VIETNAM GENERAL CONFEDERATION OF LABOR

**TON DUC THANG UNIVERSITY**

Logo

Description automatically generated**FACULTY OF INFORMATION TECHNOLOGY**

**NATURAL LANGUAGE PROCESSING MIDTERM**

*Instructor:* **Assoc. Prof. PhD. Lê Anh Cường**

*Student:* **Trương Gia Bảo - 521H0201**

**Vi Thành Đạt - 521H0390**

*Class:* **21H50302**

**21H50301**

**HO CHI MINH CITY, 2024**

VIETNAM GENERAL CONFEDERATION OF LABOR

**TON DUC THANG UNIVERSITY**

Logo

Description automatically generated**FACULTY OF INFORMATION TECHNOLOGY**

**NATURAL LANGUAGE PROCESSING MIDTERM**

*Instructor:* **Assoc. Prof. PhD. Lê Anh Cường**

*Student:* **Trương Gia Bảo - 521H0201**

**Vi Thành Đạt - 521H0390**

*Class:* **21H50302**

**21H50301**

**HO CHI MINH CITY, 2024**

# ACKNOWLEDGEMENT

I would like to express my sincere thanks to the teachers and lecturers of Ton Duc Thang University and especially Mr. Le Anh Cuong - lecturer of Natural Language Processing Group 03 for kindly helping and supporting as well as answering our questions throughout the process of making this report, so that I can get the best results.

**THE PROJECT WAS COMPLETED**

**AT TON DUC THANG UNIVERSITY**

I would like to assure you that this is my own project and guided by Le Anh Cuong. The research contents and results in this topic are honest and have not been published in any form before. The data in the tables for analysis, comments, and evaluations collected by the author himself from different sources are clearly stated in the references section.

In addition, the project also uses some comments, reviews as well as figures of other authors and other organizations with quotes and annotations of origin.

If any fraud is detected, I would like to take full responsibility for the content of my project. Ton Duc Thang University is not involved in copyright or copyright violations caused by me in the process (if any).

*Ho Chi Minh City, 28th Oct 2024*

*Author*

|  |  |
| --- | --- |
|  | *Trương Gia Bảo* |
|  | *Vi Thanh Đạt* |

**FACULTY ENDORSEMENTS AND REVIEWS**

**Phần xác nhận của GV hướng dẫn**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tp. Hồ Chí Minh, ngày tháng năm

(ký và ghi họ tên)

**Phần đánh giá của GV chấm bài**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tp. Hồ Chí Minh, ngày tháng năm

(ký và ghi họ tên)

# TABLE OF CONTENTS

[ACKNOWLEDGEMENT 1](#_Toc181131909)

[TABLE OF CONTENTS 1](#_Toc181131910)

[CHAPTER 1: CLASSIFICATION MODEL 2](#_Toc181131911)

[CHAPTER 2: LSTM ( LONG SHORT TERM MEMORY ) 7](#_Toc181131912)

[2.1 LSTM ( Long Short Term Memory ) 7](#_Toc181131913)

[2.2 BiLSTM ( Bidirectional Long Short Term Memory ) 11](#_Toc181131914)

[2.2.1 Bidirectional Processing 12](#_Toc181131915)

[2.2.2 Forward Pass 13](#_Toc181131916)

[2.2.3 Backward Pass 13](#_Toc181131917)

[CHAPTER 3: WORD EMBEDDING 14](#_Toc181131918)

[CHAPTER 4: WORD2VEC 17](#_Toc181131919)

[4.1 Model Architecture 18](#_Toc181131920)

[4.2 Neural Network Training 19](#_Toc181131921)

[4.3 Vector Representations 20](#_Toc181131922)

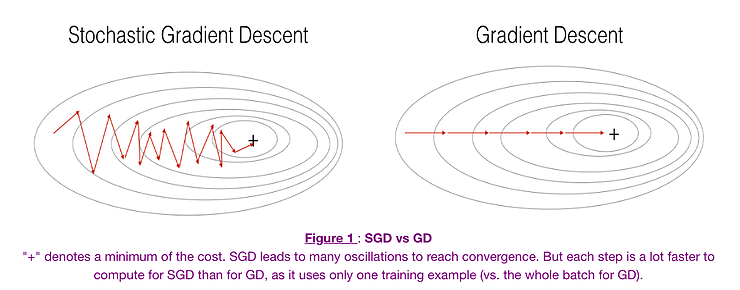
[4.4 Advantage and Disadvantage 20](#_Toc181131923)

[REFERENCE 21](#_Toc181131924)

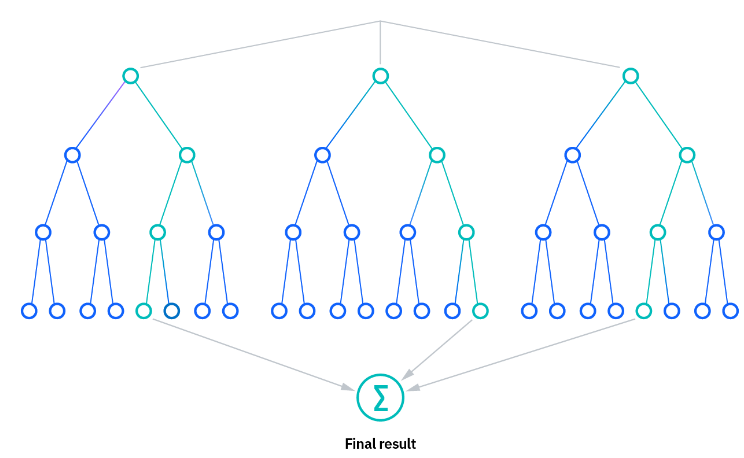
# CHAPTER 1: CLASSIFICATION MODEL

1. **Classification Model**
   1. **Decision Tree**
      1. **Introduction:**

* A common and effective tool in many domains, including statistics, data mining, and machine learning, are decision trees. By simulating the connections between many factors, they offer a straightforward and understandable method for making data-driven judgments. The focus of this article is on decision trees, including their definition, operation, benefits, and uses.
* A decision tree is a structure that resembles a flowchart and is used to make predictions or choices. It is composed of nodes that indicate choices or attribute tests, branches that show how these choices turned out, and leaf nodes that provide the results or forecasts. Every internal node represents an attribute test, every branch represents the test's outcome, and every leaf node represents a class label or a continuous value.
  + 1. **How does Decision Tree works:**
* Selecting the Best Attribute: Using a metric like Gini impurity, entropy, or information gain, the best attribute to split the data is selected.
* Splitting the Dataset: The dataset is split into subsets based on the selected attribute.
* Repeating the Process: The process is repeated recursively for each subset, creating a new internal node or leaf node until a stopping criterion is met (e.g., all instances in a node belong to the same class or a predefined depth is reached).
  + 1. **Metrics for Splitting**
* Gini Impurity: Measures the likelihood of an incorrect classification of a new instance if it was randomly classified according to the distribution of classes in the dataset.
* , where pi is the probability of an instance being classified into a particular class.
* Entropy: Measures the amount of uncertainty or impurity in the dataset.
* , where pi is the probability of an instance being classified into a particular class.
* Information Gain: Measures the reduction in entropy or Gini impurity after a dataset is split on an attribute.
* ,
* where Di is the subset of D after splitting by an attribute.
  + 1. **Benefits:**
* Simplicity and Interpretability: Decision trees are easy to understand and interpret. The visual representation closely mirrors human decision-making processes.
* Versatility: Can be used for both classification and regression tasks.
* No Need for Feature Scaling: Decision trees do not require normalization or scaling of the data.
* Handles Non-linear Relationships: Capable of capturing non-linear relationships between features and target variables.
  + 1. **Limitations:**
* Overfitting: Decision trees can easily overfit the training data, especially if they are deep with many nodes.
* Instability: Small variations in the data can result in a completely different tree being generated.
* Bias towards Features with More Levels: Features with more levels can dominate the tree structure.
  1. **Stochastic Gradient Descent**
     1. **Introduction:**
* Stochastic Gradient Descent (SGD) is a variant of the gradient descent algorithm employed for optimizing machine learning models. It addresses the computational inefficiency associated with gradient descent methods when working with extensive datasets in machine learning endeavors. SGD updates model parameters incrementally. In this method, instead of utilizing the entire dataset for each iteration, a single random instance (or a mini-batch) is chosen to compute the gradient and update the model parameters. This stochastic selection injects randomness into the optimization process, hence the term "stochastic" in stochastic gradient descent. For instance, if the model involves a dataset of 10,000, SGD will perform parameter updates 10,000 times.



* + 1. **Benefits:**
* Frequent updates of model parameter
* Requires less Memory.
* Faster updates as it uses only one training example at a time
  + 1. **Limitations:**
* The frequent can also result in noisy gradients which may cause the error to increase instead of decreasing it.
* High variance in updates can lead to noisy convergence.
* Frequent updates are computationally expensive.
  1. **Random Forest**
     1. **Introduction**
* Leo Breiman and Adele Cutler are the trademark holders of the widely used machine learning technique known as "random forest," which aggregates the output of several decision trees to produce a single outcome. Because it can handle both classification and regression problems, its versatility and ease of use have encouraged its use.
* The random forest algorithm is an extension of the bagging method as it utilizes both bagging and feature randomness to create an uncorrelated forest of decision trees. Feature randomness, also known as feature bagging or “the random subspace method”, generates a random subset of features, which ensures low correlation among decision trees. This is a key difference between decision trees and random forests. While decision trees consider all the possible feature splits, random forests only select a subset of those features.
* If we go back to the “should I surf?” example, the questions that I may ask to determine the prediction may not be as comprehensive as someone else’s set of questions. By accounting for all the potential variability in the data, we can reduce the risk of overfitting, bias, and overall variance, resulting in more precise predictions.
* How random forest works:
* Random forest algorithms have three main hyperparameters, which need to be set before training. These include node size, the number of trees, and the number of features sampled. From there, the random forest classifier can be used to solve for regression or classification problems.
* The random forest algorithm comprises multiple decision trees, each built from a bootstrap sample of the training set. Approximately one-third of each sample is left out as out-of-bag (oob) data for validation. Feature bagging adds randomness, reducing correlation among trees. For regression, predictions are averaged, while classification uses majority voting. The oob data then serves for cross-validation to finalize the prediction.



* + 1. **Benefits**
* Reduced overfitting risk: Random forests minimize overfitting by averaging uncorrelated trees, lowering variance and prediction error.
* Flexibility: Random forests handle both regression and classification tasks accurately, manage missing data well, and estimate missing values effectively.
* Feature importance: Random forests simplify feature importance evaluation using Gini importance, mean decrease in impurity (MDI), or permutation importance (MDA), assessing accuracy drop when excluding features.
  + 1. **Limitations**
* Time-consuming procedure: Random forest algorithms can produce more accurate predictions since they can handle enormous data sets, but they can also process data slowly because they are computing data for each decision tree separately.
* More resources are needed: Random forests will need more resources to store the data they process because they handle larger data sets.
* More complex: Compared to a forest of decision trees, the forecast of a single decision tree is simpler to understand.

# CHAPTER 2: LSTM ( LONG SHORT TERM MEMORY )

## LSTM ( Long Short Term Memory )

Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) architecture commonly used in deep learning. Unlike traditional feedforward neural networks, LSTMs include feedback connections, enabling them to capture temporal dependencies within sequences of data. They are specifically designed to address the challenges of vanishing or exploding gradients that can occur when training standard RNNs on sequential data. This makes LSTMs particularly effective for tasks like natural language processing (NLP), speech recognition, and time series forecasting.

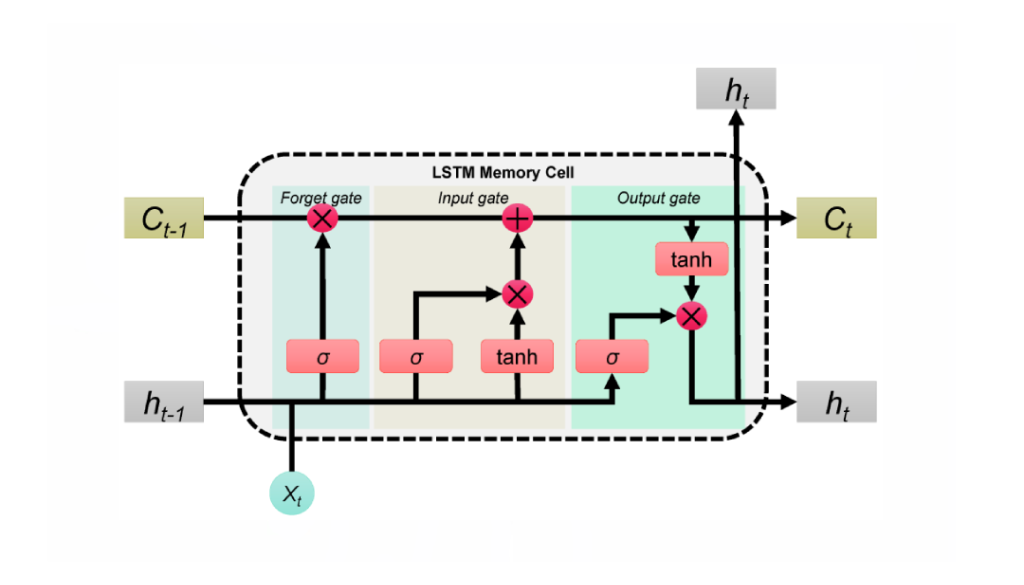


Fig 2.1: LSTM Architecture

LSTM networks introduce memory cells, which have the ability to retain information over long sequences. Each memory cell has three main components: an input gate, a forget gate, and an output gate. These gates help regulate the flow of information in and out of the memory cell.

1. **Cell State ( Memory State )**

The **cell state** is a key component of Long Short-Term Memory (LSTM) networks. It acts as a memory that carries information across the sequence as it flows through the network. The cell state allows LSTMs to maintain long-term dependencies by controlling the flow of information through various gates—input, forget, and output gates:

* **Forget Gate**: Decides what information to discard from the cell state.
* **Input Gate**: Determines which new information will be added to the cell state.
* **Output Gate**: Controls what part of the cell state will be output.

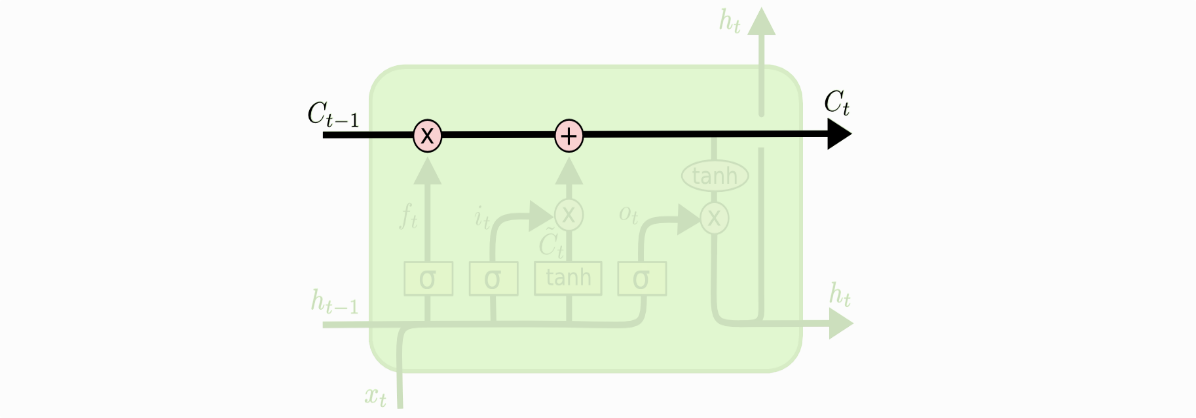
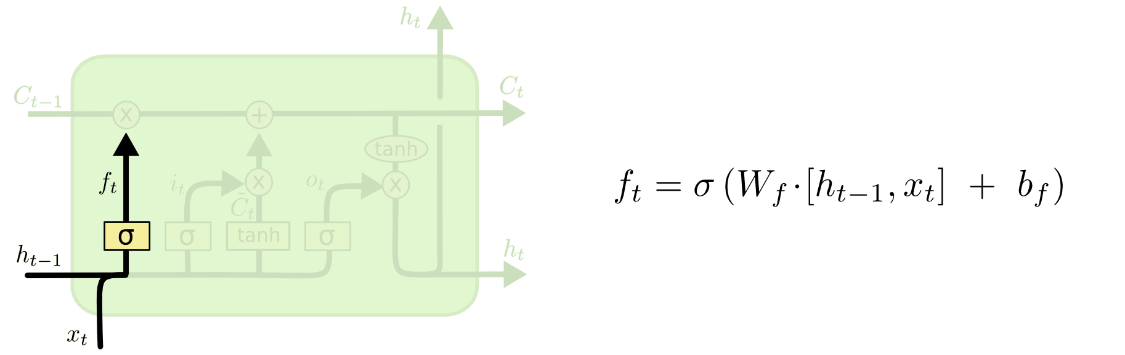


Fig 2.1: Cell State Architecture

The cell state undergoes modifications at each time step through these gates, allowing LSTMs to selectively remember or forget information, thus preserving relevant data over long sequences. This mechanism is what allows LSTMs to overcome the vanishing gradient problem, making them effective in capturing long-term dependencies in sequential data.

1. **Forget State**

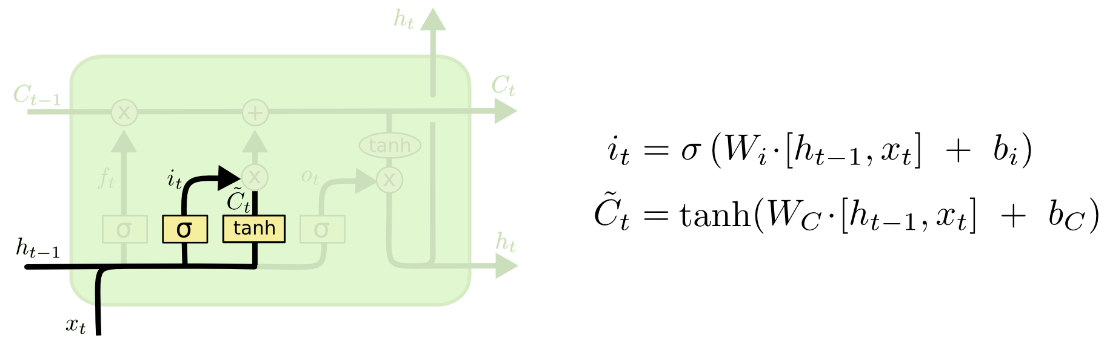
The forget gate in an LSTM plays a crucial role in determining which information should be discarded from the memory cell. It takes as input the current input vector and the hidden state from the previous time step, and processes them through a sigmoid activation function. The output of this function is a set of values ranging between 0 and 1 for each element of the memory cell. These values act as weights, where a value of 0 indicates that the information will be completely discarded, and a value of 1 means that the information will be fully retained.



By doing so, the forget gate enables the LSTM to selectively forget irrelevant or outdated information while keeping relevant data in the memory. This capability is essential for LSTMs to adaptively manage information over time, ensuring that the memory cell retains only the most important aspects of the input sequence, which helps in learning long-term patterns and dependencies in sequential data.

1. **Input Gate**

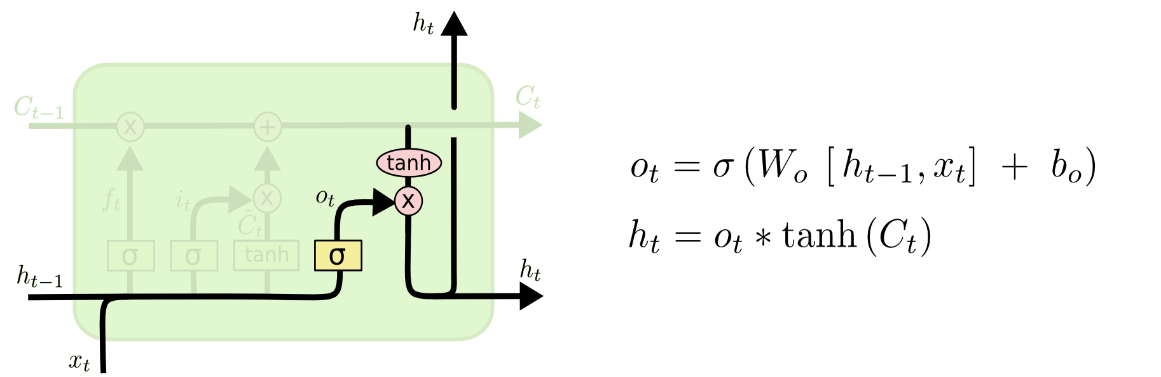
The input gate in an LSTM controls how much of the new information should be stored in the memory cell. It is responsible for deciding which parts of the current input and the previous hidden state should be retained. The input gate takes the current input vector and the hidden state from the previous time step as inputs. These inputs are passed through a sigmoid activation function, which outputs values ranging from 0 to 1 for each element in the memory cell. A value of 0 means that no new information will be added to that part of the memory, while a value of 1 indicates that the new input should be fully stored in the memory cell.



This selective process helps the LSTM decide what new information is significant and should be remembered, contributing to the network's ability to learn from long-term dependencies within sequential data. The input gate, working alongside the forget and output gates, ensures that the LSTM can dynamically update the cell state, balancing new information with previously stored data.

1. **Output Gate**

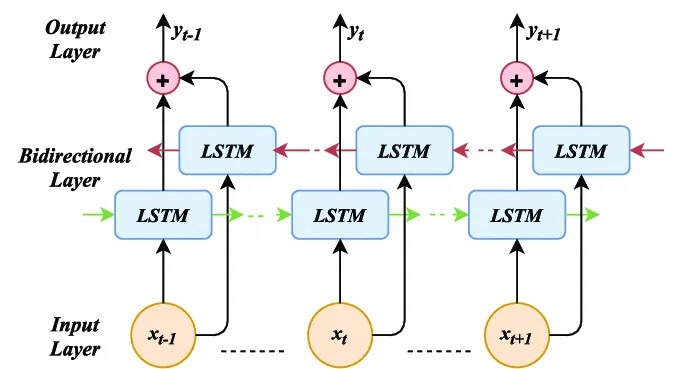
The output gate in an LSTM is responsible for determining how much of the information stored in the memory cell should be used to compute the hidden state at a given time step. It takes two inputs: the current input vector and the hidden state from the previous time step. These inputs are processed through a sigmoid activation function, which produces values between 0 and 1 for each element of the memory cell. These values act as a filter, where a value of 0 means that the corresponding information from the memory cell will be excluded from the hidden state, while a value of 1 means that the information will be fully included. After determining the level of filtering, the memory cell’s content is passed through a tanh function to normalize the values between -1 and 1. The output gate then multiplies this normalized content by the output of the sigmoid function to produce the new hidden state.



This mechanism ensures that the hidden state selectively carries only the most relevant information from the memory cell to the next time step, enabling the LSTM to effectively manage both short-term and long-term dependencies in sequential data.

## BiLSTM ( Bidirectional Long Short Term Memory )

A **Bi-LSTM (Bidirectional Long Short-Term Memory)** is an advanced type of recurrent neural network (RNN) that extends the capabilities of traditional LSTM models by processing sequential data in both forward and backward directions. Unlike standard LSTMs, which only process information from past to future, Bi-LSTMs use two separate LSTM layers—one that reads the input sequence from start to end (forward) and another that reads from end to start (backward). The outputs of these two layers are then combined, allowing the network to access context from both directions simultaneously. This bidirectional processing enables the model to capture a richer representation of the input sequence, as it can consider both past and future information for each time step. As a result, Bi-LSTMs are particularly effective in tasks that benefit from understanding the entire context, such as natural language processing (NLP), speech recognition, and text classification. By leveraging information from both directions, Bi-LSTMs improve the understanding of complex patterns within sequential data, making them powerful tools for handling a wide range of sequence modeling tasks.



### **Bidirectional Processing**

Unlike traditional RNNs, which process input sequences in a single direction—either forward (from past to future) or backward (from future to past)—a **Bi-LSTM (Bidirectional Long Short-Term Memory)** processes sequences in both directions simultaneously. This structure consists of two separate LSTM layers: one that processes the sequence in the forward direction, and another that processes it in the backward direction. Each LSTM layer maintains its own hidden states and memory cells, allowing the model to retain information from both earlier and later parts of the sequence. The outputs from these two layers are then combined, typically through concatenation or summation, providing a richer representation of the input data. This dual processing capability allows Bi-LSTMs to capture context from both the past and the future for each time step in the sequence, making them particularly powerful for tasks that require a deep understanding of surrounding context, such as natural language processing (NLP), machine translation, and speech recognition. By leveraging information from both directions, Bi-LSTMs improve performance on complex sequence modeling problems, offering a more comprehensive understanding of the data.

### **Forward Pass**

During the forward pass of a **Bi-LSTM**, the input sequence is processed by the **forward LSTM layer** from the first time step to the last. As the sequence progresses through each time step, the forward LSTM uses the current input, along with the hidden state and memory cell from the previous time step, to compute its new hidden state and update its memory cell. This process allows the forward LSTM to retain information about past elements of the sequence as it moves forward, effectively learning patterns and dependencies from earlier time steps.

At each time step, the forward LSTM’s computations involve a series of gating mechanisms—input, forget, and output gates—that help regulate what information is added to, retained in, or removed from the memory cell. The updated hidden state from each step is then passed on to the next time step, ensuring that the forward LSTM maintains an evolving understanding of the sequence as it progresses. The resulting hidden states from the forward LSTM layer capture information about the sequence from the beginning up to the current time step, forming a forward-directed representation that is later combined with the backward LSTM’s output for a comprehensive view of the input sequence.

### **Backward Pass**

Simultaneously, in a **Bi-LSTM**, the input sequence is fed into the **backward LSTM layer** in reverse order, starting from the last time step and moving toward the first. As the sequence progresses backward, the backward LSTM, like its forward counterpart, computes a hidden state and updates its memory cell at each time step. This computation is based on the current input and the hidden state and memory cell from the previous time step in the reverse sequence (which corresponds to the next time step in the original order).

Just like the forward pass, the backward LSTM uses input, forget, and output gates to regulate what information is stored, forgotten, or outputted at each step. This allows it to capture patterns and dependencies from future elements of the sequence, effectively learning information that follows the current time step. As it moves in reverse, the backward LSTM retains knowledge of elements that would have occurred later in the sequence.

The hidden states generated by the backward LSTM represent the sequence’s context from the end to the beginning. These backward-directed hidden states are combined with those from the forward LSTM, creating a richer and more context-aware representation of each time step in the input sequence. This dual perspective enables Bi-LSTM models to understand the context surrounding each time step more effectively, leveraging both past and future information.

# CHAPTER 3: WORD EMBEDDING

1. **Word Embeddings**
   1. **What is word embedding?**

* A spatial vector called word embedding is used to represent data that can explain its context, relationship, and semantic similarity. Words that share the same scene or semantics will be found near one another in this multi-dimensional realm.
  + Example: "Today eat apples" and "Today eat mango". Because "apple" and "mango" share the same value position in a phrase, they will be next to one another in the space we represent when we conduct Word Embedding.
  1. **Why using word embedding?**
* Let's try to compare with another representation that we often use in multi-label, multi-task problems, which is one-hot encoding. If we use one-hot encoding, the data we represent will look like this:

| **Document** | **Index** | **One-hot encoding** |
| --- | --- | --- |
| a | 1 | [1, 0, 0, ...., 0](9999 number 0) |
| B | 2 | [0, 1, 0, ...., 0] |
| C | 3 | [0, 0, 1, ...., 0] |
| .... | ...... | ...... |
| mom | 9999 | [0, 0, 0, ..., 1, 0] |
| dad | 10000 | [0, 0, 0, ...., 0, 1] |

* Looking at the table above, we see there are 3 problems when we represent text data in one-hot format:
  + High computational cost: The one-hot vector's length is 100 if the data contains 100 words. The length of the one-hot vector is 10,000 if the data contains 10,000 words. But in order for the model to be very general, it must In practice, the data may contain millions of words, in which case the one-hot vector's length will increase, making computation and storage challenging.
  + Vectors are nearly all zeros, hence they carry little information value. As you can see, text data has relatively little value for pixels (if the input is an image) or other formats. It mostly resides in the semantic relationship and relative positions of words. However, as one-hot vector only indexes the input dictionary order and not the position of words in a particular context, it is unable to express that. In order to get around that, the model frequently employs an LSTM or RNN layer to extract location data. Another method is the transformer model, which adds positional encoding and self-attention layers while removing the word embeddig or RNN layer entirely.
  + Weak generalization: For instance, the terms "mother," "mother," and "bruise" all allude to the mother. But in Vietnamese, the word "bruise" is rather uncommon. The word "bru" has the same meaning as the other two words when encoded using one-hot encoding, but it is categorized into a distinct class because of the differing representation methods. The word embedding will be situated near the other two words if you utilize it, as it can express both positional and semantic information. Similar to my embedding goal
  1. **How to perform Word Embedding**

There are two main methods commonly used to calculate Word Embedding: Count based method and Predictive method. Both of these methods are based on the hypothesis that words appearing in the same context or semantics will be located close to each other in the newly transformed space

* + 1. **Count-base method**
* This method calculates the semantic relatedness between words by calculating the number of co-occurrences of a word compared to other words. For example, you have two sentences as follows:
  + Cat eats fish
  + Cat eats rice
* We built the co-occurrence matrix of the words as follows and found that rice and fish have similar meanings, so they will be located close to each other in the representation vector space.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | cat | rice | fish | eat |
| cat | 0 | 1 | 0 | 2 |
| rice | 1 | 0 | 0 | 1 |
| fish | 1 | 0 | 0 | 1 |
| eat | 2 | 1 | 1 | 0 |

* However, this method has a drawback: when our data is large, some words have a high frequency but do not carry much information (like in English: a, an, the, ...) . And if we statistics this amount of data, the frequency of these words will obscure the value of words that carry more information but are less frequent.
* And to solve the problem, one solution is to re-weight the data to suit our problem. There is a very good algorithm used to solve this problem, which is TF\_IDF transform. In which: TF is the frequency of occurrence of a word in data (term frequency) and IDF is a coefficient that helps reduce the weight of words that frequently appear in data (inverse document frequency). Thanks to the combination of TF and IDF, this method can reduce the weight of words that appear a lot but do not have much information.
  + 1. **Predictive Method (Word2Vec)**

In contrast to the count-based approach, the predictive method uses the input of surrounding words (context words) to determine the semantic similarity between words in order to anticipate the next word. This is done by running the word through a neural network with one or more layers.. One or more words can be considered context words. The words "rice" and "fish," for instance, can be initialized very far apart in the two phrases above. However, in order to reduce the loss between those two words and the context word ("Cat" and "eat"), the two words' positions in vector space must be near to one another. Two widely used prediction techniques are:

* + Continuous Bag-of-Words (CBOW)
  + Skip-gram

Two methods will be introduction in chapter 4.

# CHAPTER 4: WORD2VEC

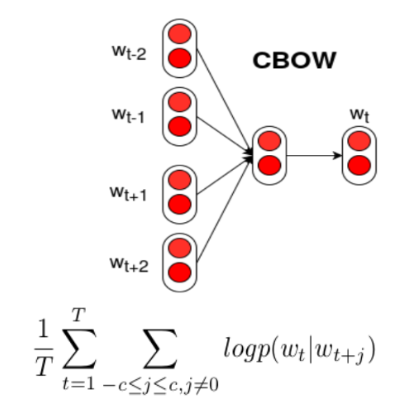
Word2Vec is a widely used technique for creating word embeddings, which transforms words into continuous vector representations in a high-dimensional space. This approach enhances the understanding of semantic relationships between words by capturing their contextual meanings. Word2Vec comprises two primary models: Continuous Bag of Words (CBOW) and Skip-gram. The CBOW model predicts the target word based on its surrounding context, effectively learning from the context to infer the word’s meaning. In contrast, the Skip-gram model operates in the opposite direction, using a given word to predict its neighboring words. Both models significantly contribute to the development of accurate and meaningful word representations.



## Model Architecture

1. **Continuous Bag of Words (CBOW)**

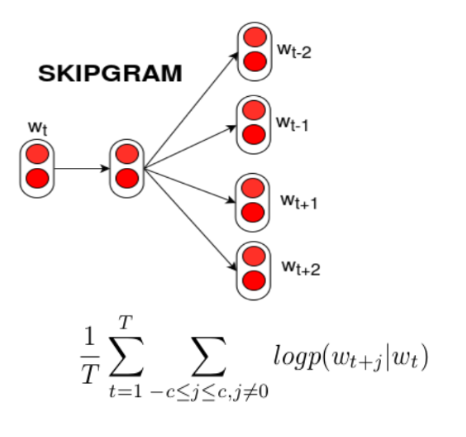
Continuous Bag of Words (CBOW) is a model that predicts a target word using its surrounding context. In this approach, the context words serve as input to the model, while the target word represents the desired output. During the training process, the model learns to minimize the discrepancy between the predicted target word and the actual target word.



By doing so, CBOW effectively captures the relationships between words based on their context, allowing it to generate meaningful word embeddings that reflect the semantic similarities and associations within the text.

1. **Skip-gram**

The Skip-gram model takes a different approach by predicting the context words based on a given target word. In this framework, the target word is used as input, and the model's objective is to identify the words that are likely to occur in its surrounding context.



Similar to the Continuous Bag of Words (CBOW) model, the Skip-gram model seeks to minimize the discrepancy between the predicted context words and the actual context words present in the data. This method allows Skip-gram to effectively learn the relationships between words, capturing their contextual associations and enhancing the quality of word embeddings.

## Neural Network Training

Both the Continuous Bag of Words (CBOW) and Skip-gram models utilize neural networks to develop vector representations of words. These neural networks are trained on extensive text corpora, where they adjust the weights of the connections based on the prediction errors encountered during training. By refining these weights, the models learn to position similar words closer together in the resulting vector space. This training process not only enhances the accuracy of the predictions but also ensures that words with similar meanings or contexts have vector representations that are geometrically close, thereby capturing their semantic relationships effectively.

## Vector Representations

After training, Word2Vec generates a unique vector for each word within a high-dimensional space. These vectors effectively encapsulate the semantic relationships between words, allowing the model to convey meaning through spatial relationships. Words that share similar meanings or frequently occur in comparable contexts will have vectors positioned close together, signifying their semantic similarity. This proximity in the vector space illustrates how Word2Vec captures nuances in language, enabling it to reflect the underlying relationships and associations between different words in a mathematically structured manner.

## Advantage and Disadvantage

**Advantages:**

1. **Effective Semantic Relationships**: Word2Vec excels at capturing semantic relationships between words, enabling the representation of nuances in meaning based on context.
2. **Efficiency with Large Datasets**: The model is designed to handle large datasets efficiently, making it suitable for extensive text corpora, which is essential for generating accurate word embeddings.
3. **Meaningful Word Representations**: It produces meaningful and contextually relevant word representations, facilitating various natural language processing tasks like sentiment analysis, text classification, and machine translation.

**Disadvantages:**

1. **Struggles with Rare Words**: Word2Vec may have difficulty effectively representing rare or infrequent words due to limited context, potentially leading to less accurate embeddings for these terms.
2. **Ignores Word Order**: The model does not take into account the order of words in a sentence, which can result in a loss of important syntactic information and nuances that depend on word arrangement.

# REFERENCE