THAI TOKENIZATION WITH ATTENTION MECHANISM

MR. JEDNIPAT ATIWETSAKUN

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR
THE DEGREE OF MASTER OF ENGINEERING
(COMPUTER ENGINEERING)
FACULTY OF ENGINEERING
KING MONGKUT'S UNIVERSITY OF TECHNOLOGY THONBURI
2021

Thai Tokenization with Attention Mechanism

Mr. Jednipat Atiwetsakun B.Eng. (Computer Engineering)

A Thesis Submitted in Partial Fulfillment
of the Requirements for
the Degree of Master of Engineering (Computer Engineering)
Faculty of Engineering
King Mongkut's University of Technology Thonburi
2021

Thesis Committee

[illegible] .... Chairman of Thesis Committee
(Asst. Prof. Boonsit Yimwadsana, Ph.D.)

[illegible] .... Member and Thesis Advisor
(Asst. Prof. Santitham Prom-on, Ph. D.)

[illegible] .... Member
(Asst. Prof. Nuttanart Facundes, Ph. D.)

[illegible] .... Member
(Assoc. Prof. Peerapon Siripongwutikorn, Ph.D.)

[illegible] .... Member
(Dr. Warasinee Chaisangmongkon, Ph.D.)

Thesis Title	Thai Tokenization with Attention Mechanism
Thesis Credits	12
Candidate	Mr. Jednipat Atiwetsakun
Thesis Advisor	Asst. Prof. Dr. Santitham Prom-on
Program	Master of Engineering
Field of Study	Computer Engineering
Department	Computer Engineering
Faculty	Engineering
Academic Year	2021

Abstract

Word segmentation is an important pre-processing step in natural language processing applications, particularly in languages with no demarcation indicators including such as Thai. Dictionary-based segmentation, while simple, does not consider the context of the sentence. This paper proposes an attention-based deep learning approach for Thai word segmentation. With an additional attention mechanism, the model can learn character correlations across the entire sentence without gradient vanishing or gradient explode problems and then tokenize them into word vectors. The goal of this research is to test two different types of attention mechanisms to determine the effectiveness of word tokenization. The visualization of attention for each attention mechanism is also included as an outcome.

Keywords: Attention/ Deep Learning/ Thai Language / Word Segmentation

หัวข้อวิทยานิพนธ์	การตัดคำภาษาไทยโดยใช้กลไกการให้ความสนใจ
หน่วยกิต	12
ผู้เขียน	นายเจตนิพัทธ์ อธิเวชสกุล
อาจารย์ที่ปรึกษา	ผศ.ดร.สันติธรรม พรหมอ่อน
หลักสูตร	วิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
ภาควิชา	วิศวกรรมคอมพิวเตอร์
คณะ	วิศวกรรมศาสตร์
ปีการศึกษา	2564

บทคัดย่อ

การตัดคำเป็นหนึ่งในขั้นตอนพื้นฐานก่อนทำการประมวลผลภาษาธรรมชาติโดยเฉพาะในภาษาที่ไม่มีตัวบ่งชี้ขอบเขตของคำอย่างชัดเจนอย่างเช่น ภาษาไทย การตัดคำโดยใช้ วิธีอิงคำจากพจนานุกรม เป็นวิธีสะดวก และ เร็ว แต่จะมีปัญหาที่อัลกอริทึมนี้ จะไม่เข้าใจ โครงสร้างของประโยค และความหมายของคำต่างๆในประโยค งานวิจัยนี้จึงเสนอ วิธีใช้กลไกการให้ความสนใจกับการเรียนรู้เชิงลึก เพื่อช่วยในการตัดคำภาษาไทย โดยกลไกการให้ความสนใจ จะช่วยให้โมเดลสามารถเรียนรู้ความสัมพันธ์ระหว่างตัวอักษรแต่ละตัวในประโยค โดยลดปัญหาของการที่ตัวโมเดลไม่สามารถคำนวณค่าความสำคัญของพารามิเตอร์ เนื่องจากค่าน้ำหนักของค่าพารามิเตอร์มีค่าน้อยเกินไป ในงานวิจันนี้ต้องการเสนอกลไกการให้ความสนใจ 2 ประเภทเพื่อสังเกตประสิทธิภาพของการตัดคำภาษาไทย นอกจากนี้ในงานวิจัยนี้ยังได้มีการทำกราฟเพื่อแสดงผลลัพธ์ของการเรียนรู้ความสัมพันธ์โดยใช้กลไกการให้ความสนใจ เป็นหนึ่งในผลลัพธ์เช่นกัน

คำสำคัญ : การตัดคำภาษาไทย/ การเรียนรู้เชิงลึก/ กลไกการให้ความสำคัญ

CONTENTS

CONTENTS (Cont'd)

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 INTRODUTION

Natural language processing (NLP) involves the study of interactions between computer and human languages. The goal is to create a computer that can understand the content of documents or generate spoken words in a way which is similar to how humans do. Human languages are filled with a lot of ambiguities which make them hard to create computer software that can determine the intent of text data. For example, humans can uses instances of sarcasm, idioms, or metaphors. Because of these problems, programmers need to teach applications to interpret text precisely to make the applications useful.

NLP processes human texts in such a way that the computer can understand them. For example, part of speech tagging (PoS Tagging) is a process of determining the part of speech of a word in a text based on its use and context which is known as part of speech tagging. For example, in "They make a cake for me," part of the speech will recognize "make" as a verb, while "What make of car do you own?" would be recognized as a noun. With this information, the computer can understand each of the words in the sentence and use them to learn association which can be useful in question and answering tasks.

This research focuses on tokenization, one of the first steps of NLP tasks. Tokenization is the process that make input texts into a sequence of tokens before applying other processes. As seen in the above example, part of speech tagging involves the algorithm having to find a way to get the word “make” out of the sequence of characters first before finding the part of speech with that word. For English, this can be done easily since words are separated by spaces. Meanwhile, languages like Chinese, Japanese, and Thai do not mark word boundaries clearly. In this case, automatic word segmentation is often required.

Word segmentation tasks in the Thai language is challenging because of linguistic ambiguity. For example, “ขนมไข่” (cupcake) can be separated into ขนม (dessert) + ไข่ (egg) by using words contained in the dictionary. However, there are some cases where separating words using a simple dictionary lookup alone without context can be ambiguous, for example, “ตากลม” can be separated into ตา (eye) + กลม (round) or ตาก (dry) + ลม (wind).

A simple and heuristic way to perform Thai tokenization is the dictionary-based method. But some sequences of words can be ambiguous. For example, of the following sentences: “จักรยานยนต์ชนเสาไฟดับ” for the first case, word segmentation can get the result “จักรยานยนต์|ชน|เสา|ไฟดับ” which means that the motorcycle caused a power outage by crashing into an electricity pole, or it can reach the result in a second case, “จักรยานยนต์|

ชน|เสาไฟ|ดับ" which instead means that the person who rode the motorbike died after hitting the electricity pole. If the tokenizer gets the first case as a result of the NLP application trying to figure out what caused the power outage, then the future processes in the NLP application may receive incorrect information.

In recent years, open-sourced Thai word segmentation have been developed and in wide use, including PythaiNLP[1], Sertis[2], and Deepcut[3]. In PythaiNLP, one of the modules for tokenization is Standalone Dictionary-based, Maximum Matching + Thai Character Cluster (Newmm) which uses the dictionary-based method and also uses help from Thai character clusters (TCCs) to find the best possible token for an unknown token that is received from the dictionary-based method. Both Sertis and Deepcut applications use the deep learning method. The deep learning approach has become more popular in NLP application than dictionary-based and machine learning methods due to its ability to learn the representation and contextual dependencies of words in a sentence. More information about these three techniques are provided in the related work and background section.

Previous approaches to Thai word segmentation use Convolutional Neural Networks (CNNs) [3,4] and Long-Short Term Memory (LSTM) networks [2,5] to improve the effectiveness of the tokenizer and enable the model to understand the context of the sentence. However, these deep learning architectures have their own inherent issues. For LSTM, even if LSTM can overcome the vanishing gradient problem in a recurrent neural network (RNN), it takes a lot of training time because it cannot be trained in parallel due to the sequential calculation hidden states in the sentences. For CNNs, it is fast to train but the way this architecture learns dependencies is limited by the number of kernels that are used in the model, meaning that when the length of input is increased, the number of different kernels required to capture all possible dependencies will also need to be higher.

One of the potential techniques to capture long-term dependencies in deep learning is the attention mechanism. The attention mechanism enables the deep learning architecture to learn contextual dependencies of neighboring words by calculating alignment scores between the current word and other words in the sentence. This mechanism remains unused in word segmentation of the Thai language.

This thesis proposes experiments on attention mechanisms in deep learning for the application of Thai word tokenization. This thesis will determine how attention mechanisms may help with learning context in the Thai word segmentation task. This will benefit the research and development of Thai natural language processing and other text-mining-related applications.

1.1 Objectives

1. To design and implement a deep learning architecture for Thai text tokenization that incorporates attention mechanism.

2. To test and benchmark the proposed method with existing Thai text tokenization techniques.

1.2 Scopes

1. Develop the attention-basis for the Thai word segmentation algorithm that will be the focus of two types of attention, alignment, and transformer.
2. Datasets used in this thesis are obtained from the NECTEC website. The data itself is already tagged with the symbol ('|') to denote word boundaries.
3. The test results will be compared with other open-source Thai word segmentation programs using the same dataset.

1.3 Expected output

1. Determine the best attention mechanism that can achieve the highest performance for word segmentation of the Thai language.
2. Validation and evaluation of the results of the proposed method using the attention mechanism.

1.4 Benefit

1. The proposed method could provide an opportunity for using attention mechanisms in other Thai sequence labeling tasks.
2. This could be a new way to design model architecture for Thai tokenization.

CHAPTER 2 RELATED WORK/BACKGROUND

Tokenization is one of the crucial steps for an NLP application because the input text that the computer receives is a sequence of characters and the machine does not understand the meaning. Tokenization is the process to get words out of the sequence of characters and this process can help the NLP application understand which sequence of characters are tied together and which are not. A tokenization application for the Thai language requires a more complex method to remove words from the sequence of characters. There are three methods for this task.

2.1 Dictionary Based Approach

The early approach to address the word segmentation problem for Thai involves using dictionaries. In this approach, the text is segmented according to words that are defined in dictionaries.

One of the most classic algorithms is Maximal Matching by Sornlertlamvansh [6]. This algorithm is based on backtracking every word that was the result of the longest matching algorithm, calculating the cost for all possible paths, and ensuring that unknown words are grouped.

The maximal matching algorithm uses the following steps. The longest matching algorithm extracts the first set of the longest word from left to right, for example, “กีฬาเป็นการออกกำลังกาย/อย่างหนึ่ง” will return following words “กีฬา”, “เป็นการ”, “ออกกำลัง”, “กาย”, and “อย่างหนึ่ง”. After getting this result, the algorithm then backtracks each word to find the smallest word possible. For example, the word “กีฬา” will not be changed because it is already the smallest word that is contained in the dictionary, but the word “เป็นการ” can change into “เป็น/การ” and the word “การ” will concatenate into the next word, so the result will change the tokenization to “กีฬา/เป็น/การออกกำลังกาย/อย่างหนึ่ง”.

Example results of the maximal matching algorithm are shown in Table 2.1. In this research, the researchers found that tokenization with this algorithm alone cannot achieve 100% accuracy because this method does not analyze the grammar structure or find any correlation between each word.

Table 2.1 Example maximal matching results

Result from Backtracking	Cost
กีฬา/เป็น/การออกกำลังกาย/อย่างหนึ่ง	4
กีฬา/เป็นการ/ออกกำลัง/กาย/อย่างหนึ่ง	5
กีฬา/เป็นการ/ออก/กำลังกาย/อย่างหนึ่ง	5
กีฬา/เป็นการ/ออกกำลัง/กาย/อย่าง/หนึ่ง	6
กีฬา/เป็นการ/ออกกำลัง/กา/ยอ/ย่างหนึ่ง	7
กีฬา/เป็นการ/ออ/ก/กำลังกาย/อย่างหนึ่ง	11
กีฬา/เป็นการ/ออก/กำลังกาย/อย่า/ง/หนึ่ง	12

PyThaiNLP [1] is a popular natural language processing tool for the Thai language that provides various tasks for performing natural language processing. A popular task is word segmentation for the Thai language. PyThaiNLP uses the maximal matching algorithm and Thai Character Cluster (TCC) [7]. TCC does not require a dictionary but needs a set of rules to apply to a group of contiguous characters in texts. To segment TCC, out of sequence characters use the following rules: A front vowel and the next character must be grouped into the same unit; a tonal mark is always located above a consonant and cannot be separated from the consonant; and a rear vowel and the previous character must be grouped into the same unit. These are example rules to segment the sequence of characters into TCC. The image in Figure 2.1 shows example information about consonants, vowels, and tonal marks in the Thai language.

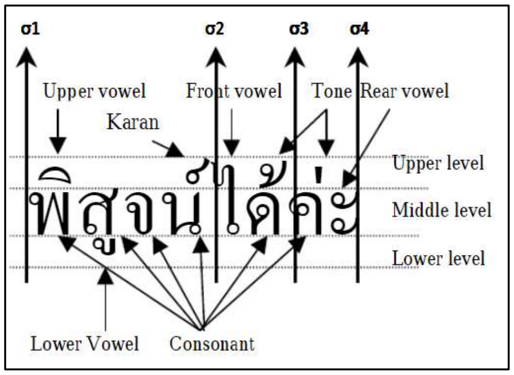


Figure 2.1 Example of Thai character type [18].

PyThaiNLP performs maximal matching first to extract the best set of words from the sequence of characters then they applies TCC to get a set of words that are not contained in the dictionary. With the help of TCC, this algorithm can now get words out of the sequence, even if the dictionary does not contain those words.

2.2 Machine Learning-Based Approach

Before the deep learning method became popular, word segmentation for the Thai language relied on a statistical model to overcome issues of dictionary-based approaches. Machine learning is a branch of artificial intelligence (AI) that makes applications or systems that can learn and improve from experience without being explicitly programmed. The machine learning process begins with observation data or instructions to find patterns in data and make future decisions based on that example. Machine learning algorithms are often categorized into supervised and unsupervised algorithms.

A supervised algorithm is used to learn past data and apply what has been learned to new data with the help of labeled examples to predict the future. This algorithm start from learning data in a training dataset and the algorithm then attempts to produce the function to make predictions about the output values. The algorithm can compare its output with the correct output and find errors to change the function of the model accordingly. This is a popular algorithm for use in classification or regression tasks that seek to predict future values based on current and older values.

An unsupervised algorithm is different because this algorithm does not require labeled data. This method attempts to learn how the system can infer a function to group or describe a hidden structure from unlabeled data. This algorithm does not need to figure out the specific or correct output like with a supervised algorithm, and it can instead draw inferences from datasets to show hidden structures from unlabeled data.

For the Thai tokenization task, the NLP task is a supervised algorithm because the algorithm that this task uses returns results to show whether a corresponding character in the text sequence is the end of a word boundary by learning from previous text data with word boundaries labeled as characters.

Because machine learning is now based on statistical models, some methods require text input to be converted into a numeric input to perform mathematical operations. Bag-of-words is one of the simplest ways to accomplish this. This approach is a simple and flexible way to extract features from documents. A bag-of-words is a representation of text in the form of the frequency of words that occur in a document. This method does not care about the order of the words or the grammatical structure of words in the document. Instead, it only keeps information about the frequency of words that occur in the document. For example, if we have two sentences which are “it is a cat” and “it is a dog, and it can bark”, the algorithm will start by finding unique words from the dictionary first, which from aforementioned example sentences find the following unique words; “it”, “is”, “a”, “cat”, “dog”, “and”, “can”, and “bark”, then it will create a vector to tell which word is which and the frequency that those words appear in each document. The result from the sentences “it is a cat” will be [1,1,1,1,0,0,0,0], with this vector showing information about the word in the dictionary that appears in the sentence and the frequency that that word appears in the document. For the sentence “it is a dog, and it can bark", the vector is now [2,1,1,0,1,0,0,0] because the word "it" occurs twice in the sentence, so the word count is two instead of one.

If the document is very large, the bag-of-words method can result in a sparse vector, a vector with many zeros, because there is a greater chance that the vector of that sentence does not contain most of the words in the vocabulary. Bag-of-words are also not very good to use when we want to understand context. For example, the two sentences, "I love a dog and hate cat" and "I love a cat but hate dog", can result in a similar vector representation despite them carrying different meanings. A better approach to change text representation to numeric is called Word2Vec [8] which is covered in the deep learning approach section.

One major problem in the machine learning approach is called overfitting. Overfitting is a term that is used to refer to an algorithm that fits exactly to its dataset. When this occurs, the algorithm will not perform well against unseen data. To prevent this type of behavior, the dataset must be split into a two part training set and a test set. The training set will be used to send to the model to help it learn and the test set will be used for evaluation. With the help of the test set, it is possible to determine whether what the model or algorithm learn from the training set are overfit or not.

Before moving onto the related work, some background information about machine learning algorithms is required. The decision tree [9] is a machine learning models that can be used for supervised tasks such as classification or regression tasks. The concept of the decision tree is that decisions are based on conditions or features. The decision tree models consist of two elements, nodes and branches. At each node, features are evaluated to split the observations in the training process. Figure 2.2 shows an example result from the customer churn classification, in which the model attempts to predict whether the customer will receive a churn or not by using these features. From the picture, if the value of techSupport is lower than or equal to 0.5 it will go to check monthly changes next, but if the value of online backup is greater than 0.5 then the next node will be the online backup check.

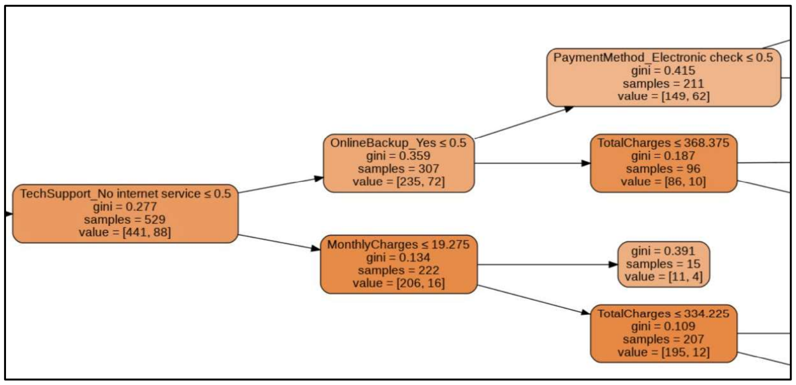


Figure 2.2 Visualization of how decision trees work.

Gini value is used to tell how good each split is. A lower Gini value indicates that the node gets better splitting. The Gini value is calculated by the following equation:

$$Gini\ impurity = \sum_{i=1}^{n} p_i\,(1 - p_i) \qquad (2.1)$$

Here, p_i is the probability of an object that is being classified to a particular class.

The next machine learning model is the Hidden Markov Model (HMM). HMM is based on augmenting the Markov chain which is a model to tell information about the probabilities of sequences in which the probability of each event depends only on the state of the previous event. But in many cases, the events we are interested in are hidden which cannot be observed directly. Table 2.2 shows the components of HMM

Table 2.2 Components of HMM.

Parameter	Description
S= $s_1, s_2, \ldots, s_n$	a set of N states
A= $a_{11}, a_{ij}, \ldots, a_{nn}$	a transition probability matrix
O= $o_1, o_2, \ldots, o_T$	the sequence of T observations
B= $b_i(o_t)$	Emission probabilities
π= $\pi_1, \pi_2, \ldots, \pi_n$	Initial probability distribution

Figure 2.3 shows an example of the Hidden Markov Model. Set of states (S) are rainy, sunny which cause the person to go for walk, shopping, or doing cleaning. The emission probability is the probability that can be observed. For example, it can be observed that a person has a 60% (0.6) chance of a person going for walk-in sunny weather, or a 30% (0.3) chance of going shopping. For transition probabilities, this model includes climate as the hidden state that influences the observations and there are correlations between sunny and rainy days. For example, there is a 20% (0.2) chance for sunny weather to be in the next few days and an 80% (0.8) chance that the weather will change from sunny to rainy.

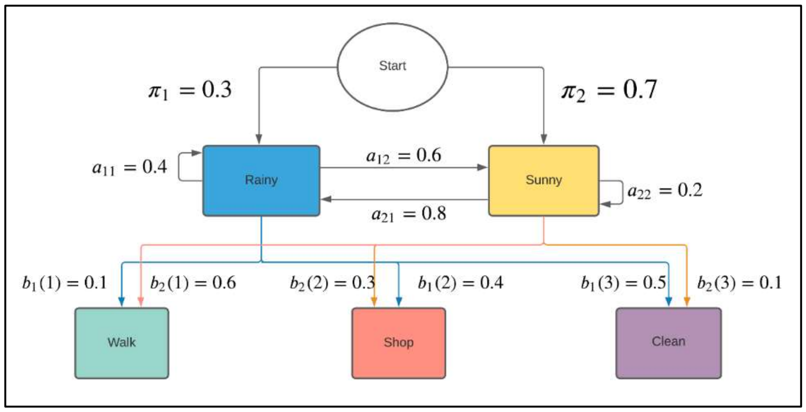


Figure 2.3 Example of Hidden Markov Model.

The Viterbi algorithm is dynamic programming that can be used to find the maximum likelihood estimate of the parameters of the HMM given the set of output sequences. This algorithm tracks the maximum probability and the corresponding state sequence.

Consider the sequence of tasks: Walk, walk, and shop for 3 consecutive days. From Figure 2.4, the probability of the weather being sunny will be the initial probability of sunny multiplied with the probability of sunny and going for a walk which will be 0.42 and the same is also done for it being rainy. For the next day, now the first set of probabilities will go to the next step by starting with whether the next day is sunny and then the probability will be the result of the first day as sunny multiplied with the probability that the next day is also sunny and then multiplied by the probability that a person will go for a walk, which will be 0.42 * 0.2 *0.6 = 0.05. Then it also needs to see the probability that the first day is rainy and next day is sunny which will be 0.03 * 0.8 *0.6 = 0.01. The algorithm also does the same with the rainy path by calculating sunny to rainy and rainy to rainy the same way as it was done before. After that, it will choose the higher probability to be the path. Then the algorithm will go through these steps until it reaches the final days and then chooses a path that has a higher probability.

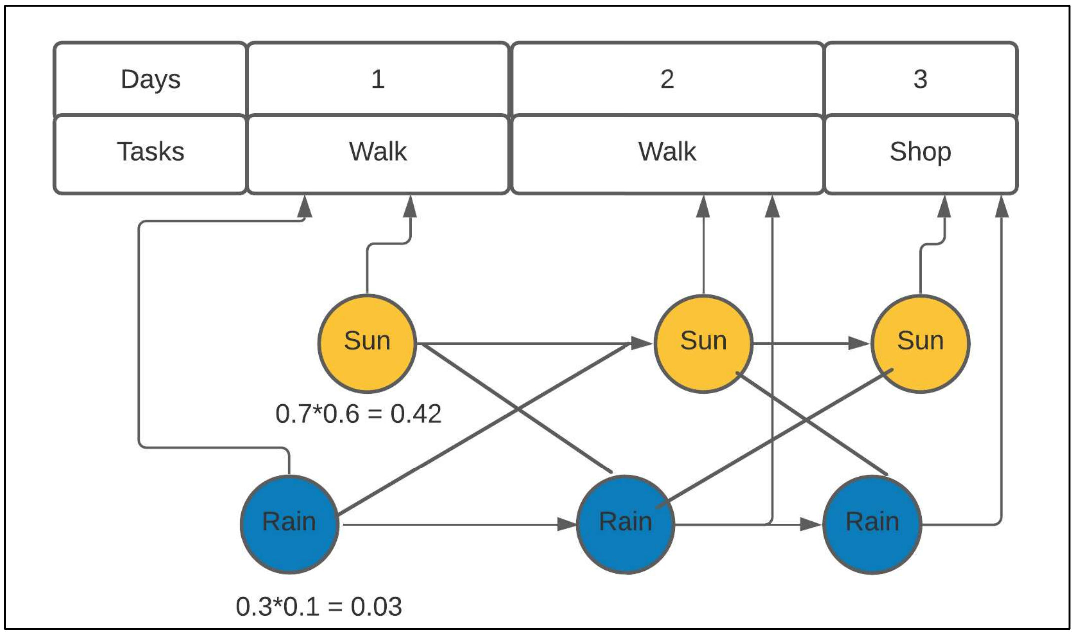


Figure 2.4 Example of Viterbi algorithm in Hidden Markov model.

In 2001, Theeramunkong and Usabanasin [10] proposed non-dictionary-based Thai word segmentation using the decision tree model. They used TCC as a feature to make the model determine whether the word should be formed from TCC based on a predefined metric. In 2009, Bheganan, et al. [11] proposed Thai word Segmentation with Hidden Markov Model (HMM) and decision tree. They created syllable segmentation by applying the standard rules and then used HMM to extract the first set of identified words. The HMM model has been chosen to find the word that are in the size of one to six syllables. In this model, a one-syllable word will pass only state 1, while a two-syllable word passes states 1 and 6. A three-syllable word passes states 1, 5, and 6. A four-syllable word passes states 1, 2, 5, and 6. A five-syllable word passes states 1, 2, 3, 5, and 6 and a six-syllable word passes states 1, 2, 3, 4, 5, and 6. Figure 2.5 show how the hidden Markov for Thai tokenization task look like. The following equation show the probability number of syllable word had been computed:

$$\lambda = (A, B, \pi) \tag{2.2}$$

Here, A is the state transition probability, B is an observation symbol used to counted in a particular state when each possible word is parsed into HMM., and π is the initial state.

They also needed it to be dictionary-based to combine the possible list of words before identifying the sentence and let the decision tree turn any syllables unidentified by the dictionary into a tagged word.

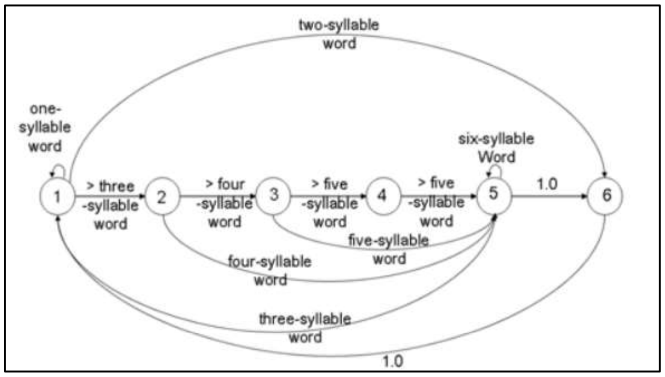


Figure 2.5 A six-state left-right Hidden Markov model [10].

Haruechaiyasak and Kongyoung introduced a Thai character type feature set for CRF-based models by categorizing them as shown in Table2.3. [12]

Table 2.3 Character type mapping.

Tag	Type	Characters
c	Characters which can be assigned as the word ending character	ก ข ฃ ค ฆ ง จ ช ซ ญ ฎ ฏ ฐ ฑ ฒ ณ ด ต ถ ท ธ น บ ป พ ฟ ภ ม ย ร ล ว ศ ษ ส ฬ อ
n	Characters that **cannot** be assigned as the word ending character	ฅ ฉ ผ ฝ ฌ ห ฮ
v	Vowel characters that cannot begin a word	า ะ ◌ำ ◌ิ ◌ี ◌ึ ◌ื ◌ั ◌ู ◌ุ
w	Vowel characters which can begin a word	เ แ โ ใ ไ
t	Tonal characters	◌่ ◌้ ◌๊ ◌๋
s	Symbol characters	◌์ ๆ ฯ
d	Digit characters	0-9, ๑-๙

Table 2.3 Character type mapping (cont'd).

Tag	Type	Characters
q	Quote characters	‘ “ ’
p	Space character within a word	
o	Other characters	a-z, A-Z

The result indicates that using a feature set that combined character and character type can provide effective information to find word boundaries. One study investigated word segmentation with the help of the bi-gram model. In the bi-gram model, we combined two words together in the sentences. For example, “I love dog” the bi-gram model will return “I love”, and “love dog”. Chaonithi, et al. [13] proposed a hybrid approach for Thai word segmentation with a crowdsource feedback system. They used heuristic exhaustive word matching and word bi-gram model with the combined of the crowdsourcing dictionary which takes advantage of an online society to update the dictionary and help the application to solve ambiguity problem. They spilt their dictionary into two dictionaries, one for basic words and another for compound words. Their word segmentation algorithm-generated word sequences based on the basic word dictionary. They fixed ambiguity problems using a bi-gram model with crowdsourcing dictionary to assign the best solution. The bi-gram model will combine result words from heuristic exhaustive word matching and let the society help to decide which word from the result of the bi-gram model is the best possible word.

The result of this method shows that it can get 97.63% in f measure with InterBEST2009 and can get up to 98.54% with the social network corpus. Ronran, et al. also proposed Thai word segmentation on social networks with time sensitivity. This method adopts the ThaiAnalyzer method provided by Apache Lucene to be the first part of segmentation then they used an n-gram combination to measure the probability of two or more segments for the first part occurring together [14].

For classification tasks, the most basic way to evaluate model performance accuracy is by comparing the number of correct predict classes or numbers of real classes, but this will result in poor accuracy for datasets that have imbalanced classes because accuracy looks at correctly classified observations in both positive and negative terms. Another measure, F1 score or F1 measure, involves balancing precision and recall on the positive class. Precision is the fraction of true positive among the model classified as positive and recall is the fraction of true positives among the total number of real positives. Equation 2.2 indicates how F1measure is calculated.

$$F1 = 2 \times \frac{precision \times recall}{precision + recall} \tag{2.3}$$

For the Thai tokenization task, model performance can be evaluated using F1 measure because in the tokenization task care, the more positive class can tell whether the model can result in a good result. In addition, F1 measure must be used if the Thai tokenization dataset is mostly unbalanced with more class 0 (characters that are not the end word of a boundary) contained in the dataset than class 1 (characters that are at the end of a word boundary). If accuracy was used, then the result can achieve very high accuracy, but the model performance does not follow it because class 0 outnumbers class 1.

2.3 Deep Learning-Based Method

Deep learning has become one of the most popular methods in the natural language processing field due to its ability to learn the representation and contextual dependencies of words in a sentence. In Thai word segmentation, deep learning-based approaches have also become popular because the model can achieve high performance despite their computational complexity.

Deep learning is another sub-field in machine learning primarily related to the artificial neural network (ANN) architecture. The simplest unit in the ANN is a neural unit, combined with the input's weight and activation function, as shown in Figure 2.5. The linear function is formed from the multiple weighted inputs that sum together and passed to an activation function which will transform linearity into non-linearity. The following equation shows how the neural unit returns the result.

$$\hat{y} = activation(wx + b) \tag{2.4}$$

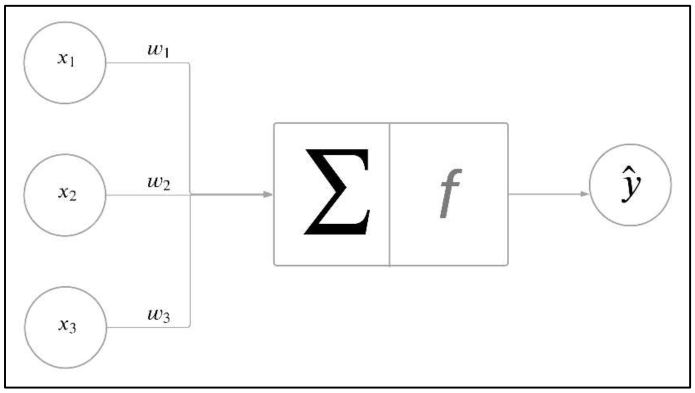


Figure 2.6 Neural unit.

There are a lot of activation functions that can be used in the neural unit. Result from the sigmoid will return in the range of 0 to 1. This type of activation is used to decide the result of the classification function. Softmax is another activation function that can be used to get the result of the classification task by returning values in the range of 0 to 1 and also each value will sum up to 1 which is good for use with multi-class data. Tanh function is another popular activation function to use because it can change the result of the neural unit to a non-linear result. Another popular activation function is ReLU that also can change the result of the neural unit to a non-linear result.

The multiple neural units form a fully connected layer. The stack of the fully connected layer forms a deep neural network or multilayer perceptron, as shown in Figure 2.6.

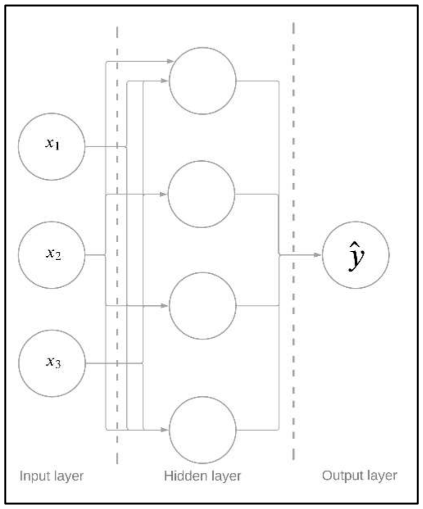


Figure 2.7 Multilayer perceptron (MLP).

Deep learning uses a gradient-based learning method as an optimization task to train the model. The goal is to minimize the cost function $J(\theta)$, where θ is a model parameter of weight in each neural unit where $\theta \in$ R. The cost function for the classification task is cross-entropy, which is a measure of the difference between two probability distributions for a given random variable or set of events and can be calculated using the following equation:

$$Cross\ entropy\ = \frac{1}{N} \sum_{i=1}^{N} -(y_i log(\hat{y}_i)\ + (1 - y_i)log(1 - \hat{y}_i)) \qquad (2.5)$$

Here, $\hat{y}_i$ is the result of the probability. When the observation belongs to class 1, the first part of the formula becomes active and the second part will not be used because $y_i\ =\ 1$ causes the second part to become zero and vice versa when the observation of the actual class is $y_i\ =\ 0$.

The objective is to find θ, which can give a model parameter vector that results in minimum cost, as follows the equation:

$$\theta^*\ =\ argmin\ J(\theta) \qquad (2.6)$$

The gradient-based learning uses a gradient to minimize the cost function by taking the first-order partial derivative to the cost function ($\nabla_\theta J(\theta)$). The result of the derivative will result in a slope at point θ. If the result at point θ is decreased, this means that the slope moves in the direction of the negative gradient as shown in equation 2.6, while if the slope is returned as positive, the update cost function is changes to equation 2.7.

$$J(\theta + \in)\ \ \approx J(\theta) - \in \nabla_\theta J(\theta) \qquad (2.7)$$

$$J(\theta + \in)\ \ \approx J(\theta) + \in \nabla_\theta J(\theta) \qquad (2.8)$$

The algorithm minimizes the cost function and updates the parameter using the following equation:

$$\theta_n = \theta_{n-1} - \in \nabla_\theta J(\theta) \tag{2.9}$$

Here, $\in$ is the learning rate, a step size of the gradient of the parameter θ_{n-1}. The deep learning model will repeat these steps until it can reach the local or global minimum. The model will stop at that point because the derivative result is returned as zero and with that value, the model cannot update the parameter anymore.

One problem with the deep learning approach is the local minimum which causes the model to no longer update. The local minimum causes model to think that it is the lowest point that can achieved, but there is still another point called the global minimum that is the real lowest point. There are many optimization techniques that can avoid this local minimum. Mini-batch gradient descent is one such technique that can be used to fix this problem, which is computed as an average of a gradient over a small batch of training data, as follows:

$$\theta_n = \theta_{n-1} - \in \nabla_\theta J(\theta_{n-1}, x^{(i:n)}, y^{(i:n)}) \tag{2.10}$$

Here, $x^{(i:n)} = \{x^i, x^{i+1}, x^{i+2}, \dots, x^n\}$ and $y^{(i:n)}$ is the corresponding label for that batch $x^{(i:n)}$. Another popular technique is called Adam optimization which can be used to fix this local minimum problem. Adam optimization (adaptive moment estimation) [15] is an adaptive learning rate for stochastic gradient optimization. It adjusts the learning rate during the model training the same as the previous optimization technique. Adam optimization applies both the momentum and the accumulated sum of square gradient, as follows:

$$v_t = \beta_1 v_{t-1} + (1 - \beta_1)g \tag{2.11}$$

$$s_t = \beta_2 s_{t-1} + (1 - \beta_2)g^2 \tag{2.12}$$

Here, terms $v_a \; and \; s_a$ are the first momentum and the second momentum of the gradient. But these values are initialized as the vector of zero and authors of Adam observe that they are biased toward zero especially during the initial time step and when the decay rates ($\beta_1 \; and \; \beta_2$) are close to one. Thus, they change the correct bias terms to as follows:

$$\widehat{v_t} = \frac{v_t}{1 - \beta_1^t} \tag{2.13}$$

$$\widehat{s_t} = \frac{s_t}{1 - \beta_2^t} \tag{2.14}$$

The learning rate is updated as follows:

$$\epsilon = \epsilon \frac{\widehat{v_t}}{\sqrt{\widehat{s_t}} + \delta} \tag{2.15}$$

Kingma, et al. [15] propose that the default value of $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\delta = 10^{-8}$ can work well in practice. This method becomes a common design choice for the optimization function to train a deep learning model.

The next component of deep learning architecture is the activation function. Without the activation function, the deep learning model is no different from the linear model. The rectified line unit (ReLU) is a popular activation function used to train a model. ReLU can solve a vanishing gradient problem (more detail in chapter 3) that can happen during backpropagation and this function can run very quickly, and can be observed in the following equation:

$$f(x) = max(0, x) \tag{2.16}$$

The ReLU function can improve the model converge rate and create a sparsity of the network by deactivating out some neurons by putting a zero to those neurons when the value from the neuron is less than zero.

The last part of the deep learning approach is called backpropagation. Backpropagation is an important step of deep learning training which uses a chain rule of differential calculus by fine-tuning the weights of a neural net based on the error rate or the result of the loss function obtained in each epoch. This starts with forwarding propagation which is defined in equation 2.13.

$$\hat{y} = x^L = \sigma(w^L x^{L-1} + b^L) \tag{2.17}$$

Here is the multilayer neural network, having a weight vector as w, input vector or result from the previous vector as x, a differentiable activation function as σ, L is a total number of the layer, and $\hat{y}$ as a result of the forward propagation. The error term is computed by using the cost function by comparing a prediction $\hat{y}$ with an actual result y which is defined as follows:

$$Error = Cost(y, \hat{y}) \tag{2.18}$$

The gradient is then computed from an error term by using the chain rule of differential calculus by starting from the output of the model back to the first layer of the model defined as:

$$\frac{\partial\, Error}{\partial w^l} = \frac{\partial\, Error}{\partial a^l} \cdot \frac{\partial\, a^l}{\partial z^l} \cdot \frac{\partial z^l}{\partial w^l} \tag{2.19}$$

$$\frac{\partial\, Error}{\partial w^l} = \frac{\partial\, Error}{\partial a^l} \cdot \acute{\sigma}(z^l) a^{l-1} \tag{2.20}$$

Here, $\frac{\partial\, Error}{\partial a_k^l}$ is the result of previous backpropagation. The deep learning model is training by doing forward propagation by estimating an output from a given input and using backpropagation until reaching a certain number of iterations (number of epochs). One epoch is equal to training a model over a dataset once.

2.3.1 Overfitting in deep learning

The deep learning approach also has an overfitting problem similar to the machine learning approach, but it can result in a greater chance of overfitting because the deep learning approach model is more complex than the machine learning method. There are also more choices to overcome this problem.

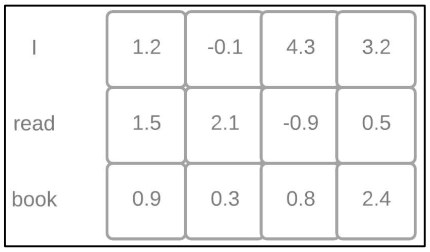
Dropout, or dropout regularization, is a method that attempts to randomly mask the output of the neural unit to zero. By doing this, it makes the model train in different neural units set in each training iteration which forces each neural unit to be independent.

Early stop mechanism is regularized technique that monitors and stops the optimization process to prevent the model from overfitting the training data. Early stopping rules give the model the number of iterations that can be run before the model become overfit or check the performance of the validation dataset if it starts to decrease compared to the performance on the training data. The validation dataset must be from the same domain or have the same distribution as the training set. The easiest way to get a validation dataset is by splitting out the data from the training set.

Simplifying the model also prevents the model from overfitting. This can be the first step to be performed when dealing with overfitting. To decrease the complexity, layers can be removed or the number of hidden units in neurons can be reduced to make the network smaller. There are no general rules about the number of layers or the number of hidden units that should be contained in the model. But attempting to make the model smaller can prevent it from overfitting.

2.3.2 Word embedding

Word embedding is the key reason why deep learning has become popular for NLP applications. Word embedding is a training representation for the text that creates a similar representation for words with the same meaning. An embedding layer is the simplest technique to accomplish word embedding in a deep learning approach. An embedding layer is a dense vector of floating values in which the length of the vector is the hyperparameter that can be chosen. The embedding layer has weights as trainable parameters the same as the fully connected layer. A higher dimensional embedding can capture more relationships between words, but it also requires more data to learn.



I	1.2	-0.1	4.3	3.2
read	1.5	2.1	-0.9	0.5
book	0.9	0.3	0.8	2.4

Figure 2.8 A 4-dimensional embedding.

Figure 2.8 presents an example result of a word embedding. Each word is represented as a 4-dimensional vector of floating values. Now the model can encode each word by looking up this dense vector it corresponds to in the table.

An embedding layer is the default method to create word representation in the deep learning approach. Most previous research discussed in the present study used this method to create character representation for the input used to train and test the model. Before going into related work, there is also another possible method to create word representation called Word2Vec.

Word2Vec [16] is a standalone statistical model that learns word embedding from a text corpus. Two different learning methods can be used as part of the Word2Vec which are Continuous Bag-of-Words (CBOW) and continuous Skip-Gram. CBOW learns the embedding by letting the model try to predict the current word based on its context or its surrounding words. Meanwhile, the continuous Skip-Gram lets the model try to predict the surrounding words by giving a current word.

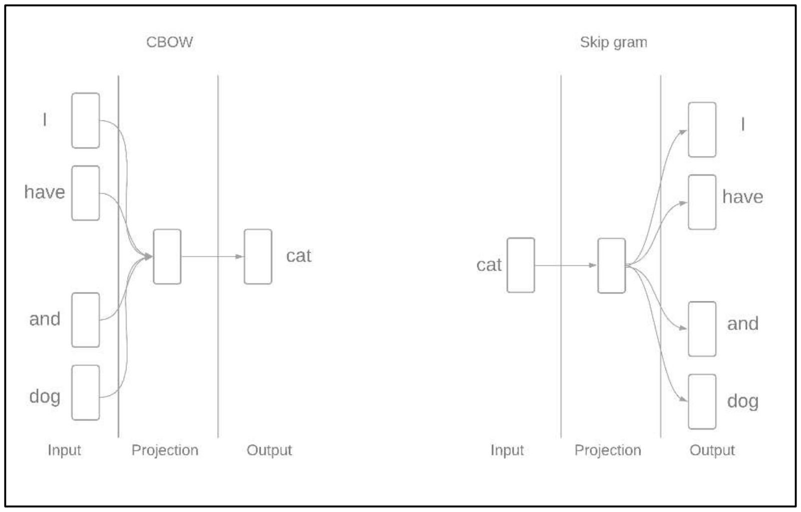


Figure 2.9 CBOW and Skip-gram method of Word2Vec.

Figure 2.9 shows the visualization of how the Word2Vec model works in both CBOW and Skip-Gram. For example, the current word is "cat" when CBOW tries to learn what surrounding words are associated with the word "cat" to make the model be able to predict the word "cat". Skip Gram is another method which uses the word "cat" and finds surrounding words that could be associated with the word "cat".

Word2Vec can give out word embedding and does not need to learn through the training process of the deep learning approach. It became a popular method for pre-trained word embedding. Training the document through Word2Vec first before sending the weight result to the word embedding layer of the deep learning approach can make the model work more efficiently.

In Thai tokenization, the deep learning method became very popular due to the capability of the deep learning method that can efficiently learn from a large dataset and can achieve very high performance compared to the machine learning approach. For word embedding, most applications use the embedding layer to make the model learn character representation instead of using Word2Vec as pre-trained due to the tokenization application of the input requiring character representation.

There are two architectures of deep learning that are mostly used in this research area due to the ability to work the long-range dependency, namely recurrent neural network and convolutional neural network.

2.3.3 Recurrent Neural Network

Recurrent neural network (RNN) is a type of neural network architecture in which the output of the previous step is fed into the model as inputs of the current step. This allows RNN to remember each piece of information over time, which is useful for tasks such as time series prediction and natural language processing. Treeratpituk, et al. [5] proposed Cutkum, which uses RNN architecture at the character level to construct word segmentation for the Thai language. On BEST2010, they received a score of 95.0% in the f measure.

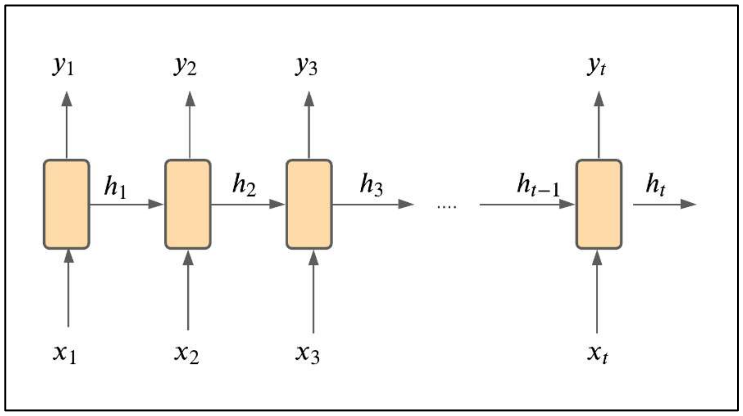


Figure 2.10 RNN architecture.

A fully connected neural network is unable to capture the sequential because each of the layers in the network considers each frame independently. RNN specializes in extracting a sequential representation. An RNN unit is like a general neural unit, but contains a recurrent state or hidden state h which is fed an activation to itself from time t to time t+1. In another term, RNN will have information on the current time step and previous time step. The equation of the RNN unit is defined as follows:

$$a^{(t)} = W * [h^{(t-1)}, x^{t}] + b \tag{2.21}$$

$$h^{(t)} = \sigma(a^{(t)}) \tag{2.22}$$

Here, W is the weight matrix composed of the weight of the input feature and the weight of the recurrent state and $a^{(t)}$ is the result of the timestep t before sending into activation function (σ). The activation function of RNN is the tanh function. For that reason, it causes a vanishing gradient problem when applied with a long sequence feature. The vanishing gradient problem happens from a derivative getting smaller and smaller as it goes backward with every layer during backpropagation which can be resolved by changing the activation function from tanh to ReLU or changing the architecture to long short-term memory.

Sertis (Sertis Co., Ltd.) [2] performs word segmentation for the Thai language that uses bi-directional RNN as architecture. Bi-directional RNN was introduced by Schuster and Pakiwal [17].

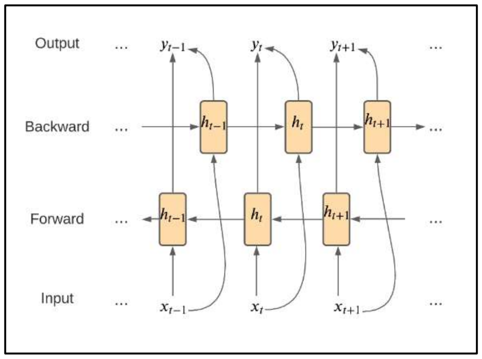


Figure 2.11 Bi-directional RNN architecture.

In any timestep *t*, given input X_t and let the activation function of the hidden layer be "σ". In bi-directional architecture, let the forward and backward hidden states for this time step be $\overrightarrow{H_t}$ and $\overleftarrow{H_t}$. The updated equations for the forward and backward hidden states are as follows:

$$\overrightarrow{H_t} = \sigma(W^{(f)} * [h^{(f)}_{(t-1)}, x^{(t)}] + b^f) \tag{2.23}$$

$$\overleftarrow{H_t} = \sigma(W^{(b)} * [h^{(b)}_{(t-1)}, x^{(t)}] + b^b) \tag{2.24}$$

$$y_t = [\overrightarrow{H_t}, \overleftarrow{H_t}] \tag{2.25}$$

The general RNN is a one-direction recurrent state which can be the forward direction or backward direction. By using both directions, the model can have information from both directions which can help the model learn the association of input that come before and after. Output y_t is the concatenation of the result from both forward and backward directions. The result from this architecture is claimed to get an F measure of 99.18% on the InterBEST2009 dataset.

Long short-term memory (LSTM) is designed to overcome the problem of RNN. LSTM has an internal mechanism called gates that can control the flow of information. These gates can help the model learn which input in a sequence is important to keep or throw away. The computation of the LSTM unit is defined as follows:

$$f_t = \sigma(W_f[h_{(t-1)}, x_t] + b_f) \quad (2.26)$$

$$i_t = \sigma(W_i[h_{(t-1)}, x_t] + b_i) \quad (2.27)$$

$$o_t = \sigma(W_o[h_{(t-1)}, x_t] + b_o) \quad (2.28)$$

$$a_t = tanh(W_a[h_{(t-1)}, x_t] + b_a) \quad (2.29)$$

$$C_t = f_t C_{t-1} + i_t a_t \quad (2.30)$$

$$y_t = h_t = o_t\, tanh(C_t) \quad (2.31)$$

LSTM has a similar input to RNN but also require more information to call the cell state (C_t). The cell state is used to carry relative information all way down the sequence of the model. Each gate of LSTM uses an activation called the sigmoid function which is used to give the probability in the range of 0 to 1. With this probability, it can tell whether to use gates or not. When the value from the sigmoid is near 0, the gate will not be used and if the value is near 1 then the gate will be active. LSTM consists of the following gates: Forget gates (f_t) which decide whether information should be kept or thrown away; input gates (i_t) which decide whether the previously hidden state information will be kept in the cell state or not; and output gates (o_t) which decides what the next hidden state should be using the new hidden state which learns from $[h^{(t-1)}, x^t]$ or by using information from the current cell state (C_t).

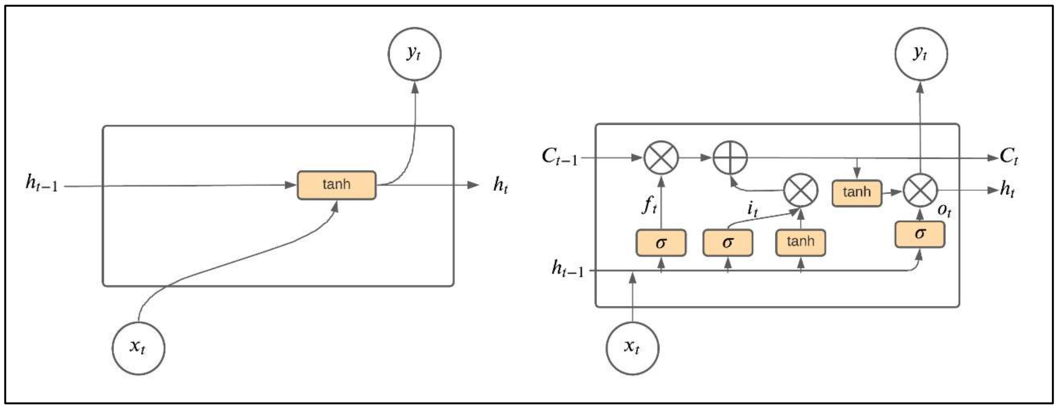


Figure 2.12 Difference of RNN unit (Left) and LSTM unit (Right).

This research used Bidirectional LSTM (Bi-LSTM) as a model basis. Bi-LSTM works the same as Bi-RNN, with the only change being that the RNN unit is changed to the LSTM unit. LSTM unit was chosen because it can reduce the vanishing gradient problem and can be bidirectional instead of one-way directional because the model can learn information in both directions. In the tokenization task, we must feed input as a character and let the LSTM learn the association with other characters in the input sentence then the model will use that information to predict whether the output class of this character is the start of the word boundary or not.

2.3.4 Convolutional Neural Network

A convolutional neural network (CNN) is a type of neural network architecture that has been mostly used for image and video processing. The main concept of CNN is the application of multiresolution analysis by applying a kernel to extract spatial or temporal features from images or video. This kernel in CNN is a convolution operator that estimates the feature map of the image through the process of convolution between an input image and a kernel in form of a small window. Multiresolution analysis is done by pooling which lowers the feature map resolution such that the resulting output represents features at lower resolution. This downsampling technique reduces data dimensionality and lowers the computational complexity of the model. The convolutional layer applies convolution operation is defined as follows:

$$a = \sum x(\tau + t)w(\tau) \tag{2.32}$$

Here, the equation of a convolution operation. The term a is a result before sending it into an activation function. The local region is calculated by multiplying kernel $w(\tau)$ to the entire input feature where τ is a hyperparameter of kernel size. There are also other hyperparameters called stride which is a skip or shift operation that identifies the number of the step of the kernel to shift and padding is used to specify the edge's padding of the input during convolution.

For the classification task, the convolutional layer must use the flatten operation before sending the result into the fully connected layer because the fully connected layer cannot operate data that has multiple channels.

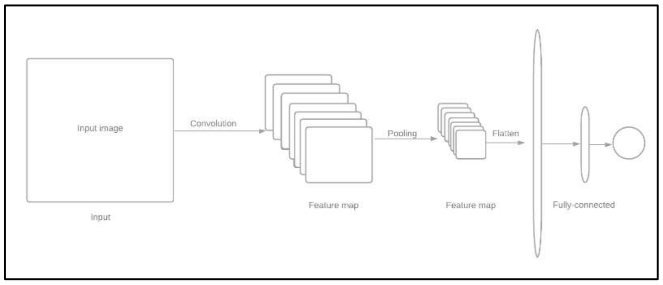


Figure 2.13 CNN architecture for classification.

From Figure 2.13, when input as tensor with shape (number of inputs) x (input height) x (input width) x (input channels) pass through a convolutional layer, the image becomes abstracted to a feature map with shape as (number of inputs) x (feature map height) x (feature map width) x (feature map channels). Pooling layers is used to reduce the dimensions of the feature maps. It reduces the number of parameters to learn and help summarize features present in a region of the feature map. Flatten layer will flatten all information of the feature map up to a single array before sending it into a fully connected layer for performing classification.

In computer vision, the image is usually in 3 dimensions, which are width, height, and depth. But for text there will only be 1 dimension which is the length of the sentence. For word segmentation in the Thai language, this method has been used in DeepCut (Kittinaradorn, et al.) [3]. DeepCut's structure combined with character type embedding, character embedding, 1-dimensional convolutional layer with kernel width from 1 to 12, pooling layer, flattening and concatenation layers, and fully connected layer. The result reported an f measure of 98.18% on the InterBEST-2009. Input data of the Deepcut method contain these 10 characters that come after the current character and 10 characters that come before the current character. The vector of one input will be [char1, char2, …, char10, char-1, char-2, …, char-10, char]. This makes the input size of each character be 21. The authors of this method did not state why they decided to do this, but it could be because they wanted the model to learn associations in both forward and backward directions.

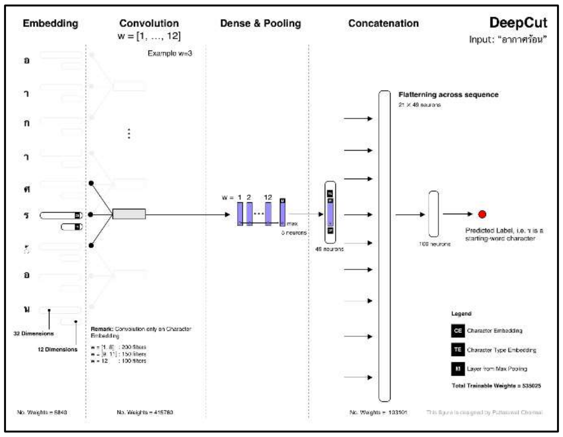


Figure 2.14 Architecture design of Deepcut application [3].

AttaCut (Chormai, et al.) [4] was inspired by DeepCut and applies the dilation technique [18] in the convolutional layers which makes the model cover sufficient contexts with fewer parameters. The difference between standard convolution and dilated convolution can see as follows:

$$(F * k)(p) = \textstyle\sum_{s+t=p} F(s) \cdot k(t) \tag{2.33}$$

$$(F *_l k)(p) = \textstyle\sum_{s+lt=p} F(s) \cdot k(t) \tag{2.34}$$

Where F is a receptive field, k is the convolution kernel, and p is the size of the receptive field. Equation 2.30 is standard convolution and equation 2.31 is dilated convolution which gets the variable dilation rate (l) added into the summation part. When l in the summation part is greater than 1, it will make the receptive field of the filter size get larger and help to capture more context with a smaller number of parameters than using standard convolution.

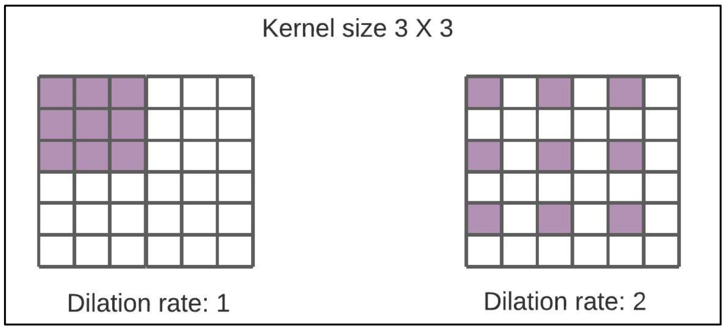


Figure 2.15 Example of how dilation techniques work.

AttaCut also adds syllable embedding as a feature to improve the contextual representation. The result reported an F measure of 89% on BEST-2010 for AttaCut with no syllable feature and 91% on BEST-2010 for AttaCut with syllable embedding as a feature.

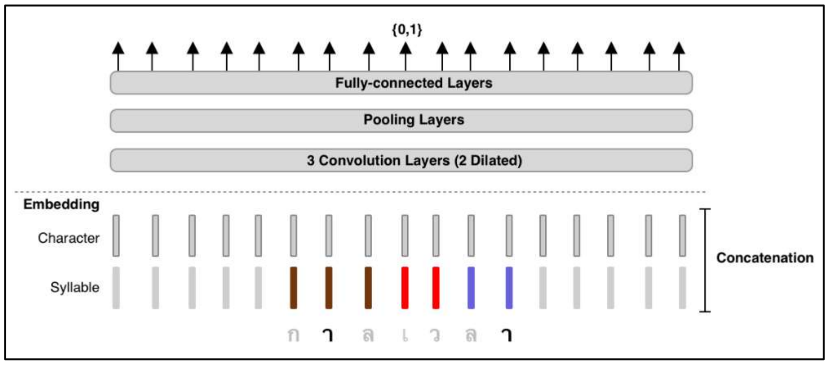


Figure 2.16 Architecture design of Attacut application [4].

In summary, the dictionary-based method is the easiest way to create Thai tokenization, but it has an ambiguity problem so this method alone cannot solve the problem and it requires a dictionary to be updated frequently. The machine learning approach can solve the dictionary problem by giving data to train the model first and that model can then be used to predict the word boundary. By doing this, the model does not require an updated dictionary.

Deep learning approach methods show that with the help of this approach, tokenization tasks can run more correctly due to the deep learning model being more complex than the machine learning approach and the word embedding can help the model learn word representation through the training process. With the help of RNN and LSTM, the model can learn the long-range association and with the help of bidirectional and now the model can learn association from both forward and backward. For CNN, the model is instead learning the association of the surrounding word and can run faster than RNN and LSTM methods.

There is a deep- learning concept called Attention which can put emphasis on the contextual dependencies as well as highlight important parts of the information. The attention mechanism in deep learning can be interpreted as a vector of weight. Attention vectors can then show how strongly the current element is correlated with other elements.

2.4. Attention Background

The attention mechanism that will be used in this research is based on the concept of the attention mechanism in the sequence- to- sequence model, which states that the attention mechanism must be used with deep learning architecture known as alignment models, as well as the concept of the transformer model.

Attention is what humans use to focus on something or take great notice of something. When we give attention to the sentence, we can understand what information the sentence wants to give. From Figure 3.1, the sentence is “I am eating a red apple”. When we read this sentence, we focus on who the sentence talks about and what that subject attempts to do. From the example, when we find the word “eating” appear, we can expect to find a food word soon and the adjective that describes the food will be focused on the food rather than the subject or verb.

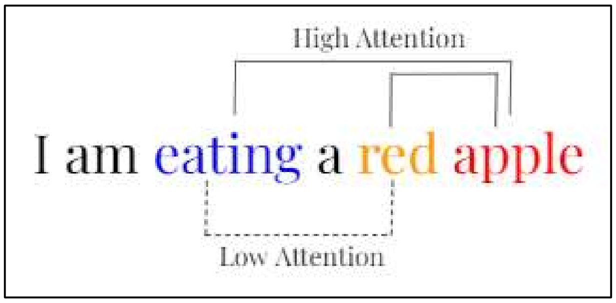


Figure 2.17 Example of how attention work.

Researchers attempt to mimic this mechanism to the deep learning approach to make the model be able to pay attention to certain factors when processing data because the context dependencies are very important when attempting to process the text input.

The attention-based model can help the deep learning approach to learn long-range dependency with the help of the context vector. The context vector contains all information in every time step and has the information about the importance of each input to the corresponding input which can help the model overcome the vanishing gradient problem and in a certain type of attention-based models, this type of model can help the model run faster than models that do not have an attention mechanism.

In this research, we attempt to create attention-based models to observe whether this type of model can improve in Thai tokenization tasks by observing both performances based on the degree of correctness and speed. We also create a visualization to see the result of character association from the attention-based model.

The attention mechanism was first proposed by Bahdanau, et al. for use in neural machine translation [19]. The model architecture in that paper is the same as the sequence-to-sequence model [20], but Bahdanau added alignment scores that calculated from the information of all hidden states of the encoders and the information of the t-1 state of the decoder which allow the model to find the relation between tokens after multiplying them with their encoder hidden states to form the context vector.

Bahdanau, et al. came up with this mechanism because he found out that the sequence-to-sequence model had a problem in the encoding part where the result of the encoder part must squeeze the information of the source sentence into a fixed-length vector which can cause the performance of the-sequence-to-sequence model to drop when the length of the input sentence lengthens.

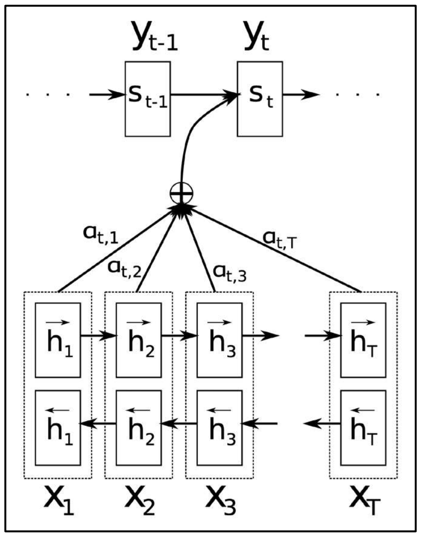


Figure 2.18 Architecture of Bahdanau attention [21].

From Figure 2.18, the architecture is Bi-directional RNN and the result for both forward and backward will be sent into the context vector to calculate attention score. The context vector is calculated using the following equations:

$$h_i = [\overrightarrow{h_i}, \overleftarrow{h_i}] \tag{2.35}$$

$$a(s_{t-1}, h_i) = v^T tanh(W_1 h_i + W_2 s_{t-1}) \tag{2.36}$$

$$\alpha_{t,i} = softmax(\ a(s_{t-1}, h_i)\ ) \tag{2.37}$$

$$c_t = \sum_{i=1}^{T} \alpha_{t,i}, h_i \tag{2.38}$$

Here h_i is the result from Bi-directional RNN, attention score $(a(s_{t-1}, h_i))$ of Bahdanau attention is using the addition of the previous result (s_{t-1}) with the result of Bi-directional RNN and v is a weight vector, the result of the attention score $(\alpha_{t,i})$ will be sent into Softmax function before doing weighted sum for the result as the context vector (c_t). The result of their work shows that the help of an attention mechanism can translate the long sentence correctly when compared to the vanilla sequence-to-sequence model. After that, Luong, et al. introduced other ways to calculate scores for attention mechanisms which are Global and Local Attention [21].

The alignment score of global attention is calculated by doing matrix multiplication between the target hidden state and each of the source hidden states. Meanwhile, the score of local attention is calculated by source hidden states within a selected window with target hidden states which will make computations less expensive than global attention. Global attention considers all the source words in the input sentence when calculating the alignment score for the context vector. The equation of global attention is similar to the alignment score from the Bahdanau method. Local attention is required to find the window centered over an aligned position (p_t) which can be calculated from the following equation:

$$p_t \;=\; S \cdot Sigmoid(v_p^T \, tanh[W_p, s_t]) \tag{2.39}$$

Here, S is the length of the source sentence, and both v and W are the model parameters. The result of this equation will return as [0, S]. They also place a Gaussian distribution centered around p_t which can be defined as:

$$align(s_t, h_i) \;=\; align(s_t, h_i) exp(-\frac{(s-p_t)^2}{2\sigma}) \tag{2.40}$$

Here, $align(s_t, h_i)$ is the function for calculating the alignment score. Table 3.1 shows the alignment score that can be used in this type of attention-based model.

Table 2.4 Alignment score function that can be used to calculate attention score.

Name	Alignment score function
Additive/Concat	$score(s_t, h_i) \;=\; v^T tanh(W_1 h_i \;+\; W_2 s_t)$
General	$score(s_t, h_i) \;=\; s^T{}_t W h_i$
Dot-Product	$score(s_t, h_i) \;=\; s^T{}_t h_i$

The difference between the Bahdauau method and the Luong method is the computation of the alignment score($score(s_t, h_i)$), in which Luong method depends on the current decoder hidden state (s_t) but Bahdauau method depends on the previous decoder hidden state (s_{t-1}) and the Luong method also investigate more in multiplicative attention rather than using only additive attention. Table 3.1 shows the alignment score that is used in the Luong method, from which it is evident that there are two multiplicative methods here, general and dot-product. In the general method, the result of the attention score needs to be multiplied with trainable weight (W), but this is not required with the dot-product method.

The alignment method requires deep learning architecture to work by creating a context vector between the encoder and the decoder in the case of machine translation task, but it also can be used for classification tasks by changing the decoder layer to a fast forward layer. In this research, additive and general are used as alignment score functions.

Vaswani, et al. proposed a new type of attention mechanism which is called transformer [22]. This attention mechanism can be built without deep learning architecture by using its architecture. The main concept in the transformer is the unit of the multi-head self-attention mechanism. The transformer encoded input as a set of query-key-value (Q, K, V). To calculate attention score, the current word in the sentence needs to do the dot product with Q and K from every word in the sentence and be sent to the SoftMax function to calculate alignment and be multiplied with the value vector of each word and then added up to get the context vector. The following equation shows the attention score of scaled dot-product attention:

$$score(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.41}$$

Here, you can see that this attention score is quite similar to multiplicative attention except for the scaling part $(\frac{1}{\sqrt{d_k}})$ where d_k is the number of the dimension of the key vector which can be simplified to $(\frac{1}{h})$, where h is the number of parallel attention layers.

Table 2.5 Example of how scaled dot-product work.

	Big	**Brother**	**is**	**watching**	**you**
Q	[1]	[2]	[3]	[4]	[5]
K	[1]	[2]	[3]	[4]	[5]
Q*K[i]	[1]	[2]	[3]	[4]	[5]
SoftMax	0.74	0.11	0.44	0.54	0.33
V	[1]	[2]	[3]	[4]	[5]
SoftMax * V	[1]	[2]	[3]	[4]	[5]
Z	[1]	[2]	[3]	[4]	[5]

Table 2.5 shows how scaled dot-product attention works. To start, we must initiate the weight value of Q, K, and V and then these weights must be multiplied with the input to make each word have its Q, K, V values. Then we will multiply the value Q of the word corresponding with value K from all of the words in the sentence. After that, we get probability out by using the Softmax function, and then we will multiply the result of the Softmax function to each value V from all of the words. This process is used to determine how important each word is to the corresponding word and sum all of them to get the result. This is an example of how on scaled dot product work and for multi-head attention is only doing each scaled dot product parallel to make the result more variant because each head can have a different focus, in part due to the different results that we get from weights Q, K, and V through the learning process.

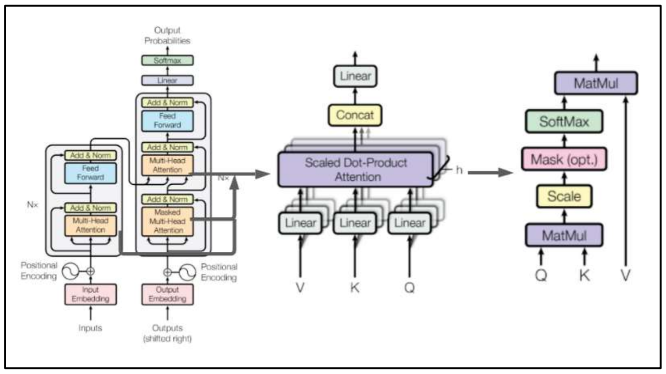


Figure 2.19 Transformer model from Vaswani, et al. [22].

Because multi-head attention can do parallel computing, before sending the data into the multi-head part the input data must first tag the position to preserve the location of each word because parallel computing input can go into the model nonsequentially. Transformer researchers use sine and cosine functions of different frequencies which are defined as follows:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}}) \tag{2.42}$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}}) \tag{2.43}$$

Here, *pos* is the position, *i* is the dimension, and *d* is the size of the word embedding. This technique is needed for the transformer model because there is no notion to tell word order which makes the model have no idea whether the words are in order.

The proposed model also has a decoder side, which has a masked mechanism inside of the scaled dot-product which will mask out the future token. The following equation is an example result from masking:

$$mark(QK^T) = mark\begin{pmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{pmatrix} = \begin{matrix} e_{11} & -\infty & -\infty \\ e_{21} & e_{22} & -\infty \\ e_{31} & e_{32} & e_{33} \end{matrix} \tag{2.44}$$

The decoder must have a masked mechanism because it needs to prevent the model from having bias towards a specific word when the model attempts to predict the future word.

The result of the experiment in the machine translation task of the transformer shows that it is the best method with the highest BLEU (the Bilingual Evaluation Understudy) score of 28.4. Moreover, the complexity per layer of transformer is O(n^2d) and the Recurrent Layer Type is O(nd^2), meaning that the transformer can be performed much faster than recurrent with the same dimension (d), but the length of the input (n) must not be very long.

The transformer model has become the base model for many new attention-based models. The pre-training of deep bidirectional transformers (BERT) [23] involves applying bi-directional to the transformer model. In the BERT model, only the encoder side of the transformer model is used. The BERT model requires the pre-training part to predict the next word in a sequence by randomly masking out 15% of the words in each sequence. Arthur stated that this procedure forces the model to keep a distributional context representation of every input. After the BERT model does pretraining with unlabeled sentences, the next part is fine-tuning which is used to let BERT do specific tasks by adding a layer next to the result of the transformer model. For example, adding a feedforward layer with Sigmoid or Softmax as the activation function to create a classification task.

Because the computation of the transformer can become very slow when the length of the input increases, using this method alone for doing character-level representation may not be sufficient. Gao, et al. introduced Convtransformer as a way to do character-level representation in the transformer by stacking multiple Convolutional Layers on top of multi-head attention [24]. The result shows that the BLEU scores of this method (29.23) are better than the standard transformer (28.8). During training time, this method can be 30% slower than the standard transformer but it can reach comparable performance in less than half of the number of epochs.

There is another method to reduce the complexity of the transformer that was introduced by Guo, et al. Which involves replacing the fully connected structure with a convolutional layer [25]. By doing this method, the author sees that the complexity per layer of star-transformer decreased to O($6nd$). The authors verified the ability of their model in sequence labeling tasks by comparing word- level representation and character representation of standard transformer and star- transformer. The result shows that the word- level accuracy of star- transformer (97. 14%) was higher than the standard transformer which is 96. 31 and at the character level with adding CRF layer getting the highest result with an accuracy score of 97.68.

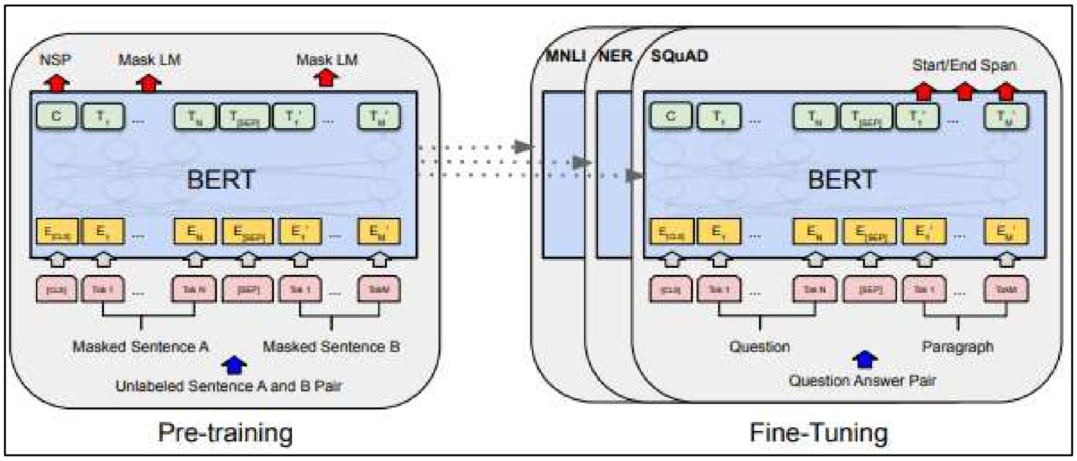


Figure 2.20 Example of BERT model [25].

Self-attention is an attention mechanism learning dependency between each position of a single sequence. Self-attention is the main concept of the transformer and has been used in a variety of tasks in the alignment method by changing the attention score to learn the dependency of each position in a single sequence instead of dependency between input and output. Self-attention was first proposed by Cheng, et al. [26] to do machine reading. The result shows that the self-attention mechanism can help the model learn the correlation between the current words and the previous part of the sentence. The alignment method in this research will follow this concept to create a model for word segmentation.

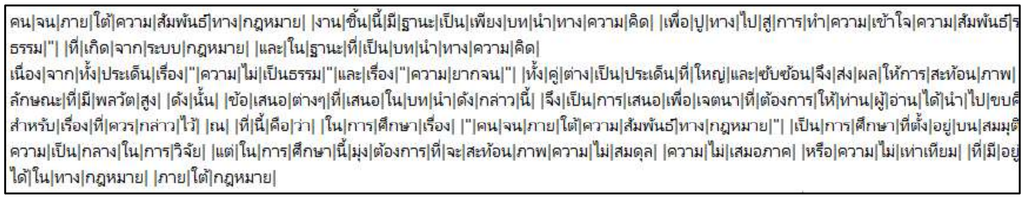
In this research, the transformer model was also selected to create Thai tokenization and the result will be compared to the alignment method.

CHAPTER 3 METHODOLOGIES

This chapter contains details regarding the method we'll be employing in our research. We also provide details on the datasets, evaluation techniques, and model architectures used in this study. This research not only investigates model effectiveness, but also creates visualizations to observe how attention mechanisms can help in tokenization tasks. Two main attention mechanisms will be tested in this research, which are alignment models and transformer models.

3.1 Dataset

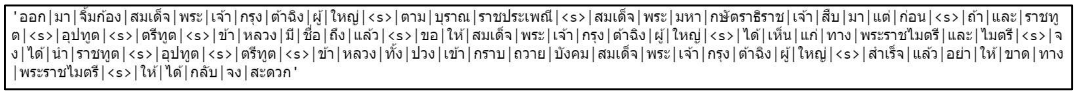
In this research, we use datasets as follows: the BEST2010 dataset is a tagged corpus for word segmentation from the National Electronics and Computer Technology Center (NECTEC) and was used in this study. The corpus contains 5 million words with word boundary markers from news articles, novels, encyclopedias, and articles. The BEST2010 dataset was divided into two parts: training sets to train the model and test sets for evaluation. The dataset was split to prevent the model from overfitting.



คน|จน|ภาย|ใต้|ความ|สัมพันธ์|ทาง|กฎหมาย| |งาน|ชิ้น|นี้|มี|ฐานะ|เป็น|เพียง|บท|นำ|ทาง|ความ|คิด| |เพื่อ|ปู|ทาง|ไป|สู่|การ|ทำ|ความ|เข้าใจ|ความ|สัมพันธ์|ร
ธรรม|"| |ที่|เกิด|จาก|ระบบ|กฎหมาย| |และ|ใน|ฐานะ|ที่|เป็น|บท|นำ|ทาง|ความ|คิด|
เนื่อง|จาก|ทั้ง|ประเด็น|เรื่อง|"|ความ|ไม่|เป็นธรรม|"|และ|เรื่อง|"|ความ|ยากจน|"| |ทั้ง|คู่|ต่าง|เป็น|ประเด็น|ที่|ใหญ่|และ|ซับซ้อน|จึง|ส่ง|ผล|ให้การ|สะท้อน|ภาพ|
ลักษณะ|ที่|มี|พลวัต|สูง| |ดัง|นั้น| |ข้อ|เสนอ|ต่างๆ|ที่|เสนอ|ใน|บท|นำ|ดัง|กล่าว|นี้| |จึง|เป็น|การ|เสนอ|เพื่อ|เจตนา|ที่|ต้องการ|ให้|ท่าน|ผู้|อ่าน|ได้|นำ|ไป|ขบค
สำหรับ|เรื่อง|ที่|ควร|กล่าว|ไว้| |ณ| |ที่|นี้|คือ|ว่า| |ใน|การ|ศึกษา|เรื่อง| |"|คน|จน|ภาย|ใต้|ความ|สัมพันธ์|ทาง|กฎหมาย|"| |เป็น|การ|ศึกษา|ที่ตั้ง|อยู่|บน|สมมุติ
ความ|เป็น|กลาง|ใน|การ|วิจัย| |แต่|ใน|การ|ศึกษา|นี้|มุ่ง|ต้องการ|ที่|จะ|สะท้อน|ภาพ|ความ|ไม่|สมดุล| |ความ|ไม่|เสมอภาค| |หรือ|ความ|ไม่|เท่าเทียม| |ที่|มี|อยู่
ได้|ใน|ทาง|กฎหมาย| |ภาย|ใต้|กฎหมาย|

Figure 3.1 Example of BEST2010 dataset.

This study also employed the Thai National Historical Corpus (TNHC) dataset, which was manually tokenized and collected by Chulalongkorn University's Faculty of Arts, as an additional evaluation criterion. This dataset consists of 47 documents totaling 599K words in classic literature written using poetic techniques. We also used the TEST_100K dataset from NECTEC for speed comparison.



'ออก | มา | จิ้มก้อง | สมเด็จ | พระ | เจ้า | กรุง | ต้าฉิง | ผู้ | ใหญ่ | <s> | ตาม | บุราณ | ราชประเพณี | <s> | สมเด็จ | พระ | มหา | กษัตราธิราช | เจ้า | สืบ | มา | แต่ | ก่อน | <s> | ถ้า | และ | ราชทู
ต | <s> | อุปทูต | <s> | ตรีทูต | <s> | ข้า | หลวง | มี | ชื่อ | ถึง | แล้ว | <s> | ขอ | ให้ | สมเด็จ | พระ | เจ้า | กรุง | ต้าฉิง | ผู้ | ใหญ่ | <s> | ได้ | เห็น | แก่ | ทาง | พระราชไมตรี | และ | ไมตรี | <s> | จ
ง | ได้ | นำ | ราชทูต | <s> | อุปทูต | <s> | ตรีทูต | <s> | ข้า | หลวง | ทั้ง | ปวง | เข้า | กราบ | ถวาย | บังคม | สมเด็จ | พระ | เจ้า | กรุง | ต้าฉิง | ผู้ | ใหญ่ | <s> | สำเร็จ | แล้ว | อย่า | ให้ | ขาด | ทาง
| พระราชไมตรี | <s> | ให้ | ได้ | กลับ | จง | สะดวก'

Figure 3.2 Example of TNHC dataset.

Figure 3.2 shows an example of the TNHC dataset. This data set is mostly tokenized in order to read it harmonically, which is different from BEST2010 in that the data is tokenized by the meaning of the word.

We also use TEST_100K as another test set that will be used to measure speed performance. Speed performance must be measured because tokenization is one of the first processes to create an NLP application. The TEST_100K dataset does not have a “|” symbol to index the location of the word boundary so it cannot be used to evaluate correctness. This dataset contains about one hundred thousand words.

Table 3.1 Summary of datasets.

Name	**Description**
BEST2010	The corpus is collected from news articles, novels, encyclopedias, and articles. Each word is mostly found in daily life.
TEST_100K	BEST2010 test set, but without word boundary markers.
TNHC	The corpus is collected from literature that was written with poetic techniques. Each word in the corpus is often short and harmonically matched.

We preprocessed the input in the same way that Deepcut did by using the same character type as shown in Table 2.2 of the previous chapter and the same input size was used as Deepcut by having information of 10 previous characters, 10 future characters, and the current character which to give an input size of 21 characters. Figure 3.8 shows the flow of how we created the input data with a size of 21 that contain the information of the current, previous, and future inputs.

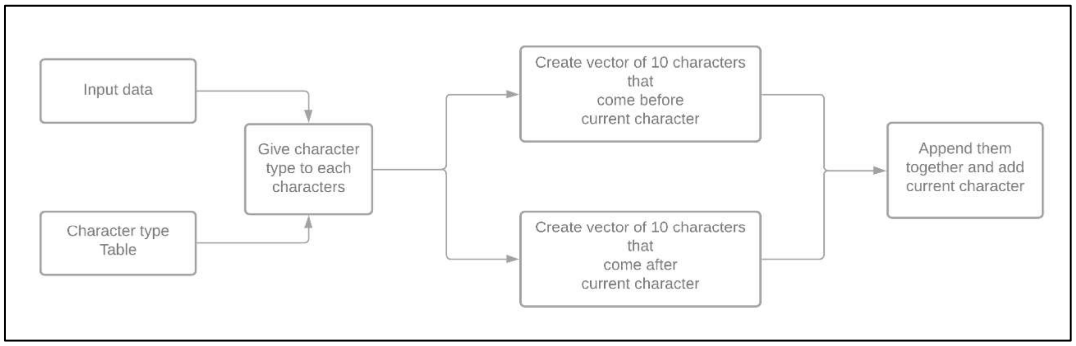

Input data
Character type Table
Give character type to each characters
Create vector of 10 characters that come before current character
Create vector of 10 characters that come after current character
Append them together and add current character

Figure 3.3 The Preprocess of the input data.

For experiment design, we sent all datasets into the preprocessing algorithm and then sent the training set of the BEST2010 dataset to train 5 models, namely Bi-LSTM, Bi-LSTM with additive attention, Bi-LSTM with multiplicative attention, Transformer with character type feature, and Transformer without character type feature. Each of the models will be evaluated with the BEST_2010 test set and then we will select the best result of our attention-based model to compare the result with popular Thai tokenization open-source systems which are Deepcut, Newmm from PyThaiNLP, and Attacut. We also created a visualization for the attention-based model to observe how each attention-based model learns character association with each other. The following figure shows the flow of experimental design.

To compare the result, the F1 score is used to determine how good the model is at predicting word boundaries which will give more information in the evaluation metric section.

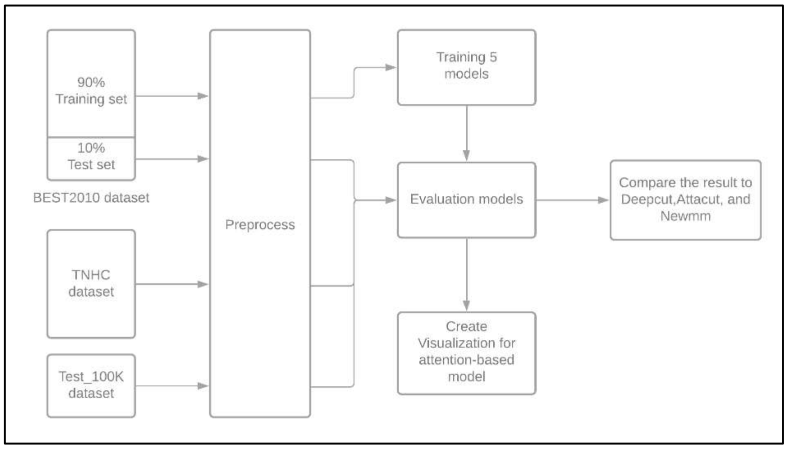


Figure 3.4 The flow of our experimental design.

3.2 Models

The attention mechanism is quite useful for various NLP tasks. Only a few studies have investigated the use of attention mechanisms in word segmentation. The goal of this paper is to test how employing character level in the attention mechanism can assist the model to learn character dependencies and potentially increase segmentation accuracy to predict word boundaries in different contexts.

Five models were tested in this study. For each model, the text's n-grams serve as the model's input. Each model has 2 embedding layers. The first layer is an embedding layer with character level representation and character type features. The character type feature is the same feature that using in the Deepcut method. The second layer is the attention layer. The base model architecture for alignment models is the Bi-LSTM layer. Different methods for calculating the alignment score for the context vector were tested to determine their effectiveness.

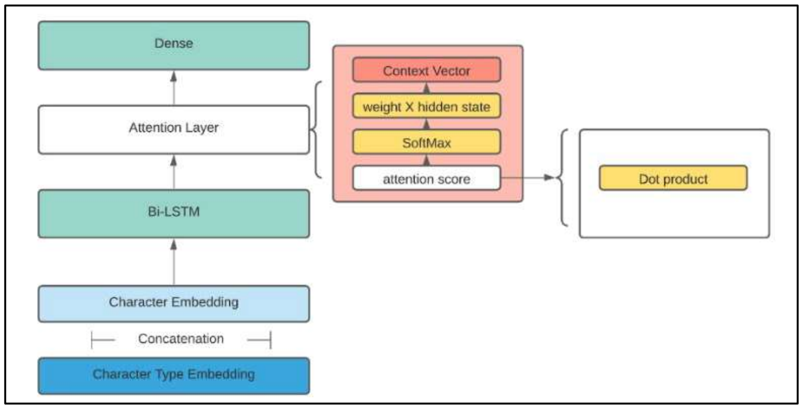


Figure 3.5 The model architecture of multiplicative attention.

For Luong attention or multiplicative attention, we used dot product with input hidden states and each of the hidden states to generate the context vector, as shown in the following equation.

$$score(s_t,h_i) = s^T{}_t W h_i \tag{3.1}$$

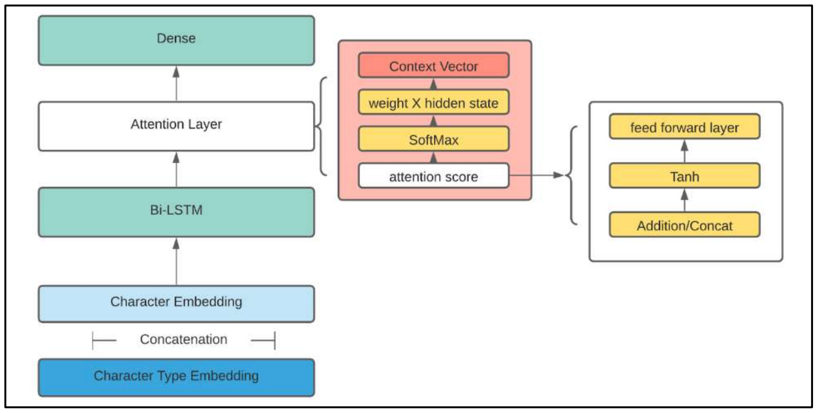


Figure 3.6 The model architecture of additive attention.

Bahdanau attention or additive attention calculated the alignment score by adding input hidden states and each of the hidden states, applying non-linear activation function, and feeding to one unit of the feed-forward layer.

$$score(s_t,h_i) = v_a^T tanh\left(W_1 s_t + W_2 h_i\right) \tag{3.2}$$

The transformer model used the same architecture as the architecture proposed in the paper “Attention is all you need” which used only the encoding side to perform a classification task. In this study, two transformer models were tested, including one that includes character type and one that excludes character type.

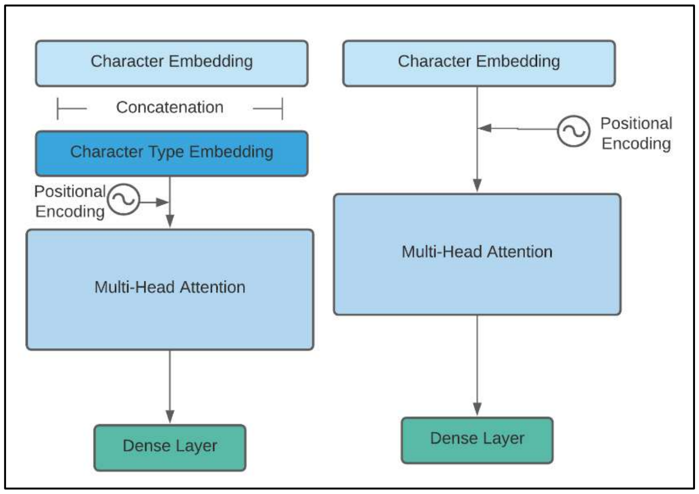


Figure 3.7 The model architecture of transformer.

Figure 3.8 shows overall the attention-bases model design that was used in this experiment. For optimization algorithms, Adam optimization was used as an optimizer for each model architecture.

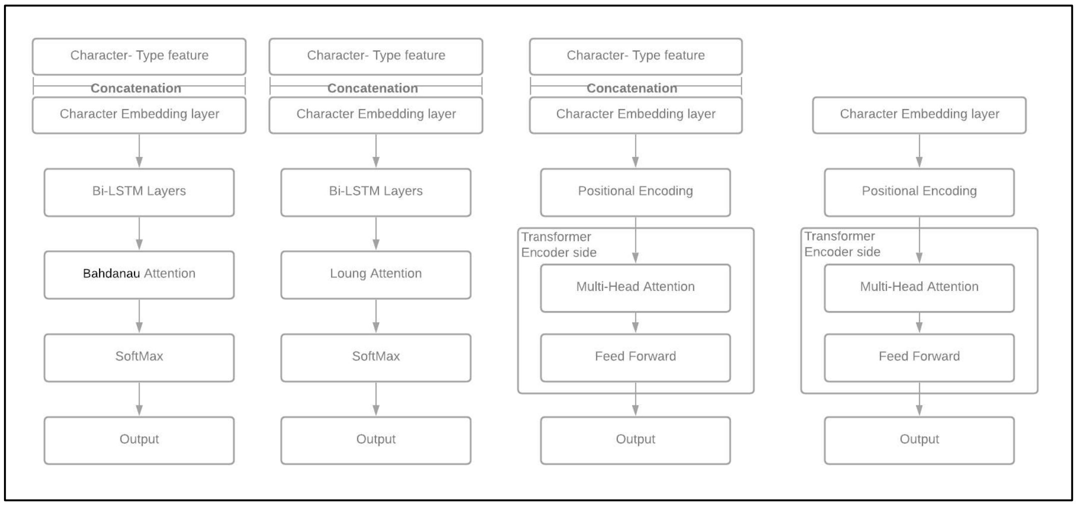


Figure 3.8 Overall of experimental models in this experiment.

For parameter tuning, the based model had the following structure: One Bi-LSTM layer with 32 hidden units; one fast forward layer with 32 hidden units; and the embedding layer with 32 hidden units for character and 16 hidden units for character type. The reason we are using this number of hidden units because we want to make the model small to avoid overfitted problem. If we increased number of the hidden units and hidden layer, the model may have higher chance to increase the accuracy of the training set but also have a higher chance of getting overfitting problem.

Table 3.2 shows the tuning information of other models. The attention-based model of alignment method use the same hyperparameter as the vanilla Bi-LSTM, but the hidden unit at the attention layer will be tuning. For the transformer model, this architecture is different from the Bi-LSTM which requires more parameters for it to be tuned.

Table 3.2 Hyperparameter for each model.

Model	**Hyperparameter for tuning**
Bi-LSTM	Not tuning (base model)
Bi-LSTM+ additive attention	Hidden unit of attention layer
Bi-LSTM+ multiplicative attention	Hidden unit of attention layer
Transformer	Number of heads, the hidden unit of fast forward, and hidden unit of the embedding layer

3.3 Visualization

The attention mechanism can give the attention score as a result that the model has for both the input and output. The concept of self-attention was applied in all models in this study. The resulting attention score shows information about how the model pays attention to each input.

The information about the attention score of each input can be observed to better understand how attention works. This study utilized this to visualize the attention score in the two-dimensional plane.

3.4 Experimental Design

There are two types of hyperparameter tuning in the experiment because there are two model types. For the alignment model, the hyperparameter will be the number of hidden units of the attention layer, and their Bi-LSTM layer will have the same number of layers and the same number of hidden units as the based model. For the transformer model, the hyperparameter will be the number of parallel scaled Dot-Product attention layers. All models will be trained with the BEST2010 dataset.

For evaluation, this research will use the TNHC dataset and the test set that was split from the BEST_2010 dataset that the models had not previously seen. Each model architecture uses the F1 score (further details in section 3.6) to measure the performance by comparing with PyThaiNLP with its maximal matching engine, DeepCut and AttaCut.

Word segmentation is one of the first tasks for performing NLP, so speed is an important factor to consider when designing a tokenizer. This will compare the prediction time of each model on the TEST_100K dataset and compare the result with PyThaiNLP, DeepCut, and AttaCut.

3.5 Evaluation Metrics

Each model was evaluated using precision, recall, and F1 score at the word boundary-level which can be defined as:

$$Precision = \frac{\#\ correctly\ predicted\ word\ boundaries}{\#character\ predicted\ as\ word\ boundaries} \tag{3.3}$$

$$Recall = \frac{\#\ correctly\ predicted\ word\ boundaryies}{\#\ real\ word\ boundaries} \tag{3.4}$$

$$F1 = 2 \times \frac{precision \times recall}{precision + recall} \tag{3.5}$$

Evaluate the model performance, the test set that we split out of BEST2010 was using and the TNHC dataset was also used as the dataset for testing out-of-domain performance and TEST_100K dataset using speed performance with other popular techniques, including PyThaiNLP, Deepcut, and Attacut. Precision can tell the boundaries that the model inserted and how many of them are correct, while recall can tell how many of all the boundaries that the model could predict correctly.

In this research, the result will be in word-level only because at the character-level, the accuracy of the model has been higher than what the model can do. From the following figure, it can be seen that the accuracy of the character-level is 75% correct, but when doing word-level the accuracy drops to 25% correct.

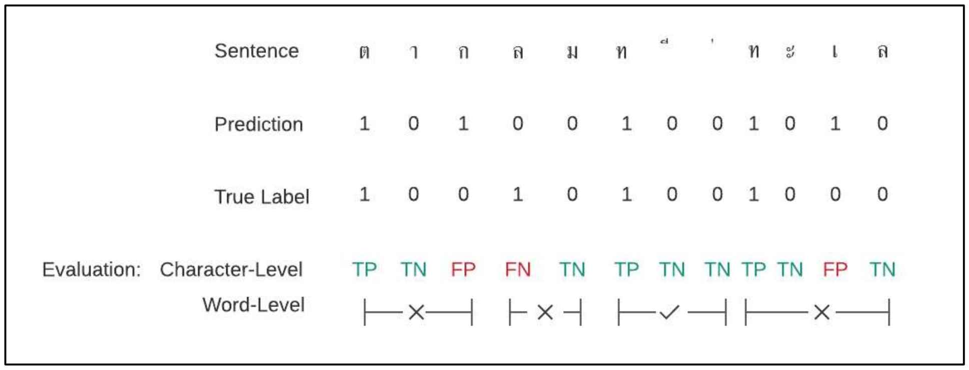


Figure 3.9 Evaluation of character-level and word-level.

CHAPTER 4 RESULTS AND DISCUSSION

The outcome of the experiment is demonstrated and discussed in this chapter. The first section presents and discusses results for each model in terms of performance by using F1 measure and prediction time speed. The results are also compared with other Thai tokenization open-source systems. The following section presents visualization graphs of each attention model and discusses the information from the attention mechanism shown in each graph.

4.1. Word Segmentation Performance

The results of each model performance when evaluated against the BEST2010 test set are shown in Table 4.1. The results show that in the alignment method, additive attention can help the model get a higher score when compared to the multiplicative method, and both of the alignment methods can outperform vanilla Bi-LSTM. In the transformer method, the without character type feature model can achieve a higher score than with character type feature.

Table 4.1 Word segmentation quality on the BEST2010 test set.

Algorithm	**Precision**	**Recall**	**F1score**
Bi-LSTM	93.29	84.88	94.08
Bi-LSTM + additive attention	94.12	95.59	94.85
Bi-LSTM + multiplicative attention	94.06	95.22	94.64
Transformer + char type	93.17	95.35	94.53
Transformer (no char type)	93.93	95.23	94.58

The performance results of each model when evaluated against the TNHC dataset are shown in Table 4.2. The results show the same conclusion as the results from the BEST2010 test set.

Table 4.2 Word segmentation quality on the TNHC dataset.

Algorithm	**Precision**	**Recall**	**F1score**
Bi-LSTM	74.10	55.05	63.17
Bi-LSTM + additive attention	76.11	57.56	65.53
Bi-LSTM + multiplicative attention	76.25	57.33	65.45
Transformer + char type	73.49	55.33	63.13
Transformer (no char type)	75.16	56.62	64.58

As shown in Table 4.3, the results of other popular open source systems of Thai tokenizers are compared with the results from this experiment. From the results, it is evident that the Deepcut approach achieved the highest score with the BEST2010 dataset and Newmm from PythaiNLP got the highest score in TNHC.

Table 4.3 Evaluation of other Thai tokenizers.

Dataset	Algorithm	Precision	Recall	F1score
BEST2010	Newmm	76.49	68.99	72.55
	Deepcut	97.86	98.22	98.04
	Attacut	96.12	96.65	96.38
TNHC	Newmm	67.56	74.28	70.76
	Deepcut	67.48	54.59	60.35
	Attacut	63.04	50.64	56.16

Bi-LSTM with additive attention got highest F1score in attention-based model. Subsequently, we used this model to compare against other techniques. Table 4.4 shows that Bi-LSTM with additive attention outperformed Deepcut and Attacut in TNHC, but not in the BEST-2010 datasets. This is potentially because the attention-based model is more economical than others in terms of complexity, which prevents it from being overfitted in the BEST2010 dataset.

Table 4.4 Word segmentation quality on different methods.

Dataset	Algorithm			
	Bi-LSTM + additive attention	Deepcut	Attacut	Newmm
BEST-2010	94.85	98.04	96.38	72.55
TNHC	65.53	60.35	56.16	70.76

The next part is the speed performance because word segmentation is one of the first steps to do any text mining task in the Thai language. The predicted time of the experiment model was observed and compared to other methods.

Table 4.5 shows that Attacut has the shortest prediction time among the Thai word segmentation algorithms. The transformer with char type can run 2.1 times faster than Deepcut, and Bi-LSTM with multiplicative attention can run 1.4 times faster than Deepcut and 1.3 times faster than Bi-LSTM with additive attention. Because of the way the attention score is calculated, Bi-LSTM with multiplicative attention outperformed additive attention in terms of prediction time. For multiplicative attention, the attention score is derived from dot production which consumes fewer resources than additive attention which requires multiple operations.

Table 4.5 Speed comparison between each model on TEST_100K.

Algorithm	**Prediction time (sec.)**
Attacut	12.44
Newmm	35.37
Transformer + char type	49.589
Bi-LSTM + multiplicative attention	74.498
Transformer (No char type)	90.84
Bi-LSTM	94.567
Bi-LSTM + additive attention	102.72
Deepcut	105.41

We also attempted to provide more examples for the result of Thai tokenization from our attention-based model, Deepcut, Attacut, and Newmm. We used longer sentences for this.

Table 4.6 Result from tokenizing the first sentence.

Algorithm	**Sentence : "ฉันนั่งอยู่ที่ริมทะเล"**
Attacut	ฉัน | นั่ง | อยู่ | ที่ | ริม | ทะเล
Newmm	ฉัน | นั่ง | อยู่ | ที่ | ริมทะเล
Deepcut	ฉัน | นั่ง | อยู่ | ที่ | ริม | ทะเล
Bi-LSTM + additive attention	ฉัน | นั่ง | อยู่ | ที่ | ริม | ทะเล
Bi-LSTM + multiplicative attention	ฉัน | นั่ง | อยู่ | ที่ | ริม | ทะเล
Transformer (No char type)	ฉัน | นั่ง | อยู่ | ที่ | ริม | ทะเล
Transformer + char type	ฉัน | นั่ง | อยู่ | ที่ | ริม | ทะเล

From the result, it can be seen that the Newmm method returns a different result when compared to other methods which could be from the maximal matching algorithm that attempts to find the longest word.

Table 4.7 Result from tokenizing the second sentence.

Algorithm	**Sentence : "หมอยงแจ้งงบประมาณสำหรับวัคซีน"**
Attacut	หมอยงแจ้ง | งบ | ประมาณ | สำหรับ | วัคซีน
Newmm	หมอ | ยง | แจ้ง | งบประมาณ | สำหรับ | วัคซีน
Deepcut	หมอยงแจ้ง | งบ | ประมาณ | สำหรับ | วัคซีน
Bi-LSTM + additive attention	หมอยง | แจ้ง | งบ | ประมาณ | สำหรับ | วัคซีน
Bi-LSTM + multiplicative attention	หมอยง | แจ้ง | งบ | ประมาณ | สำหรับ | วัคซีน
Transformer (No char type)	หมอยง | แจ้ง | งบ | ประมาณ | สำหรับ | วัคซีน
Transformer + char type	หมอยง | แจ้ง | งบ | ประมาณ | สำหรับ | วัคซีน

From this result, Deepcut and Attacut returned incorrect words ("หมอยงแจ้ง") which may be caused by the models understanding that the word "หมอยงแจ้ง" is a compound word. Our model result returned a more appropriate answer, the word "หมอยง" which is the words career and name concatenated together. For this case, PythaiNLP separating this word to "หมอ" and "ยง", which are the smallest possible words for each character because the dictionary does not have a word "หมอยง" which could be why the maximal matching algorithm does not return the word "หมอยง".

Table 4.8 Result from tokenizing the third sentence.

Algorithm	**Sentence : "ศธ.และสพฐ.ร่วมมอบอุปกรณ์สู้ภัยโควิด19"**
Attacut	ศธ. | และ | สพฐ. | ร่วม | มอบ | อุปกรณ์ | สู้ | ภัยโควิด | 19
Newmm	ศธ |. | และ | สพฐ. | ร่วม | มอบ | อุปกรณ์ | สู้ | ภัย | โควิด | 19
Deepcut	ศธ. | และ | สพฐ. | ร่วม | มอบ | อุปกรณ์ | สู้ | ภัย | โค | วิด |19
Bi-LSTM + additive attention	ศธ. | และ | สพฐ. | ร่วมมอบ | อุปกรณ์ | สู้ | ภัย | โค | วิด |19
Bi-LSTM + multiplicative attention	ศธ. | และ | สพฐ. | ร่วมมอบ | อุปกรณ์| สู้ | ภัย | โควิด |19
Transformer (No char type)	ศธ. | และ | สพฐ. | ร่วม | มอบอุปกรณ์| สู้ | ภัย |โควิด |19
Transformer + char type	ศธ. | และ | สพฐ. | ร่วม | มอบ | อุปกรณ์ | สู้ | ภัย | โควิด | 19

As result shown in Table 4.8, The Newmm method return an incorrect word at the start because it is an abbreviation which may not be contained in the dictionary, so the maximal matching algorithm cannot combine those two words. Our additive attention and Deepcut returned an incorrect word at "โควิด" which got separated to "โค" and "วิด". The word "19" may be why those two algorithms decided to return those results because without the inclusion of "19", these two algorithms will return "โควิด" instead of "โค" and "วิด". Further reasons behind this are included in the discussion section by creating a visualization of the attention score of the word "โควิด19".

Table 4.9 Result from tokenizing the fourth sentence.

Algorithm	**Sentence : "ปลูกฝังวัฒนธรรมรักการอ่าน"**
Attacut	ปลูกฝัง | วัฒนธรรม | รัก | การ | อ่าน
Newmm	ปลูกฝัง | วัฒนธรรม | รัก | การ | อ่าน
Deepcut	ปลูกฝัง | วัฒนธรรม | รักการ | อ่าน
Bi-LSTM + additive attention	ปลูก | ฝัง | วัฒนธรรม | รัก | การ | อ่าน
Bi-LSTM + multiplicative attention	ปลูกฝัง | วัฒนธรรม | รัก | การอ่าน
Transformer (No char type)	ปลูกฝัง | วัฒนธรรม | รัก | การ | อ่าน
Transformer + char type	ปลูกฝัง | วัฒนธรรม | รักการ | อ่าน

From the results in Table 4.9, even though the Deepcut approach that has the highest score in the BEST2010 dataset, it can still return incorrect results by it returning the word "รักการ" instead of "รัก" and "การ". Most of the examples use basic words and the results indicate that the best Thai tokenization method is a dictionary-based method such as Newmm from PythaiNLP. The next example is a sentence with more complex words.

Table 4.10 Result from tokenizing the fifth sentence.

Algorithm	**Sentence : "เฟสบุ๊คผลักดันระบบเมตาเวิร์ส"**
Attacut	เฟสบุ๊ค | ผลักดัน | ระบบ | เมตา | เวิร์ส
Newmm	เฟส | บุ๊ค | ผลักดัน | ระบบ | เม | ตา | เวิร์ส
Deepcut	เฟสบุ๊ค | ผลักดัน | ระบบ | เมตาเวิร์ส
Bi-LSTM + additive attention	เฟสบุ๊ค | ผลักดัน | ระบบ | เมตาเวิร์ส
Bi-LSTM + multiplicative attention	เฟสบุ๊ค | ผลักดัน | ระบบ | เมตาเวิร์ส

Table 4.10 Result from tokenizing the fifth sentence(cont'd).

Algorithm	Sentence : "เฟสบุ๊คผลักดันระบบเมตาเวิร์ส"
Transformer (No char type)	เฟสบุ๊ค | ผลักดัน | ระบบ | เมตา | เวิร์ส
Transformer + char type	เฟสบุ๊ค | ผลักดัน | ระบบ | เมตาเวิร์ส

The results in Table 4.10 show that when words become more complex, Newmm can give the worst result of all the algorithm due to Newmm being dictionary-based. Without an updated dictionary, the performance of this algorithm can drop.

4.2. Attention Visualization

By incorporating an attention mechanism into the deep learning architecture, performance be improved and it can also provide information about the model's attention to inputs and outputs. The attention mechanism in this study is self-attention, which means that the attention score is the score that the attention-based model uses to learn the correlation between the inputs.

There are four attention score visualizations in this section. Each graph presents information about the attention score of the input text (y-axis) for each of the characters in the text (x-axis).

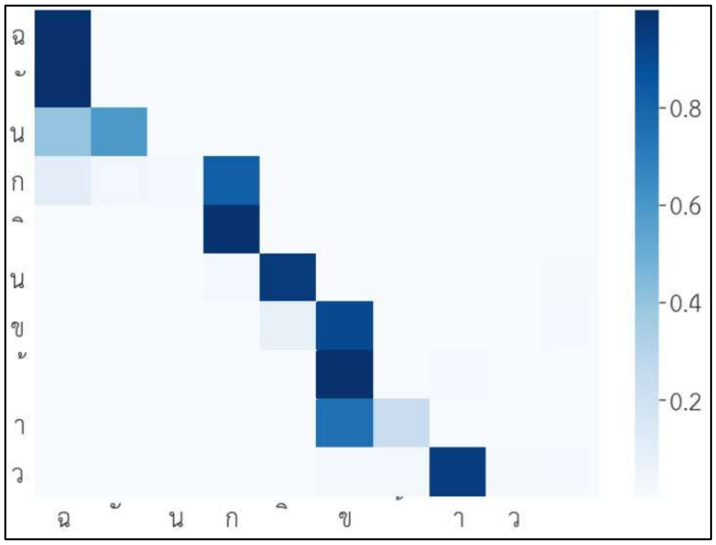


Figure 4.1 Result of additive attention.

1. Bi-LSTM with additive attention. As shown in Figure 4.1, the model can cluster a set of characters with the help of additive attention, for example the character "น" was not the beginning of a new word in this context, thus the attention score bound this character to the previous character. Characters that were the start of a new word had very high

attention scores for themselves. This implies that additive attention may help Bi-LSTM perform better.

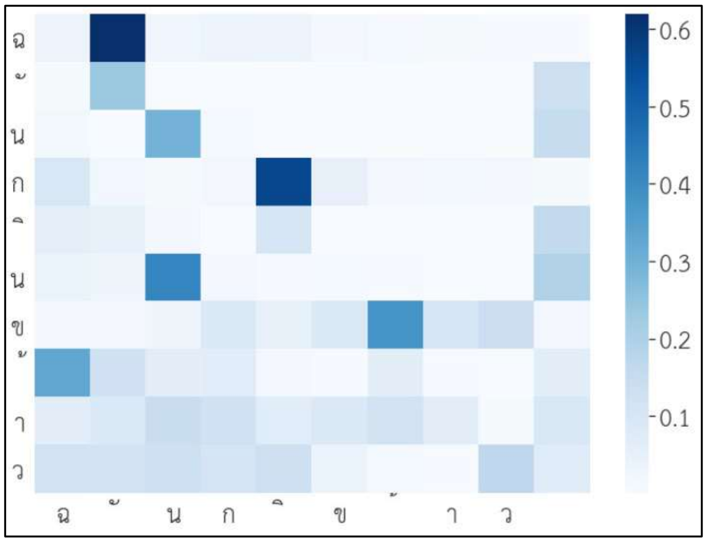


Figure 4.2 Result of multiplicative attention.

2. Bi-LSTM with multiplicative attention. Figure 4.2 shows that multiplicative attention has a different focus point that is more ambiguous than the result shown in Figure 4.1 This may be because of the dot product that was used in the attention score calculation.

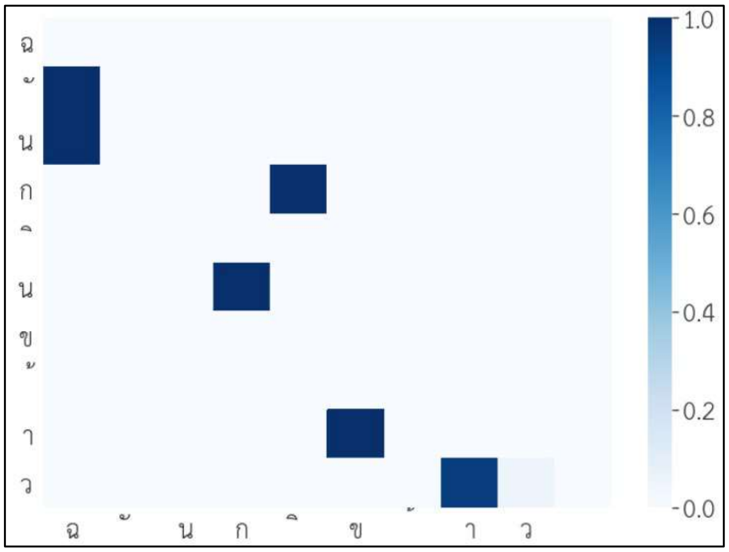


Figure 4.3 Result of transformer with the character type.

3. Transformer with character type. The visualization for the Transformer-based was different from the alignment model because the attention scores were obtained from each attention head. Each head focused on different parts of the input. Figure 4.3 shows the

visualization of 1 from 16 heads, with the result showing that this head put more emphasis on some of the characters, similar to additive attention.

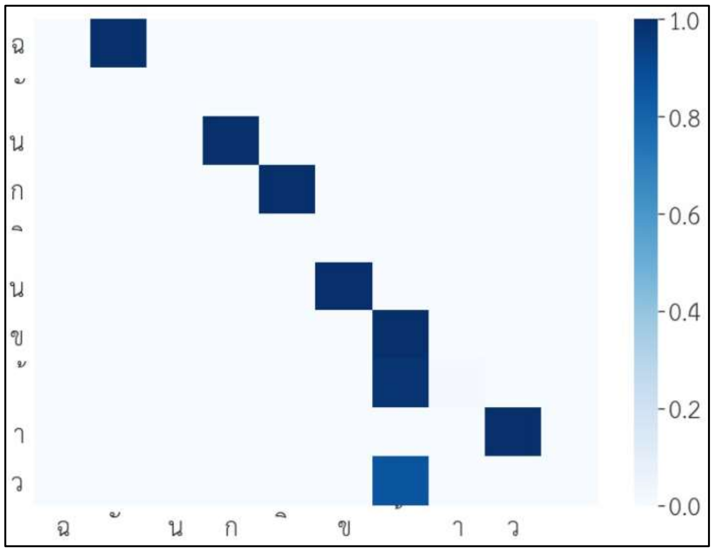


Figure 4.4 Result of transformer without character type.

4. Transformer without character type. The number of heads employed in this model was the same as in multi-head attention with the character type. Figure 4.4 shows that there were more characters that the model paid attention to than the transformer with character type. This could be because the model did not know which character could potentially be the start of the word boundary.

4.3 General Discussion

According to the results of the experiment, the model with the attention mechanism can outperform the based model. In these experiments, the attention-based model is more economical than others in terms of complexity, which prevents it from being overfitted in the BEST2010 dataset. In terms of speed comparison, the attention-based model can run faster than the Bi-LSTM model. In the experiment, with the help of multiplicative method, the model can run 20 .07 seconds faster than the Bi-LSTM model. With the transformer method, the model can run 45 seconds faster than the Bi-LSTM model. Each attention-based models' visualization can provide information about how attention learns dependencies to create tokenizers.

We also trained the attention-based model using the TNHC dataset. We split this dataset in the same way that the BEST2010 dataset was split. For the evaluation set, we used the BEST 2010 test set that was already split in the main experiment to be the evaluation set for this test.

Table 4.11 Word segmentation quality with the TNHC dataset test set.

Algorithm	Precision	Recall	F1score
Bi-LSTM + additive attention	85.25	86.27	84.12
Bi-LSTM + multiplicative attention	81.41	86.51	83.88
Transformer + char type	80.54	86.91	83.60
Transformer (no char type)	80.57	87.12	83.72

Table 4.12 Word segmentation quality on the BEST2010 test set.

Algorithm	Precision	Recall	F1score
Bi-LSTM + additive attention	66.81	66.61	66.71
Bi-LSTM + multiplicative attention	64.20	64.20	64.20
Transformer + char type	66.06	69.94	67.95
Transformer (no char type)	68.39	70.52	69.44

Table 4.11 and Table 4.12 show the results when training each model with the TNHC dataset instead of BEST2010. Each model uses the same parameters and hyperparameters as the model that trained with the BEST2010 dataset. The results show that if we change the training dataset to TNHC, the model can learn to predict the result of the TNHC dataset well but it also results in the model making poor predictions from the BEST2010 dataset. The TNHC dataset is quite small compared to the BEST2010 dataset, which could be why the f1 score of these models was not very high compared to the model that was trained with the BEST2010 dataset.

Visualization graphs were created to see why additive attention incorrectly cut the sentence (“ศธ.และสพฐ.ร่วมมอบอุปกรณ์สู้ภัยโควิด19”). Because this sentence was too long for visualization, we cut out the first part of the sentence and left out the specific part that caused the additive method and Deepcut to get the incorrect result. “อุปกรณ์สู้ภัยโควิด19” is the part that was used for visualization. We tried adding more words back to observe which word caused the additive attention to return this result.

Table 4.13 Result of tokenization from additive attention.

Sentences	The result from Bi-LSTM with additive attention
โควิด19	โควิด, 19
ภัยโควิด19	ภัย, โควิด, 19
สู้ภัยโควิด19	สู้, ภัย, โควิด, 19
อุปกรณ์สู้ภัยโควิด19	อุปกรณ์, สู้, ภัย, โค, วิด, 19

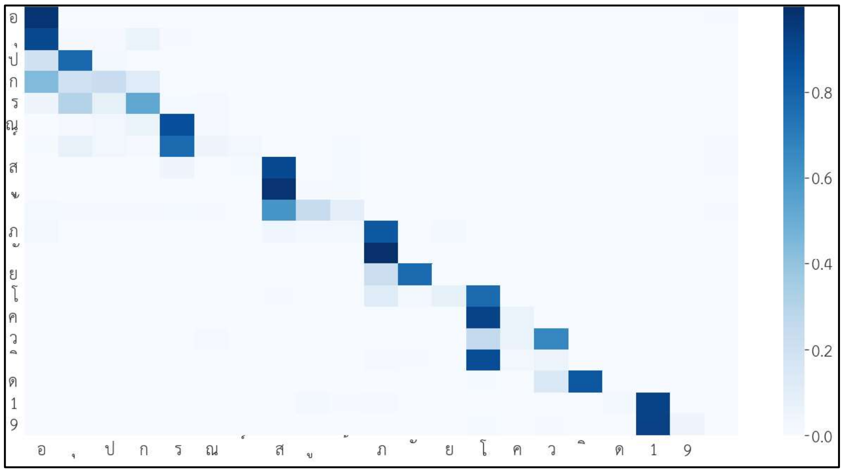


Figure 4.5 Result of additive attention in the problem sentence.

Figure 4.5 shows the result of tokenization from additive attention. The result shows that this type of attention tries to give attention to the character nearby and tends to result in the smallest word rather than compound words. But it is difficult to determine what causes the model to get the incorrect result from this graph.

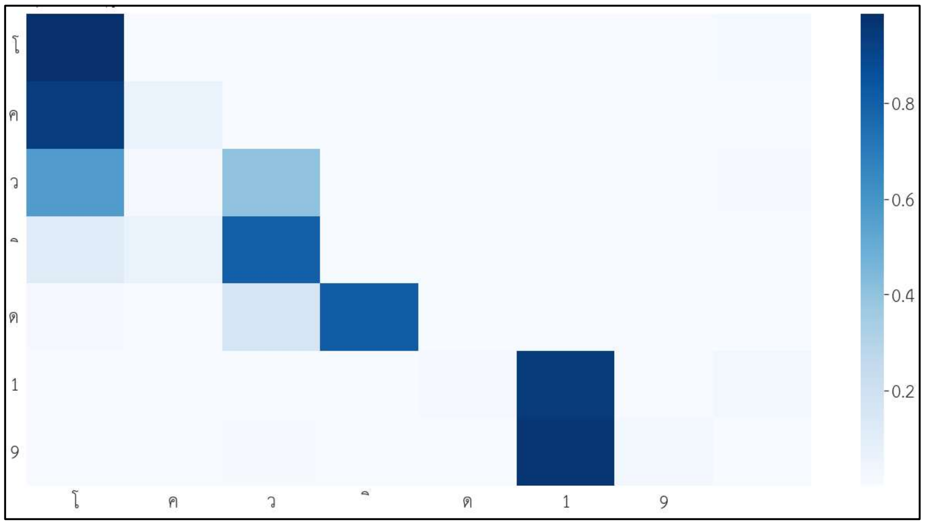


Figure 4.6 The correct result of additive attention in the problem sentence.

Figure 4.6 shows the correct result of the tokenization from additive attention. This visualization shows the tokenization of "โควิด19" which additive attention can correctly tokenize. The difference between these two figures is at the tonal mark "อิ" wherein the correct one this tonal mark has more focus on the character "ว" than the wrong one and the character "ว" has less focus on itself in the correct one than the wrong one.

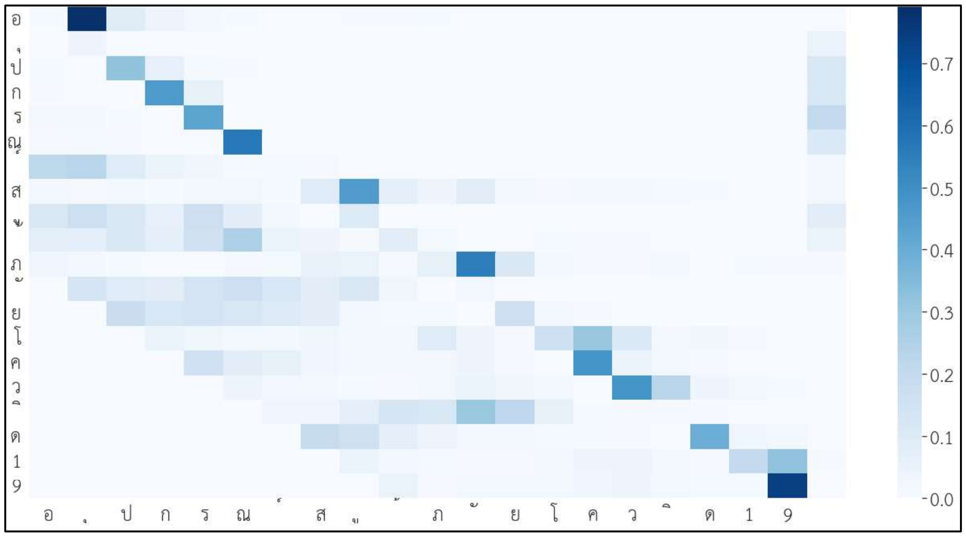


Figure 4.7 Result of multiplicative attention in the problem sentence.

Figure 4.7 is the result of multiplicative attention. The result shows that multiplicative attention tries to learn character association across the sentence which is different from additive attention that tries to learn association from nearby characters. This may be why multiplicative attention can group longer words than additive attention.

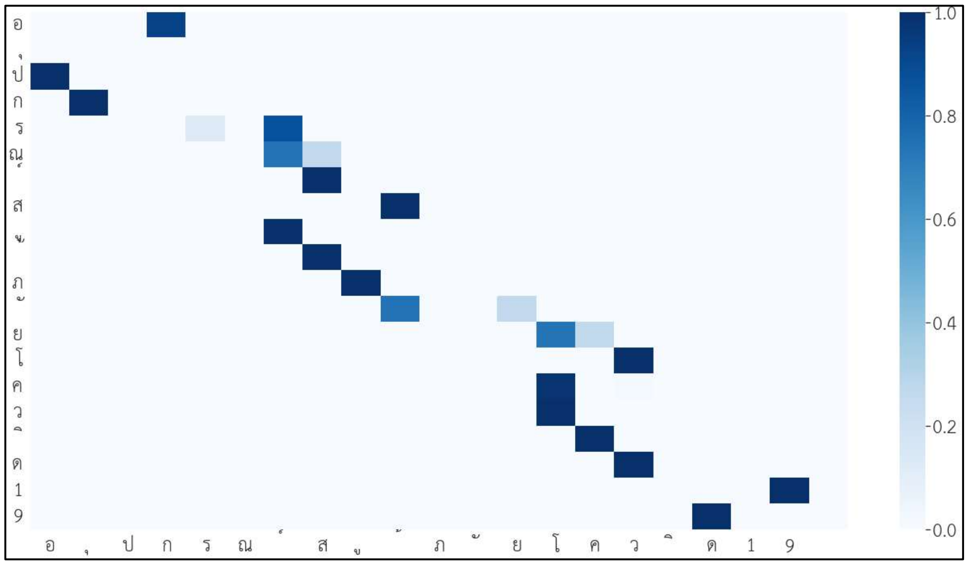


Figure 4.8 Result of transformer without character type in the problem sentence.

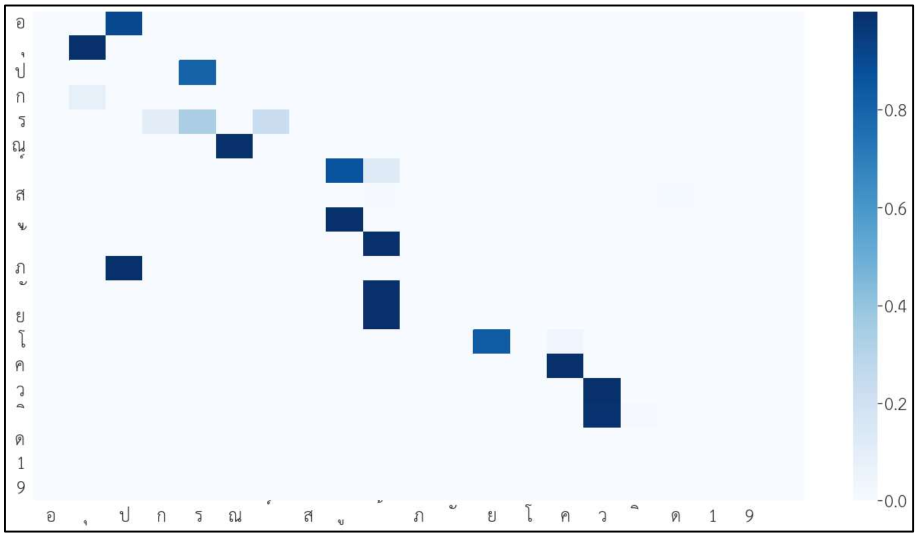


Figure 4.9 Result of transformer with character type in the problem sentence.

Figures 4.8 and 4.9 example visualizations in the transformer-based model. This type of attention is difficult to understand due to the attention score of the transformer having more than 1 set which comes from the different heads in the training process. Both results are samples from one of the multi-head attention. Both results show that this type of attention tries to learn associated with the previous group of characters and the current group.

The reason that our transformer model could learn associations both backward and forward is because of the input itself that has both information.The transformer-based method still needs further investigation by changing to the other transformer-based methods such as Star-Transformer which is more appropriate to use with character-level inputs than the vanilla transformer that was created for word-level inputs.

After observing each of the example sentences and visualizations, the additive attention sometimes tends to tokenize the compound word out to subwords. Meanwhile, multiplicative has a greater chance to return compound words due to this type of attention tending to study the association of a previous group of words rather than additive attention which attempts to learn associations within their group. It is still difficult to determine the best model for Thai tokenization because each method has strengths. For the deep learning method, this approach can tokenize complex words better than the dictionary-based method, but it can sometimes result in incorrect words. Yet the dictionary-based method can return very good results if the dictionary already contains that possible word. To make a very good Thai tokenization application, it may be necessary to find a way to use both of these approaches together to make a model that can result in good results for both complex and simple words. The easiest way to achieve this would be by adopting the deep learning approach first to extract the first set of each word then use a dictionary-based method to compare the deep learning approach results with the dictionary.

CHAPTER 5 CONCLUSION

5.1. Overall conclusion

Word segmentation is an important pre-processing step in natural language processing applications, particularly in languages with no demarcation indicators including such as Thai. Dictionary-based segmentation, while simple, does not consider the context of the sentence. This research proposes an attention-based deep learning approach for Thai word segmentation. With an additional attention mechanism, the model can learn character correlations across the entire sentence without gradient vanishing or gradient explode problems and then tokenize them into word vectors.

This study tested various attention mechanisms that can be beneficial to Thai word segmentation by using the alignment model and the transformer model. These two types of attention mechanisms were tested in this study because they are basic methods and can be used to help the model learn the long-range dependencies task. The goal of this study was to identify how attention mechanisms can aid Thai tokenization tasks. The input of the model was concatenated with the character type feature, the same as the Deepcut and Attacut methods which exclude for one transformer model because this model was used to determine the different way that the attention mechanism works in the transformer model. Each model was trained with the BEST-2010 dataset and tested with the TNHC dataset.

The proposed model demonstrates how an attention mechanism can help the model perform better. With the help of additive attention, the model's f1score is increased 0.78% higher than the Bi-LSTM model and Multiplicative attention can predict the result of 100k characters 20 seconds faster than the Bi-LSTM model and multi-head attention is 45 seconds faster. But when attempting to let each model predict example sentences, we found that our attention-based model did not help much in Thai tokenization because some of the sentences could still get the incorrect token similar to the Deepcut and Attacut methods.

After observing the results from the tokenization examples, we can see that our attention model did not help much with the Thai tokenization task. A result of the additive attention shows that when the sentences are longer, there is a greater chance that this method attempts to learn local dependencies than long range dependence. For multiplicative attention, the resulting trend is that it can predict more correctly than the additive method and can predict faster than the additive method because the multiplicative method only uses dot product to calculate the attention score. But the results of the BEST2010 dataset show that multiplicative attention has a lower score than additive attention, so there are some cases in the BEST2010 dataset in which additive attention can result in better tokenization. For the transformer type, the result can vary in which some sentences can return a good result while others do not. We must investigate other transformer types that are more appropriate for use at the character level. Both the additive and multiplicative methods must make the model more complex to see if performance can be increased.

In terms of visualization, the results show that the model with additive attention has a greater chance to cluster a set of characters to their sub word than a compound word when the sentence is longer. Multiplicative attention, on the other hand, attempts to learn the association of each character across the sentences which could be why the model can result in compound words even the sentence becomes longer. The transformer-based model must find another way to create a visualization because it has a different attention score in each head and, must use other transformer-based models that have been proposed for use at the character level such as Convtransformer and Star-Transformer.

5.2. Limitation and Future work

The limitation of this research is that we attempted to observe the result from each attention-based method to the based model (the vanilla Bi-LSTM) which is a very small architecture. The result shows that the attention-based model can improve accuracy in the Thai tokenization task when compared with the based model, but when comparing back to other Thai tokenization open-source methods that use the deep learning method, the accuracy is lower. The complexity of the model may have to be increased by increasing the number of layers or the number of hidden units and observing if the performance changes.

Further work is required and further exploration of integrating the attention mechanism with Thai tokenization is necessary. The alignment method needs to increase the number of layers in both Bi-LSTM layer and the attention layer. Other approaches should be adopted tested for the transformer method, such as the star-transformer method which is designed for the character level. We should also attempt to develop a new visualization method to visualize the transformer method's attention score, since the attention score of this type of attention is dependent on the number of heads in multi-head attention.

Moreover, further investigation into the potential applications of attention in other Thai NLP tasks should also be undertaken, for instance by applying the attention mechanism to classification and observing the visualization of the attention score. The two tasks, named entity recognition and part of speech tracking, utilize the same many-to-many method as the tokenization task, but the input is at the word level instead of the character level. These two NLP tasks can be used to identify the entity and the part of speech of each word in the sentence. By creating an attention-based model for those tasks, the visualization results of attention score may provide more information about how the attention-based model works at the word level.

REFERENCES

1. Phatthiyaphaibun, W., Korakot, C., Polpanumas, C., Suriyawongkul, A. and Lowphansirikul, L., 2017, **PyThainlp**, [Online], Available: https://github.com/PyThaiNLP/pythainlp [2020, November 15].

2. Sertis Co., Ltd., 2017, **Thai Word Segmentation with Bi-directional RNN,** [Online], Available: https://www.sertiscorp.com/november-20-2017 [2020, November 15].

3. Kittinaradorn, R., Chaovavanich, K., Achakulvisut, T., Srithaworn, K, Chormai, P., Kaewkasi, C., Ruangrong, T. and Oparad, K., 2019, **Deepcut,** [Online], Available: https://github.com/rkcosmos/deepcut [2020, November 15].

4. Chormai, P., Prasertsom, P. and Rutherford, A. T., 2019, **AttaCut: a Fast and Accurate Neural Thai Word Segmenter,** [Online], Available: https://arxiv.org/abs/1911.07056, [2020, November 10].

5. Treeratpituk, P., Suriyawongkul, A. and Phatthiyaphaibun, W., 2018, **Cutkum**, [Online], Available: https://github.com/pucktada/cutkum [2020, November 15].

6. Sornlertlamvanich, V., 1993, " Word Segmentation for Thai in Machine Translation System, Machine Translation", **National Electronics and Computer Technology Center, Bangkok**, January 1993, pp. 50–56.

7. Theeramunkong, T., Sornlertlamvanich, V., Tanhermhong, T. and Chinnan, W., 2013, "Character Cluster Based Thai Information Retrieval", **Proceedings of the 5th International Workshop on Information Retrieval with Asian Languages, IRAL 2000, No. March 2013** pp. 75– 80, 2000, DOI: 10.1145/355214.355225.

8. Mikolov, T., Chen, K., Corrado, G. and Dean, J., 2013, "Efficient estimation of word representations in vector space," **1st International Conference on Learning Representations, ICLR 2013 - Workshop Track Proceedings**, May 2013, Arizona, USA, pp.1–12.

9. Quinlan, J. R., "Induction of decision trees," **Machine Learning**, vol. 1, no. 1, pp. 81–106, 1986, doi: 10.1007/bf00116251

10. Theeramunkong, T. and Usanavasin, S., 2001, **Non-dictionary-based Thai Word Segmentation Using Decision Trees**, pp. 1– 5, DOI: 10.3115/1072133.1072209.

11. Bheganan, P., Nayak, R. and Xu, Y., 2009, “Thai Word Segmentation with Hidden Markov Model and Decision Tree”, **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**, Vol. 5476 LNAI, pp. 74–85, DOI: 10.1007/978-3-642-01307-2_10.

12. Haruechaiyasak, C. and Kongyoung, S., 2009, **TLex: Thai Lexeme Analyser Based on the Conditional Random Fields**, [Online], Available: http://thailang.nectec.or.th/interbest/downloads/InterBEST_3.pdf [2020, November 15].

13. Chaonithi, K. and Prom-On, S., 2016, “A Hybrid Approach for Thai Word Segmentation with Crowdsourcing Feedback System”, **2016 13th International Conference on Electrical Engineering/ Electronics, Computer, Telecommunications and Information Technology, ECTI-CON 2016,** DOI: 10.1109/ECTICon.2016.7561298.

14. Ronran, C., Unankard, S., Nadee, W., Khomwichai, N. and Sirirangsi, R., 2016, “Thai Word Segmentation on Social Networks with Time Sensitivity”, **Proceedings of Knowledge Management International Conference (KMICe) 2016**, 29-30 August 2016, Chiang Mai, Thailand, pp. 362–367.

15. Kingma, D. P. and Ba, J. L., 2015, “Adam: A method for stochastic optimization,” **3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings**, May 2015, San Diego, USA, pp. 1–15.

16. Mikolov, T., Yih, W. T. and Zweig, G., 2013, “Linguistic regularities in continuous spaceword representations,” **NAACL HLT 2013 - 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Main Conference**, June 2013, Atlanta, USA, pp. 746–751.

17. Schuster, M. and Paliwal, K. K., 1997, “Bidirectional Recurrent Neural Network”, **IEEE Transactions on Signal Processing**, Vol. 45, No. 11, pp. 2673–2681, DOI: 10.1109/78.650093.

18. Yu, F. and Koltun, V. 2016, “Multi-scale context aggregation by dilated convolutions,” **4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings**, May 2016, Puerto Rico.

19. Bahdanau, D., Cho, K. H. and Bengio, Y., 2015, “Neural Machine Translation by Jointly Learning to Align and Translate,” **3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings**, 7-9 May 2015, San Diego, USA, pp.1–15.

20. Sutskever, I., Vinyals, O. and Le, Q., 2014, "**Sequence to Sequence Learning with Neural Networks**," **Advances in Neural Information Processing Systems**, Vol. 4, No. January, pp. 3104–3112.

21. Luong, M. T., Pham, H. and Manning, C. D., 2015, "Effective Approaches to Attention- based Neural Machine Translation," **Conference Proceedings - EMNLP 2015: Conference on Empirical Methods in Natural Language Processing**, 1412–1421, DOI: 10.18653/v1/d15-1166.

22. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., Kaiser, L. and Polosukhin, I., 2017, "Attention is All You Need," **arXiv: 1706.03762.**, [2020, November 10].

23. Devlin, J., Chang, M. W., Lee, K. and Toutanova, K., 2019, "BERT: Pre-training of deep bidirectional transformers for language understanding," **NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference**, vol. 1, no. Mlm, June 2019, Minneapolis, Minnesota, pp. 4171–4186.

24. Gao, Y., Nikolov, N. I., Hu, Y. and Hahnloser, R.H.R., 2020, **Character-level Translation with Self- attention,** 1591– 1604, DOI: 10. 18653/ v1/ 2020. acl-main.145.

25. Guo, Q., Qiu, X., Liu, P., Shao, Y., Xue, X. and Zhang, Z., 2019, "Star-transformer," **NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference**, 1, 1315– 1325, DOI: 10.18653/v1/n19-1133.

26. Cheng, J., Dong, L. and Lapata, M., 2016, "Long short-term memory-networks for machine reading," **EMNLP 2016 - Conference on Empirical Methods in Natural Language Processing, Proceedings**, November 2016, Austin, Texas, pp. 551–561, doi: 10.18653/v1/d16-1053.

CURRICULUM VITAE

NAME	Mr. Jednipat Atiwetsakun
Date of Birth	11 March 1997

EDUCATIONAL RECORD

HIGH SCHOOL	High School Graduation Wat Suthi Wararam School, 2014
BACHELOR'S DEGREE	Bachelor of Engineering (Computer Engineering) King Mongkut's University of Technology Thonburi, 2018
MASTER'S DEGREE	Master of Engineering (Computer Engineering) King Mongkut's University of Technology Thonburi, 2022
SCHOLARSHIP/ RESEARCH GRANT	BX scholarship