gate

is the concatenation of the previous hidden state and the current input

is the bias term for the forget gate.

**Operation of the Forget Gate**

**i. Concatenation:** The previous hidden state and the current are concatenated to form a single vector. . This concatenated vector represents the combined information from both the past and the present

**ii. Linear Transformation**: The concatenated vector is multiplied by the weight matrix and then the bias vector is added. This linear transformation produces a new vector

**iii. Sigmoid Activation:** The resulting from the linear transformation is passed through the sigmoid activation function. The sigmoid function squashes the value of the vector to lie between 0 and 1.

**Forget Gate Output:** The output F is a vector of values between 0 and 1, where each value indicates how much the corresponding component in the previous state should be retained (value close to 1) or forgotten (values close to 0)

By selectively forgetting and retaining information, the forget gate ensures that the LSMT can dynamically adjust its memory, and maintain relevant information over long sequences while discarding unnecessary details. This ability to manage long term dependencies is what makes LSTMs particularly powerful for tasks involving sequential data, such as time-series predictions.

**3.2.1.1.2 INPUT GATE**.

The input gate in a Long Short-Term Memory (LSTM) network is responsible for controlling how much of the new information generated at the current time step should be added to the cell state. This gate works in conjunction with the forget gate to ensure that the LSTM network can effectively update its memory based on new inputs while preserving important information from previous states.

The input gate operation in an LSTM in involves two main components: the input gate itself and the candidate cell state. The input gate controls the flow of new information into the cell. It consists of two main components: the gate itself, which decides how much of the new information to add, and the candidate cell state, which represents the potential new information. The input **(*l*)** uses a sigmoid activation function to determine the extent to which new information should be added to the cell state. The process can be expressed as:

**a. Input gate**

3.3

Where

I is the input gate vector at time step t

is the sigmoid function

is the weight matrix for the input gate

is the concatenation of the previous hidden state and the current input

is the bias term for the input gate

**b. The candidate cell state (*l’*)** uses a tanh activation function to create a vector of new

information with values ranging between -1 and 1.

3.4

Where

tanh is the hyperbolic tangent function

is the weight matrix for the candidate cell state

is the concatenation of the previous hidden state and the current input

**Operation of the Input Gate**

**i. Concatenation:** The previous hidden state and the current input are concatenated into single vector , representing combined past and present information.

**ii. Weight Multiplication:** This concatenated vector is multiplied by the weight matrices and , which are learned during training. These operation combine the input and previous hidden state information to compute the activations for the input gate and candidate cell state.

**iii. Bias Addition:** Bias vectors and are added to the results of the weight multiplications. The biases help in adjusting the outputs independently of the input and hidden state.

**Iv Activation functions:**

(a) The input gate activation l is passed through a sigmoid function, which squashes the value into the range (0,1)

(b) The candidate cell state l’ is passed through a tanh function, tanh which produces values in the range (-1.1).

The input gate (I) and the candidate cell state (I’), are combined to update the cell state :

3.5

Where: is the forget gate vector, which modulates the previous cell state (. The input gate I determines the extent to which the candidate cell state should influence the new state.

**3.2.1.1.3 OUTPUT GATE**

The output gate in a long short term memory (LSTM) network controls the information that is output from the cell state and passed to the next hidden state. This gate determines what part of the cell state should be exposed to the hidden state, thereby influencing the subsequent time step’s computations and ultimately the network’s output. The process can be expressed as:

3.6

3.7

3.8

Where:

O is the output gate vector at time step t, containing values between 0 and 1, it determines how much of the cell state should be output

is the hidden state at the current time step, which also serves as the output of the LSTM cell an input for the next time step

is the sigmoid activation function

tanh is the hyperbolic tangent activation function

is the weight matrix for the output gate learned during training

is the concatenation of the previous hidden state and current input

is the bias vector for the output gate, learned during training

is the cell state at the current time step

represent the output of this time.

**Operation of the Output Gate**

**i. Concatenation:** The previous hidden state and the current input are concatenated to form a single vector. This concatenated vector represents the combined information from both the past and the present.

**ii. Linear Transformation:** The concatenated vector is multiplied by the weight matrix and the bias vector is added. This linear transformation produces a new vector

**iii. Sigmoid Activation:** The resulting vector from the linear transformation is passed through the sigmoid activation function. The sigmoid function squashes the values of the vector to lie between 0 and 1, resulting in the output gate vector (o)

**iv. Modulate Cell State:** The current cell state (is passed through the tanh activation function to produce a vector with values between -1 and 1. This operation ensures that the cell state’s values are within a stable range.

**v. Hidden State Calculation:** The output gate vector (o) is element-wise multiplied with the tanh of cell state. This operation selectively fiters the information from the cell state to produce the new hidden state (

**3.2.2 An Optimized Gated Recurrent Unit (OGRU) model and The Standard GRU Neural Network and the OGRU Neural Network**

3.2.2.1 **The Standard GRU Neural Network and the OGRU Neural Network**

There is no way we want to talk about OGRU without starting from GRU. The GRU neural network is a special variant of the recurrent neural network, which can maintain a longer-term information dependence and has been widely used in industry.

The LSTM neural network model is composed of three gate units, such as the forgetting gate, the input gate, and the output gate. The design of the gate unit is used to process the time series data. Although the gradient disappears to a certain extent, the parameters are more likely to lead to training. Longer time. The GRU neural network model is a variant of the LSTM neural network. It optimizes the structure of the LSTM neurons and combines the three gating units of the LSTM into two gating units, namely the update gate and the reset gate. Therefore, the parameters of the GRU model are relatively small, the training overhead is reduced, and the information dependency of a longer distance can be maintained. The neuronal structure of the standard GRU is shown in Figure 2. The GRU model consists of input layers, output layers and the implicit.

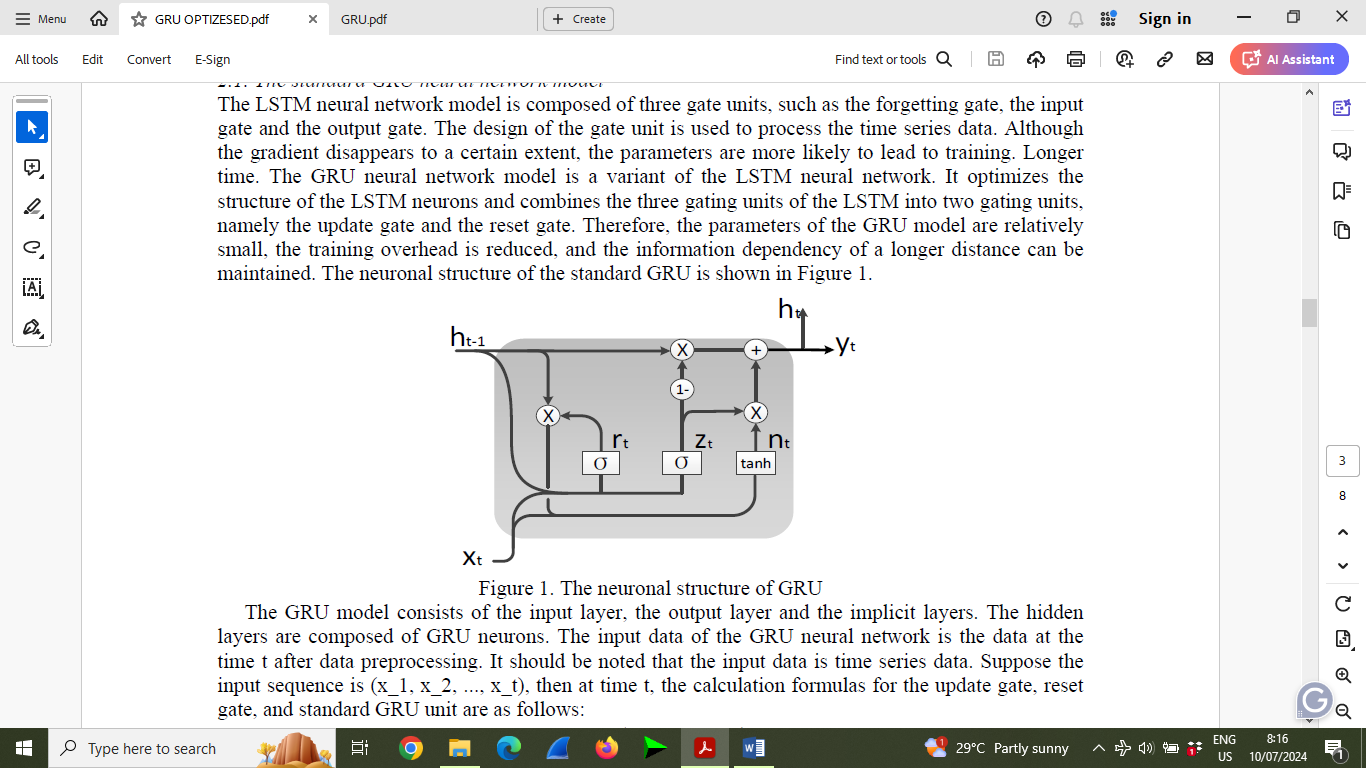


Figure 2. The neuronal structure of GRU

**Update Gate (**

The update gate in GRU plays a critical role in regulating the flow of information through the network allowing it to capture long term dependencies effectively while mitigating the vanishing gradient problem often encountered in traditional RNNs. This makes GRUs suitable for tasks involving sequential data where understanding context across different time steps is essential. The Update Gate ( determine how much of the previous hidden state to retain and how much of the new candidate hidden state to incorporate

The Update Gate ( is computed using a sigmoid function applied to a linear transformation of the concatenation of and

*3*.9

Where:

is the sigmoid function defined as

is the weight matrix specific to the update gate

denotes the concatenation of (previous hidden state) and (current input).

**Operation of the Update gate**

**i. Concatenation:** The previous hidden state and current input are concatenated into a single vector

= [ 3.10

and are components (or features) of and respectively.

**ii. Linear Transformation**: The concatenated vector is multiplied by the weight matrix

3.11

Where

and are specific components of

**iii. Sigmoid Activation:** The resulting vector from the linear transformation is then passed through a sigmoid activation function :

3.12

The sigmoid function squashes the value between 0 and 1, thereby producing the update gate

a determine how much of the previous hidden state should be retained (if is closed to 1) or forgotten (if is closed to 0)

b it also controls how much of the new candidate hidden state (computed by considering the reset gate and the current input) should be included in the updated hidden state

**Reset Gate**

The reset gate in GRU is essential for managing the flow of information through the network over time enabling the model to capture and utilize long-term dependencies in sequential data effectively. Together with the update , the reset gate contributes to the GRU's ability to address the challenges of vanishing gradient and to model complex relationships in tasks such as natural language processing, speech recognition, and time series prediction. The reset gate de-termines how much of the previous state should be forgotten or rest based on the current input . The reset gate is computed using a sigmoid activation function applied to a linear transformation of the concatenation of

3.13

where

is the sigmoid function

is the weight matrix specific to the reset gate

denotes the concatenation of and .

**Operation of the Reset gate**

**Concatenation:** The previous hidden state and current input are concatenated into a single vector

= [ 3.14

and are components (or features) of and respectively.

**Linear Transformation**: The concatenated vector is multiplied by the weight matrix

3.15

Where

and are specific components of

**Sigmoid Activation:** The resulting vector from the linear transformation is then passed through a sigmoid activation function :

3.16

The sigmoid function squashes the value between 0 and 1, thereby producing the reset gate

i determine how much of the previous hidden state should be forgotten or reset (if is closed to 0) or retained (if is closed to 0)

ii it helps the GRU model to selectively reset the information in the hidden state based on the current input , allowing the network to adaptively focus on relevant information in different contents.

**Hidden State**

The hidden state update in a Gated Recurrent Unit (GRU) is the process by which the hidden State at time step ***t*** is computed using the previous hidden stat the current input and the gating mechanisms of the GRU(the update gate and the reset gate )

The hidden state at time step t in GRU is computed as follows:

3.17

Where:

is the hidden state at time step t.

is the hidden state at the previous time step ***t*** - 1

is the candidate hidden state at time step t

is the update vector at time step t.

denotes the element-wise (Hadamard) product.

**The update gate ( )** determines the extent to which the previous hidden state ( ) should be updated to the new hidden state it is computed as:

3.18

Where , , and are the weights and bias for update gate, and is the sigmoid activation function

**Candidate Hidden State ()** is a potential new state computed using the reset gate . It is calculated as:

= tanh 3.19

Where is the reset gate, , and are the weights and bias, and tanh is the hypebolic tangent activation.

**Element-wise Multiplication**: The term 1 - is the complement of the update and determines how much of the previous hidden state should be retained.

The new hidden state is a linear interpolation between the previous hidden state and the candidate hidden state , controlled by the update gate . The update gate decides the proportion of the hidden state and the candidate hidden state to use in forming the new hidden state.

**When is close to 1:** The hidden state is mostly influenced by the candidate hidden state , this means the network is more receptive to new information from current input

**Where is close to 0:** The hidden state is mostly influenced by the previous hidden state . This means the network retains more of the information from the past, effectively preserving the long term dependencies.

**Candidate Hidden State ()**

The candidate hidden state in a GRU is a potential new state that combines the current input and the selectively forgotten previous state. The actual new hidden state is then a linear interpolation between the previous hidden state and the candidate hidden state controlled by the update gate ( ). This mechanism allows the GRU to maintain and update long-term dependencies in the sequence data efficiently

The candidate hidden state ( ) is calculated as follows:

= tanh 3.20

Where:

is the candidate hidden state at time step t.

is the weight matrix for the input .

is the input at time step *t*

is the reset gate vector at time step *t*

denotes the element-wise (Hadamard) product.

is the weight matrix for the hidden state

is the bias vector

is the hidden state at the previous time step ***t*** - 1

tanh is the hyperbolic tangent activation function.

**Input and Weight Matrices:** The input at time step *t* is multiplied by the weight matrix , this transformation maps the input to the hidden state space

**Reset Gate:** The reset gate rt controls how much of the previous hidden state to use in the computation of the candidate hidden state. It is computed as follows

3.21

Where , and are the weight and bias for the reset gate, and is the sigmoid activation function.

**Element-wise Multiplication:** The reset gate rt is element-wise multiplied with the previous hidden state . This allows the network to forget parts of the previous state selectively.

**Transformation with Previous State:** The result of the element-wise mu;tiplication is the multiplied by the weigh matrix . This transormation combines the selected parts of the previous hidden state with the current input.

**Candidate Hidden State:** Finally, the candidate hidden state is computed by applying the tabh activation function to the sum of weighted input and the transformed previous hidden state.

*3.22*

yt represents the output of the GRU neural network at time t, that is, the predicted result,

Wo represents the weight of ht; for sigmoid activation function, sigmoid and tanh

h(t-1) represents the standard GRU unit output at time t-1;

xt represents the input at time t;

**An Optimized Gated Recurrent Unit (OGRU) model**

GRU has the disadvantages of slow convergence and low learning efficiency. The OGRU neural network originates from the GRU neural network. The optimized gated recurrent unit (OGRU) neural network model uses the reset gate to optimize the learning mechanism of GRU, improving the learning efficiency and prediction performance. The OGRU neural network improves information processing capability and learning efficiency by optimizing the unit structure and learning mechanism of GRU, and avoids the update gate being interfered by the current forgetting information.

The optimized Gated Reccurrent Unit (OGRU)is an improved version of standard Gated Recurrent Unit(GRU) designed to further enhance the effciency and performance of the recurrent neural network. The OGRU introduces modifications to the original GRU equations to streamline operations and potentially improve training and inference speed while maintaining or enhancing performance. The primary components of an OGRU, similar to a standard GRU, include the update gate, reset gate, and candidate hidden state. The key difference lies in how these components are computed and interact

nt

x

σ

x

**+**

**x**

**x**

1-

**tanh**

**σ**

**t**

Figure 3. The neuronal structure of OGRU. Source: Xin et al(2019)

The OGRU aims to enhance the GRU by introducing optimization that maintain or improve performance while reducing computational complexity and improving efficiency. The specific mathematical modification can vary, but they generally focus on streamlining gate computations and interactions between inputs and hidden states.

**Key Optimization in OGRU**

**Parameter Reduction:** Fewer parameters can reduce the model size and computational requirements.

**Simplified Computations:** Adjusting operations to reduce complexity while maintaining performance

**Efficient Training:** Faster convergence due to optimized operations, potentially leading to shorter training times.

The update gate of the GRU neural unit is improved, the in the original update gate input is changed to multiplied by , and the output of the reset gate is used to feedback adjust the update gate. By filtering the current input information by the reset gate, the adverse effects caused by the redundant information are avoided to a greater extent, thereby accelerating the convergence speed and achieving the purpose of efficient learning. The OGRU neural network originates from the GRU neural network, and its neuron structure diagram is shown in Figure 3. The standard OGRU unit output calculation formula is as follows

= tanh

Among them, symbols such as and rt in the formula indicate the same meaning as standard GRU neurons, as shown in Fig 2, the OGRU neuron is different from the GRU neuron in that, at the update gate the rt is multiplied and then multiplied with previous time to hide the state weight, so that the rest gate re-screens the current input information that is. The Output of the reset gate is used to adjust the update gate to optimize the neuron structure. The neuron structure of the OGRU neural network is more reasonable than that of the GRU, and the hidden state at each moment can be simplified, and the gradient attenuation is suppressed to some extent. Therefore, the OGRU model can maintain a larger distance information dependency, and its learning efficiency and prediction accuracy are higher, The OGRU neural network consists of the input layer, the output layer and the hidden layer. The hidden layers are composed of OGRU neurons. By optimizing the learning mechanism of the GRU model, the recursive transmission of information between neurons is promoted, and the information preservation ability is stronger.

**1.5.3 Restricted Boltzmann machine (RBM)**

ARestricted Boltzmann machine (RBM) is a generative stochastic artificial neural network that can learn a probability distribution over its set of inputs. It consists of two layers a visible layer (representing observable data). and a hidden layer (representing latent features). There are no connections between units within a layer, making it a restricted version of the more general Boltzmann machine.

An RBM is composed of the visible units *V* = *(Vj ) j*∈*M* with visible state vector *v* = *(v j ) j*∈*M*, the hidden units *H* = *(Hi )i*∈*N* with hidden state vector *h* = *(hi )i*∈*N* , and the weight state matrix *w* = *(wji ) j*∈*M,i*∈*N* , which connects the visible units and hidden units, where *M* is the number of visible units and *N* is the number of hidden units. The joint probability distribution *P(v, h)* in RBM is defined as follows:

` 3.23

**1. Energy Function:**

The energy function E(v,h) for the visible units v and the hidden units h is defined as:

3.24

Where:

and are the states of the visible and hidden units. (where: v is binary state of visible unit of I, h is a binary state of hidden unit at j)

is the weight between visible unit and hidden unit

and are the biases of the visible and hidden units respectively.

The structure itself assigns a probability to each connection vector between hidden and visible units. This probability can be written in a mathematical equation with the help of energy function.

**2. Probability Distribution:**

The joint probability distribution of the visible and hidden units is given by:

Z here is the partition function is given by summing over all possible pairs of visible and hidden vector.

3.25

The marginal probability of a visible vector v is :

3.26

**3. conditional Distribution.**

The conditional probability of a hidden unit being active given the visible units, and vice versa are:

3.27

3.28

Where: (x) = is the logistic sigmoid function.

**4. Training (Contrastive Divergence):**

Training an RBM involves adjusting the weights and biases to minimize the difference between the data distribution and the distribution represented by the model. One common training algorithm is contrastive Divergence (CD).

\* Positive Phase: Compute the expected value of the outer product of the visible and hidden vectors given the data.

3.29

\* Negative Phase: Sample a reconstruction of the visible units from the hidden units and then compute the expected value of the outer product of the visible vectors given the reconstruction.

3.30

The weights are updated as:

3.31

Where a is a learning rate, means the expectation over the associated distributions

**1.6 Optimization Techniques**

Optimization techniques are the techniques used to discover the best solution out of all the possible solutions available under the constraints present. Optimization techniques are methods used to find the best possible solution or decision for a given problem, often involving the minimization or maximization of a function (Alwan, 2019)

### **Basic Concepts**

* **Objective Function:** This is the function that needs to be optimized. It could represent cost, profit, time, etc.
* **Constraints:** These are the conditions that the solution must satisfy.
* **Variables:** These are the unknowns that we need to determine to optimize the objective function.

### 2. **Types of Optimization**

* **Linear Optimization (Linear Programming):** The objective function and the constraints are linear. Common methods include the Simplex method and Interior Point methods.
* **Non-Linear Optimization:** Either the objective function or the constraints or both are non-linear. Techniques include Gradient Descent, Newton's Method, and Quasi-Newton methods.
* **Integer Optimization:** The variables are restricted to integer values. Techniques include Branch and Bound, Cutting Planes, and Dynamic Programming.
* **Combinatorial Optimization:** Focuses on optimizing a discrete system. Examples include the Traveling Salesman Problem and Knapsack Problem. Techniques include Greedy Algorithms, Dynamic Programming, and Metaheuristics like Genetic Algorithms and Simulated Annealing.
* **Stochastic Optimization:** Deals with problems under uncertainty. Techniques include Stochastic Gradient Descent and Monte Carlo methods. (Alwan, 2019)

**1.6.1 Genetic algorithm**

Optimization techniques are the techniques used to discover the best solution out of all the possible solutions available under the constraints present. The genetic algorithm is one such optimization algorithm built based on the natural evolutionary process of our nature. **Genetic Algorithms** will be used in this research.

Genetic algorithms are search algorithms based on mechanics of natural selection and natural genetics. Philosophically GAs are based on Darwins′theory. Genetic algorithms have the following advantages over traditional methods:

i. GAs search from a population of points, not a single point. Hence GAs are said to be Global optimization techniques.

ii. GAs use only the value of convex (minimize) objective function. The derivatives are not used in the search process.

iii. GAs use probabilistic transition rules, not deterministic rules.

iv. Genetic algorithms are the most popular form of evolutionary algorithms

A population of chromosomes represents a set of possible solution. These solutions are

Classified by an evaluation function, giving better values, or fitness to better solutions. The simplest representation is a *value representation* where the chromosome consists of the values of the design variables placed side by side. For example, suppose we have 6 discrete design variables with integer values ranging from 1 to 5. Suppose we also have 4 continuous design variables whose values are real numbers ranging from 3.000 to 9.000. A possible chromosome is shown in Fig.4:

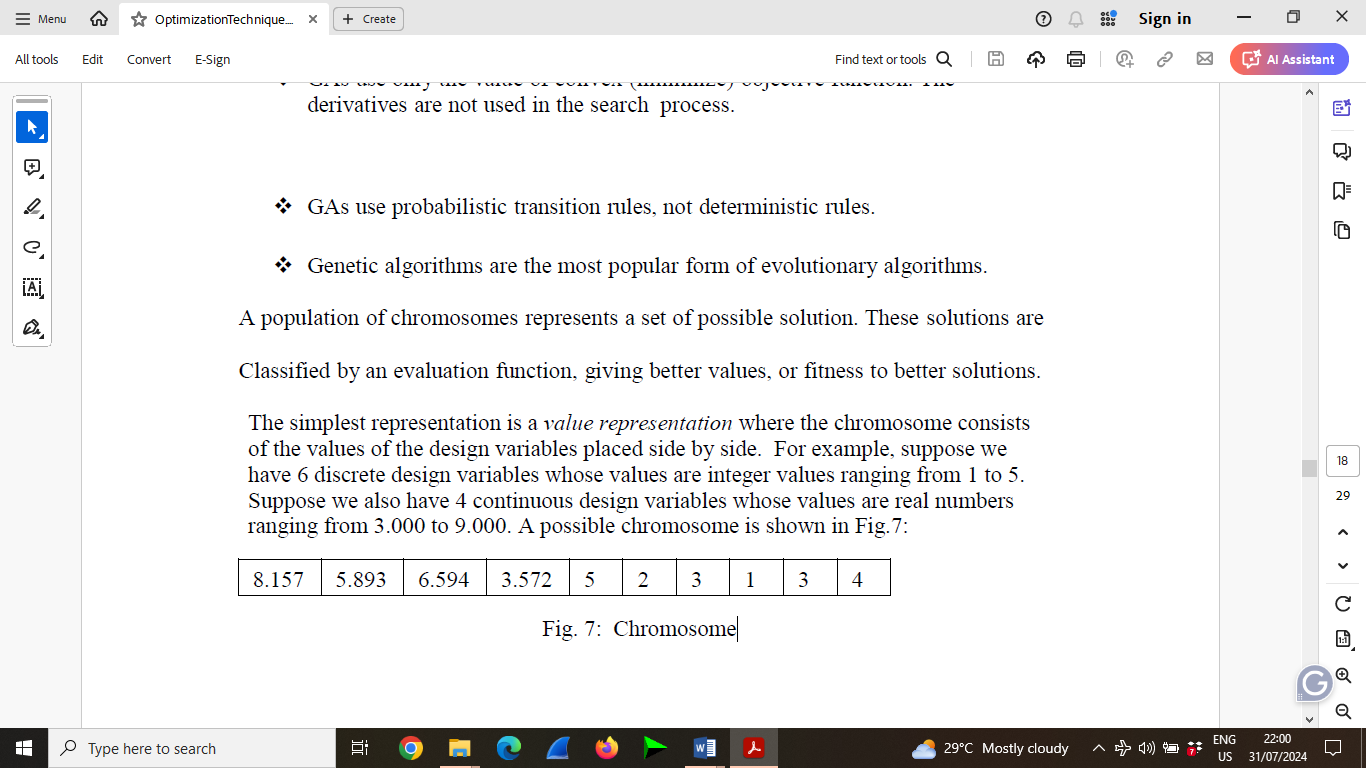


Fig. 4: Chromosome

**Key Concepts**

1. **Population**: A set of potential solutions to the problem. Each individual in the population is called a chromosome.
2. **Chromosome**: A representation of a solution, typically encoded as a string of bits, numbers, or characters.
3. **Gene**: A part of a chromosome, representing a specific parameter of the solution.
4. **Fitness Function**: A function that evaluates and assigns a fitness score to each chromosome based on how well it solves the problem.
5. **Selection**: The process of choosing chromosomes from the current population to create offspring for the next generation.
6. **Crossover (Recombination)**: A genetic operator that combines parts of two parent chromosomes to produce offspring with characteristics of both parents.
7. **Mutation**: A genetic operator that introduces random changes to individual genes in a chromosome, promoting diversity within the population.

**Steps in a Genetic Algorithm**

**1. Initialization (**Chromosome encoding)

A population of potential solutions (chromosomes) is initialized randomly. Each chromosome represents a candidate solution to the problem.

3.32

Where: P(0) is the initial population and represents an individual chromosome.

**2 Fitness Function**

Fitness Function F:

Each chromosome is evaluated using a fitness function to determine how good a solution it is

3.33

For a given chromosome x, the fitness function f(x) returns a fitness score.

Where the fitness function and x is a chromosome.

**3 Selection**

Selection is based on the fitness of each chromosome. Chromosomes are selected from the population to be the next generation's parents based on their fitness scores. A common method is roulette wheel selection, where the probability of selecting a chromosome is proportional to its fitness:

3.34

**4. Crossover:**

Crossover combines parts of two parent chromosomes to produce offspring.

Single-point at position k:

Offspring1 = 3.35

Offspring2 = 3.36

Where XA and XB are present chromosomes.

**5. Mutation Operation:**

With a certain probability, some genes (bits) in the offspring are mutated. Mutation introduces random changes to genes in a chromosome to maintain genetic diversity.

For a binary chromosome, mutation at gene j:

3.37

Where pm is the mutation probability.

**6. Replacement**

The new generation of chromosomes is formed by replacing some or all the old generation with the new offspring. This can be done in various ways, such as by replacing the least fit individuals or by combining old and new generations.

P(t + 1) = New Generation.

**7. Termination**

The algorithm repeats the selection, crossover, mutation, and replacement steps until a termination condition is met. The termination condition can be a maximum number of generations, a satisfactory fitness level, or stagnation ( no improvement over a certain number of generations)

**Algorithm workflow (GA)**

**Initialization:**

Generate an initial population of chromosome randomly.

Population0 = {x1,x2,………xN} 3.38

**Evaluation:**

Compute the fitness of each chromosome.

F(Xi), ꓯ I Ꜫ {1,2, ……N} 3.39

**Selection:**

Select chromosomes based on their fitness using a method like roulette wheel selection.

Selectedt = { xi : I ~ pi, ꓯ I Ꜫ {1,2, ……N}} 3.40

**Crossover:**

Apply crossover to selected pairs to create offspring.

Offspringt = Crossover (Selectedt)

**Mutation**

Apply mutation to the offspring.

Offspring’t = Mutation (Offspringt)

**Replacement:**

From the new population by replacing some or all of the old population with the new offspring.

Population

**Termination:**

Repeat the process until a stopping criterion is met, such as a maximum number of generations G or a satisfactory fitness level.

T = t + 1 until

The process continues from step 2 with t= t =1 until termination conditions is met, This iterative cycle simulates the process of natural evolution, progressively improving the quality of solutions in the population. Genetic algorithms effectively use these mathematical principles to evolve solutions over iterations making them powerful tools for solving complex optimization problems.

**Performance indicators / Performance Evaluation**

To illustrate the effectiveness of the prediction model for network traffic, the following performance Evaluation metrics or Standard Metrics are introduced to measure the prediction accuracy of the prediction model

**Coefficient of Determination**

The Coefficient of Determination, or R2 value is a normalized function popular when determining the goodness of a fit to data. It is defined by

Where,

With y representing the reference values and е is the difference between the predicted values and the actual. Value Equation 22 is exactly the sum of Square errors and Equation 21 is the sum of all deviations squared.

Hence, the R2 value gives a combined performance indication of the actual size and deviation of the errors as opposed to what the RMSE and STD indicate individually. In a perfect fit

, and thus the R2 value would be equal to 1.

**R**oot Mean Square Error (**RMSE)**

**M**ean Absolute Error (**MAE)**

**Mean absolute percentage error (MAPE)**

**Square sum error (SSE)**

Where N is the number of Network traffic samples, T(k) is actual value of network traffic, Ṫ(k) is the predicted value of network traffic.