VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF INFORMATION TECHNOLOGY

DEPARTMENT OF INFORMATION SYSTEMS

PROJECT REPORT

E-COMMERCE - IS334.N21.TMCL

SPAM REVIEWS DETECTION ON VIETNAMESE E-COMMERCE PLATFORMS

Teacher Intructions: Master DO DUY THANH

BUI THI HUONG - 21522130

Ho Chi Minh City, July 17, 2023

ACKNOWLEDGMENTS

First of all, I would like to express my sincere thanks to Mr. MSc. Do Duy Thanh for guiding and instructing me since the first days of studying and researching. Your dedication, extensive knowledge and dedicated guidance have helped me to better understand the concepts and techniques in the field of Machine Learning, Deep Learning and corresponding models. In particular, I am very grateful for the comments and instructions from you, which helped me see my own mistakes and weaknesses, thereby helping me to improve and develop myself.

Through the process of completing this report, I have had the opportunity to practice and improve many important skills. The process of solving problems arising in the project helped me learn how to handle arising difficulties, and also exercised my ability to think logically and analyze problems.

In addition, completing this report also helps me grasp important methods and techniques in the field of machine learning. I had the opportunity to apply the knowledge and programming techniques I learned to real projects, thereby improving my programming skills and better understanding the architecture and working of machine learning and learning models. deep.

Moreover, I am very grateful to IS LAB, for giving me the opportunity to participate, study and accompany my friends during the time I was working on this report. The dedication of the brothers and sisters, the discussions of experience, as well as the professional theory, helped me to update a lot of new knowledge, strengthen my old knowledge, change my opinion a lot. its shortcomings.

Once again, I would like to thank the lecturers for their support, contributions and dedication during the preparation of this report.

Ho Chi Minh City, July 17, 2023

Table of contents

List of images

List of acronyms

ANN Artificial Neural Network. 2, 5, 15

CNN Convolutional Neural Network. 2, 5, 23, 35

DNN Deep Neural Network. 5, 23

LSTM Long short-term memory. 5, 28

NLP Natural Language Processing. 5

RNN Recurrent Neural Network. 2, 5, 23, 24, 28

Report summary

In this report, I focus on research on the application of deep learning techniques to natural language processing problems and present the results of the research. The report is divided into the following chapters:

- **Chapter 1:** Overview of the topic: This chapter introduces the content of the topic, including research motivation, problem statement, challenges, objectives, scope and research object. In addition, we also present the main contributions of this study.
- **Chapter 2:** Theoretical Basis: This chapter provides an overview of the basic theories related to natural language processing. In addition, we also present surveys on NLP approaches from previous studies.
- **Chapter 3:** Experimental Methods: This chapter introduces the basic concepts of NLP and presents traditional statistical machine learning algorithms as well as advanced deep learning algorithms used in the research.
- **Chapter 4:** Data Processing: This chapter focuses on data preprocessing, including data cleaning methods, and basic data processing. This process ensures that the data is well prepared for application to algorithms for spam reviews detection.
- **Chapter 5:** Experimentation and evaluation: This chapter presents the experimental and evaluation of the study. It describes the experiments that have been carried out to evaluate the performance of spam reviews detection algorithms and methods. Evaluation results are presented as metrics such as Accuracy, and F1-macro.
- **Chapter 6:** Conclusion and development direction: The last chapter summarize the main results achieved in the topic and the future development direction of the topic.

1 OVERVIEW OF THE TOPIC

1.1 Research motivation:

In today's digital age, e-commerce websites play an important role in providing information and creating a favorable online shopping experience for users.

As we all know, comments and reviews from users who have purchased a product online play an important role in providing feedback on the quality, features and user experience of the product. Shared opinions and experiences help others have a more objective view of the product and it plays an important role in helping buyers better understand the advantages and disadvantages of the product, thereby giving Make smarter buying decisions. However, an increasingly serious problem is the proliferation of spam comments and reviews that are rampant on these sites. Spam reviews are not only annoying and wasting users' time, but also affect the trustworthiness and reputation of e-commerce websites. Review spam may contain irrelevant advertising content, malicious links, or misinformation intended to defraud users. This makes it difficult for users to filter useful information from a large number of spam reviews.

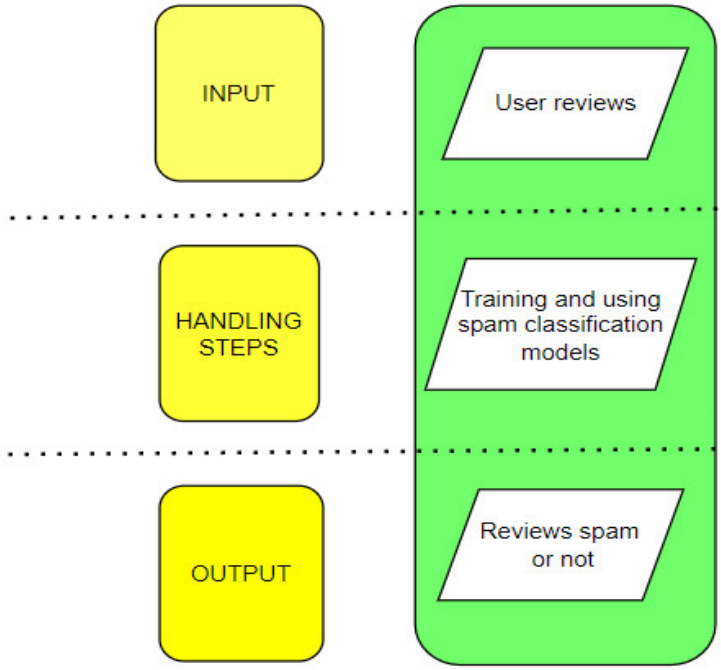
Therefore, the goal of the project is to build an effective spam reviews detection system for e-commerce websites. By applying advanced machine learning methods and diverse training data, we hope to be able to create a robust and flexible system that will be able to detect new forms of spam and minimize its impact. their impact on users and the reputation of the e-commerce site.

1.2 Problem statement

- Input: Input data for the problem includes user reviews on e-commerce websites. Each review will be represented as text and encoded as a set of arithmetic features, in this case a binary classification with the value 1 representing the spam

review and the value 0 representing the review that are not spam.

- Handling Steps: The processing involves building a machine learning model to classify reviews into two groups: spam and non-spam. First, the training data will be used to train the model, where the model will learn from the features of the labeled reviews (spam or non-spam).
- Output: The model will apply the feature of each new review to the trained model and predict the corresponding label for the review: spam or non-spam.



Hình 1: Problem statement with Input, Handling Steps, Output

1.3 Challenges

- Variety of Spam Reviews: Detecting spam reviews on e-commerce sites is a significant challenge due to increasingly complex and sophisticated forms of spam.

- Real-time and big data processing: E-commerce websites can receive thousands or even millions of reviews daily. To handle this large amount of data and ensure fast response times, spam detection algorithms need to be optimized and use efficient data processing methods.

- Accuracy and false detection rate: the accuracy and processing speed of the algorithm are also important factors in the successful implementation of this solution. Although spam detection algorithms have been developed, there can still be cases of errors when classifying comments. Algorithms need to ensure high accuracy in spam detection, while reducing false detection rates to avoid confusing valid comments.

- Unbalanced training data: The majority of comments on e-commerce sites are non-spam, while only a small percentage are spammy. This leads to an imbalance in the training data and can affect the performance of spam detection models.

1.4 Objectives of the topic

- Get a detailed overview of traditional statistical and deep learning machine learning algorithms.

- Gain knowledge of data preprocessing techniques to improve the performance of spam detection systems

- Install and test the spam rating detection algorithm on the built data set.

- Analyze, evaluate and synthesize research results into reports.

1.5 Research scope and research object

- Research scope:

  - The topic focuses on detecting spam comments on e-commerce websites. E-commerce sites may include online sales platforms, online stores, buying and selling forums, and similar e-commerce platforms. The topic is not limited to a specific field or industry, but can apply to any e-commerce website.
  - Research and survey published works on how features and methods are chosen to detect spam reviews
  - Researching machine learning models towards deep learning (ANN, RNN, LSTM, CNN)

- Research object:
  - The main object of the research is comments and reviews from users on e-commerce websites. These reviews can be submitted by users who have purchased online or have not, regardless of age, gender, or geographic origin. The object of the study also included spam review samples from various sources.

2 THEORETICAL BASIC

In the field of spam reviews detection on e-commerce websites, many previous studies have been done:

- "Detecting spam customer reviews using machine learning techniques" (2014): This research focuses on using machine learning algorithms such as SVM and Naive Bayes to detect spam reviews in customer reviews on an e-commerce website.

- "Detecting Review Spam: A Survey and Future Directions" (2015): This study presents a survey of spam detection methods in customer reviews. The research focuses on analyzing comment characteristics and using machine learning algorithms such as SVM, Random Forest and K-means to detect spam comments.

- "A survey on techniques for spam detection" (2016): This study presents a survey of spam detection methods in general, including comment spam on e-commerce websites. The study introduces data preprocessing methods, such as stemming and stop words removal, as well as the use of machine learning algorithms such as SVM, Naive Bayes, and deep learning techniques for spam detection.

- "An evaluation of machine learning methods for spam review detection" (2019): This study compares the performance of different machine learning algorithms in spam detection in customer reviews. Research uses algorithms like SVM, Random Forest, and MLP to classify reviews as spam or non-spam.

- “Detecting Spam Reviews on Vietnamese E-commerce Websites” (2022): propose a method to detect spam reviews about products on online shopping platforms. First, constructing a corpus for spam detection from users’ reviews by texts. Second, using machine learning approaches to build classification models for detecting spam reviews and evaluate classification models’ performances on the constructed dataset.

3 EXPERIMENTAL METHODS

3.1 Classification technique:

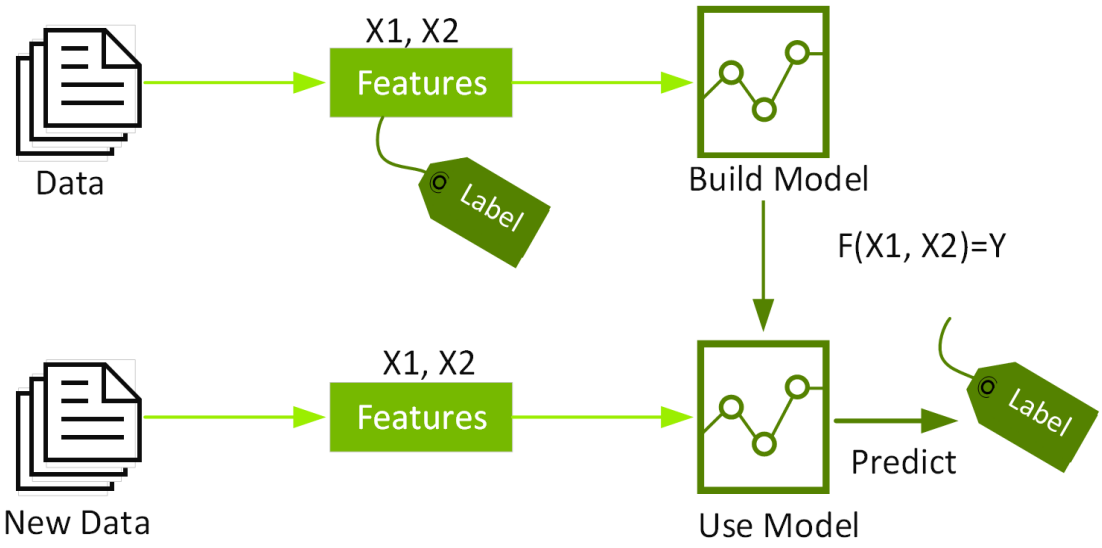
- In machine learning, a distinction has traditionally been made between two major tasks: supervised and unsupervised learning.
  - Supervised learning accounts for a lot of research activity in machine learning and many supervised learning techniques have found application in the processing of multimedia content. The defining characteristic of supervised learning is the availability of annotated training data. The name invokes the idea of a 'supervisor' that instructs the learning system on the labels to associate with training examples. Typically, these labels are class labels in classification problems. Supervised learning algorithms induce models from these training data and these models can be used to classify other unlabeled data.
  - Supervised learning entails learning a mapping between a set of input variables X and an output variable Y and applying this mapping to predict the outputs for unseen data. Supervised learning is the most important methodology in machine learning, and it also has a central importance in the processing of multimedia data.
  - In unsupervised learning, on the other hand, no specific output value is provided. Instead, one tries to infer some underlying structure from the inputs. For instance, in unsupervised clustering, the goal is to infer a mapping from the given inputs (e.g., vectors of real numbers) to groups such that similar inputs are mapped to the same group.
- For this classification problem, I use supervised learning method. Based on labeled training data (spam and non-spam), I can build a classification model to predict whether a new review is spam or not.

3.2 Machine Learning Algorithms

3.2.1 eXtreme Gradient Boosting

- XGBoost (eXtreme Gradient Boosting) is a machine learning algorithm that belongs to the Gradient Boosting family. It was developed by Tianqi Chen at the University of Washington and has become one of the most popular and powerful machine learning algorithms in the data science community.

- XGBoost combines the strengths of two key components: Gradient Boosting and Decision Trees. Gradient Boosting focuses on building predictive models by continuously optimizing a loss function through the addition of decision trees to the model. Decision Trees, on the other hand, are simple decision-making structures used to derive decision rules based on the features of the data



Hình 2: Supervised machine learning uses algorithms to train a model to find patterns in a dataset with labels and features and then uses the trained model to predict the labels on a new dataset's features

Some important features of XGBoost are:

  - Regularization: XGBoost supports regularization techniques such as L1 regularization (Lasso) and L2 regularization (Ridge) to control overfitting and

enhance the generalization ability of the model.

- Gradient-based Optimization: XGBoost utilizes a gradient-based optimization algorithm to improve the model. It calculates the gradient and hessian of the loss function to efficiently find local minima.
- Parallel Processing: XGBoost supports parallel processing to speed up the training and prediction processes. It can run on multiple threads and effectively utilize computational resources.
- Handling Missing Values: XGBoost has the ability to handle missing values in the input data automatically.
- Feature Importance: XGBoost provides tools to evaluate the importance of features in the data. This helps identify the most important features and understand their role in making predictions.

  XGBoost has been proven to be effective and powerful in various machine learning tasks, including classification, regression, and ranking. It has received numerous awards and is widely used in both research and real-world applications

3.2.2 K Nearest Neighbors

- What is KNN and how does it work?
  - KNN , or K Nearest Neighbor is a Machine Learning algorithm that uses the similarity between our data to make classifications (supervised machine learning) or clustering (unsupervised machine learning).
  - With KNN we can have a certain set of data and from it draw patterns that can classify or group our data.
- But how exactly does it work?
  - The concept of neighborhood depends on the idea that those close to us tend to be more like us.

  - From this notion, what KNN (very generically) does is create neighborhoods in our dataset and as we pass other data samples to the model it will return us on "which neighborhood our sample would best fit" !

3.3 Deep Learning Algorithms

3.3.1 Artificial Neural Network (ANN)

- Why is it called a neural network?
  - It is called a neural network because it is a network of interconnected elements. These elements were inspired from studies of biological nervous systems. In other words, neural networks are an attempt at creating machines that work in a similar way to the human brain by building these machines using components that behave like biological neurons.
- What does a neural network do?
  - The function of a neural network is to produce an output pattern when presented with an input pattern. This concept is rather abstract, so one of the operations that a neural network can be made to do - pattern classification - will be described in detail. Pattern classification is the process of sorting patterns into one group or another.
- Neural networks are made up of units of neurons called perceptors. A perception can take many inputs. However, it only outputs a single output value. For any output of a perceptron will be passed an activation function to transform the output value to fit the problem semantics.
- A Multi-layer perceptron (MLP) network is a model with layers in which the perceptrons in the current layer are fully connected from the perceptions of the previous layer. Each model has an input layer, an output layer, and one or more intermediate layers called hidden layers. The number of layers of an MLP model

is calculated as the number of hidden layers and an output layer. Input is not counted as a layer. In the MLP model the perceptron units in a layer are called units or nodes.

- As mentioned earlier, the nodes in the later layer will be fully connected to the nodes in the previous layer. Connections will have a distinct weight. The value of each previous node is multiplied with the weight to pass to the node of the next layer. The value of a node in any layer will be equal to the sum of the product of all values with the weights of the nodes from the previous layer, passing through a particular activation function. The general formula is presented as follows:

$$a_i^{(l)} = f(w_i^{(l-1)T} a^{(l-1)} + b_i^{(l)})$$

  Where, a is the value of the node where l is the layer to which the node belongs and i is the position of the node in that layer, w is the weight of the node participating in the connection with the node of the next layer, and b is the coefficient. bias of a node.

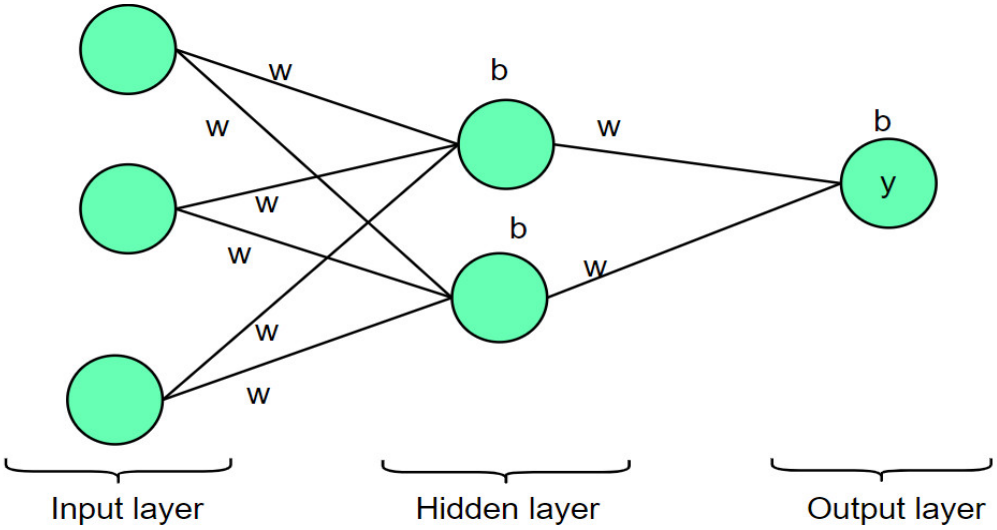
- When the artificial neural network consists of a large number of processing parts connected together in many layers, it will develop into a deep learning neural network - Deep Neural Network (DNN). The basic neural network item structure consists of 3 parts:
  - **Input:** Acts as a way to put input information into the processing network.
  - **Hidden layer:** Works by transforming the input information with the corresponding link weight, then passing an activation function. When the hidden layer is larger than two, the neural network is now a deep learning network.
  - **Output:** Through the previous hidden layers, the output state will be determined according to the input of the link weights, going to the corresponding output unit.

- The state of the output and hidden units in the neural network is also determined by the activation function. The purpose of the activation function is, besides introducing nonlinearity into the neural network, to bound the value of the neuron so that the neural network is not paralyzed by divergent neurons. A common example of the activation function is the sigmoid (or logistic) function. The activation function is responsible for adjusting and transforming the output and state of the computational units (neurons) in the network. The activation function helps the neural network model become more diverse and suitable for many real-life tasks. There are three main groups of activation functions:
  - **Linear Function or identity function:** This function keeps the sum of the products between the input and the weight associated with the calculation unit. In other words, the output of the computational unit will be equivalent to the input it receives. The linear activation function has no effect on the derivative during the backpropagation during the training of the model, and the consecutive linear connections are also not significant since the final output is also equivalent to the linear output of the first layer.
  - **Non-linear Function:** A nonlinear activation function that transforms the sum of the input products and the associated weights into a corresponding value. There are two basic groups of nonlinear functions:
    - *Limit function group:* The activation function limits the output state to a certain range of values, such as the limit in the range [0, 1] or [-1, 1]. Typical examples are the Sigmoid and Tanh functions.
    - *Unlimited function group:* The activation function is not limited or partially unlimited, can go to negative infinity or positive infinity. Some representative activation functions in this group include the Rectified Linear Unit (ReLU), Leaky ReLU, and Exponential Linear Unit (ELU).
  - **Step Function:**This type of activation function converts the output into two unique states "active" or "inactive". Usually, the activation value is normal-

ized to 0 and 1 for convenience in the calculation process. The process of activating the output value is based on a specified threshold, where if the input exceeds the threshold, the output status will be "enabled", otherwise it will be "not activated".

The choice of activation function depends on the nature of the problem and the characteristics of the data. The use of nonlinear functions makes the neural network model more powerful in learning complex features and representing nonlinear relationships.

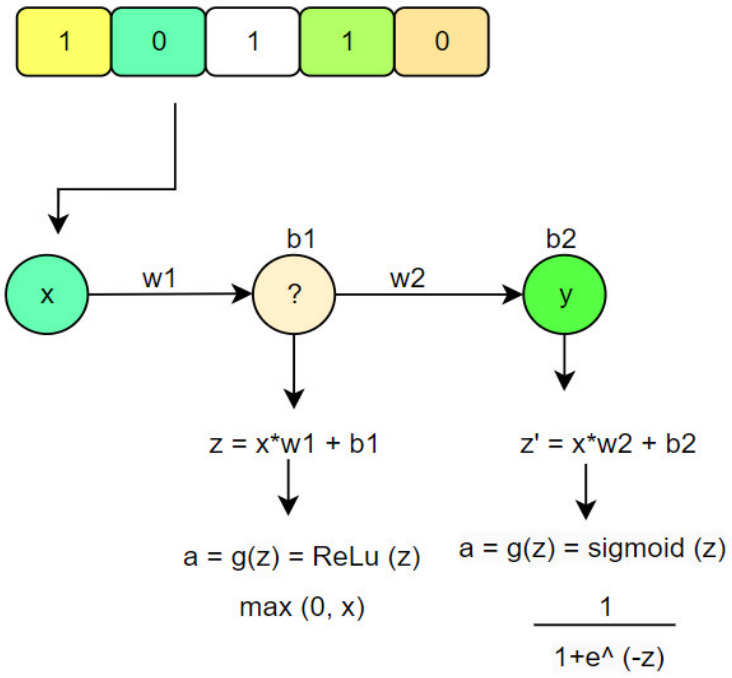
A basic neural network model can be described specifically as follows:



Hình 3: Illustrate the architecture of a neural network

Neural network model with input consisting of 3 features, two hidden layers, an output layer with a single output, where w is the weight of the connection between the computational unit of this layer and the computational unit in the layer. Next, b is the bias value of a unit of computation.

Suppose we have a neural network with the following structure, in which the weights and bias are randomly initialized. The first data sample is flattened into a single vector and fed into the model: [1 0 1 1 0]



Hình 4: A sample of a neural network

The output of a computational unit in a convolutional neural network is generalized by the following formula:

$$Output = Activation(x * w + b)$$

Where, x is the input or output characteristic of the previous computational unit, w is the connection weight, and b is the bias value of the current computation unit. Suppose we have a computational unit in a convolutional neural network with a ReLU activation function in the hidden layer and a Sigmoid activation function in the output layer. We have the following values:

- Hidden layer (ReLU activation):
  - Input of the calculation unit (x): the input or output characteristic value of the previous calculation unit.
  - Connection weight (w): weight value associated with the current calculation unit.
  - Bias value (b): bias value of the current calculation unit.

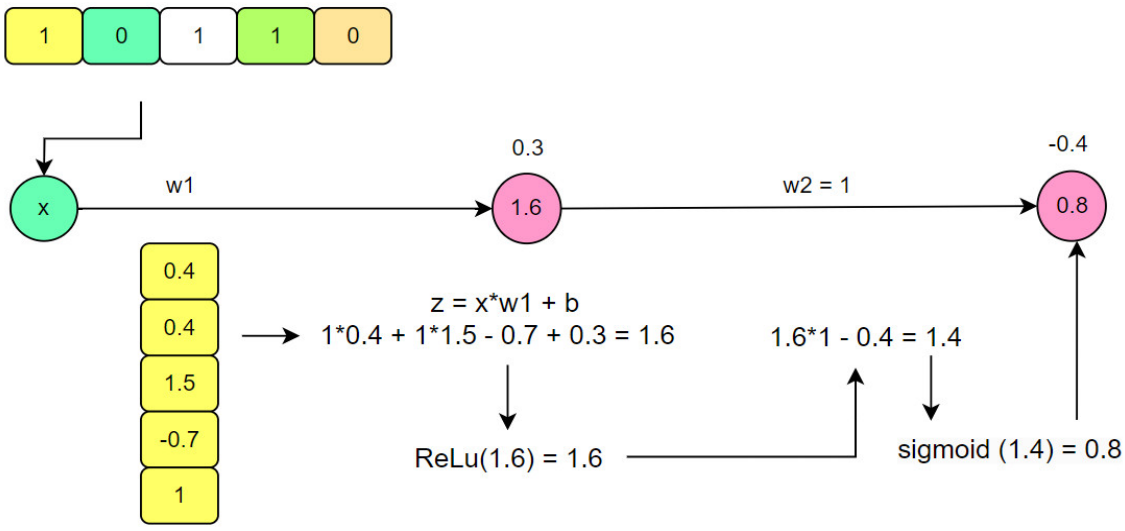
* Output: $Output = ReLU(x * w + b)$

– Output layer (Sigmoid activation):

* Input of the calculation unit (x): the input or output characteristic value of the previous calculation unit.
* Connection weight (w): weight value associated with the current calculation unit.
* Bias value (b): bias value of current calculation unit.
* Output: $Output = Sigmoid(x * w + b)$

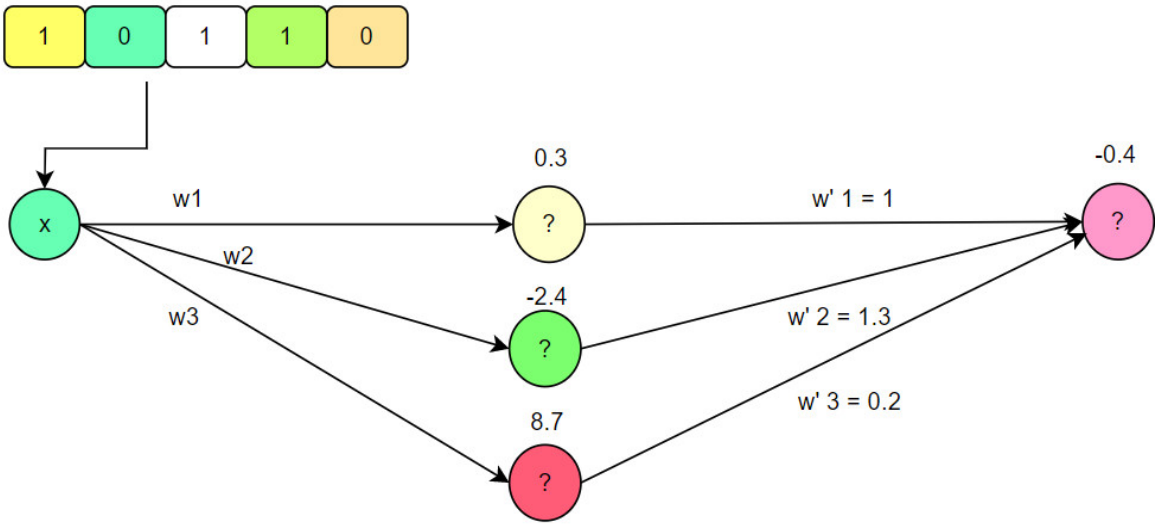
In both layers, the input (x) can be the input feature value of the convolutional neural network or the output of the previous computational unit. The connection weight (w) and bias value (b) are used to adjust and produce the output of the computational unit. The ReLU activation function is applied in the hidden layer to produce a non-linear output, while the Sigmoid activation function is applied in the output layer to produce a bounded output in the interval [0, 1], which is usually used to represent probabilities.

Illustrate a neural model with input weights, biases, and features fed into the network. After the calculation steps, we will have the following result:



Hình 5: Illustration of a neural network model with a fully computed unit

Assuming the hidden layer has more than 1 unit, how will the calculation be done?

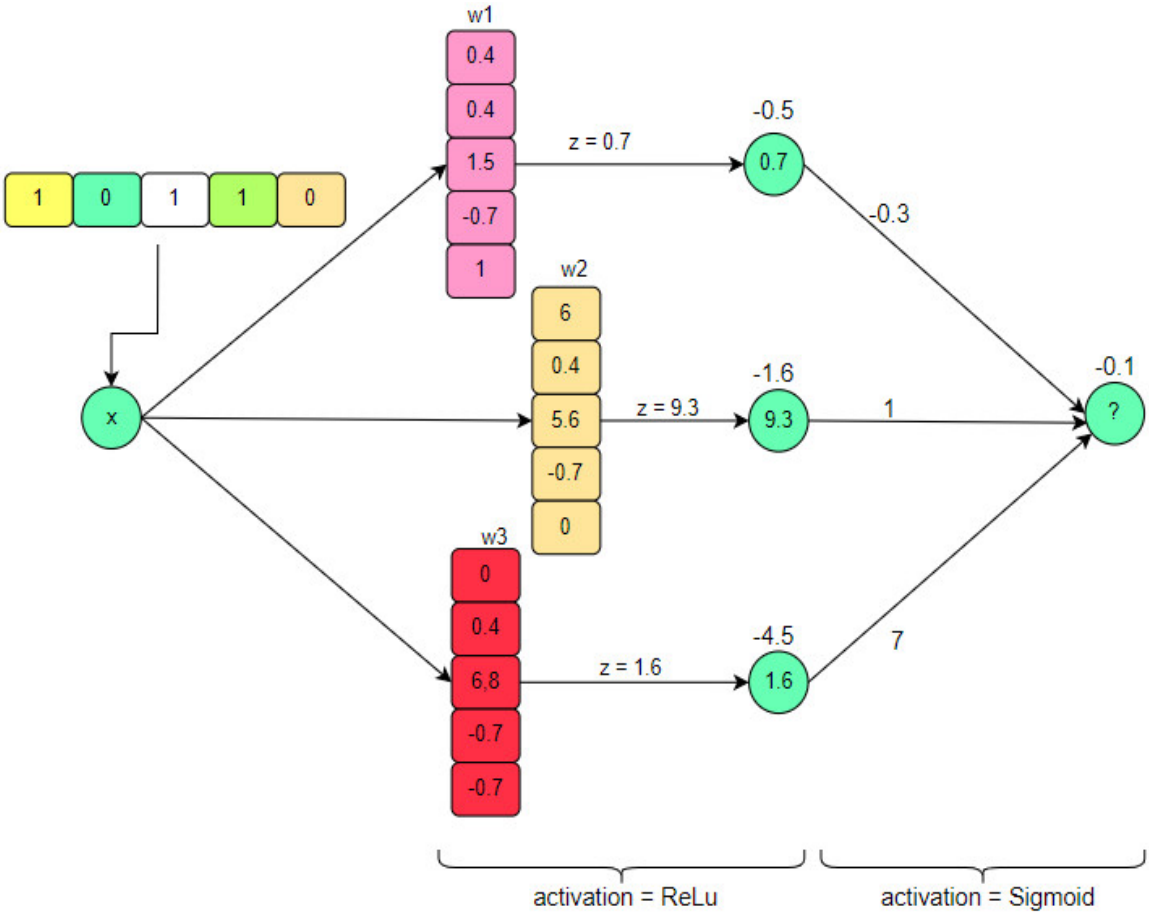


Hình 6: Neural network illustration with many units

The calculation will be similar to having 1 unit:

The values of the 3 calculation units in the hidden layer are presented as follows:

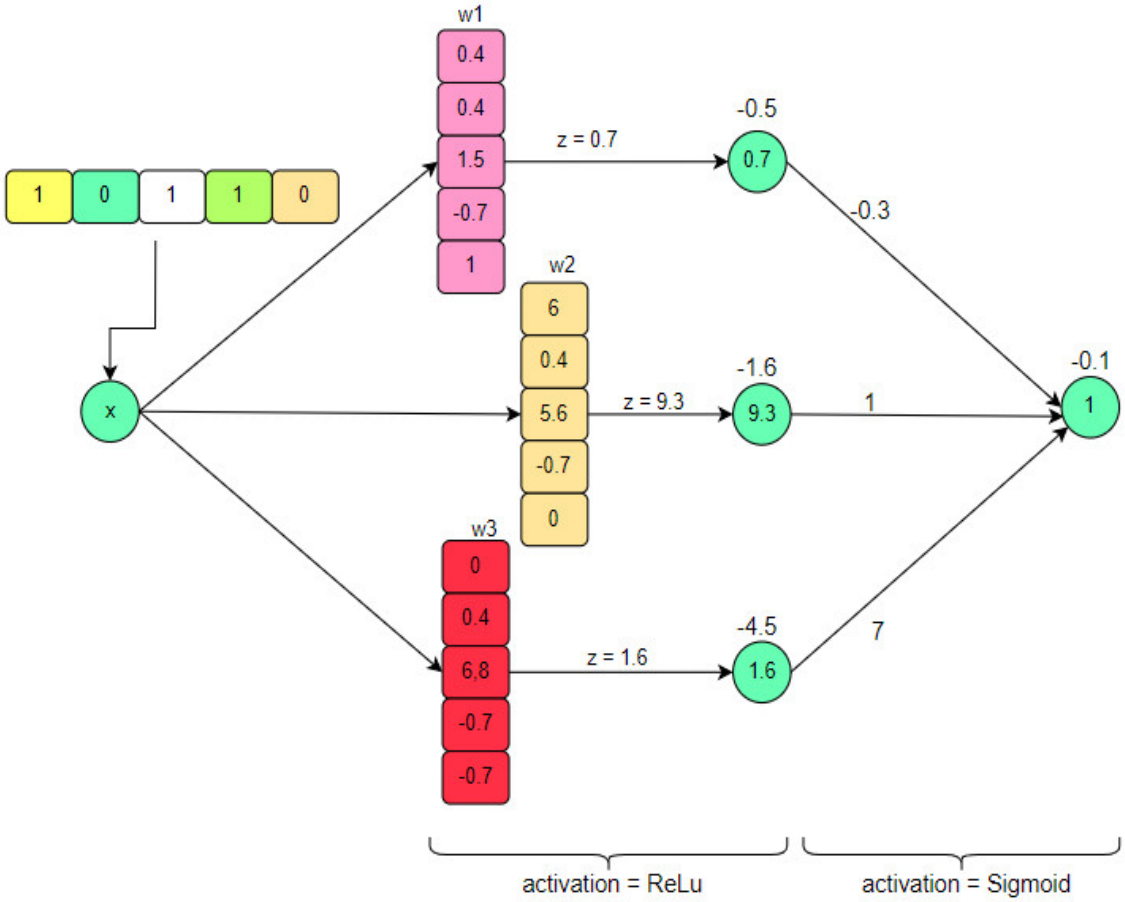
- $ReLU(0.4 + 1.5 \check{} 0.7 \check{} 0.5) = 0.7$
- $ReLU(6 + 5.6 - 0.7 - 1.6) = 9.3$
- $ReLU(6.8 - 0.7 - 4.5) = 1.6$



Hình 7: Value of 3 calculation units in hidden layer

Finally, the output is also calculated with the same idea where the output layer uses the Sigmoid activation function:

– $Sigmoid(0.7 * (-0.3) + 9.3 + 1.5 * 7\tilde{}0.1) = 0$



Hình 8: illustrate the neural network model with 3 fully calculated units

3.3.2 Recurrent Neural Network (RNN)

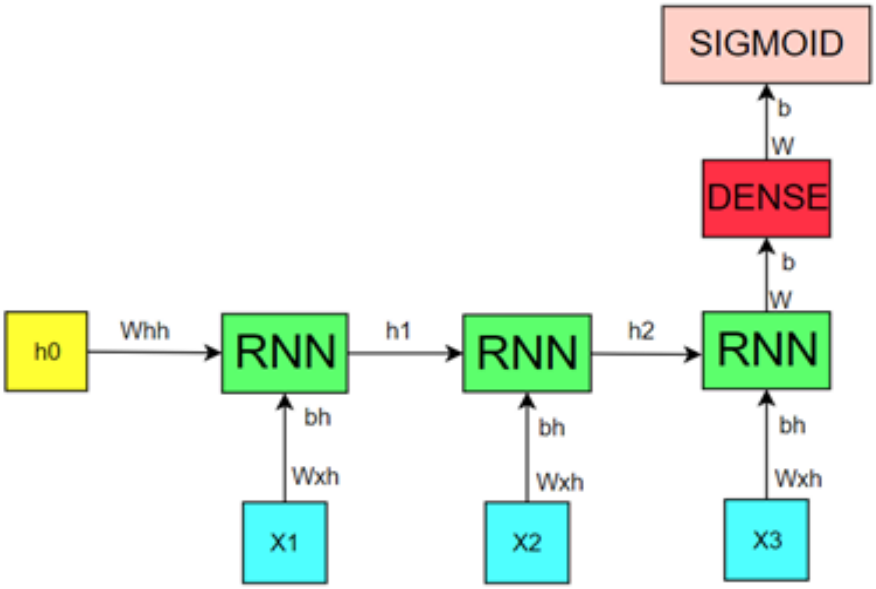
- Recurrent neural network (RNN) is an artificial neural network model designed to process sequential or time series data in the context of a classification problem. The special feature of RNN compared to forward models such as basic Deep Neural Network (DNN) or CNN is that it has a component that acts as memory, which helps to store information from previous inputs to add semantics to the current input. While other neural network models assume that the input and output are independent of each other for each sample, any output of a recurrent neural network depends on previous sequential samples over time.

- Another feature of the regression network is the sharing of information back and

forth between the layers through the multiplication between the output value of the current node and the weight for the shared node.

- By using data serialization and recurrent connections in the network, the 0 model can capture time dependencies and contextual information important for sequence classification tasks. This makes RNN particularly well-suited for tasks such as text classification, speech recognition, sentiment analysis, and machine translation, where the order and context of input data play an important role in define output.

- A basic regression network model can be specifically described as follows:



Hình 9: Illustration of a Recurrent Neural Network

The above diagram illustrates a recurrent neural network (RNN) model with an input layer, recurrent layers (RNN layers), a hidden Dense layer, and an output layer with a sigmoid activation function. The parameters involved in the RNN model are as follows:

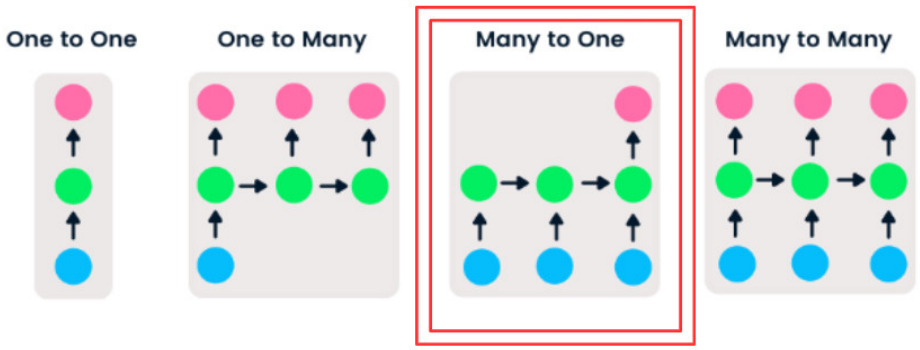
  - Wxh: The weight connecting the input units to the hidden units in the recurrent layers.

  - Whh: The weight connecting the hidden units to themselves within the recurrent layers and between the units in the same layer.
  - b: The bias value for the computation units.

- The input layer processes the initial input and passes it to the middle layer RNN. The middle layer consists of multiple hidden layers, each with its activation functions, weights, and biases.

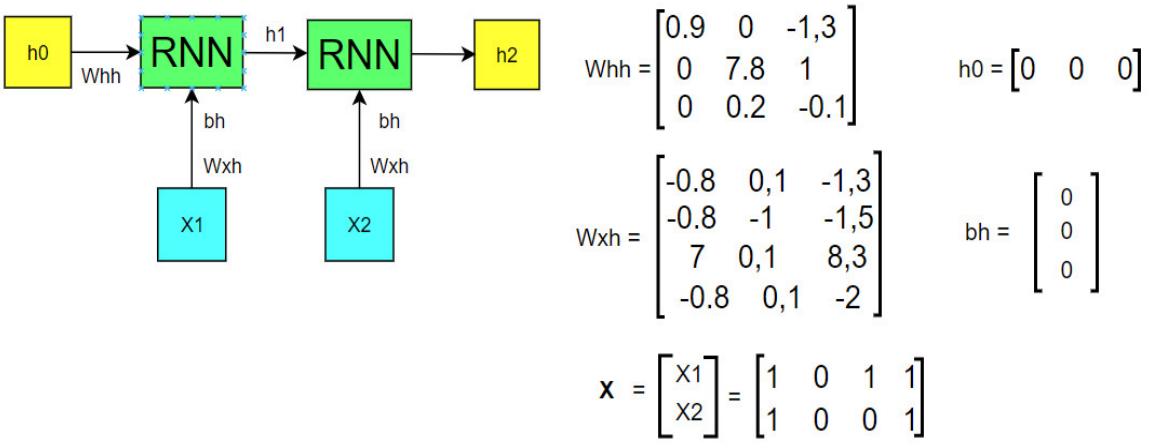
- These parameters are standardized across the hidden layer so that instead of creating multiple hidden layers, it will create one and loop it over.

- There are four types of RNN based on different lengths of inputs and outputs.
  - One-to-one is a simple neural network. It is commonly used for machine learning problems that have a single input and output.
  - One-to-many has a single input and multiple outputs. This is used for generating image captions.
  - Many-to-one takes a sequence of multiple inputs and predicts a single output. It is popular in sentiment classification, where the input is text, and the output is a category.
  - Many-to-many takes multiple inputs and outputs. The most common application is machine translation.



Hình 10: Four types of RNN

Let's consider a recurrent neural network (RNN) model with the following structure, input data samples with randomly initialized weights and biases as depicted in the diagram below:



Hình 11: A recurrent neural network (RNN) with randomly initialized weights and biases

The output of a computational unit in a recurrent neural network is generalized by the following formula:

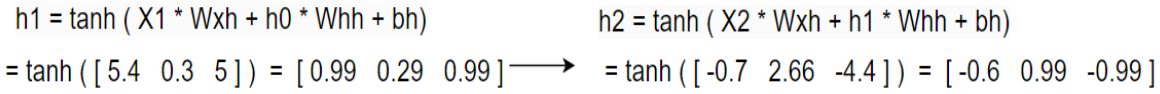
$$Output = (W * X + b) + (Wo * h(t-1))$$

Where:

- X is the matrix of input feature values that we want to extract information from.
- W is the weight matrix connecting the computational units of the current layer to the next layer.
- Wo is the weight matrix for the h state value used for recursion and sharing information with other computational units in the same layer.
- b is the bias vector containing the bias values for each computational unit.
- h(t-1) is the h state value of the previous computational unit, representing the previous state of the RNN network.

The process of computing the state of the computational units in the RNN network is done by using the input X and the previous state h(t-1), combined with the weights W and Wo along with the bias b. The result is the output value (Output) of the current computational unit.

This allows the RNN network to store and propagate information through previous states, creating the ability for recursion and information sharing within the network. The initial value of h state t - 1 is set to 0, and the state of computational units in the RNN layer is computed as follows:



Hình 12: Illustration of a complete computation in a recurrent neural network

$$\text{Where,} \qquad tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

In the process of backpropagation in an RNN model, two important issues can arise: Gradient Explosion and Gradient Vanishing.

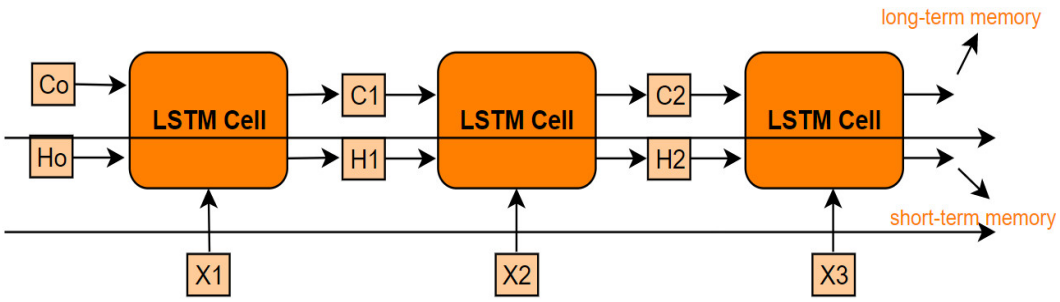
- Gradient Explosion: This occurs when the gradient values become very large, exceeding the permissible threshold. When the gradient values are too large, the weight update process becomes unstable, and the training of the model becomes unpredictable. As a result, the loss function during training can increase rapidly, even surpassing the NaN (not a number) threshold. This issue often occurs when the weights of the RNN are too large or when the gradient is propagated through multiple previous states and accumulates uncontrollably.
- radient Vanishing: This occurs when the gradient values become very small, close to zero. When the gradients are too small, the weight update process does not significantly change the states of the network. This means that the model cannot learn information from previous states and struggles to learn

long-term dependencies in sequential data. This issue often occurs when the RNN has too many layers or when using activation functions like Sigmoid or Tanh.

Handling Gradient Explosion and Gradient Vanishing is a crucial challenge in training RNNs and requires careful consideration and appropriate adjustments of the model's architecture and parameters.

3.3.3 Long Short-Term Memory (LSTM)

- Long short-term memory (LSTM)is an improvement of RNN designed to overcome the limitations of RNNs in learning long-term dependencies. Long-term dependencies refer to the influence of input values from the beginning to the output values. LSTM has the ability to remember and maintain information over a long period of time, which is challenging for traditional RNNs.

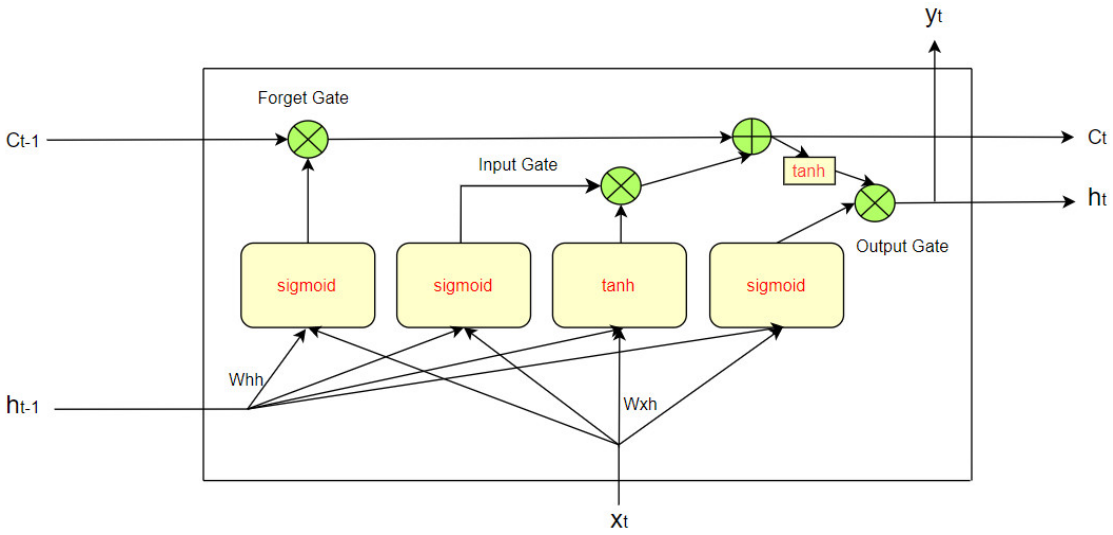


Hình 13: LSTM Model

- One important feature of LSTM is its ability to automatically retain information over long periods of time without the need for special training processes. It has a sequence-like structure similar to RNN, but LSTM modules have a different architecture.

- An LSTM unit is composed of four main components: an input gate, a self-recurrent connection, a forget gate, and an output gate. These gates regulate the

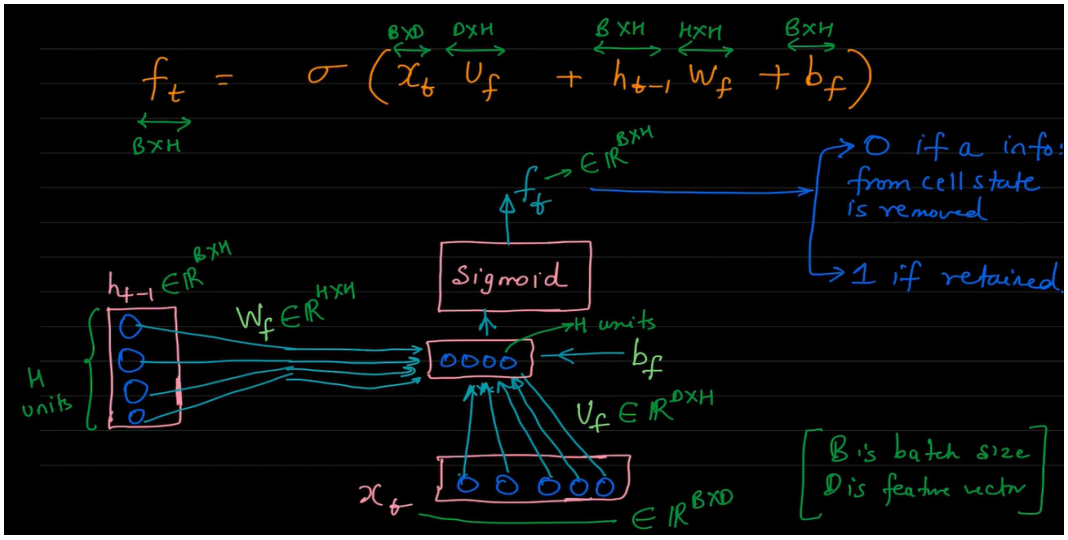
flow of information through each LSTM module and determine which information should be stored and which information should be disregarded. The input gate controls whether an incoming signal can modify the cell state. Similarly, the output gate determines whether the cell state has an impact on other units. The forget gate enables the cell to remember or forget its previous state by controlling the cell's self-recurrent connection. An LSTM layer with multiple units can be seen as a deep network across time steps, where each time step represents a layer.



Hình 14: Detailed LSTM Model

Based on the architecture illustrated in the figure above, we see:

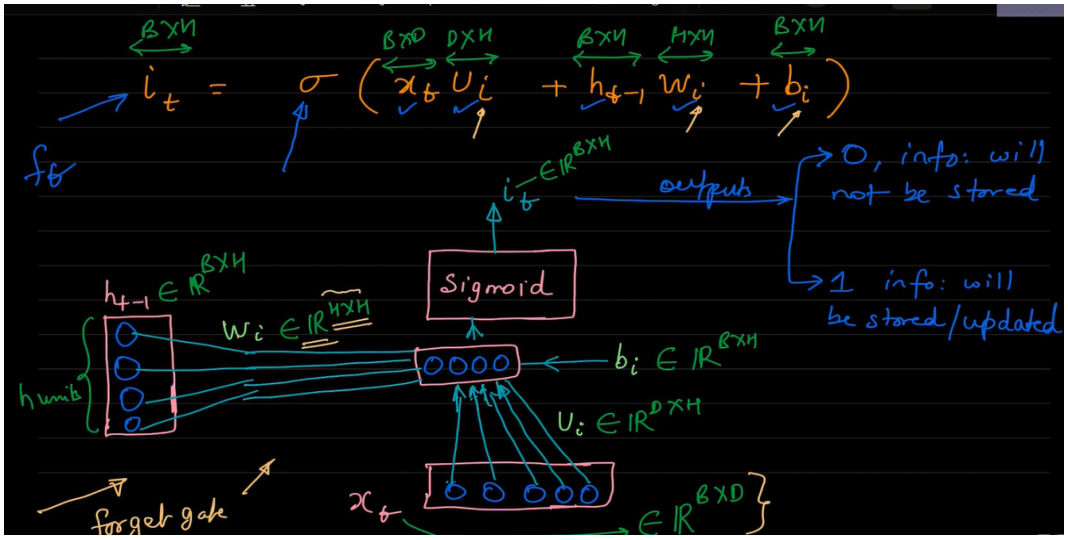
– **Forget Gate:** This gate decides what information should be thrown away or kept. Information from the previous hidden state and information from the current input is passed through the sigmoid function. Values come out between 0 and 1. The closer to 0 means to forget, and the closer to 1 means to keep.



Hình 15: Forget Gate

* Input: Previous cell state (c) and current input feature (x).
* Function: Determine which information needs to be forgotten from the previous cell state.
* Computation: Use a sigmoid function to map the input (c and x) to a value between 0 and 1. This value indicates the extent to which the information from the previous state should be forgotten.

– **Input Gate:** First, we pass the previous hidden state and current input into a sigmoid function. That decides which values will be updated by transforming the values to be between 0 and 1. 0 means not important, and 1 means important. We also pass the hidden state and current input into the tanh function to squish values between -1 and 1 to help regulate the network.



Hình 16: Input Gate

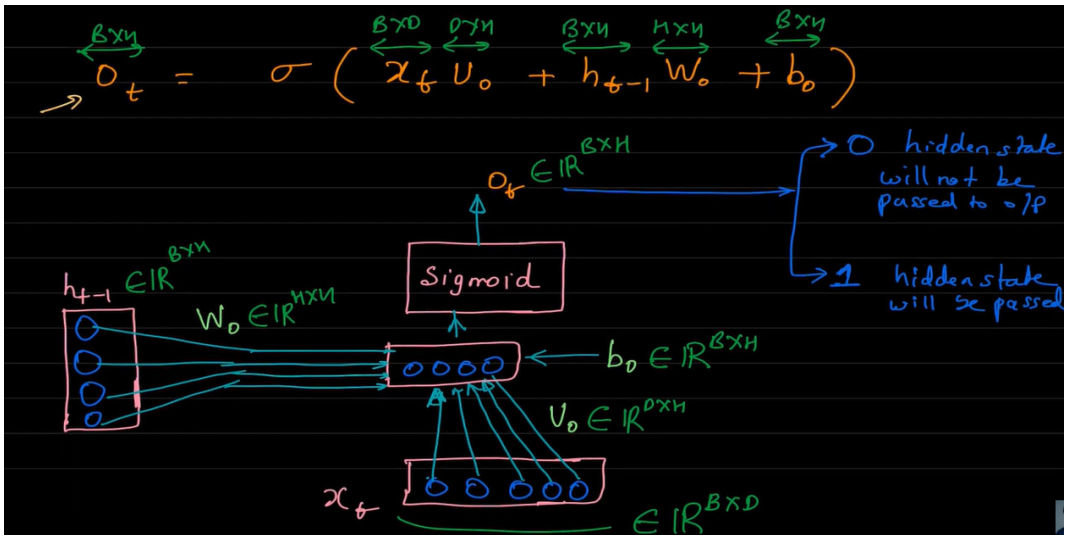
* Input: Previous cell state (c) and current input feature (x).
* Function: Responsible for deciding what information should be stored in the cell state.
* Computation: It consists of two parts. First, it uses a sigmoid function to determine which values in the input (c and x) should be updated. Second, it uses a tanh function to generate a vector of proposed new values for the cell state.

Example: Mark is a good singer. He lives in California. Jacob is also a good singer.

$\Longrightarrow$ The memory of Mark should be removed, and Jacob is the new subject. We can see the subject has changed from Mark to Jacob.

– **Output Gate:** The output gate decides what the next hidden state should be. Remember that the hidden state contains information on previous inputs. The hidden state is also used for predictions. First, we pass the previous hidden state and the current input into a sigmoid function. Then we pass the newly modified cell state to the tanh function. We multiply the tanh output with the sigmoid output to decide what information the hidden state should carry. The output is the hidden state. The new cell state and the new hidden is then

carried over to the next time step.



Hình 17: Output Gate

* Input: Previous state of LSTM (h(t-1)) and current input feature (Xt).
* Sigmoid Activation Function: The input is multiplied by a weight matrix and passed through the Sigmoid function to generate a vector of values ranging from 0 to 1 for each component of the current state (ht).
* Tanh Activation Function: The input is multiplied by a weight matrix and passed through the Tanh function to generate a vector of values ranging from -1 to 1 for each component of the current state (ht).
* Output Vector: The value after passing through the output gate is computed by multiplying the output vector of the output gate with the current state (ht).

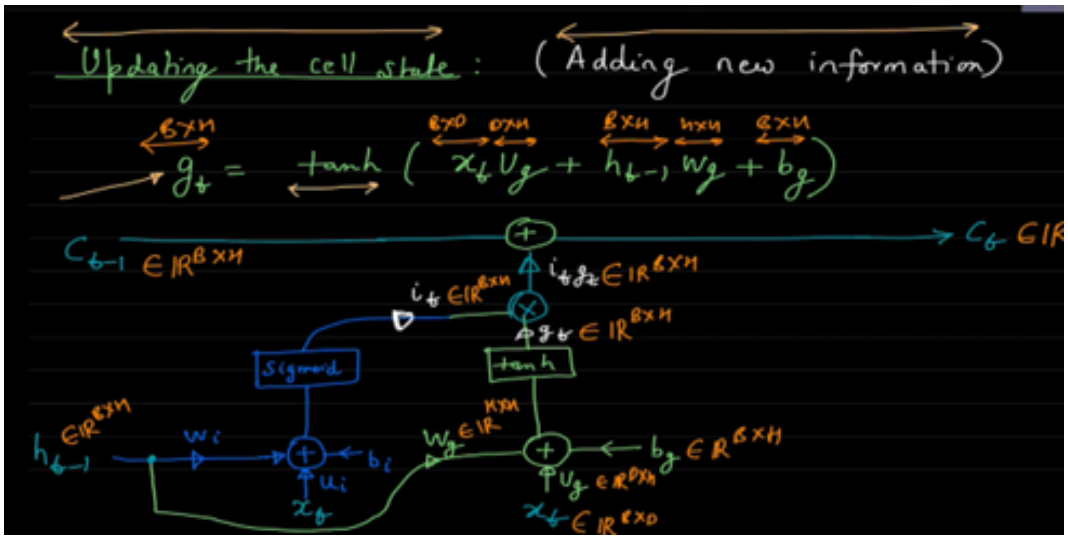
Consider the following sentence: Jacob debut album was a hug success. Congrate ..... $\Longrightarrow$ Jacob

– **Updating the Cell State:**

* · Input: Previous state of the cell (c), current input feature (x), forget gate, and input gate.
  · Function: Update the state of the cell (c) based on the decision information from the forget gate and input gate.

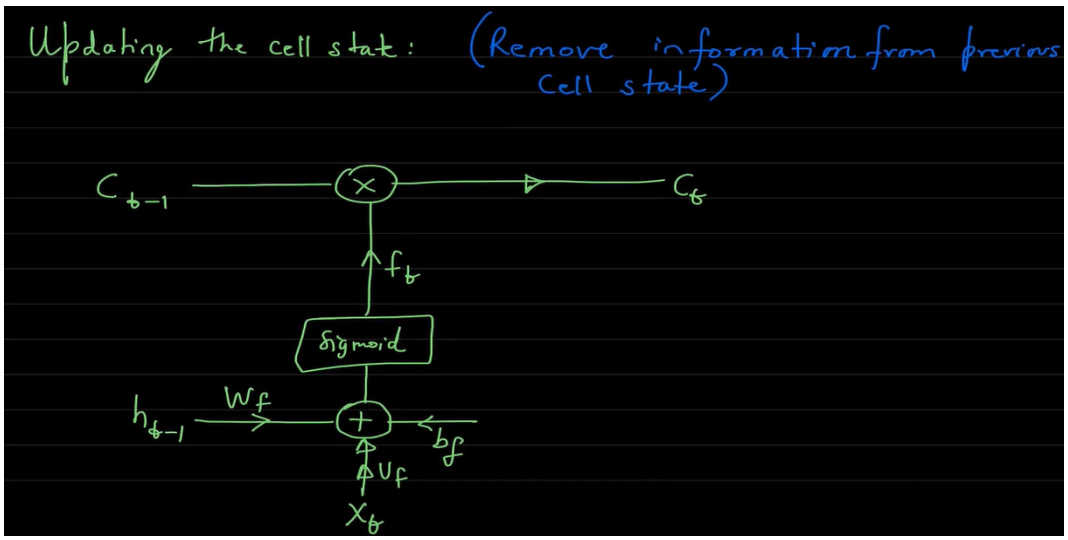
· Computation: Multiply the information from the forget gate with the previous state of the cell and add it to the information from the input gate, then adjust using the Tanh function.

∗ Adding new information:



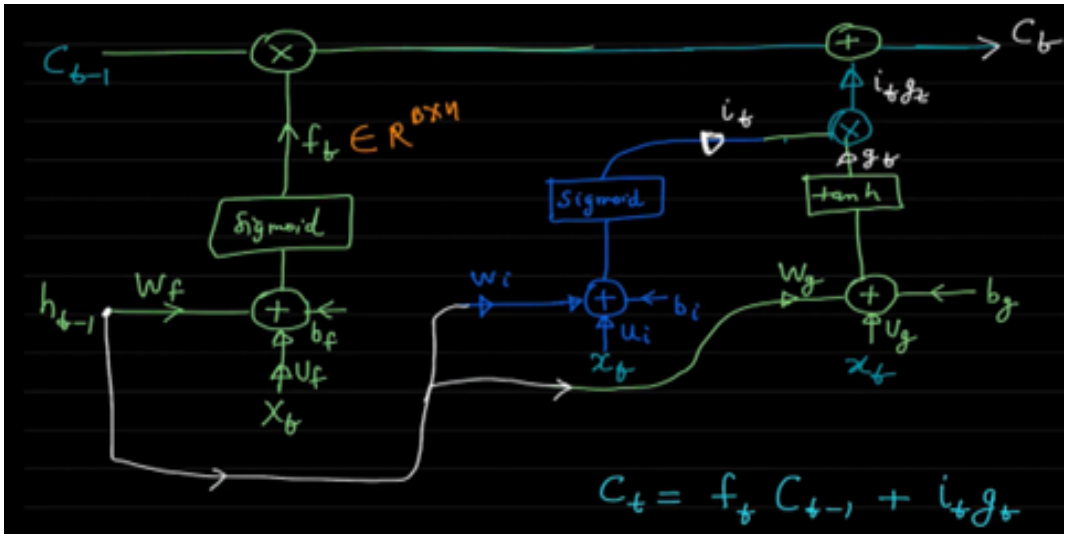
Hình 18: Updating the cell state: Adding new information

∗ Remove information from previous cell state



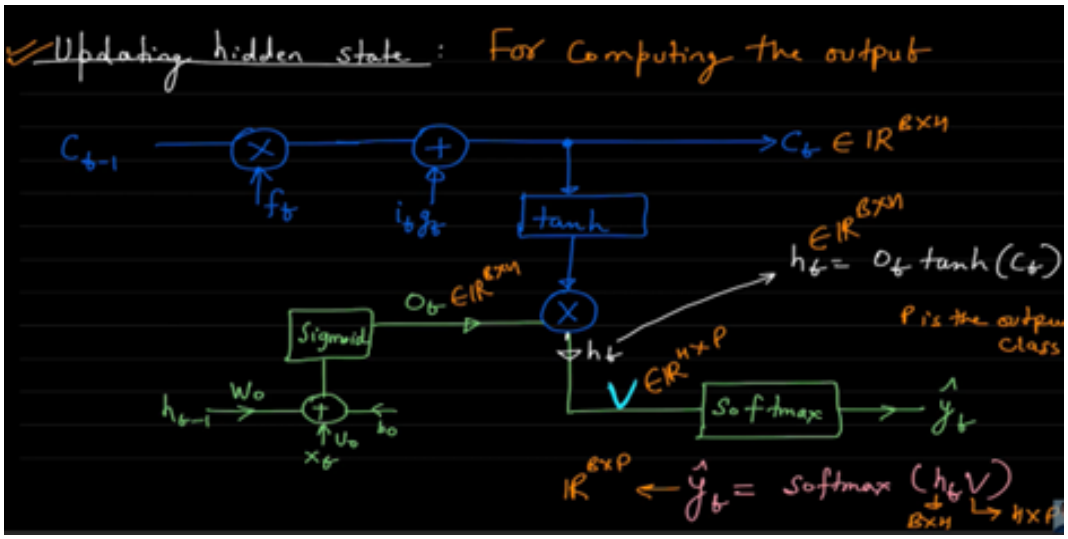
Hình 19: Updating the cell state: Remove information from previous cell state

* $Ct = ft * Ct - 1 + it * gt$



Hình 20: Updating the cell state: Calculate Ct

– **Updating the Hidden State:** Computing the output

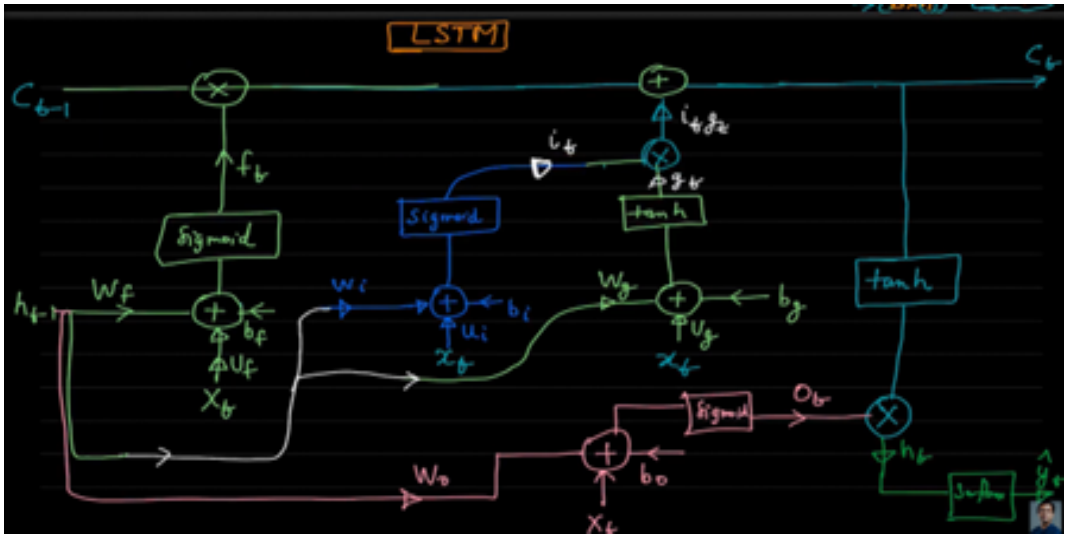


Hình 21: Updating the Hidden State: compute the output

* Input: Previous cell state (c), current input feature (x), and previous hidden state (h).
* Function: Update the new hidden state (h) based on the information from the cell state and input feature.

* Computation: Use the output gate to determine which information from the cell state should be passed to the hidden state. Then, apply the Tanh function to the cell state and multiply it with the output gate to generate the new hidden state.

– After the above steps, here is a complete LSTM model:



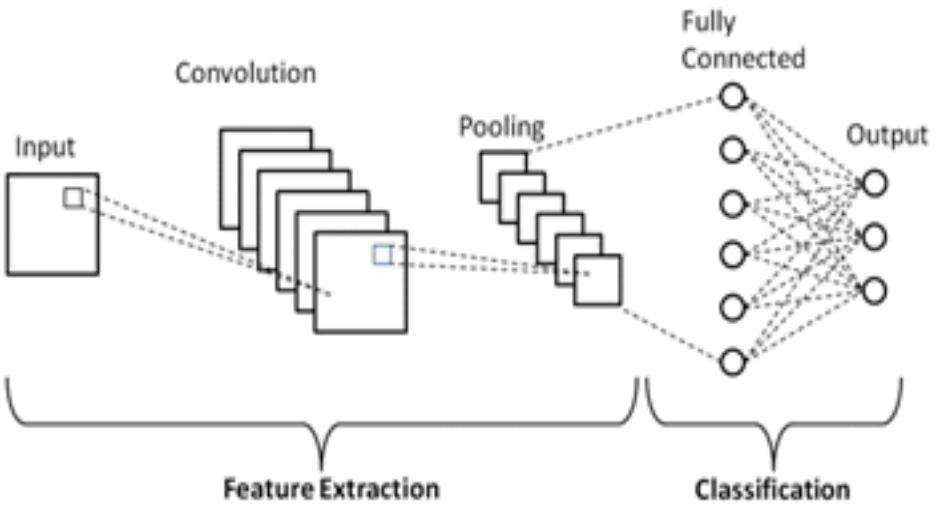
Hình 22: A complete LSTM Model

3.3.4 Convolutional Neural Network (CNN)

- CNN is a deep learning model primarily used in image analysis and spatial data processing. It is designed to automatically learn and extract important features from image data.

- The input to a CNN is a matrix containing raw information of the image. The network utilizes a series of small filters (also known as kernels or filters) to scan through the image. Each filter searches for specific patterns, edges, or features in the image. As the filters move across the image, they generate feature maps by computing convolutions between the filter and different parts of the image.

- These feature maps then pass through non-linear activation layers (e.g., ReLU) to create non-linear features. This process helps the CNN learn better filters for detecting and extracting important features from the image.
- Next, through pooling layers, the size of the feature maps is reduced. Pooling layers often use max pooling or average pooling to extract the maximum or average values from sub-regions of the feature maps. This helps reduce the data size and introduce spatial invariance.
- Finally, the extracted features are fed into fully connected layers to perform classification or prediction for a specific task. Fully connected layers utilize weights to generate appropriate outputs based on the task, such as classifying objects in the image



Hình 23: Architecture of CNN Model

The Convolutional Neural Network (CNN) is widely regarded as one of the most popular and advanced deep learning models today. The network is constructed from the following components:

  - Convolution Layer (Conv): The goal of the Conv layer is to extract high-level features from the raw input information using filters/kernels. A CNN

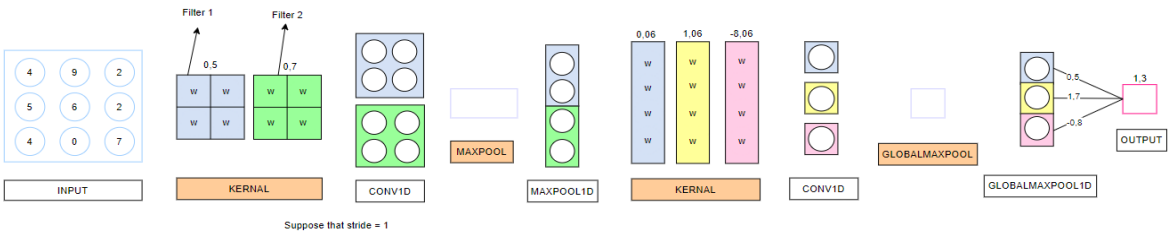
is not limited to just one Conv layer. Typically, the first Conv layer captures low-level features such as edges, colors, gradient orientations, etc. With subsequent layers, the architecture is designed to gradually capture higher-level features. This enables the network to learn more meaningful information and deeper understanding, similar to how humans perceive.

– Pooling Layer (Pool): The pooling layer is a downsampling operation, typically used after the Conv layer, to increase spatial invariance, reduce computation, and training time while retaining important features. There are several types of pooling, such as Sum pooling, L2 pooling, Max pooling, and Average pooling. Among them, Max pooling and Average pooling are the most common forms.

– Fully Connected Layer (FC): The fully connected layer takes input data that has been flattened, where every neuron in the current layer is connected to every neuron in the previous layer. In the convolutional model, after feature extraction through Conv and Pool layers, the data is passed to the fully connected layer to produce the final output relevant to the given task.

With its feature extraction and processing capabilities, CNNs can also be used for analyzing other data domains such as text or sequential time-series data. For image data, 2D grids of filters are used, while 1D grids are typically employed for data like text or time-series.

A basic CNN model can be described as follows:



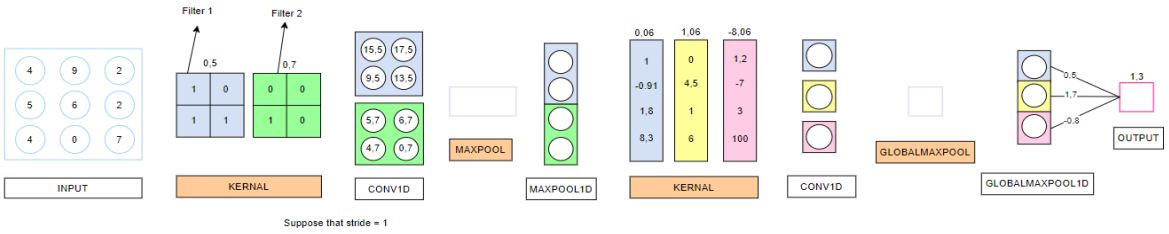
Hình 24: A basic CNN model

Figure 23: Illustration of a convolutional neural network model with a 3x3 feature matrix input, convolutional layers, and an output, with w representing the weights in the filters and b representing the bias of the filters - for the convolutional component, w denotes the connection weights and b denotes the bias of the computing unit - for the fully connected component.

- Suppose we have a convolutional neural network with the structure shown in Figure 23, where the weights and biases are randomly initialized, and we have a sample data:

$$\begin{bmatrix} 4 & 9 & 2 \\ 5 & 6 & 2 \\ 4 & 0 & 7 \end{bmatrix}$$

- The sample data in the form of a 3D matrix is directly input into the model:



Hình 25: Illustration of a CNN model with input features, weight and bias values, and input features fed into the network.

The output of a computational unit in a CNN is generalized by the following formula:

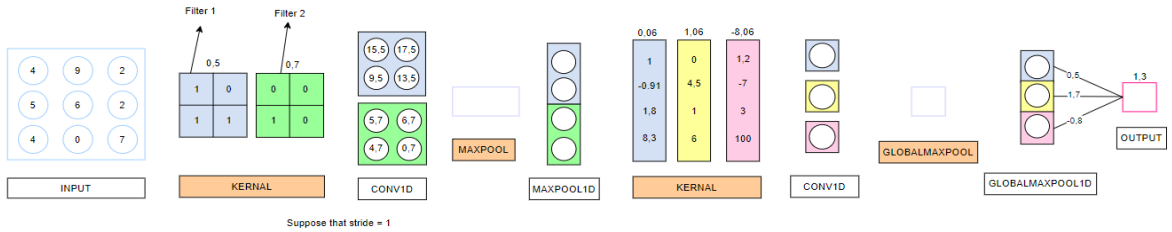
$$Output = X * W + b$$

Where X is the matrix of input features, W is the weight of the filter used for feature extraction, and b is the bias value of the filter. From this, the value of the first convolutional layer with the first kernel is computed as follows:

- 4 + 5 + 6 + 0.5 = 15.5
- 9 + 6 + 2 + 0.5 = 17.5
- 5 + 4 + 0.5 = 9.5
- 6 + 7 + 0.5 = 13.5

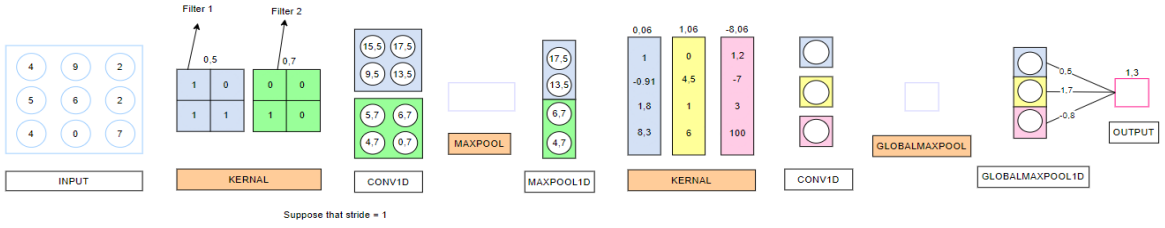
The value of the first convolutional layer with the second kernel is computed in a similar manner as follows:

- 5 + 0.7 = 5.7
- 6 + 0.7 = 6.7
- 4 + 0.7 = 4.7
- 0 + 0.7 = 0.7



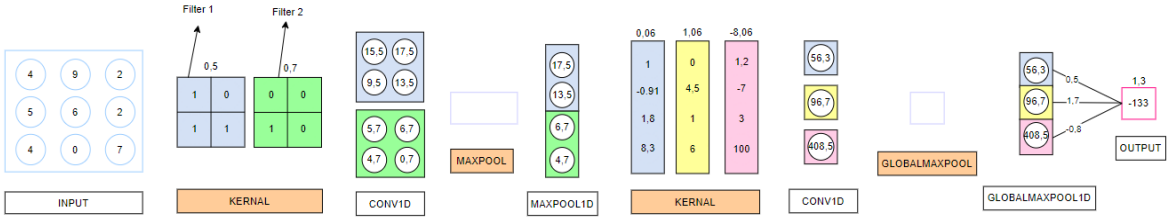
Hình 26: Illustration of a convolutional neural network with the values of the first convolutional layer computed.

Through the MAXPOOL layer only the significant value is kept, so the result of the pooled layer includes only the maximum values of each previous convolution layer:



Hình 27: Illustration of a CNN model with the output values after the MAXPOOL pooling layer.

Following the convolutional layer calculations and similar computations as mentioned in the DNN (fully connected layers), a complete convolutional model is computed with the following values:



Hình 28: Illustration of a convolutional neural network (CNN) model with the complete computed values.

4 DATA PREPROCESSING

The data in the spam comment detection problem consists of over 19,000 product reviews, which include the number of star reviews, comments about the product, and the link to that product. The data has been divided into three sets: data-train : data-dev : data-test $\approx$ 14306 : 1590 : 3974 , with the respective ratios of approximately 72%, 8%, and 20%. The data preprocessing pipeline consists of two sequential steps:

- Data cleaning
- Data preprocessing

4.1 Data cleaning

- Use LabelEncoder() from sklearn.preprocessing to encode the labels of the output data. This performs the conversion of textual labels into numerical encoding for ease of use in the training process.
- For data-train, data-dev, and data-test as the data to be encoded, the processing code can be performed as follows:

```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
data_train.Label = encoder.fit_transform(data_train.Label)
data_dev.Label = encoder.fit_transform(data_dev.Label)
data_test.Label = encoder.fit_transform(data_test.Label)
```

- Remove duplicate records: Use the drop-duplicates() method to remove duplicate records from data-train, data-dev, and data-test. Only keep the first record. This helps maintain only one unique record for each case and ensures the integrity of the data. For data-train, data-dev, and data-test as the data to be encoded, the processing code can be performed as follows:

```
data_train=data_train.drop_duplicates(keep='first')
data_dev=data_dev.drop_duplicates(keep='first')
data_test=data_test.drop_duplicates(keep='first')
```

4.2 Data preprocessing

The following steps are performed to clean and preprocess text data for further tasks such as model training and evaluation. These steps help remove unnecessary components and generate a preprocessed raw vocabulary ready for feature extraction and model training. The data preprocessing process includes the following basic steps:

- Lowercasing: This step ensures that all characters in the text are converted to lowercase. This helps standardize words and avoids differentiation between uppercase and lowercase when processing the data.
- Tokenization: This step divides the text data into individual words, known as "tokens." In a simple manner, tokenization can be done using the function to split a string into a list of words:

  ```
  nltk.word_tokenize()
  ```

- Removing special characters: This step removes special characters in the text, such as punctuation marks, special symbols, and non-alphabetic or non-numeric characters. This can be done by iterating through the list of tokens and keeping only words that contain alphabetic or numeric characters.
- Removing stopwords and punctuation: This step removes stopwords and punctuation marks from the text. Stopwords are common words such as "và", "là", "ở", "một" etc. Punctuation marks include periods, commas, question marks, and other special characters. To remove stopwords and punctuation, you can use a list of Vietnamese stopwords and a list of punctuation characters. Iterate through the list of tokens and keep only words that are not in the stopwords list and are not punctuation marks.

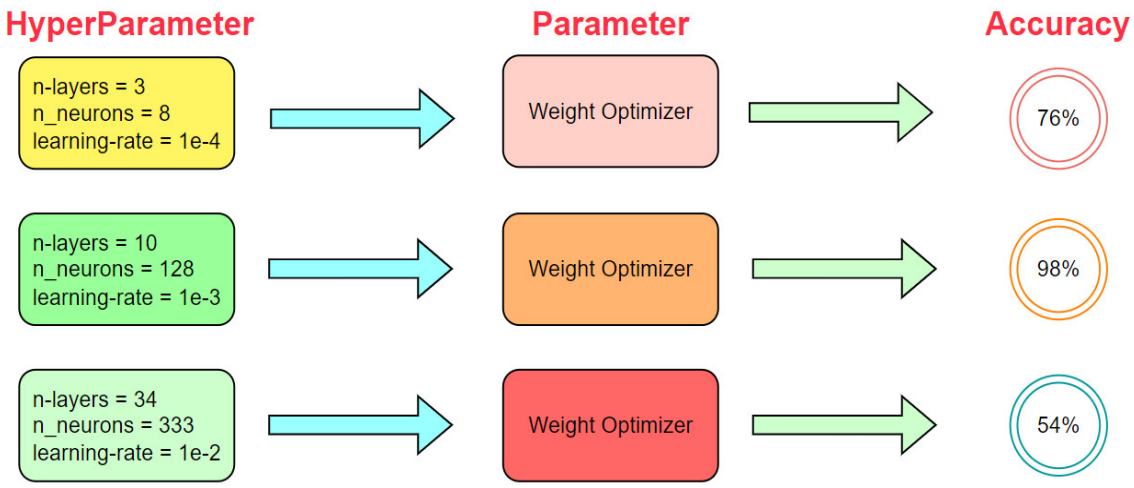
– The code that performs the data processing is as follows:

```
stopwords_path = "/content/vietnamese-stopwords-dash.txt"
stopwords=get_stopwords_list(stopwords_path)
import string
import nltk
from nltk.tokenize import word_tokenize
string.punctuation
nltk.download('punkt')
# Preprocessing function:
def transform_text(text):
# Lower case
text=text.lower()
# Tokenization
text=nltk.word_tokenize(text)
# Removing special characters
y=[]
for i in text:
#kiểm tra xem một chuỗi có chứa duy nhất các kí tự chữ và
#số hay không
if i.isalnum():
    y.append(i)
# Removing stopwords and punctuation
text=y [:]
y.clear()
for i in text:
  if i not in stopwords and i not in string.punctuation:
    y.append(i)
return " ".join(y)
```

5 EXPERIMENTATION AND EVALUATION

The models are sequentially optimized for hyperparameters. Hyperparameters refer to parameters that cannot be updated during the training of a machine learning model. They can be involved in defining the model's structure, such as the number of hidden layers and the activation function, or in determining the efficiency and accuracy of model training, such as the learning rate (LR) of stochastic gradient descent (SGD), batch size, number of epochs, and optimizer. The hyperparameters are evaluated based on the model's prediction losses. In other words, the hyperparameters are set for the model, the model is trained on the data, and the model's performance is assessed using the loss function.

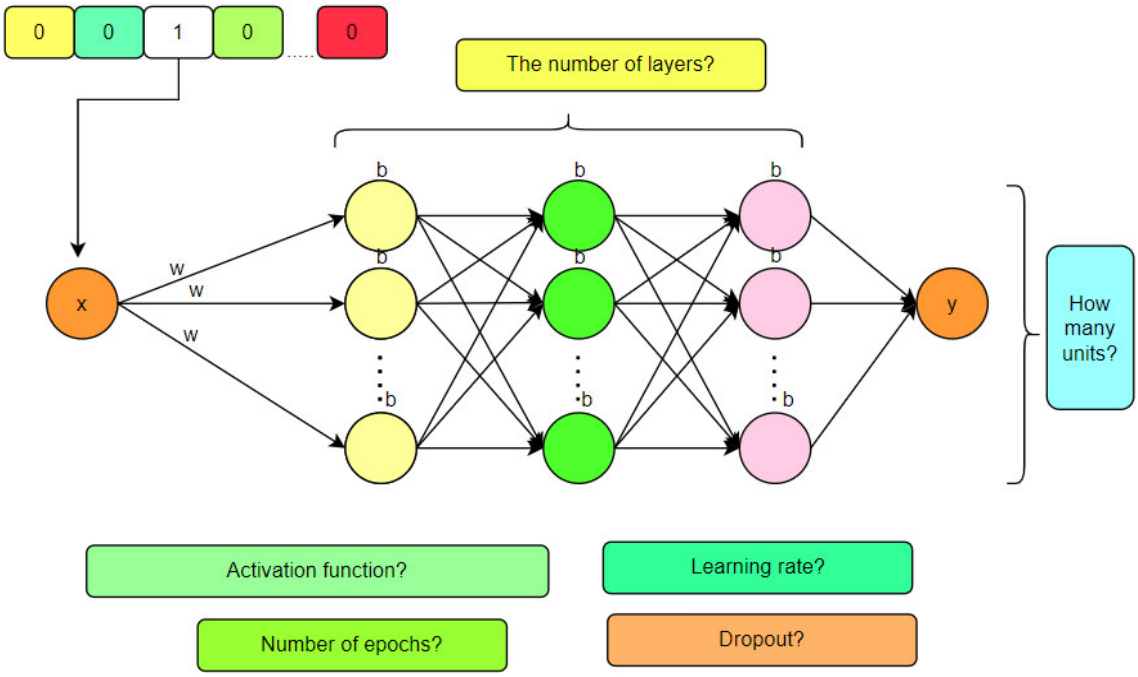
These steps are iteratively followed to find the optimal set of hyperparameters. However, it is important to note that there might not always be a globally optimal solution, and different sets of hyperparameters can yield different results:



Hình 29: Each different set of hyperparameters will give different results

The process of optimizing hyperparameters utilizes the KerasTuner library, an open-source library developed by the Keras team. It provides tools for searching and tuning the hyperparameters of deep learning models, such as the network architecture, the number of layers, the number of computational units, the learning rate, etc.

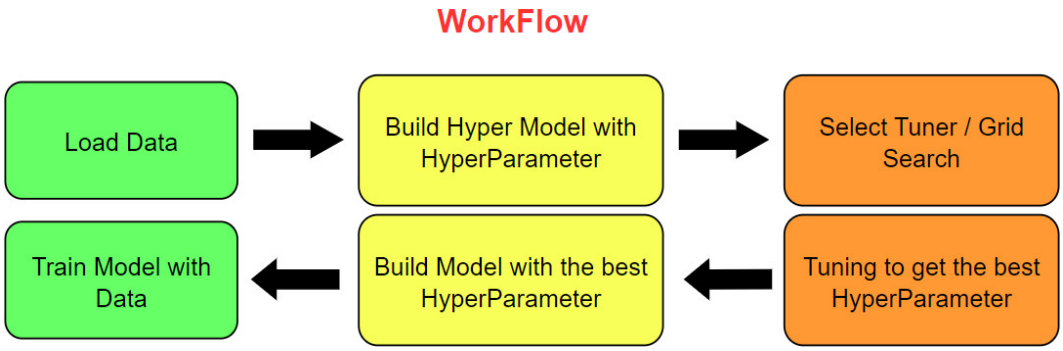
The KerasTuner library enables the exploration of various hyperparameter configurations to find the best set of values. It automates the process of hyperparameter tuning and assists in determining the optimal hyperparameter values for a given deep learning model.



Hình 30: The hyperparameters of deep learning models may need to be optimized

For deep learning algorithms such as ANN, RNN, and LSTM, the hyperparameters to be optimized include the number of layers, the number of units, the activation function, the number of epochs, the learning rate, and the dropout rate.

Once the set of hyperparameters is determined, the corresponding training models are built. The experimental procedure involves the following steps:



Hình 31: The experimental procedure

5.1 Evaluation Metrics

5.1.1 Accuracy:

It is calculated as the ratio of the number of correct predictions (true positives and true negatives) to the total number of samples. Specifically, accuracy is calculated using the formula:

$$\text{Accuracy} = \frac{\text{True Positives + True Negatives}}{\text{Total Samples}}$$

Where:

- True Positives (TP) is the number of spam samples correctly classified as spam.
- True Negatives (TN) is the number of non-spam samples correctly classified as non-spam.
- Total Samples is the total number of data samples (including both spam and non-spam).

Accuracy is a performance metric that measures the overall effectiveness of the model in classifying spam. It indicates the percentage of correct predictions out of the total number of data samples. However, it still has some limitations, such as:

- Imbalanced Data: Accuracy does not account for class imbalance, where one class has significantly more samples than the others.

- Failure to Capture Error Types: Accuracy does not reflect the specific types of errors made by the model.
- Inapplicability to Time Series Data: Accuracy cannot measure the performance of models on time series data, such as time series prediction or forecasting.

5.1.2 F1 Macro:

F1-macro is a metric used to evaluate the performance of a classification model in a multi-class classification problem. It calculates the F1-score for each class individually and then takes the average of these scores, giving equal weight to each class.

The F1-score for a single class is the harmonic mean of precision and recall. It is calculated using the following formula:

$$\text{F1-score} = \frac{\text{Precision * Recall}}{\text{Precision + Recall}}$$

Where:

- Precision is the number of true positives divided by the sum of true positives and false positives for that class.
- Recall is the number of true positives divided by the sum of true positives and false negatives for that class.

F1-macro calculates the F1-score for each class separately and then takes the average of these scores. It gives equal importance to each class, regardless of its size or frequency in the dataset. This makes it suitable for imbalanced datasets where some classes may have fewer samples. The drawback of F1-macro is that it doesn't consider class imbalances. If there are significant imbalances between classes, the F1-score of the minority classes may be overshadowed by the majority classes.

5.2 Experimental results by measure:

The experimental results based on accuracy and F1-macro metrics for my deep learning models provide detailed information about the performance and accuracy of the model in classification and prediction tasks.

It allows assessing the model's ability to make accurate and consistent predictions across different data classes. Based on these results, I can compare and select the model with the best performance for my specific task and make adjustments to hyper-parameters or model architecture if necessary to improve performance.

5.2.1 Accuracy before tunning:

Accuracy (before Tuning)			
Data / Model	Train	Validation	Test
CNN	0.996045	0.8101266	0.8103098
LSTM	0.995056	0.8025317	0.8125952
ANN	0.928309	0.7639241	0.7671407
RNN	0.921458	0.7803798	0.7775521

Hình 32: Accuracy comparison table of deep learning models before tunning

Based on the results, we can make a comparison among the deep learning models in terms of their performance on the training, validation, and test datasets.

- The CNN model achieved the highest training accuracy (0.996) among all the models, indicating that it has learned the training data well. However, its validation and test accuracies are slightly lower (0.810), suggesting a possible overfitting issue.

- The LSTM models performed with high training accuracies 0.995 and relatively good validation and test accuracies (around 0.810). These models seem to have achieved a good balance between learning the training data and generalizing well to unseen data.

- The ANN and RNN models have lower training accuracies (0.928 and 0.921) compared to the other models. This suggests that these models may need more training data or further optimization to improve their performance. However, they still achieved reasonable validation and test accuracies (around 0.764-0.778).

In summary, the CNN model shows the highest training accuracy but may suffer from overfitting. The LSTM models perform well in terms of both training and generalization. The ANN and RNN models have relatively lower training accuracies but still achieve reasonable validation and test accuracies. Based on these observations, the LSTM models could be considered as good options for this particular task. Further analysis and experimentation may be required to fine-tune these models and optimize their performance.

5.2.2 F1-Macro before tunning:

F1-Macro (before Tuning)			
Data / Model	Train	Validation	Test
LSTM	0.989240424	0.72129373	0.72781059
ANN	0.904609439	0.69212702	0.69639037
RNN	0.894243807	0.70984764	0.70064739

Hình 33: F1-Macro comparison table of deep learning models before tunning

From these results, we can observe the following comparisons:

- The LSTM model shows the highest F1-Macro scores among the models, indicating that they have better performance in terms of precision and recall for both positive and negative classes.

- The CNN, ANN and RNN models have lower F1-Macro scores compared to the LSTM model, suggesting that they may have lower overall performance in capturing the true positives and true negatives.

In summary, the LSTM model demonstrate better performance in terms of F1-Macro scores compared to the CNN, ANN and RNN models. However, it is recommended to consider additional evaluation metrics and further analysis to make a comprehensive comparison and select the most suitable model for the specific task.

5.3 Improving the result:

To improve the accuracy of the models, I have utilized Keras Tuner to optimize the hyperparameters. By leveraging Keras Tuner, I can efficiently search through the hyperparameter space and find the best combination of settings that maximize the accuracy of my deep learning models. This iterative process of tuning the hyperparameters allows me to continually improve the models and achieve higher levels of accuracy.

Here are the results of hyperparameter optimization for the ANN, RNN and LSTM models:

5.3.1 Accuracy after tunning:

Accuracy (after Tuning)			
Data / Model	Train	Validation	Test
LSTM	0.9965	0.8095	0.8134
ANN			0.8009
RNN			0.804

Hình 34: Accuracy comparison table of deep learning models after tunning

From these results, we can observe the following comparisons among the models:

- The LSTM model achieved the highest Accuracy scores on both the validation and test datasets, indicating its superior performance in capturing patterns and making accurate predictions.

- The ANN model achieved a relatively lower Accuracy score compared to the LSTM model on the test dataset.
- The RNN model achieved a slightly lower Accuracy score compared to the LSTM model on the test dataset.

Overall, the LSTM model shows the best performance in terms of Accuracy on the test datasets. It demonstrates its effectiveness in capturing the underlying patterns and making accurate predictions for the given task. The ANN and RNN models also show reasonable performance, but they fall slightly behind the LSTM model.

5.3.2 F1-Macro after tunning:

F1-Macro (after Tuning)			
Data / Model	Train	Validation	Test
LSTM	0.99534128	0.7356677	0.74590209

Hình 35: F1-Macro comparison table of deep learning models after tunning

- With the optimized hyperparameters, the LSTM model has shown strong performance on the training dataset, achieving a high F1-Macro score of 0.99534128. This indicates that the model has successfully learned the patterns and relationships present in the training data:
- On the validation dataset, the LSTM model achieved a moderate F1-Macro score of 0.7356677. This suggests that the model's performance may be slightly less accurate when generalizing to unseen data, as compared to the training dataset.
- On the test dataset, the LSTM model achieved a similar F1-Macro score of 0.74590209. This indicates that the model's performance is consistent when evaluated on an independent dataset.

Overall, the LSTM model, after using Keras Tuner to optimize its hyperparameters, has demonstrated strong learning capabilities and reasonable generalization performance. It can effectively capture complex patterns and make accurate predictions on the given task.

6 CONCLUSION AND DEVELOPMENT DIRECTION

6.1 Conclusion

In conclusion, this study explored the application of various deep learning models for the task of spam comment detection in natural language processing (NLP). The models that were evaluated include CNN, LSTM, ANN, and RNN. The performance of these models was assessed using metrics such as accuracy and F1-Macro score on the training, validation, and test datasets.

Based on the results, deep learning models such as CNN, LSTM, ANN, and RNN demonstrated the ability to learn complex patterns and dependencies in text data. However, the results indicated that the LSTM model consistently outperformed the other models in terms of Accuracy score and F1-Macro across all datasets. It achieved the highest F1-Macro score on the training dataset (0.99534128) and showed competitive performance on the validation (0.7356677) and test (0.74590209) datasets. LSTM achieved the highest Accuracy on the testing dataset (0.8125952).

It can be seen that, the models achieved high accuracy and F1-Macro on the training data, indicating their ability to learn from the available training samples. However, the performance on the validation and test datasets was slightly lower. This suggests that the models might benefit from a larger training dataset to further improve their generalization performance.

Finally, it is worth noting that all models achieved relatively high accuracy and F1-Macro scores, indicating their effectiveness in spam reviews detection. These findings demonstrate the potential of deep learning models in tackling NLP tasks, specifically in spam comment detection.

6.2 Development direction

Moving forward, there are several directions for future development in this research area. Firstly, exploring more advanced architectures and variations of the LSTM model, such as bidirectional LSTM or stacked LSTM, could be beneficial. These models may capture more complex relationships and improve the overall performance in spam comment detection.

Additionally, incorporating pre-trained language models, such as BERT or GPT, could be a promising avenue for improvement. These models have been shown to capture contextual information and semantic nuances effectively, which could enhance the models' understanding of spam comments and improve their classification accuracy.

Furthermore, expanding the dataset by collecting more diverse and representative samples would enhance the models' ability to generalize and handle various types of spam comments. Data augmentation techniques, such as synthetic oversampling or text perturbation, could also be explored to address class imbalance issues and improve model performance.

In addition to the NLP problem, data preprocessing is an extremely necessary step. Some of the steps in the data preprocessing process that need to be improved include enhanced data cleaning techniques, advanced tokenization methods ...

Finally, conducting further analysis and interpretation of the models' predictions, such as error analysis and feature importance, would provide valuable insights into the characteristics of spam comments and guide the development of more robust and interpretable models.

Tài liệu

[1] Mahesh, B. (2020). Machine learning algorithms-a review. International Journal of Science and Research (IJSR). [Internet], 9(1), 381-386.

[2] Van Engelen, J. E., & Hoos, H. H. (2020). A survey on semi-supervised learning. Machine learning, 109(2), 373-440.

[3] Cunningham, P., Cord, M., & Delany, S. J. (2008). Supervised learning. In Machine learning techniques for multimedia: case studies on organization and retrieval (pp. 21-49). Berlin, Heidelberg: Springer Berlin Heidelberg.

[4] Wang, S. C., & Wang, S. C. (2003). Artificial neural network. Interdisciplinary computing in java programming, 81-100.

[5] Picton, P., & Picton, P. (1994). What is a neural network? (pp. 1-12). Macmillan Education UK.

[6] Yin, C., Zhu, Y., Fei, J., & He, X. (2017). A deep learning approach for intrusion detection using recurrent neural networks. Ieee Access, 5, 21954-21961.

[7] Delsy, T. T. M., Nandhitha, N. M., & Rani, B. S. (2021). Feasibility of Recurrent Neural Network for the Binary classification of Non stationary signals. Microprocessors and Microsystems, 82, 103955.

[8] Rao, A., & Spasojevic, N. (2016). Actionable and political text classification using word embeddings and LSTM. arXiv preprint arXiv:1607.02501.

[9] Van Dinh, C., Luu, S. T., & Nguyen, A. G. T. (2022, December). Detecting Spam Reviews on Vietnamese E-Commerce Websites. In Intelligent Information and Database Systems: 14th Asian Conference, ACIIDS 2022, Ho Chi Minh City, Vietnam, November 28–30, 2022, Proceedings, Part I (pp. 595-607). Cham: Springer International Publishing.

[10] Othman, N. F., & Din, W. I. S. W. (2019). Youtube spam detection framework using naïve bayes and logistic regression. Indonesian Journal of Electrical Engineering and Computer Science, 14(3), 1508-1517.

[11] Foland, W., & Martin, J. H. (2016, June). CU-NLP at SemEval-2016 task 8: AMR parsing using LSTM-based recurrent neural networks. In Proceedings of the 10th international workshop on semantic evaluation (semeval-2016) (pp. 1197-1201).

[12] Chen, Y. (2015). Convolutional neural network for sentence classification (Master's thesis, University of Waterloo).

[13] Huynh, H. D., Do, H. T. T., Van Nguyen, K., & Nguyen, N. L. T. (2020). A simple and efficient ensemble classifier combining multiple neural network models on social media datasets in Vietnamese. arXiv preprint arXiv:2009.13060.

[14] viblo.asia. 2023. Phân loại văn bản tự động bằng Machine Learning như thế nào?. [ONLINE] Available at: https://viblo.asia/p/phan-loai-van-ban-tu-dong-bang-machine-learning-nhu-the-nao-4P856Pa1ZY3.

[15] viblo.asia. 2023. Ứng dụng Machine Learning để phân loại spam SMS. [ONLINE] Available at: https://viblo.asia/p/ung-dung-machine-learning-de-phan-loai-spam-sms-07LKXwg85V4. [Accessed 11 April 2023].

[16] guendouz.wordpress.com. 2015. Implementation of TF-IDF in JAVA – Guendouz Mohamed. [ONLINE] Available at: https://guendouz.wordpress.com/2015/02/17/implementation-of-tf-idf-in-java/.

[17] caihuuthuc.wordpress.com. 2020. Precision, Recall và F1-score là gì? – Cái Hữu Thức's notes. [ONLINE] Available at: https://caihuuthuc.wordpress.com/2020/02/23/precision-recall-va-f1-score-la-gi/.

[18] Karim, M., & Rahman, R. M. (2013). Decision tree and naive bayes algorithm for classification and generation of actionable knowledge for direct marketing.

[19] Rezaeinia, S. M., Ghodsi, A., & Rahmani, R. (2017). Improving the accuracy of pre-trained word embeddings for sentiment analysis. arXiv preprint arXiv:1711.08609.

[20] realpython.com. 2023. Practical Text Classification With Python and Keras – Real Python. [ONLINE] Available at: https://realpython.com/python-keras-text-classification/#convolutional-neural-networks-cnn.

[21] dennybritz.com. 2015. Recurrent Neural Networks Tutorial, Part 1 – Introduction to RNNs · Denny's Blog. [ONLINE] Available at: https://dennybritz.com/posts/wildml/recurrent-neural-networks-tutorial-part-1/.

[22] Othman, N. F., & Din, W. I. S. W. (2019). Youtube spam detection framework using naïve bayes and logistic regression. Indonesian Journal of Electrical Engineering and Computer Science, 14(3), 1508-1517.

[23] www.machinelearningnuggets.com. 2022. TensorFlow Recurrent Neural Networks (Complete guide with examples and code). [ONLINE] Available at: https://www.machinelearningnuggets.com/tensorflow-lstm/.

[24] machinelearningmastery.com. 2022. Binary Classification Tutorial with the Keras Deep Learning Library - MachineLearningMastery.com. [ONLINE] Available at: https://machinelearningmastery.com/binary-classification-tutorial-with-the-keras-deep-learning-library/.

[25] www.machinelearningnuggets.com. 2022. TensorFlow Recurrent Neural Networks (Complete guide with examples and code). [ONLINE] Available at: https://www.machinelearningnuggets.com/tensorflow-lstm/.

[26] www.linkedin.com. 2023. Count Vectorizer vs TFIDF Vectorizer | Natural Language Processing. [ONLINE] Available at:

https://www.linkedin.com/pulse/count-vectorizers-vs-tfidf-natural-language-processing-sheel-saket.

[27] Larabi-Marie-Sainte, S., Ghouzali, S., Saba, T., Aburahmah, L., & Almohaini, R. (2022). Improving spam email detection using deep recurrent neural network. Indonesian Journal of Electrical Engineering and Computer Science, 25(3), 1625-1633.

[28] Liu, P., Qiu, X., & Huang, X. (2016). Recurrent neural network for text classification with multi-task learning. arXiv preprint arXiv:1605.05101.

[29] Yu, T., & Zhu, H. (2020). Hyper-parameter optimization: A review of algorithms and applications. arXiv preprint arXiv:2003.05689.

[30] Rogachev, A. F. (2021). Optimization of Artificial Neural Network Hyperparameters For Processing Retrospective Information. In CEUR Workshop Proceedings.

[31] O'Malley, T.: Hyperparameter tuning with Keras Tuner. https://blog.tensorflow.org/2020/01/hyperparameter-tuning-with-keras-tuner.html, last accessed 2020/10/21.

[32] https://www.analyticsvidhya.com/blog/2021/06/tuning-hyperparameters-of-an-artificial-neural-network-leveraging-keras-tuner/

[33] https://keras.io/api/keras_tuner/hyperparameters/