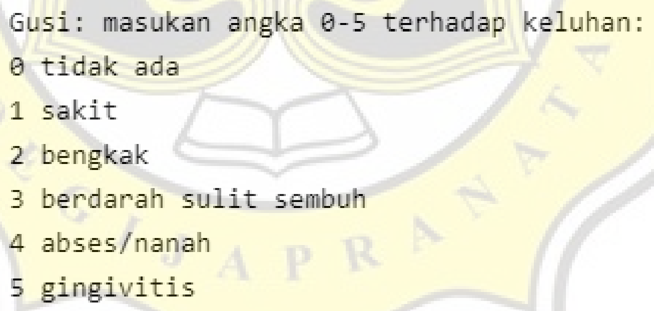
CHAPTER 4
ANALYSIS AND DESIGN

4.1. Analysis

This chapter describes how the algorithms work, and how the program is implemented. The earlier iteration of the program only uses three features which are the gums (*gusi*), teeth (*gigi*), and lips (*bibir*). The data was only gathered from web searches and medical websites. This early iteration creates a lot of ambiguity as there are similar symptoms that occur for multiple illnesses and because the illnesses' symptoms only came from limited sources, not a lot of unique symptoms were noted. In the iteration that follows, features were added and more sources are also added, those sources coming from new trusted websites and also health journals. The new features are gums (*gusi*), teeth (*gigi*), lips (*bibir*), throat (*tenggorokan*), and area of mouth (*area mulut*). This makes the data more diverse and the algorithms have more data on which to base their predictions.

With six distinct parts of the mouth with each having unique symptoms, the way the prediction is done is by inputting six values that correspond to each symptom on each part of the mouth as seen in Figure 4.1.



```
Gusi: masukan angka 0-5 terhadap keluhan:
0 tidak ada
1 sakit
2 bengkak
3 berdarah sulit sembuh
4 abses/nanah
5 gingivitis
```

Figure 4.1 **Example of symptoms choice**

To make the prediction process more streamlined, all of the algorithms that are used use the same method of input which is taking six inputs, each input for each part of the mouth, with the differences only being in the input's processing as each algorithm may require a particular type input such as array, or has to be reshaped first in order for the prediction to work.

4.2. Design

The program will begin by asking several questions and listing several symptoms as the symptoms that will be inputted into the program, so the program itself is very straightforward as seen in Figure 4.1.

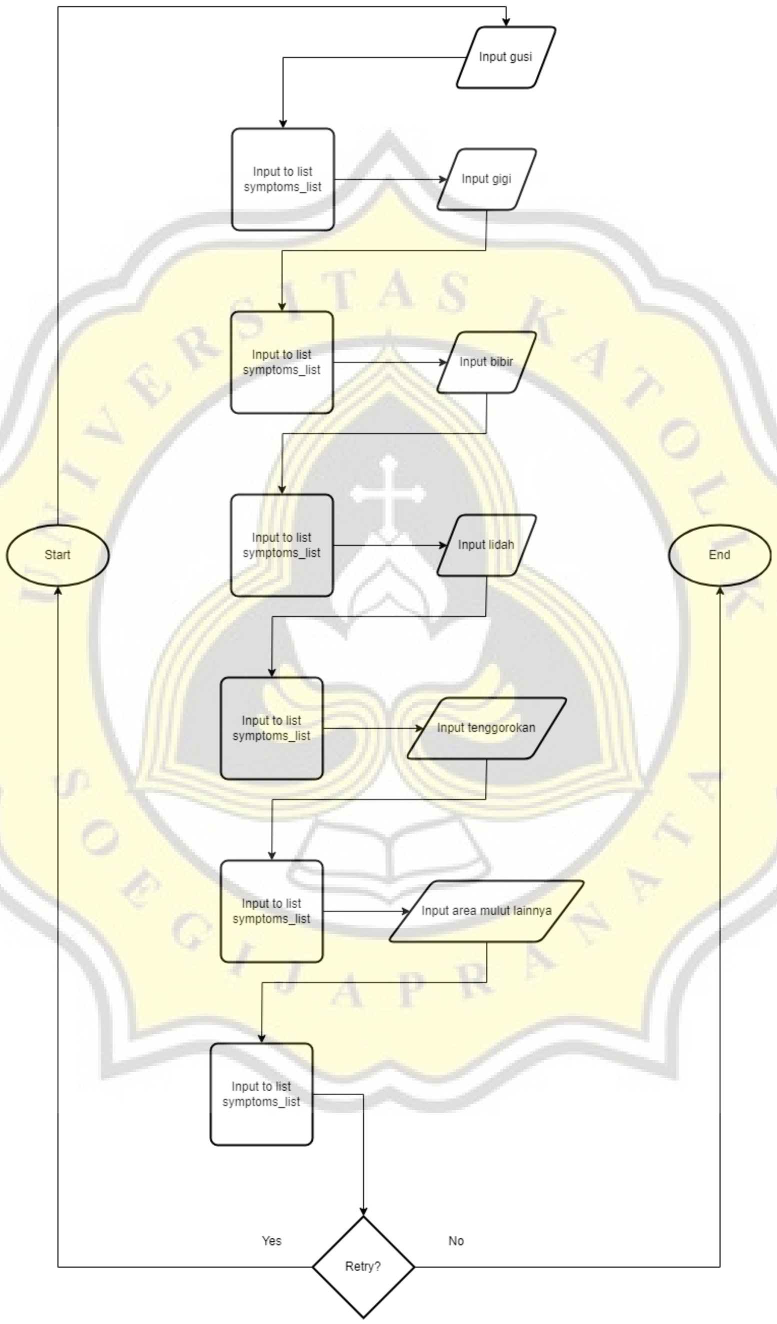

Input gusi
Input to list symptoms_list
Input gigi
Input to list symptoms_list
Input bibir
Input to list symptoms_list
Input lidah
Input to list symptoms_list
Input tenggorokan
Input to list symptoms_list
Input area mulut lainnya
Input to list symptoms_list
Start
End
Yes
No
Retry?

Figure 4.1 **User input for the program**

As there are three different algorithms used to perform prediction, this subchapter will explain the algorithms deeper and how they work. Starting with the main algorithm XGBoost and then its comparative algorithms, random forest, and TensorFlow's MLP. XGBoost being picked for the main program, and not for the comparative, is because XGBoost is the middle point between random forest and TensorFlow in a sense that XGBoost is bagging as an algorithm that learns from residuals of each iteration as seen in Figure 4.2 which constitute XGBoost as a machine learning algorithm, different from the random forest in that random forest uses ensemble learning (majority system) to get its result. To compare XGBoost with another machine learning algorithm, TensorFlow's multilayer perceptrons (MLP) is used.

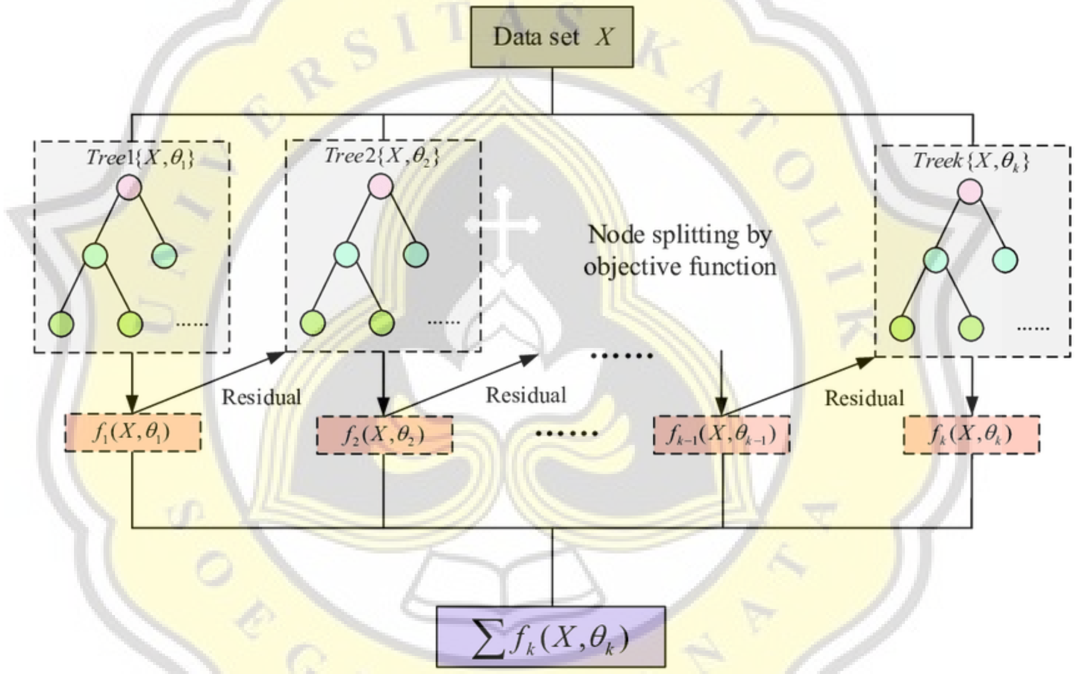


Figure 4.2 **Flowchart of XGBoost**

The comparison is done to see whether the result in XGBoost will be more similar to its predecessor random forest or whether the machine learning's counterpart TensorFlow will get a more similar result.

XGBoost derives from the random forest algorithm, which in itself derives from the decision tree. A decision tree is a flowchart-like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label. An example of a decision tree can be seen in Figure 4.3.

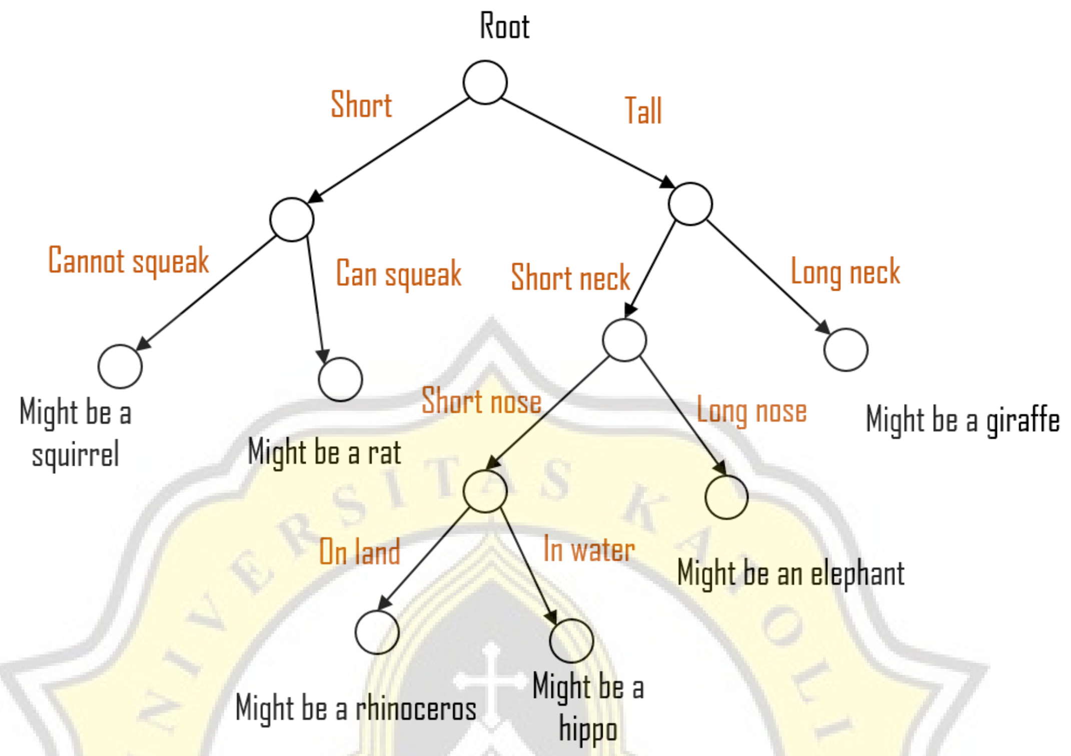


Figure 4.3 **An example of a decision tree for classification**

A tree can be "learned" by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called recursive partitioning. The recursion is completed when the subset at a node all has the same value of the target variable, or when splitting no longer adds value to the predictions[18]. The difference between XGBoost and the random forest is also reflected in how each tree is valued, and how the trees' order affects the algorithms' outcome. TensorFlow's multilayer perceptron is also used, MLP can be seen in Figure 4.4.

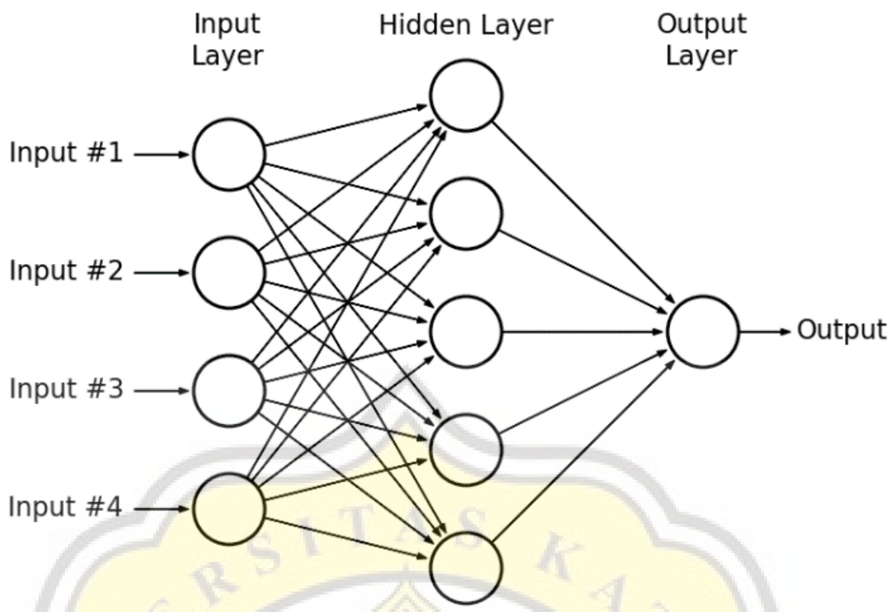


Figure 4.4 **Multilayer perceptrons with one output layer**

The MLP that is used for this program consists of three layers, two ReLu layers, and the output layer is softmax to show the probability for each prediction as it is multi-class, not binary.

4.3. Function

Starting from XGBoost. The only difference between XGBoost's classification and regression is the loss function. An example of XGboost's classification can be seen in Figure 4.1. The loss function for XGBoost's classification is as follows

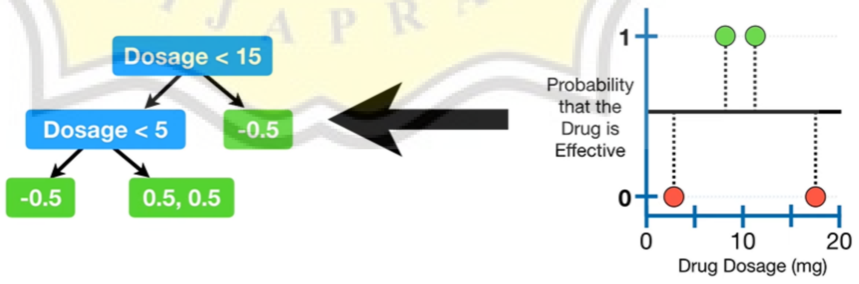


Figure 4.1 **XGBoost classification example**

$$L(y_i, p_i) = [y_i log(p_i) + \ y_i(1 - y_i) log(1 - p_i)] \quad (1)$$

$$\frac{(\Sigma Residual_i)^2}{[P_i \times (1-P_i)]+\lambda} \quad (2)$$

$$\frac{(\Sigma Residual_i)}{[P_i \times (1-P_i)]+\lambda} \quad (3)$$

$$[\sum_{i=1}^{n} \quad L(y_i, p_i)] + \frac{1}{2}\lambda O_{value}^2 \quad (4)$$

Function (1) is the loss function of XGBoost classification $L(y_i, p_i)$ in its entirety is the negative log-likelihood for the classification. The function (2) and (3) are functions for XGBoost classification for similarity score and output value respectively. Similarity score, $(\Sigma Residual_i)^2$ is the summation of residuals squared, and (P_i) is the previous probability, and lastly λ is the regularization parameter. For output value, the summation of residuals is not squared. Function (4) is the function to build the first tree, with an explanation similar to the previous function, $L(y_i, p_i)$ is the loss function, λ is the regularization parameter, and O_{value} is the output value.

For random forest some breakdowns for the algorithm in mathematical function will also be explained, an example of how random forest classification works can be seen in Figure 4.2.

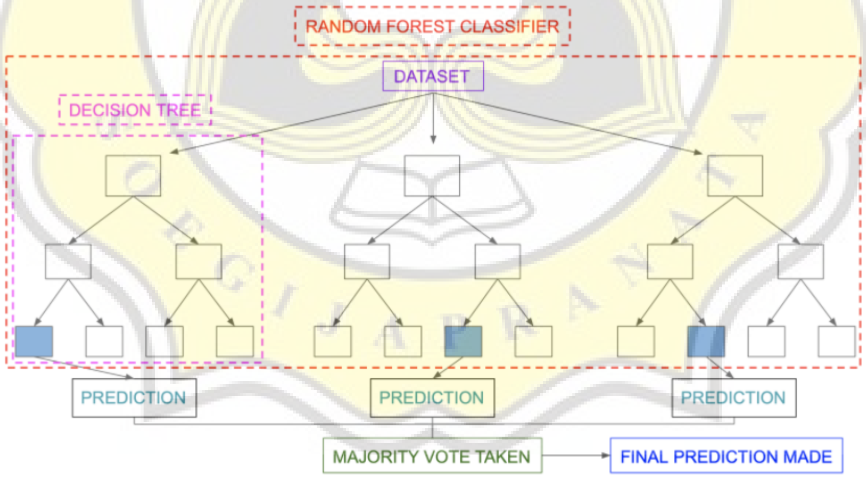


Figure 4.2 **Random forest classification example**

$$1 - \sum_{i=1}^{n} \quad (P_i \quad)^2 \quad (5)$$

$$E(S) = -p_{(+)} log p_{(+)} - p_{(-)} log p_{(-)} (6)$$

$$fi_i = \frac{\Sigma_{j:node\ j\ splits\ on\ feature\ i}\, ni_j}{\Sigma_{k \in all\ nodes}\, ni_k} \qquad (7)$$

Starting from function (5), Gini index which is used to count the impurity of the splitting nodes as seen in Figure 4.3, the root node in the right has one decision that is not purely divided to yes or no but contains both, the function will then calculate how much the impurity of each node. Then in function (6) is entropy which is related to Gini impurity as this function is used to count the entropy, which is used to measure the impurity of the split, where $p_{(+)}$ is the probability of a positive class and $p_{(-)}$ is the probability of a negative class. . And the last function for random forest is function (7) which is the function for feature importance, fi_i is the importance of node i, while ni_j is the importance of node j, Figure 4.4 is to see how it works.

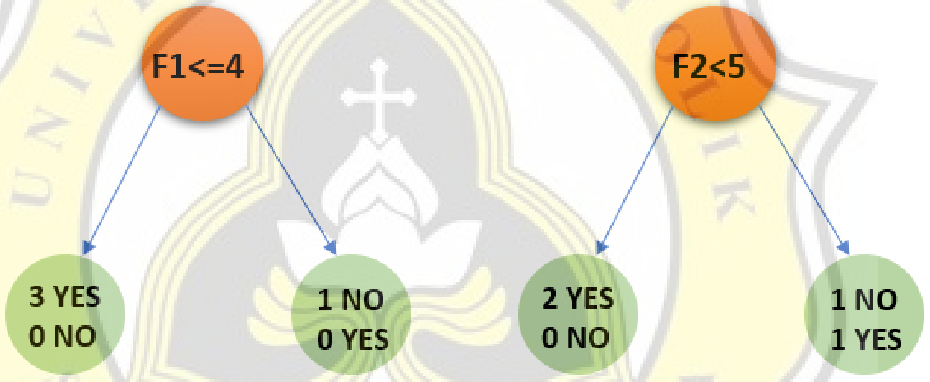


Figure 4.3 **Random forest classification example**

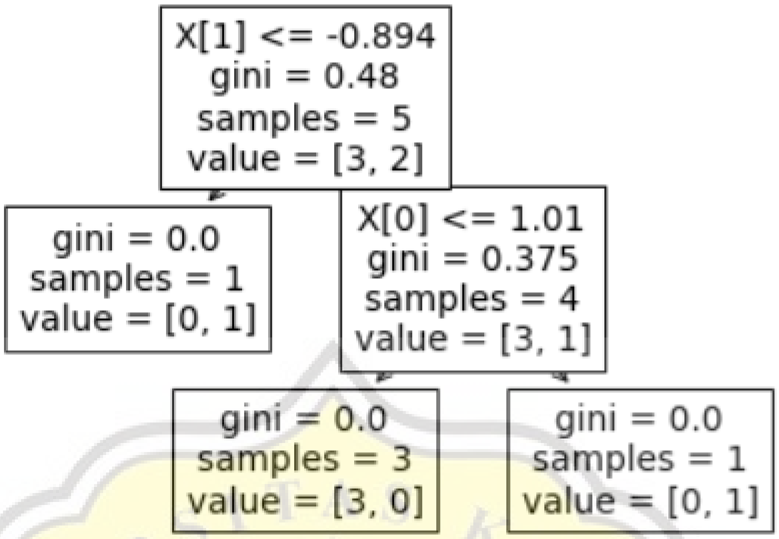


Figure 4.4 **How function (7) looks like**

For the last algorithm, we will take a look at multilayer perceptrons (MLP). To clearly see how MLP works, we need to see a single linear threshold unit (LTU) with its mathematical function shown in Figure 4.5. While an example of MLP has been shown before, a more detailed picture will be used to give an explanation of certain parts of the MLP in figure 4.6 and figure 4.7. A Perceptron is a simple artificial neural network (ANN) based on a single layer of LTUs, where each LTU is connected to all inputs of vector x as well as a bias vector b a.

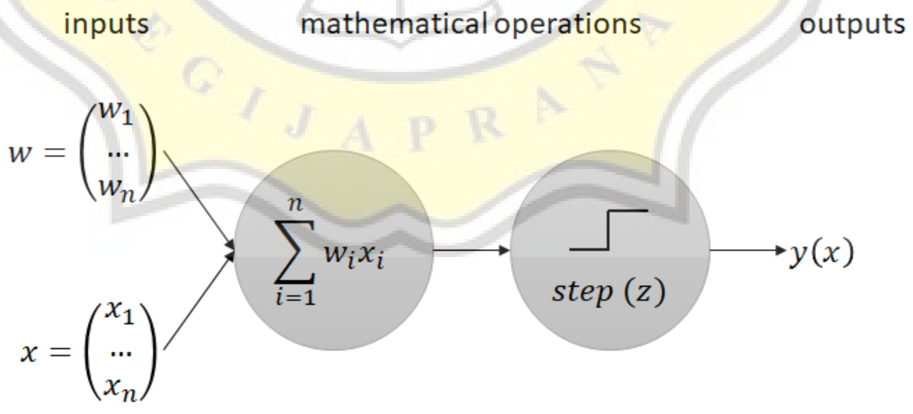

inputs
mathematical operations
outputs
$w = \begin{pmatrix} w_1 \\ \cdots \\ w_n \end{pmatrix}$
$x = \begin{pmatrix} x_1 \\ \cdots \\ x_n \end{pmatrix}$
$\sum_{i=1}^{n} w_i x_i$
$step\ (z)$
$y(x)$

Figure 4.5 **Example of LTU**

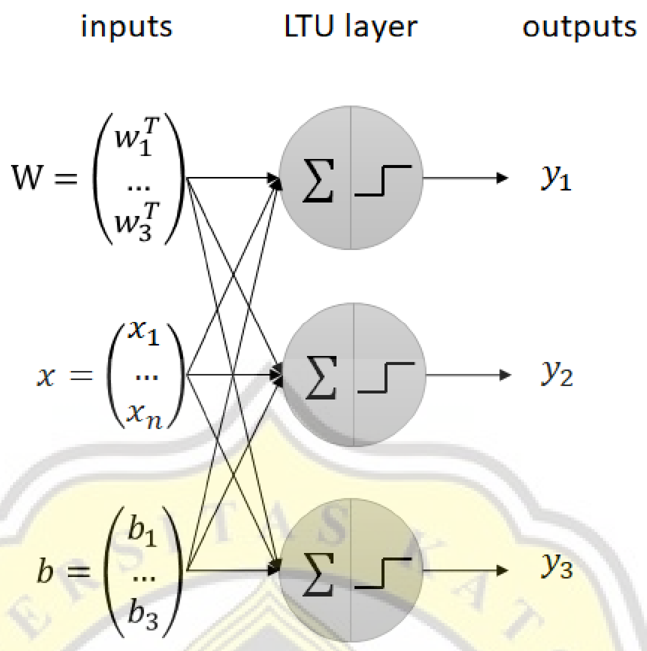

inputs
LTU layer
outputs
$W = \begin{pmatrix} w_1^T \\ \dots \\ w_3^T \end{pmatrix}$
$x = \begin{pmatrix} x_1 \\ \dots \\ x_n \end{pmatrix}$
$b = \begin{pmatrix} b_1 \\ \dots \\ b_3 \end{pmatrix}$
Σ
y_1
y_2
y_3

Figure 4.6 **Weight, input, and bias for each node**

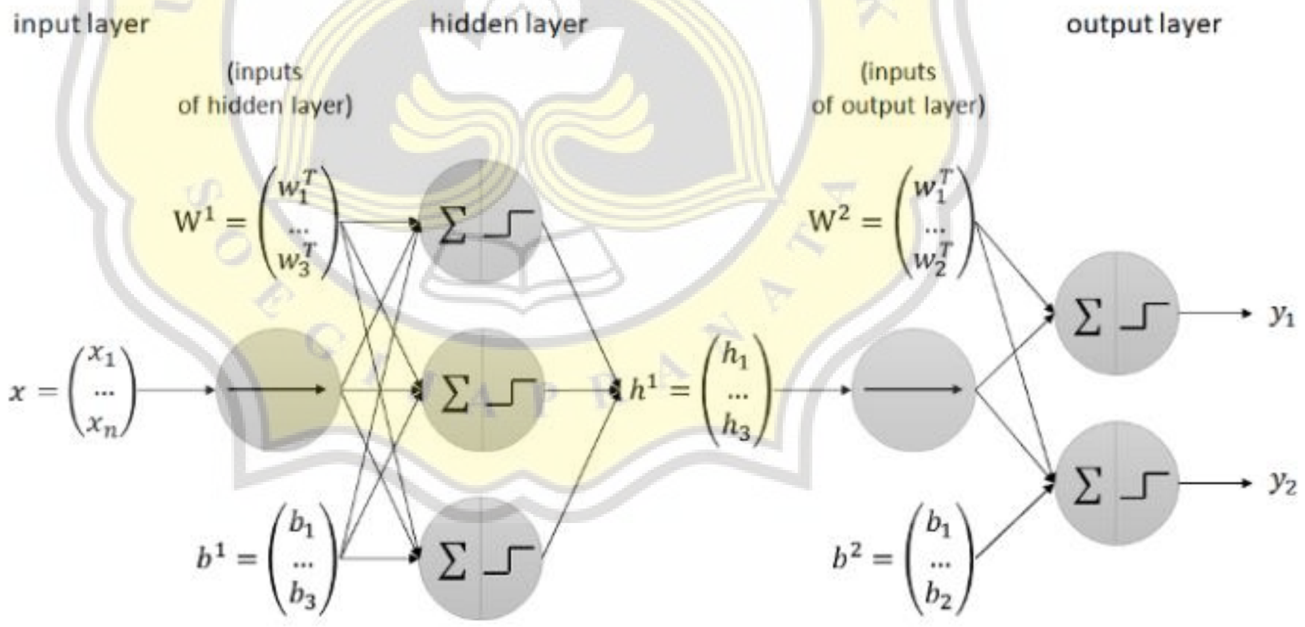

input layer
hidden layer
output layer
(inputs of hidden layer)
(inputs of output layer)
$W^1 = \begin{pmatrix} w_1^T \\ \dots \\ w_3^T \end{pmatrix}$
$x = \begin{pmatrix} x_1 \\ \dots \\ x_n \end{pmatrix}$
$b^1 = \begin{pmatrix} b_1 \\ \dots \\ b_3 \end{pmatrix}$
$h^1 = \begin{pmatrix} h_1 \\ \dots \\ h_3 \end{pmatrix}$
$W^2 = \begin{pmatrix} w_1^T \\ \dots \\ w_2^T \end{pmatrix}$
$b^2 = \begin{pmatrix} b_1 \\ \dots \\ b_2 \end{pmatrix}$
Σ
y_1
y_2

Figure 4.7 **A more detailed example of multilayer perceptrons**

$$z = w^T . x = \sum_{i=1}^{n} w_i x_i \tag{8}$$

$$f(x) = max(0, x) \quad (9)$$

$$h^1 = step(z^1) = step(W^1 . x + b^1) \quad (10)$$

$$y = step(z^2) = step(W^2 . h^1 + b^2) \quad (11)$$

We will start with the simpler function of LTU, then to MLP. Function (8) is the weighted sum (z) of LTU in which w is weight, hence $w^T$is the transpose of the weight and x is the input. Function (9) is the activation function of LTU in the hidden layer, which uses ReLU (Rectifying Linear Unit) ($f(x) = max(0, x)$) in which the minimum value that is accepted is 0, and if the number is less than 0 it will be converted to 0. Function (10) is the first hidden layer (h^1), W (as seen in Figure 4.8) is a matrix of shape (u, n), where u = number of LTUs and n = number of input values. The input vector $\boldsymbol{x}$ is of shape (n, 1), the bias vector b is of shape (u, 1) and the output vector y is of shape (u, 1). By that, the Perceptron can be used for multi-class classification. Function (11) is the output layer.

$$W = \begin{pmatrix} w_1^T \\ \cdots \\ w_u^T \end{pmatrix} = \begin{pmatrix} w_{1,1} & \cdots & w_{1,n} \\ & \cdots & \\ w_{u,1} & \cdots & w_{u,n} \end{pmatrix}$$

Figure 4.8 **Matrix of weight**