UNIVERSITAS INDONESIA

SEMANTIC ROLE LABELING IN INDONESIAN CONVERSATIONAL LANGUAGE USING RECURRENT NEURAL NETWORKS

SKRIPSI

VALDI RACHMAN
1306381862

FAKULTAS ILMU KOMPUTER
PROGRAM STUDI ILMU KOMPUTER
DEPOK
JUNE 2017

UNIVERSITAS INDONESIA

SEMANTIC ROLE LABELING IN INDONESIAN CONVERSATIONAL LANGUAGE USING RECURRENT NEURAL NETWORKS

SKRIPSI
Diajukan sebagai salah satu syarat untuk memperoleh gelar
Sarjana Ilmu Komputer

VALDI RACHMAN
1306381862

FAKULTAS ILMU KOMPUTER
PROGRAM STUDI ILMU KOMPUTER
DEPOK
JUNE 2017

HALAMAN PERNYATAAN ORISINALITAS

Skripsi ini adalah hasil karya saya sendiri,
dan semua sumber baik yang dikutip maupun dirujuk
telah saya nyatakan dengan benar.

Nama	:	**Valdi Rachman**
NPM	:	**1306381862**
Tanda Tangan	:	
Tanggal	:	**13 Januari 2017**

HALAMAN PENGESAHAN

Skripsi ini diajukan oleh	:	
Nama	:	Valdi Rachman
NPM	:	1306381862
Program Studi	:	Ilmu Komputer
Judul Skripsi	:	Semantic Role Labeling in Indonesian Conversational Language using Recurrent Neural Networks

Telah berhasil dipertahankan di hadapan Dewan Penguji dan diterima sebagai bagian persyaratan yang diperlukan untuk memperoleh gelar Sarjana Ilmu Komputer pada Program Studi Ilmu Komputer, Fakultas Ilmu Komputer, Universitas Indonesia.

DEWAN PENGUJI

Pembimbing 1	:	Rahmad Mahendra	(	)
Pembimbing 2	:	Alfan Farizki Wicaksono	(	)
Penguji	:	-	(	)
Penguji	:	-	(	)

Ditetapkan di	:	Depok
Tanggal	:	-

REMARKS

Segala puji dan syukur bagi Allah Subhanahu wa Ta'ala, Tuhan semesta alam. Semoga keselamatan dan kesejahteraan senantiasa terlimpahkan atas junjungan kita Nabi Muhammad Shallallahu Alaihi Wasallam, sebaik-baik teladan bagi umat manusia.

Penulisan skripsi ini ditujukan untuk memenuhi salah satu syarat untuk menyelesaikan pendidikan pada Program Sarjana Ilmu Komputer, Universitas Indonesia. We sadar bahwa dalam perjalanan menuntut ilmu di universitas hingga dalam menyelesaikan skripsi ini, we tidak sendiri. We ingin berterima kasih kepada pihak-pihak yang selalu peduli, mendampingi, dan mendukung we, yaitu:

1. Kedua orang tua we, Sumanto dan Suparti yang selalu memberikan dukungan dan doa kepada we.
2. Dra. Mirna Adriani, Ph.D. dan Alfan Farizki Wicaksono S.T., M.Sc. selaku dosen pembimbing yang banyak memberikan arahan, masukan, dan bantuan dalam menyelesaikan skripsi ini.
3. Rahmad Mahendra, S.Kom., M.Sc. yang telah memberikan banyak masukan dan saran dalam menyelesaikan skripsi ini.
4. Andreas Febrian yang telah membuat *template* dokumen skripsi ini, sehingga we menjadi terbantu dalam menulis skripsi.
5. Erik Dominikus yang telah mempublikasikan dan mempopulerkan *template* dokumen skripsi ini, sehingga we menjadi tahu bahwa ada *template* tersebut.
6. Alfan Nur Fauzan, S.Kom. yang telah memberikan *template* dokumen skripsi yang telah diperbaiki ini, sehingga we sangat terbantu dalam melakukan penulisan
7. Teman-teman Lab Information Retrieval (Dipta Tanaya, Putu Wira Astika Dharma, Andi Fajar Nur Ismail, dan Ken Nabila Setya), sebagai rekan yang banyak memberi masukan dan berbagi ide dengan we.
8. Segenap teman-teman angkatan 2013 (Angklung) yang memberi dukungan dan semangat kepada we untuk menyelesaikan skripsi ini.

9. Pihak-pihak lain yang tidak dapat we sebutkan satu-persatu yang sudah memberikan bantuan dan dukungannya kepada we.

Depok, Januari 2017

Valdi Rachman

HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI TUGAS AKHIR UNTUK KEPENTINGAN AKADEMIS

Sebagai sivitas akademik Universitas Indonesia, saya yang bertanda tangan di bawah ini:

Nama	:	Valdi Rachman
NPM	:	1306381862
Program Studi	:	Ilmu Komputer
Fakultas	:	Ilmu Komputer
Jenis Karya	:	Skripsi

demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Universitas Indonesia **Hak Bebas Royalti Noneksklusif (*Non-exclusive Royalty Free Right*)** atas karya ilmiah saya yang berjudul:

Semantic Role Labeling in Indonesian Conversational Language using Recurrent Neural Networks

beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Noneksklusif ini Universitas Indonesia berhak menyimpan, mengalihmedia-/formatkan, mengelola dalam bentuk pangkalan data (*database*), merawat, dan memublikasikan tugas akhir saya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyatan ini saya buat dengan sebenarnya.

Dibuat di : Depok
Pada tanggal : 13 Januari 2017

Yang menyatakan

(Valdi Rachman)

ABSTRAK

Nama	:	Valdi Rachman
Program Studi	:	Ilmu Komputer
Judul	:	Semantic Role Labeling in Indonesian Conversational Language using Recurrent Neural Networks

Saat ini, seseorang dapat memanfaatkan forum kesehatan *online* untuk mencari tahu perihal penyakit tanpa perlu tatap muka dengan dokter. Melalui forum tersebut, seseorang hanya perlu menuliskan keluhan dan pertanyaan pada formulir yang tersedia. Banyak sekali informasi bermanfaat yang dapat diperoleh dari forum tersebut seperti keluhan, obat atau langkah penyembuhan. Penelitian ini mencoba untuk melakukan ekstraksi entitas *disease, symptom, treatment* dan *drug* secara otomatis. We memandang permasalahan ini sebagai permasalahan *sequence labeling* sehingga we mengusulkan penggunaan teknik *Deep Learning* dengan menggunakan *Recurrent Neural Networks* (RNNs), karena RNNs merupakan *state-of-the-art* untuk permasalahan *sequence labeling*. We mengusulkan fitur kata itu sendiri, kamus kesehatan, *stop word*, POS-Tag, frasa kata (nomina dan verba), kata sebelum dan kata sesudah. Selain itu we juga mengusulkan dua arsitektur RNNs, yaitu LSTMs 1 layer dan LSTMs 2 layer multi-*input*. Hasil eksperimen menunjukkan bawah model yang diusulkan mampu memberikan hasil yang cukup baik. Berdasarkan eksperimen dengan kombinasi fitur kata itu sendiri, kamus kesehatan, *stop word*, frasa kata (nomina dan verba), 1 kata sebelum dan 1 kata sesudah dengan arsitektur LSTMs 1 layer mampu mencapai rata-rata *f-measure* 63.06% dan LSTMs 2 layer mampu menghasilkan rata-rata *f-measure* 62.14%. Hasil tersebut lebih baik dibandingkan dengan *baseline* yang digunakan, yaitu penelitian Herwando (2016) dengan *f-measure* 54.09%.

Kata Kunci:
MER, RNNs, *disease*, *symptom*, *treatment*, *drug*

ABSTRACT

Name	:	Valdi Rachman
Program	:	Computer Science
Title	:	Semantic Role Labeling in Indonesian Conversational Language using Recurrent Neural Networks

Semantic Role Labeling (SRL) has been extensively studied, mostly for understanding English formal language. However, only a few reports exist for informal conversational language, especially for language being used in the chatbot system. The challenges of informal texting language include a wide variety of slangs and abbreviations, short sentences, as well as disorganized grammars. In this work, we propose a new set of semantic roles and a Context-Aware Bi-Directional Long Short-Term Memory Networks model for solving SRL task on informal conversational language. We utilized word embedding and linguistic components as our main features. The SRL task was mainly evaluated on Indonesian informal conversational language used on chatting platform. Although this is a pilot task, we obtained a really promising result with F1 score of 74.78%.

Keywords:
MER, Semantic Role Labeling, deep learning, conversational language, RNNs

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR KODE

BAB 1
INTRODUCTION

1.1 Background

Semantic Role Labeling (SRL) is a task in Natural Language Processing (NLP) which aims to automatically assign semantic roles to each argument for each predicate in a given input sentence. As for a brief definition, given an input sentence, SRL system will give an output of *"Who did what to whom"* with *what* as the predicate and *who* and *whom* being the argument of the predicate. SRL is an integral part of understanding natural language as it helps machine to retrieve semantic information from the input. In practice, SRL has been widely used as one of the intermediate steps for many NLP tasks, some of which are information extraction Emanuele et al. (2013); Surdeanu et al. (2003), machine translation Liu dan Gildea (2010); Lo et al. (2013), question-answering Shen dan Lapata (2007); Moschitti et al. (2003).

In the chat bot industry, the bots need to understand semantic information of the user's text in order to generate more personalized response. To illustrate, suppose that the user send a text chat to the bot as follows.

Input: *"I just ate chicken rice! Haha"*

The SRL system then extracts the semantic roles of the text.

Roles:

Predicate: *eat*

Agent: *I*

Patient: *chicken rice*

By knowing that the user just ate a chicken rice, the bot can thus response with *"That's great! how was the chicken?"*. This way, the user will be more engaged to the conversation with the bot.

As we can see from the example above, it is worth noting that the style of language used on chatting platform is different than those in formal text. In this work, we call this as *conversational language*. While formal language has been extensively studied in terms of SRL system, conversational language is yet to explore. The language is informal and thus, it has some unique characteristics including a wide variety of slangs and abbreviations, short sentences, and disor-

ganized grammars. These characteristics are the challenges an SRL system should tackle in understanding conversational language.

SRL can be seen as either a classification or sequence labeling problem. The earlier research on SRL was conducted with the classification approach, meaning that each argument is being predicted independently from the others. Those research focused on how to extract meaningful features out of syntactic parsers Gildea dan Jurafsky (2002); Gildea dan Palmer (2002); Pradhan et al. (2005), such as the path to predicate and constituent type. This syntactic information plays a pivotal role in solving SRL problem Punyakanok et al. (2008) as it addresses SLR's long distance dependency Zhou dan Xu (2015). Thus, traditional SRL system heavily depends on the quality of the parsers. The analysis done by Pradhan et al. shows that most errors of the SRL system were caused by the parser's error Pradhan et al. (2005). In addition, those parsers are costly to build, since it needs linguistic experts to annotate the data. If we want to create an SRL system on another language, one should build a new parser all over again for it. Zhou dan Xu (2015).

In order to minimize the number of hand-crafted features, Collobert et al. utilized deep learning for solving NLP tasks including Part-of-Speech Tagging (POS), Chunking (CHUNK), Named Entity Recognition (NER), and Semantic Role Labeling (SRL) with classification approach Collobert et al. (2011). The research aims to prevent using any task-specific feature in order to achieve state-of-the-art performance. The word embedding is used as the main feature across tasks, combined with Convolutional Neural Networks (CNN) architecture to train the model. They achieve promising results for the POS Tagging and Chunking, while features from the parsers are still needed to achieve competitive results for SRL.

Different from the previous works, Zhou et al. view SRL as a sequence labeling problem in which the arguments are labeled sequentially instead of independently Zhou dan Xu (2015). They proposed an end-to-end learning of SRL using Deep Bi-Directional Long Short-Term Memories (DB-LSTM), with word embedding as the main feature. Their analysis suggests that the DB-LSTM model implicitly extracts the syntactic information over the sentences and thus, syntactic parser is not needed. The research result outperforms the previous state-of-the-art traditional SLR systems as it achieves F1 score of 81,07%. The research also shows that the performance of the sequence labeling approach using DB-LSTM is better than the classification approach using CNN, since the DB-LSTM can extract syntactic information implicitly.

On the other hand, the number of research focusing on SRL for Indonesian language (next, will be called as *Indonesian*) is still low. One example would be

a research done by Dewi [x], which proposed SRL for Indonesian using Support Vector Machine. The research done by Dewi used the TreeBank data (in English) translated to Indonesian language using Google Translate. The research result opens a window of improvement as the best result consists of 61,6

Kata.ai is a technology company focusing on Artificial Intelligence (AI) and NLP development in Indonesian. Its goal is to empower businesses by leveraging the power of AI and NLP towards customer engagement in a form of chat-bot. In order to achieve it, there has been an ongoing research project by the company focusing on Indonesian NLP. Since it uses chatting platforms as the medium, the scope of the project is for informal Indonesian short text. Informal language is the most natural way we use to communicate in our daily life and thus, we found it interesting to understand it better through SRL.

Telling from the characteristics, informal Indonesian short text has its own challenges. It has many informal words, known as '*slang*' in English, for daily conversations. For example, the verb *belikan* (*buy*) has its informal form which is 'beliin'. Another example would be *berbicara* (*talk*) as *ngobrol*. It happens to many words in Indonesian. Not to mention the variety of ways to express pronoun *aku *(*I*) which are 'gw', 'gue', and 'aq'. Yet, they have many kinds of interjection such as âĂIJehâĂİ, âĂIJduuhâĂİ, âĂIJdongâĂİ, âĂIJkokâĂİ which complexify of the structure. Moreover, daily conversations include non-sentential utterances, which could be tricky for the SRL task. These characteristics need deep understanding on how the SRL works towards informal Indonesian short text.

Based on the fact that we still lack of Indonesian SLR research, it is an interesting opportunity to build SLR system for Indonesian. Our main contribution in this work would be applying SRL to informal Indonesian short text. We will deep dive into the semantic role characteristics found in informal Indonesian short text. After that, we will use deep learning as the state-of-the-art approach that has been emerging in NLP field for doing the SRL task.

While many of the previous works studied SRL on formal language, our research aims to explore SRL on conversational language, which is still under-resourced. Following Zhou et al. Zhou dan Xu (2015), we view SRL as a sequence labeling problem. We thus introduce a new set of semantic roles for this language type. Furthermore, we propose a new architecture named Context-Aware Bi-Directional Long Short-Term Memories, designed with attention mechanism in order to capture context information of the sentence at a higher level.

This work explores the SRL on conversational language, including creating a new set of semantic roles and proposing a new architecture, the so-called Context-

Aware Bi-Directional Long Short-Term Memory Networks. We utilized word embedding and linguistic components as our main features. The SRL task was mainly evaluated on Indonesian conversational language used on chatting platform. Although this is a pilot task, we obtained a really promising result with F1 score of 74.78%.

1.2 Problem Statement

Based on the motivation described in the background, we therefore propose following problem statements:

1. Which feature combination outputs the best performance?
2. Which model architecture gives the best result?

1.3 Objectives and Contributions

The objectives of this research includes understanding the semantic role characteristics of informal Indonesian short text and performing Semantic Role Labeling for informal Indonesian short text using deep learning approaches.

1.4 Methodology

The methodology of this work consists of literature review, data gathering, model development, experiment, evaluations and analysis, and conclusion.

1. Literature Review
   In this step, we did a comprehensive study on Natural Language Processing (NLP) and Machine Learning (ML) aspects. The NLP aspect includes language model and semantic role labeling. For machine learning, we learned deep learning approach such as recurrent neural networks and convolutional neural networks. These knowledge are the basis to support our research
2. Data Gathering
   Since there seems to be no available corpus for SRL on Indonesian, especially conversational language, we therefore annotated our own corpus. We retrieved the real word data from one of Kata.ai's chat bots. For this annotation, we build a new set of semantic roles crafted for Indonesian conversational language.

3. Model Development
   After we gathered our corpus, we then design the model for the experiment in this research. We define the feature extractions and the deep learning model architecture that will be tested. In this section, we also propose our own architecture.

4. Experiment
   In this step, we design our experiment scenarios in order to answer the questions being asked in the /perumusan masalah/. There are two set of scenarios consisting of feature and architecture experiments. The first one aims to find which feature combination outputs the best result, meanwhile the later focuses on comparing deep learning architecture models.

5. Evaluation and Analysis
   The experiment results are then to be evaluated and analyzed. We use precision, recall, and F1 as the metrics for the evaluation. We also conduct error analysis to get a deeper insight on the results.

6. Conclusion
   In the end, we conclude our findings in our research based on the evaluations and analyses of the experiments. We then describe some future works that can be done following the results of this research.

1.5 Scope

1. **Linguistic:**
   Data set includes: * Single clause with: * Single verb as the predicate * Single adjective or single noun as the predicate * Data set does not include: * Multiple clauses

1. **Computationa Linguistics:**
   * Algorithm used are LSTM, CNN, ...etc.

1.6 Organization

The organization of the rest of this thesis will be divided into 6 chapters. In chapter 2, literature review on the previous works will be shown. The methodologies used to do the annotation and the design of the deep learning is presented in chapter 3. This chapter will also provide the new idea of contribution for this research. In the

next chapter, implementation is described with the details of tools used, experiment scenarios, as well as the measurement setup. In chapter 5, all the experiment results based on the scenario defined in chapter 4 is presented. The result then will be evaluated and analyzed in chapter 6. Lastly, the conclusion and possible future works are described in chapter 7.

This report is organized as follows. We first explain the previous works on SRL systems in section 2. In section 3, the methodology of the research is described, including the features and the model architectures being used. The results and analysis are then explained in section 4. Finally, we report our conclusion and potential future works in the last section.

- Chapter 1 INTRODUCTION
  Pada bab ini we menjelaskan mengenai motivasi dalam melakukan penelitian ini dan komponen-komponen utama penelitian seperi latar belakang, perumusan masalah, tujuan dan manfaat penelitian, metodologi penelitian, ruang lingkup penelitian dan sistematika penulisan.

- Chapter 2 LITERATURE REVIEW
  Pada bab ini we melakukan studi literatur mengenai beberapa teori dan penelitian yang dilakukan oleh penulis lain.

- Chapter 3 METHODOLOGY
  Pada bab ini we menjelaskan alur dari penelitian ini, yaitu pengumpulan data, pra-pemrosesan, pelabelan, pengembangan model, eksperimen dan evaluasi.

- Chapter 4 IMPLEMENTATION
  Pada bab ini we menjelaskan proses implementasi sistem dan eksperimen berdasarkan rancangan yang telah Valdi Rachman tentukan pada bab sebelumnya. Selain itu we juga menjelaskan implementasi dari masing-masing tahapan yang dilakukan.

- Chapter 5 EXPERIMENTS
  Pada bab ini we menjelaskan analisis dari hasil eksperimen yang telah we kerjakan pada tahap sebelumnya. Hasil eksperimen we sajikan dalam bentuk tabel dan grafik.

- Chapter 6 CONCLUSIONS
  Pada bab ini we memberikan kesimpulan berdasarkan hasil eksperimen dan analisis yang telah dilakukan pada penelitian ini. Selain itu we juga memberikan saran dan masukan untuk penelitian dan pengembangan sistem mengenai MER berbahasa Indonesia selanjutnya.

BAB 2
LITERATURE REVIEW

This chapter focuses on literature study on 3 aspects including language models, deep learning, and semantic role labeling. In language model section, Part-of-Speech Tag (POS Tag) and word embedding are described. Deep learning section focuses on the architecture widely used for sequence labeling problem. Finally, we explain semantic role labeling in the last section, including the semantic roles definition, annotation corpus, problem definitions, common features, and the historical perspectives.

2.1 Language Models

This section explains the language models usually used in Natural Language Processing (NLP) applications. We first describe the traditional yet important language model, Part-of-Speech Tag (POS Tag), followed by word embedding that is often used in recent NLP application with deep learning.

2.1.1 Part-of-Speech Tag (POS Tag)

2.1.2 Word Embedding

Word representation is an important feature when one wants to build deep learning model for NLP tasks. The idea is to convert words into vectors. There are two approaches for this vector representation, which are traditional and word embedding approach. Traditional approach uses one-hot vectors for the representation, meanwhile word embedding approach uses real values vectors that contain information about the words.

In the traditional approach, the vectors are retrieved based on the index of the word found in the dictionary. The dictionary consists of the word and its index. Suppose that we have four words: I, eat, chicken, beef. Each of these words has their own index, with I:0, eat:1, chicken:2, beef:3. These indices will represent the one-hot vectors for the words. For instance, word with index 0 has a one-hot vector [1 0 0 0], word with index 1 has a one-hot vector [0 1 0 0], and so on. The length of the vector is determined by the size of our dictionary. In this case, the size of our dictionary is 4, hence the length of the vector is also 4.

However, this representation has two drawbacks. First, it hardly depends on the dictionary. If the dictionary does not have the desired word, then it could not represent the word into vector. If we want the dictionary to cover all the words, the size of the dictionary will be extremely huge. Second, the vector representation is sparse. Since we just give the index to the all the words based on the dictionary, it does not really represent an important information from the words. The word âĂŸhotelâĂŹ and âĂŸhostelâĂŹ, though have similar context, could be represented by two far indices, say 1 and 100.

Word embedding aims to address the second issue. Word embedding converts similar words with similar vectors. From the previous example, the word hotel and hostel will have vectors that are close to each other. Hence, the vector representations are dense. Unfortunately, it still could not solve the first drawback that out-of-vocab words could not be represented as vectors.

There has been a lot of research on word embedding (Mikolov et al, 20xx) (Mikolov et al, 20xx) (Mikolov et al, 20xx). In this section, we will explain word embedding architectures proposed by Mikolov, called as Word2Vec. Word2Vec uses unsupervised approach so that we only need a lot of unlabeled data for building word embedding model. Word2Vec has two architectures, which are Context Bags of Words (CBOW) and Skipgram. Fig XX shows the difference between CBOW and Skipgram architectures. In CBOW, the model learns to predict a word based on its neighbouring words. In contrast, Skipgram aims to predict the neighbouring words of a word.

(GAMBAR CBOW DAN SKIPGRAM)

Both architectures mainly aims to build language model. For word embedding model, we do not need the whole architecture model after we finish training the model. Instead, we only need to extract the weight matrix W when the model converts the word index into vector. This weight matrix W is our word embedding model that we use to represent our input words.

2.2 Deep Learning

2.2.1 Recurrent Neural Networks

Recurrent Neural Networks, shortened as RNN, is part of neural network family for processing sequential data. It is thus perfect for modeling the sequence labeling problem. Suppose that we have sequence of inputs, RNN will take each input in a time step t to process it in a function. Figure XX shows a general RNN.

(GAMBAR SEDERHANA RNN)

The left picture illustrates the folded RNN model applied to all time steps. Note that the black rectangle represents one time step delay, meaning that that input is coming from the output of the previous time step.

The right picture shows the unfolded RNN that is more intuitive since it visualizes the time steps. There are three layers in every time step t, which are input, hidden, and output layers, denoted as x, h, and o respectively. The input layer is for the input representations. In the hidden layer, it contains information from the input layer as well as those coming from hidden layers in the previous time steps. The output layer consists of the output of the model. These three layers are in a form of vectors. In every time step t, RNN has input layer x(t) E R A, hidden layer h(t) E R H, and output layer y(t) E B. The values of A, H, and B, represent the length of the input vector, the number of unit in a hidden layer, and the length of the output vector. There are three parameters that will be trained, which are U, V, and W. These parameters are the weight matrices for connecting two layers. U E R HxA connects input with hidden (input-hidden), W E R HxH connects hidden with the previous hidden (hidden-hidden) and V E R BxH connects hidden with ouput (hidden-ouput). These parameters are shared across time steps.

Every input layer x(t) is mapped into output layer o(t) in every time step t. In the middle of the process, it calculates the hidden layer h(t) from two layers, x(t) and h(t-1). The output layer o(t) then is retrieved by performing a function to the hidden layer h(t). The general equations for RNN is presented as follows: âĂć h(t) = f1(U. x(t) + W . h(t-1) + b) âĂć o(t) = f2(V. h(t) + c) âĂć Where h(0) = f1(U . x(0))

Note that there are two additional parameters to train, which are the bias vectors b and c. In the first equation, the input x(t) and h(t-1) is weighted by matrices U and W respectively, added by a bias vector b. The result is then inserted to an activation function f1 in order to produce hidden layer h(t). In the second equation, h(t) is multiplied by the weight matrix V and added by a bias vector c, before being processed by the activation function f2 to produce o(t). The examples of activation function f1 and f2 are tanh and softmax.

Based on this illustration, there are two main characteristics of RNN:

1. It has a cycle in the graph for every time step. Hidden layer h(t) will be one of the inputs for forming h(t+1).

2. It has shared parameters across time steps.

Fig XX illustrates a more complete RNN model on how it is being trained.

(GAMBAR BAGAIMANA RNN DI TRAIN)

The goal of training the model is to find the estimated values of parameters W,

U, V, b, and c which produce outputs o(t) as close as the expected outputs y(t) in the training data.

The loss function L acts as the measure the difference between the predicted output o(t) and the expected output y(t) in every time step t. The more little the difference, the better the model. The machine thus has to minimize the result of loss function as small as possible. The parameters W, U, V, b, and c are unknown in the beginning. At first, these parameters are initiated randomly.For every iteration (called as epoch), the machine aims to learn the best values for each parameters.

The way to do so is by computing the gradient for each iteration. The idea behind computing the gradient values is to show us which parameter setting that brings us into smaller loss function result. By having this information, the machine then sets the better values for each parameter in the next iteration in order to reduce the loss function. From one iteration into another, the machine will find better parameter values to minimize the loss function. The learning method based on the gradient information is called optimization algorithm such as Stochastic Gradient Descent (), Adam (Kingma and Ba, 2014), and RMSProp (Hinton, 2012)

2.2.2 Long Short-Term Memories

There is an issue in traditional RNN that our networks should overcome, it is called vanishing and exploding gradient problem. The RNN architecture repeatedly uses same parameters for each time steps. Suppose that we use W as the parameter used for each time step between the hidden units. After t time steps, the matrix would be multiplied t times, hence it is the same as multiplying the hidden units with Wt. Assuming that W has an eigendecomposition W = X diag(lambda) X-1, Wt is equal to:

Wt = (X diag(lambda) X-1)t = (X diag(lambda)t X-1)

The eigenvalues lambda in diag(lambda) will either vanish if they are less than 1 in magnitude or explode if they are greater than 1 in magnitude. The gradient counted in each time step is aligned with the eigenvalues. Hence, the gradient may also vanish or explode. This is what we called as vanishing and exploding gradient problem. When the gradient vanishes, it is hard for the machine to find the direction to reduce the cost function. In the case of exploding gradient, the learning algorithm will become unstable.

To address this issue, there are solutions proposed such as leaky units (Mozer, 1992), simulated annealing and discrete error propagation (Bengio et al., 1994), time delays (Lang et al., 1990), and hierarchical sequence compression (Schmidhuber et al., 2007). Among these approach, one of the most robust solutions

is called Long Short Term Memories (LSTM) (Hochreiter et. al., 1997).

The modification used in LSTM to address the issue is by using gates. It is basically RNN, but the nonlinear units in the hidden layer is replaced by the memory blocks. Fig XX shows the difference between RNN and LSTM for the hidden layer. One nonlinear unit tanh in RNN is replaced by a more complex memory blocks in LSTM. Besides the hidden layer h(t), LSTM also has m(t) which is called memory cells. The idea of LSTM is to learn when to forget or remember the memory from previous time steps through multiplicative gates. It thus prevents the vanishing and exploding gradient problem. For example, if the input gate is closed, then the memory will be unchanged.

(GAMBAR ONE BLOCK MEMORY IN LSTM)

Fig XX illustrates a one block memory in LSTM. There are three main gates which are forget gate, input gate, and output gate. These gates are responsible to determine whether an information is added, kept, or deleted in a cell. Each gate has sigmoid layer and element-wise operations. The sigmoid layer converts the input into a probability between 0 and 1. This probability describes the gate behavior towards the input, whether to accept it (probability close to 1) or not (probability close to 0).

The equations of the sigmoid layers for each of the gates are explained as follows:

1. Forget Gate

This gate is responsible to determine how much the information from the past should be kept in the memory. The equation of sigmoid layer in forget gate is:

At = sigmoid(WAX X + WAH . h tâĂŞ 1 + WAM . m t-1)

2. Input Gate

This gate is responsible to determine how much the current information x(t) should be kept in the memory. The equation of sigmoid layer in input gate is:

Bt = sigmoid(WAX X + WAH . h tâĂŞ 1 + WAM . m t-1)

3. Output Gate

This gate is responsible to determine the output of a time step based on current cell state. The equation of sigmoid layer in output gate is:

Yt = sigmoid(WAX X + WAH . h tâĂŞ 1 + WAM . m t-1)

In every time step t, the equation for computing cell state m(t) and hidden layer h(t) is presented below:

Mt = At (x) mt-1 + Bt (X) . tanh (Wxmx Xt + Whm. H t-1) Ht = Yt (x) tanh(mt)

2.3 Semantic Role Labeling

Semantic role labeling (SRL) is a task in Natural Language Processing to assign semantic roles for each argument for each predicate in given input sentence. In this sub-chapter, the definition of semantic roles will be explained. This sub-chapter then explains the most commonly used annotation corpus for SRL. In the end, the details on semantic role labeling task is described.

2.3.1 Semantic Roles

Semantic roles are the representations that express the abstract role of that arguments of a predicate can take in the event (Jurafsky, 2015). When it comes to understanding natural language, one would want to understand the events and their participants of a given input sentence. In this case, the events refer to the predicate and the participants refer to the argument. The example below illustrates the connection between a predicate and its arguments. Andy eats fried chicken Argument Predicate Argument In this example, eat is the predicate with Andy and fried chicken as its argument. With this point of view, the predicate can be seen as the center of the sentence, followed by the arguments that depend on it.

Knowing the predicate and its arguments is not enough to understand the sentence since the roles of the arguments towards the predicate are unknown. In the previous example, it would be more meaningful to differentiate that Andy is the Eater and fried chicken is the EatenThing. Eater and thing eaten are the examples of semantic roles for the predicate eat. These semantic roles could be used to identify the roles of the arguments regardless its position in the sentence. The previous example could be represented with 2 ways: Andy eats fried chicken Eater Predicate EatenThing The fried chicken is eaten by Andy EatenThing Predicate Eater Both sentences represent the role of Andy and fried chicken as eater and thing eaten respectively, regardless of their position in the sentence as a subject or object.

There are many ways to define such semantic roles. From the examples above, the semantic roles are very specific for its predicate, known as deep roles (Jurafsky, 2015). Eater and ThingEaten are semantic roles for the predicate eat, Kicker and KickedThing are semantic roles for the predicate kick, and so on. In order to further knowing more about the semantics of these arguments, these semantic roles could be generalized into more abstract roles. Eater and Kicker have something in common: they are volitional actors having direct causal responsibility for the predicate. For this reason, thematic roles are introduced as a set of semantic roles designed to capture semantic commonality between Eater and Kicker (Jurafsky,

2015). With this in mind, Kicker and Eater can be represented as AGENT, which represents the abstract concept that is a volitional causer of an event (or predicate). On the other hand, EatenThing and KickedThing both represent the direct objects that are affected by the event. The semantic role for EatenThing and KickedThing is THEME.

Table x shows the thematic roles often used across computational papers (Jurafsky, 2015).

(TABLE CONTOH SEMANTIC ROLES)

2.3.2 Annotation Corpus

There are available annotated corpus for SRL consists of sentences labeled with semantic roles. Researchers are using these annotated corpus for building supervised machine learning model for SRL. The two most commonly used annotation corpus for SRL are Proposition Bank and FrameNet.

2.3.2.1 Proposition Bank

Proposition Bank, shortened as PropBank, is a corpus in which sentences are annotated with semantic roles. PropBank corpus is available for English, Chinese, .., ..., ... The main approach used for its semantic roles grouping is based on proto-roles and verb-specific semantic roles. Every verb sense has its set of semantic roles with argument numbers rather than names, for example: Arg0, Arg1. Arg2, etc. Generally, Arg0 represents PROTO-AGENT while Arg1 represents PROTO-PATIENT. Other argument number representations may vary based on each verb sense.

The PropBank entries are called frame files. One example of the frame files for one sense of verb eat is presented below.

Frame File: Eat.01 Arg0: Eater Arg1: Things Eaten Arg2: Instrument used

Example: Ex1: [Arg0 Andy] eats [Arg1 fried chicken] [Arg2 with spoon] Ex2: [Arg1 That fried chicken] is eaten by [Arg0 Andy] [Arg2 with spoon]

For verb sense Eat.01, Arg0 acts as the Eater (PROTO-AGENT), and Arg1 represents the Things Eaten (PROTO-PATIENT). As we can see from the example above, we can infer the commonality between examples Ex1 and Ex2 regardless its structure, be it in a passive or active voice. In both examples, Andy is the Eater and fried chicken is the Things Eaten. In this frame file, there is also another argument, Arg2, that represents the instrument used by the Eater. In example Ex1 and Ex2, the instrument is spoon.

Other non-numbered arguments are available in PropBank, the so-called ArgMs, representing modifiers that could be used across frame files. Some examples of ArgMS include:

TMP: When? LOC: Where? DIR: Where to/from?

The next annotation corpus is called FrameNet which has different approach on how to group set of semantic roles. Instead of using verb-specific, it uses frame-specific grouping.

2.3.2.2 FrameNet

FrameNet is an annotation corpus for semantic roles that are specific to a frame. In PropBank, the semantic roles are defined based on each sense of a verb. In contrast, a frame in FrameNet could include more than one predicate (verbs or nouns) that have the same background context. Each frame consists of two elements: 1. A set of semantic roles related to this frame, and 2. A set of predicates using the respective semantic roles.

One example is a frame called change position on a scale defined as: This frame consists of words that indicate the change of an ItemâĂŹs position on a scale (the Attribute)from a starting point (Initial value) to an end point (Final value) The set of semantic roles for a frame is divided into two roles: Core roles and Non-Core Roles. Core Roles are specific to a frame while Non-Core Roles are more general across frames (like ArgMs in PropBank). The set of semantic roles of the frame change position on a scale is explained as bellow:

Core Roles ITEM: The entity that has a position on the scale. ATTRIBUTE: The ATTRIBUTE is a scalar property that the ITEM possesses DIFFERENCE: The distance by which an ITEM changes its position on the scale. FINAL STATE: A description that presents the ITEMâĂŹs state after the change in the ATTRIBUTEâĂŹs value as an independent predication. FINAL VALUE: The position on the scale where the ITEM ends up. INITIAL STATE: A description that presents the ITEMâĂŹs state before the change in the ATTRIBUTEâĂŹs value as an independent predication. INITIAL VALUE:The initial position on the scale from which the ITEM moves away. VALUE RANGE: A portion of the scale, typically identified by its end points, along which the values of the ATTRIBUTE fluctuate. Non-Core Roles DURATION SPEED GROUP The length of time over which the change takes place. The rate of change of the VALUE.

The possible predicates of the frame change position on a scale are: VERBS: dwindle move advance edge climb decline dip double drop reach decrease fluctuate rise diminish gain soar mushroom swell explode plummet swing fall triple tumble

rocket grow shift slide increase skyrocket decline jump escalation shift explosion tumble fall fluctuation ADVERBS: gain increasingly NOUNS: hike decrease ris The example of semantic roles of the frame change position on a scale could be seen as follows:

(22.20) [ITEM Oil] rose [ATTRIBUTE in price] [DIFFERENCE by 2

(22.21) [ITEM It] has increased [FINAL STATE to having them 1 day a month].

(22.22) [ITEM Microsoft shares] fell [FINAL VALUE to 7 5/8].

(22.23) [ITEM Colon cancer incidence] fell [DIFFERENCE by 50

(22.24) a steady increase [INITIAL VALUE from 9.5] [FINAL VALUE to 14.3] [ITEM in dividends]

(22.25) a [DIFFERENCE 5

As we can see from the examples above, rose, fell, and increase have the same set of semantic roles under the frame change position on a scale. Instead of defining the semantic roles for each verb sense one by one, FrameNet groups predicates (not limited to verbs) that have the same semantic roles as one frame.

2.3.3 Problem Definitions

Semantic Role Labeling (SRL) is one of Natural Language Processing task which aims to automatically assign semantic roles for each constituent (argument) for each predicate in a sentence (Jurafsky, 20XX). Current approach to solve this task is by using supervised machine learning. Given a labeled data, the machine learns from it and builds a generalization model. Researches often used PropBank or FrameNet corpus as the sources of annotated data. In this section, we describe the approaches to define the problem of SRL task, followed by the common features used for building supervised model for SRL.

There are two ways to see SRL problem, either as Classification or Sequence Labeling problem (Someone, 20XX). Classification approach assigns semantic roles for each word independently. Meanwhile, Sequence Labeling approach traverses from assigning semantic role for the first word until the last one in a sentence sequentially. In Sequence Labeing, the next label (semantic role) prediction of time step t is dependent to labels predicted on previous time steps (1..t-1). The differences of these two approaches to solve SRL task are visualized in Fig X.

[Fig X. The visualizations of the differences of Classification approach and Sequence Labeling approach]

The general algorithm for SRL based on Classification approach is explained as follows. Function SEMANTICROLELABEL(words) returns labeled tree Parse <-

Parse(words) For each predicate in parse do For each node in parse do Featurevector <- EXTRACTFEATURES(node, predicate, parse) CLASSIFYNODE(node, featurevector, parse)

Explanation on SRL based on Classification approach: âĂć Parsing input sentence into a parse tree âĂć Traverse the parse tree to find all the predicates âĂć For each predicate, traverse each node (word) in a parse tree to o Extract features o Classify the node based on features extracted. ïĆğ 1-of-N classifier is trained here to predict the semantic role for each node (word) in a parse tree. ïĆğ N is the number of possible semantic roles added with 1 Other role for word that has no semantic roles.

The classification algorithms that have been used to train 1-of-N classifier include logistic regression (Someone, 20xx) and SVM (Someone, 20xx).

[Penjelasan untuk algoritme Sequence Labeling approach]

2.3.4 Common Features for SRL

âĂć The first set of features for SRL is proposed by Gildea and Jurafsky (2000). o They are the first ones who used supervised machine learning approach to solve SRL. o Over the years, many research proposed new set of features to improve the result, but they still used the basic features proposed by Gildea and Jurafsky (2000). o Common features used for solving SRL task are: ïĆğ The predicate. âĂć Usually in a form of verb. ïĆğ The phrase type of the constituent. âĂć NP, PP, etc ïĆğ The headword of the constituent. âĂć The black bird. Headword: bird. ïĆğ The headword part of speech of the constituent. âĂć Example: NNP. ïĆğ The path of the parse tree from constituent to predicate. âĂć This is to represent the grammatical relationships between the constituent and the predicate. âĂć Example: NP S VP VBD ïĆğ The voice of the clause, active or passive. âĂć Example: I eat chicken rice (active), Chicken rice is eaten by me (passive). ïĆğ The binary linear position of the constituent from the predicate. âĂć Could be before or after the predicate. ïĆğ The subcategorization of the predicate âĂć Set of arguments that appear in the verb phrase VP. âĂć Example: NP and PP in âĂŸVP -> VBD NP PPâĂŹ ïĆğ The named entity type of the constituent âĂć Example: Organization, Person, Location, .. ïĆğ The first and last words of the constituent. o There are also other additional features that could be used for SRL. ïĆğ Sets of n-grams inside the constituent. ïĆğ Using dependency parser instead of syntactic parser for extracting features.

2.3.5 Historical Perspectives

SRL can be seen as either a classification or sequence labeling problem. The earlier research on SRL was conducted with the classification approach, meaning that each argument is being predicted independently from the others. Those research focused on how to extract meaningful features out of syntactic parsers Gildea dan Jurafsky (2002); Gildea dan Palmer (2002); Pradhan et al. (2005), such as the path to predicate and constituent type. This syntactic information plays a pivotal role in solving SRL problem Punyakanok et al. (2008) as it addresses SLR's long distance dependency Zhou dan Xu (2015). Thus, traditional SRL system heavily depends on the quality of the parsers. The analysis done by Pradhan et al. shows that most errors of the SRL system were caused by the parser's error Pradhan et al. (2005). In addition, those parsers are costly to build, since it needs linguistic experts to annotate the data. If we want to create an SRL system on another language, one should build a new parser all over again for it. Zhou dan Xu (2015).

In order to minimize the number of hand-crafted features, Collobert et al. utilized deep learning for solving NLP tasks including Part-of-Speech Tagging (POS), Chunking (CHUNK), Named Entity Recognition (NER), and Semantic Role Labeling (SRL) with classification approach Collobert et al. (2011). The research aims to prevent using any task-specific feature in order to achieve state-of-the-art performance. The word embedding is used as the main feature across tasks, combined with Convolutional Neural Networks (CNN) architecture to train the model. They achieve promising results for the POS Tagging and Chunking, while for SRL features from the parsers are still needed to achieve competitive results.

Different from the previous works, Zhou et al. view SRL as a sequence labeling problem in which the arguments are labeled sequentially instead of independently Zhou dan Xu (2015). They proposed an end-to-end learning of SRL using Deep Bi-Directional Long Short-Term Memories (DB-LSTM), with word embedding as the main feature. Their analysis suggests that the DB-LSTM model implicitly extracts the syntactic information over the sentences and thus, syntactic parser is not needed. The research result outperforms the previous state-of-the-art traditional SLR systems as it achieves F1 score of 81,07%. The research also shows that the performance of the sequence labeling approach using DB-LSTM is better than the classification approach using CNN, since the DB-LSTM can extract syntactic information implicitly.

While many of the previous works studied SRL on formal language, our research aims to explore SRL on conversational language, which is still under-resourced. We thus introduce a new set of semantic roles for this language type.

Furthermore, we propose a new architecture named Context-Aware Bi-Directional Long Short-Term Memories, designed with attention mechanism in order to capture context information of the sentence at a higher level.

Previous research have found useful to use RNN for NLP task Semantic Role Labeling (SRL). Before we discuss about the use of RNN on SRL, we describe the historical perspective of solving SRL with supervised machine learning. We divide the historical perspective based on SRL systems without and with deep learning.

The non-deep learning approach uses specific hand-crafted features for SRL, which mainly depend on syntactic or dependency parser as explained in section 2.XX. It started from Gildea et. Al (2002) who firstly build supervised machine learning model for SRL. The goal of the research was to create the first shallow semantic role parser which is not domain specific, since at that time all the semantic roles research were too domain specific. The features used are extracted from the syntactic tree Collins Parser (XX, 1997), such as Phrase Type, Parse tree path, voice, and head word. Then the predicate of a sentence is also added as a feature. The research used semantic role annotation based on FrameNet. The algorithm used was statistical classifier with backoff approach. The result is 65% precision and 61% recall.

Then, Gildea et al (2002) continues the research to quantify the effect of parser accuracy on SRL systemâĂŹs performance. The research also examines whether a flatter âĂIJchunkedâĂİ representation (which is less costly) of the input can be as effective as syntactic tree parser. The data used is from PropBank dataset, since it is from Wall Street Journal corpus that has a gold-standard syntactic parse trees for the entire dataset from the Penn Treebank Project. The finding shows that the parser accuracy affects the SRL system, since it is seen that the system with gold-standard parse tree impacts directly to build a better SRL system. Hence, the syntactic parser is an integral intermediary model to build a robust SRL system. If the parser is not good, one would not get a good SRL system.

Surdeanu et al (2003) proposed a new set of features for SRL system, such as POS Tag of Head Word, POS Tag of content word, and Named Entity Class of Content Word. They use inductive learning through decision trees C5 for the algorithm.

Xue et al (2004) aims to explore more information extracted from the parse tree in order to propose new set of features crafted to improve SRL. In their research, there are three steps for the model, pruning, argument identification, and argument classification. Pruning filters out constituents that are clearly not semantic arguments to the predicate. Argument identification classifies candidates as either

semantic arguments or non arguments. Argument classification then runs a multi-category classifier to classify the constituents with semantic roles. The features proposed for the argument classification are syntactic frame, lexicalized constituent type, lexicalized head word, and the head of Preposition Phrase parent.

Since the source of SRL system errors mostly based on syntactic parserâĂŹs error, Pradhan et al (2005) combines features from different syntactic parsers (Charniak parser and Collins Parser). The idea of combining two parser is that they train separate SRL systems for each tree parser. The role output from these two systems is used as additional features in a SRL system using flat syntactic view. They then use SVM classifier to train SRL based on PropBank data.

Aside from using syntactic parse tree like Charniak or Collins parser, one can build SRL system by extracting features from dependency parser. Some of the features extracted are word property, syntactic connection, semantic connection, and dependency path.

The drawbacks of using the non-deep learning approach are 1.) building syntactic or dependency parsers is costly, 2.) the SRL system hardly depends on the robustness of the parsers. Building tree parsers is costly because it is language-dependent and it needs experts in each language to create it. When we move to another language, we have to build these parsers for the new language from scratch. Not to mention a new problem arises when such parsers are not robust, hence creating error propagation in our SRL system. The analysis in Pradhan et al., (2005) says that the major source of errors in SRL system comes from the errors of the syntactic parsers from which we extract the features.

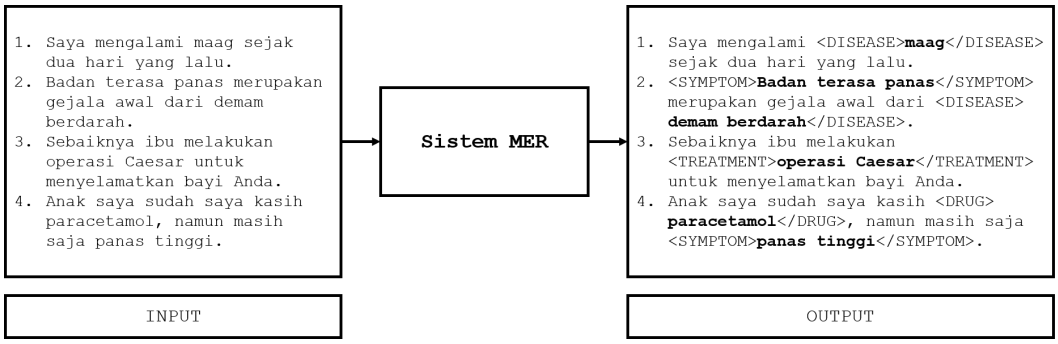
To address this issue, Collobert et al. (20XX) firstly introduced the use of deep learning for SRL and other core NLP tasks such as POS Tagging and Chunking. They use Convolutional Neural Network (CNN) with no task-specific features for the system. For example, in non-deep learning approach, we use POS Tagging features for Chunking, and we use both of which as the features for SRL. Instead, the main feature used in this research is word embedding. As explained in the previous section, word embedding model converts words to vectors. The word vectors then are fed into CNN architecture. However, though the model achieved state-of-the-art performances for POS Tagging and Chungking, that is not the case for SRL. For SRL, it is still needed to use features from the tree parser to achieve robust performance.

Zhou et al., (2015) proposed new architecture for the SRL system. Instead of seeing the SRL as a classification problem like the previous research including CollobertâĂŹs, Zhou considers SRL as the sequence labeling problem. Hence,

the suitable architecture for such problem is Recurrent Neural Networks (RNN). In their research, a more specific RNN architecture is used, which is Long Short Term Memories (LSTM) in order to prevent the vanishing and exploding gradient problem in RNN. They used the deep bi-directional LSTM. The âĂIJdeepâĂİ is for extracting more hidden features and the âĂIJbi-directionalâĂİ is for extracting information from the past and future. On top of the LSTM architecture, they used Conditional Random Field (CRF) for the output layer. For the word representation, they also used word embedding as one of the main features, along with predicate and context predicate. Our research is mainly inspired by this research.

2.4 Pengenalan Entitas Kesehatan

Pengenalan Entitas Kesehatan atau disebut juga dengan *Medical Entity Recognition* (MER) merupakan salah satu cabang dari Pengenalan Entitas Bernama (*Named Entity Recoginition*) atau disingkat NER dengan dokumen sumber berupa teks kesehatan. NER sendiri merupakan suatu sistem/aplikasi yang memanfaatkan teknik pada *Natural Language Processing* dan *Information Extraction* untuk mengenali entitas yang telah dikategorikan sebelumnya seperti nama, lokasi, organisasi, waktu dan sebagainya. Sedangkan pada sistem MER, entitas yang akan dikenali yaitu entitas yang berada pada domain kesehatan seperti nama penyakit (*disease*), gejala penyakit (*symptom*), obat (*drug*), langkah penyembuhan (*treatment*), nama protein, DNA, RNA dan lain sebagainya. Gambar 2.1 merupakan ilustrasi dari sebuah sistem MER.



Gambar 2.1: Ilustrasi Sistem MER

Dari ilustrasi di atas, sebuah sistem MER akan diberikan *input* berupa dokumen kesehatan, kemudian sistem diharapkan dapat memberikan *output* berupa dokumen yang sudah diberi label dengan benar. Dokumen kesehatan yang menjadi *input* dapat berupa dokumen formal seperti dokumen suatu rumah sakit atau dokumen non-formal seperti dokumen forum kesehatan *online*.

Implementasi sistem MER dapat memberikan manfaat pada beberapa bidang, seperti pada aplikasi *Question Answering* (Abacha dan Zweigenbaum, 2011) yang hasil pelabelan dari sistem MER dapat mempermudah identifikasi entitas yang ditanyakan. Selain itu, hasil pelabelan sistem MER juga dapat dimanfaatkan untuk pembuatan sistem *indexing* dokumen forum sehingga pencarian dokumen kesehatan dapat dilakukan dengan lebih efisien. Sistem MER juga dapat digunakan untuk mendukung aplikasi *entity linking* yang memungkinkan seseorang untuk mengetahui hubungan antar entitas (Hachey et al., 2013). Misalnya dengan adanya aplikasi *entity linking*, kita dapat mengetahui obat apabila hanya diberikan *query* nama penyakit dengan *resource* dokumen-dokumen kesehatan yang telah mendapatkan pelabelan dari sistem MER. Masih banyak manfaat lain dari implementasi sistem MER ini.

Sebelumnya Abacha dan Zweigenbaum (2011) telah melakukan penelitian terkait sistem MER pada dokumen berbahasa Inggris. Sistem MER yang dibuat bertujuan untuk melabeli entitas *treatment*, *problem* dan *test* dengan menggunakan 3 metode, yaitu (i) metode semantik dengan menggunakan *tools* MetaMap (*domain knowledge*), (ii) ekstraksi frasa berdasarkan *chunker* dan klasifikasi dengan SVM (*Support Vector Machine*) dan (iii) gabungan 2 metode sebelumnya dengan menggunakan CRF (*hybrid*). Metode *hybrid* yang dimaksud yaitu dengan menggunakan *tools* CRF sebagai *tools machine learning* yang ditambahkan fitur *domain knowledge*, yaitu fitur semantik yang diekstraksi dengan *tools* MetaMap. Hasil yang terbaik didapatkan dengan menggunakan metode *hybrid* yang menggabungkan 2 metode sebelumnya (*domain knowledge* dan *machine learning*) dan dengan *precision* 72.18%, *recall* 83.78% dan *f-measures* 77.55%.

Selain penelitian di atas, Mujiono et al. (2016) juga melakukan penelitian terkait MER pada dokumen berbahasa Indonesia. Model MER yang dikembangkan adalah untuk melabeli entitas *drug* saja. Penelitian tersebut bertujuan untuk mendapatkan representasi data yang berdasarkan karakteristik *training data*. Mujiono et al. (2016) mengusulkan tiga teknik representasi data yang berdasarkan karakteristik distribusi kata dan kemiripan kata dari hasil *training* dari model *word embedding*. Representasi data yang dimasud adalah: (i) semua kalimat diformat sebagai *sequence* token, (ii) semua kalimat di-*generate* menjadi beberapa *sequence*, dan (iii) data direpresentasikan sebagai vektor dengan *tools Word Embedding*. Masing-masing representasi kata tersebut dievaluasi dengan masing-masing evaluator, yaitu (i) evaluasi dengan model *neural networks* standar, (ii) evaluasi dengan dua *deep network classifiers*, yaitu DBN (*Deep Belief Networks*), dan SAE (*Stacked Denoising Encoders*) serta (iii) representasi kalimat sebagai vektor *word embedding*

yang dievaluasi dengan *recurrent neural networks* yaitu LSTM (*Long Short Term Memory*). Hasil yang didapatkan yaitu kalimat sebagai *sequence* yang dievaluasi dengan LSTM memberikan hasil yang terbaik, yaitu *f-measure* 86.45%.

Penelitian terkait MER pada dokumen berbahasa Indonesia sudah dilakukan sebelumnya oleh Herwando (2016). Dalam penelitiannya, Herwando (2016) menggunakan CRF (*Conditional Random Fields*) untuk proses pelabelan. Kemudian, pada pekerjaan yang Herwando (2006) lakukan, sebagian besar digunakan untuk mencari fitur-fitur yang memang diskriminatif untuk masalah MER yang menghasilkan akurasi terbaik. Entitas yang akan diberi label yaitu nama penyakit (*disease*), gejala penyakit (*sympton*), obat (*drug*) dan langkah penyembuhan (*treatment*). Dokumen yang menjadi *input* penelitian merupakan hasil *crawling* dari forum kesehatan *online* dari berbagai situs yang berisi tanya jawab. Hasil yang didapatkan yaitu *precision* 70.97%, *recall* 57.83% dan *f-measeure* 63.69% dengan fitur *its own word*, frasa, kamus (*symptom*, *disease*, *treatment* dan *drug*), *window feature (previous word)* dan panjang kata.

Selain itu, Suwarningsih et al. (2014) juga melakukan penelitian terkait MER pada dokumen berbahasa Indonesia dengan menggunakan SVM (*Support Vector Machine*), dengan SVM yang digunakan untuk klasifikasi per-kata. Entitas yang akan dikenali yaitu *location*, *facility*, *diagnosis*, *definition* dan *person*. Data yang digunakan sebagai korpus merupakan data dari situs *http://health.detik.com/*, *http://detikhealth.com/* dan *http://health.kompas.com/konsultasi/* dengan total keseluruhan sebanyak 1000 kalimat. Akurasi yang dihasilkan yaitu 90% dengan menggunakan fitur *baseline*, *word level (morphology, POS-Tag, dll)* dan fitur dari dalam dokumen tersebut.

2.5 Deep Learning

Deep Learning, atau disebut juga *deep structured learning, hierarchical learning,* dan *deep machine learning* merupakan salah satu cabang dalam *machine learning* yang model komputasinya terdiri dari beberapa layer. *Deep learning* mampu mempelajari dan mengekstrak representasi data/fitur secara otomatis pada abtraksi tingkat tinggi (LeCun et al., 2015). Model tersebut memberikan hasil yang sangat baik dalam penelitan di berbagai bidang seperti *speech recognition*, *object detection*, *sequence labeling* dan lain sebagainya.

Struktur pembelajaran pada *deep learning* berbentuk hierarki karena termotivasi dari bagaimana neokorteks pada otak maunusia bekerja secara mendalam. Neokorteks tersebut melakukan proses pemelajaran berlayer dan secara otomatis

mampu mengketrak fitur dan melakukan abstraksi dari *resource* yang diberikan (Bengio et al., 2007). Struktur tersebut terdiri atas *input layer*, *hidden layer* dan *output layer*. *Input layer* memiliki fungsi sebagai tempat masuknya data yang akan dipelajari oleh model. *Hidden layer* melakukan aproksimasi fungsi untuk mendapatkan target dari data *training* yang diberikan. Disebut *hidden layer* karena pada layer ini, *output* tidak bisa kita lihat (Goodfellow et al., 2016). *Hidden layer* inilah yang menjadi *key role* dalam *deep learning*. Sedangkan *output layer* merupakan layer untuk mengembalikan target yang diinginkan.

Deep learning ini mampu memberikan model yanng memiliki performa sangat baik dalam *supervised learning* (Goodfellow et al., 2016). Dengan menambahkan lebih banyak layer dan unit di dalam layer, *deep network* dapat merepresentasikan fungsi dengan kompleksitas yang tinggi. Secara umum, *deep learning* memetakan *input vector* ke *output vector*. Walaupun hal ini mudah dilakukan oleh manusia secar manual, namun untuk *dataset* yang sangat besar, tentu hal ini tidak mungkin dilakukan. Ada banyak macam model *Deep Learning* yang sesuai dengan kebutuhan komputasi, seperti *Deep Belief Network* (Hinton et al., 2006), *Recurrent Neural Networks* (Elman, 1990), *Long Short Term Memory* (Hochreiter dan Schmidhuber, 1997), *Restricted Boltzman Machine* (Pennington et al., 2014) dan lain sebagainya.

2.6 Recurrent Neural Networks

Recurrent neural networks (RNNs) merupakan merupakan salah satu arsitektur *Deep Learning* yang memiliki koneksi siklik (Graves, 2012). RNNs memiliki *neuron* yang terkoneksi dengan *neuron* lain sehingga membentuk *loop* umpan balik (Haykin et al. (2009)), tidak seperti *feedforward neural network* (FNNs) dimana aliran informasi hanya berjalan searah. RNNs memungkinkan *output* yang dihasilkan akan menjadi *input* untuk menghasilkan *output* yang lain. Hal ini menyebabkan perilaku RNNs tidak hanya bergantung pada *input* saat ini saja, namun juga bergantung pada *output* sebelumya. Oleh karena itu, RNNs memiliki kemampuan yang sangat bagus sebagai model dalam permasalahan *sequence data* dibandingkan dengan FNNs. RNNs sendiri memiliki kemampuan yang sangat bagus dalam beberapa *task* terkait *sequence data*, seperti *language model* (Mikolov et al. (2010)) dan *speech recognition* (Graves et al. (2013)).

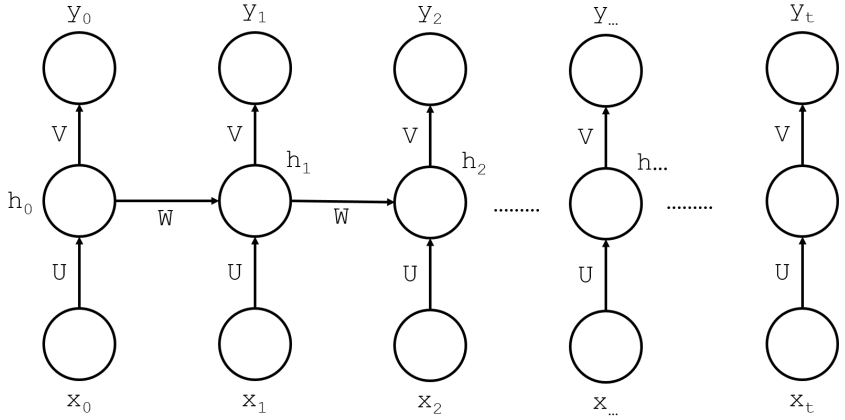
Dibandingkan dengan FNNs, RNNs memiliki beberapa kelebihan (Mikolov et al., 2010), yaitu:

1. Pada RNNs, kata-kata sebelumnya direpresentasikan dengan *recurrent*

connections, sehingga RNNs dapat menyimpan informasi kata sebelumnya dalam jumlah tak hingga. FNNs tidak bisa secara alami memodelkan hubungan kontekstual antara sebuah kata dengan kata-kata pada posisi sebelumnya dan representasi kata sebelumnya berupa konteks dari $n-1$ kata. Oleh karena itu, FNNs terbatas dalam penyimpanan informasi kata sebelumnya terbatas seperti pada model *n-gram*.

2. RNNs dapat melakukan kompresi keseluruhan riwayat kata menjadi ruang dimensi yang lebih kecil, sedangkan FNNs melakukan kompresi/proyeksi hanya dengan sebuah kata saja.

Banyak variasi RNNs yang telah diusulkan oleh beberapa peneliti, seperti Elman *networks* (Elman, 1990), Jordan *networks* (Jordan, 1986), *time delay neural networks* (Lang et al., 1990) dll. Gambar berikut merupakan contoh dari RNNs secara umum



Gambar 2.2: *Recurrent Neural Networks* sederhana

Dari gambar 2.2, sebuah jaringan pada RNNs memiliki 3 layer pada setiap *timestep*, yaitu *input layer*, *hidden layer* dan *output layer*. *Input layer* merupakan layer sebagai tempat masuk *resource*. Di dalam *hidden layer* tersebut terdapat beberapa unit untuk menyimpan informasi dari *timestep* sebelumnya. Sedangkan pada *output layer* merupakan layer yang memberikan *output* dari model. Pada setiap *timestep t*, RNNs di atas memiliki sebuah *input layer* $x(\vec{t}) \in \mathbb{R}^{N}$, *hidden layer* $h(\vec{t}) \in \mathbb{R}^{H}$, dan *output layer* $y(\vec{t}) \in \mathbb{R}^{M}$. Nilai N, H, dan M merupakan panjang vektor *input*, jumlah unit di dalam *hidden layer* tersebut, dan panjang vektor *output* yang diinginkan. Terdapat tiga parameter yang akan diestimasi, yaitu $U \in \mathbb{R}^{H \times N}$, $V \in \mathbb{R}^{M \times H}$, dan $W \in \mathbb{R}^{H \times H}$. Tiga parameter tersebut bersifat *shared*, yang artinya masing-masing *timestep* menggunakaan dan mengestimasi tiga parameter tersebut.

Apabila tiga parameter di atas sudah diketahui, $h(\vec{t})$ dan $y(\vec{t})$ dapat dihitung dengan persamaan:

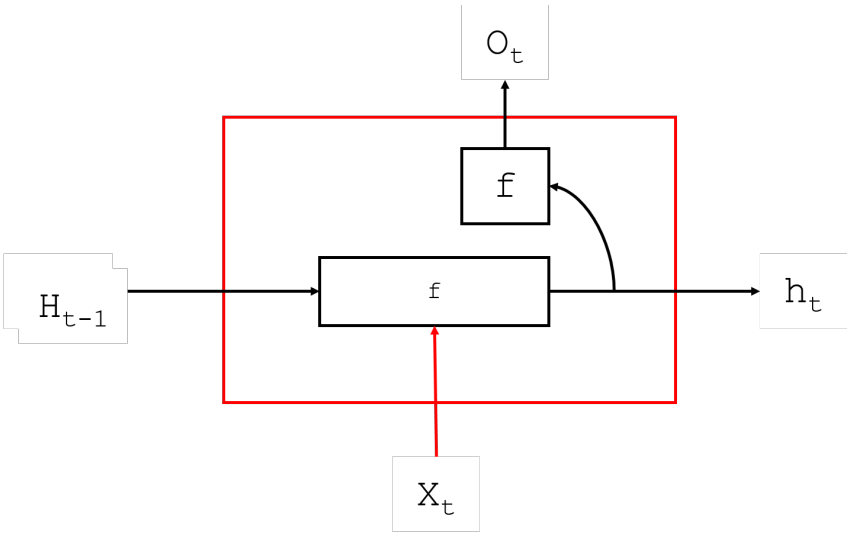
$$y(\vec{t}) = f(V \cdot (\vec{t})) \tag{2.1}$$

$$h(\vec{t}) = f(U \cdot x(\vec{t}) + W \cdot h(\vec{t-1})) \tag{2.2}$$

dimana

$$h(\vec{0}) = f(U \cdot x(\vec{0})) \tag{2.3}$$

dengan f sebagai *activation function*, misalnya $tanh$ atau $softmax$. Untuk lebih jelasnya, berikut merupakan gambar dari satu buah *timestep* di dalam RNNs.

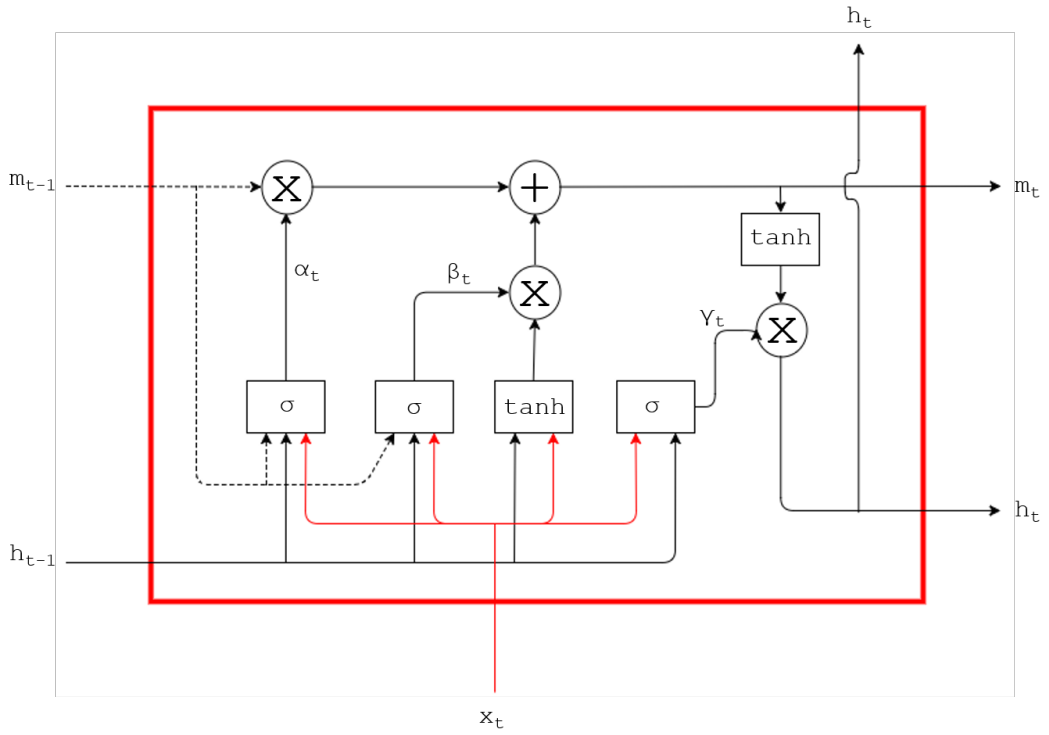


Gambar 2.3: 1 buah *timestep* dalam RNNs

2.6.1 Long Short Term Memories (LSTMs)

Pada penjelasan di atas, RNNs sederhana memiliki kelebihan mempertimbangkan konteks untuk mengolah *input* menjadi *output*. Sayangnya, *range* konteks yang dapat digunakan dalam satu blok terbatas (Graves, 2012). Efek dari keterbatasan ini yaitu informasi pada suatu blok akan hilang atau terganggu dalam perjalanan *timestep* sehingga *output* yang dihasilkan tidak sesuai harapan. Oleh karena itu RNNs sederhana tidak dapat menangani permasalahan dependensi jangka panjang. Permasalahan ini disebut dengan *vanishing gradient problem* (Hochreiter (1991); Hochreiter et al. (2001); Bengio et al. (1994)). Banyak upaya untuk mengatasi masalah ini, seperti dengan menggunakan *simulated annealing* dan *discrete error propagation* (Bengio et al., 1994), menggunakan *time delays* (Lang et al. (1990); Bakker (2001)) atau *time constant* (Mozer et al., 1997), dan *hierarchical sequence compression* (Schmidhuber et al., 2007). Namun sejauh ini solusi yang paling

bagus yaitu dengan arsitektur *Long Short Term Memory* (LSTM) (Hochreiter dan Schmidhuber, 1997).



Gambar 2.4: 1 buah blok memori dalam LSTM

LSTMs diperkenalkan oleh Hochreiter dan Schmidhuber (1997) dan saat ini banyak digunakan dalam berbagai *task*. Gambar 2.4 merupakan ilustrasi satu buah blok memori di dalam LSTMs. Pada dasarnya, arsitektur LSTMs mirip dengan RNNs sederhana, namun unit *nonlinear* pada *hidden layer* di dalam RNNs sederhana diganti menjadi blok memori. Sebuah blok memori memiliki gerbang *multiplicative* yang berfungsi untuk menyimpan dan mengakses informasi dari blok sebelumnya namun dengan batasan yang jauh lebih besar dibanding RNNs, sehingga mampu menghindari *vanishing gradient problem*. Apabila *input gate* selalu tertutup, maka memori tidak akan perah ditimpa sehingga isi memori tidak berubah.

Pada gambar 2.4, kita dapat melihat bahwa 1 blok memori pada LSTMs tersebut memiliki 3 buah gerbang, yang berfungsi untuk sebagai pengatur suatu informasi apakah ditambahkan, dipertahankan atau dihapus di dalam sebuah sel. Masing-masing gerbang terdiri dari komponen *sigmoid layer* dan komponen untuk melakukan operasi penjumlahan atau perkalian untuk masing-masing *element-wise*. *Sigmoid layer* tersebut memiliki nilai antara nol sampai dengan satu, yang mendeskripsikan perilaku gerbang dalam menerima *input*. Semakin kecil nilai dari

layer tersebut maka semakin kecil pula informasi masuk ke gerbang terkait dan sebaliknya.

1. *Forget Gate*
   Gerbang ini memiliki fungsi untuk menentukan informasi yang akan disimpan di dalam memori dengan formula berikut

$$\alpha_t = \sigma(W_{x\alpha} \cdot x_t + W_{h\alpha} \cdot \ h_{t-1} + W_{m\alpha} \cdot \ m_{t-1}) \tag{2.4}$$

2. *Input Gate*
   Gerbang ini berfungsi untuk menentukan apakah informasi baru $x(t)$ akan disimpan dalam *cell state* atau tidak.

$$\beta_t = \sigma(W_{x\beta} \cdot x_t + W_{h\beta} \cdot \ h_{t-1} + W_{m\beta} \cdot \ m_{t-1}) \tag{2.5}$$

3. *Output Gate*
   Gerbang ini berfungsi untuk menendukan *output* dari sebuah *timestep* berdasarkan *cell state* saat ini.

$$\gamma_t = \sigma(W_{x\gamma} \cdot x_t + W_{h\gamma} \cdot \ h_{t-1} + W_{m\gamma} \cdot \ m_{t-1}) \tag{2.6}$$

Dalam setiap *timestep* t, berikut merupakan formula untuk menghitung $m(t)$ dan $h(t)$:

$$m_t = \alpha_t(\times)m_{t-1} + \beta_t(\times)f(x_t, t-1) \tag{2.7}$$

$$h_t = \gamma_t(\times)tanh(m_t) \tag{2.8}$$

dimana

$$f(x_t, t-1) = tanh(W_{xm} \cdot x_t + W_{hm} \cdot h_{t-1}) \tag{2.9}$$

Notasi $(\times)$ merupakan operasi perkalian untuk setiap pasang elemen, dan $(+)$ merupakan operasi penjumlahan setiap pasang elemen.

2.6.2 Penerapan RNNs untuk MER

Terdapat beberapa penelitian terkait MER yang dikembangkan menggunakan RNNs, seperti *drug entity recognition* (Mujiono et al., 2016), *medical event detection on EHR* (Jagannatha dan Yu, 2016), *biomedical entity recognition* (Limsopatham dan Collier, 2016), dan *Named Entity Recognition in Swedish Health*

Records (Almgren et al., 2016). Penelitian *drug entity recognition* oleh Mujiono et al. (2016) sudah dijelaskan pada subbab 2.4.

Dalam penelitiannya, Jagannatha dan Yu (2016) menggunakan LSTMs untuk memprediksi label entitasnya. Penelitian tersebut bertujuan untuk mendeteksi kejadian medis pada *Electronic Health Records* seperti *medication, diagnosis (Indication), adverse drug events (ADEs) severity, other SSD, frequency, drugname* dan *duration*. Sebagai pembanding, penulis tersebut juga mengimplementasikan CRF dan GRU. Ada beberapa kesulitan yang dihadapi dalam mengolah EHR tersebut, yaitu EHR lebih *noisy* dibandingkan dengan teks biasa, banyak kalimat yang tidak komplet dan penggunaan frasa. Hasil dari penelitian tersebut menunjukkan bahwa semua model RNNs (LSTMs dan GRU) memiliki akurasi yang lebih baik daripada CRF. Apabila dibandingkan dengan *baseline* yang digunakan, GRU mampu meningkatkan *recall* (0.8126), *precision* (0.7938) dan *F-score* (0.8031) sebesar 19%, 2% dan 11% dari *baseline*.

Limsopatham dan Collier (2016) menggunakan *Bidirectional-LSTMs* untuk mengidentifikasi kalimat dengan menggunakan karakter dan kata yang diubah menjadi vektor menggunakan *word embedding*. Untuk setiap kalimatnya, peneliti tersebut mengusulkan adanya *ortographic feature* supaya modelnya dapat mempelajari fitur tersebut secara eksplisit. Evaluasi yang digunakan menggunakan tiga buah koleksi *biomedical test*, yaitu *Gene Mention task corpus*, *BioNLP 2009* dan *NCBI disease corpus*, dengan perhitungan *F1-score*. ada empat *baseline* yang digunakan sebagai pembanding, yaitu *feedforward*, *bidirectional-LSTM*, *CNN-Bidirectional-LSTM* yang hanya menggunakan karakter dan *CNN-Bidirectional LSTM*. Hasil yang didapatkan mengatakan bahwa penggunaan *Bidirectional-LSTM* yang dikombinasikan dengan CNN dengan diberikan *word embedding* dan *orthographic* merupakan model yang paling bagus. Penulis tersebut juga menyimpulkan bahwa penggunaan fitur *hand-crafted* tersebut mampu memberikan akurasi yang lebih tinggi.

Almgren et al. (2016) menggunakan *deep bidirectional LSTM* dalam mengembangkan NER di bidang medis. Entitas yang akan diidentifikasi adalah *disorders and findings*, *pharmaceutical drugs*, *body structure* dan *non-entity term*. Model menggunakan teks medis berbahasa Swedia sebagai *dataset*, di-*train* dengan menggunakan *end-to-end backpropagation* dan Adam *optimizer*, dan *input* yang diberikan berbentuk urutan karakter. Hasil yang didapatkan adalah Char-BiLSTM pada Stockholm EPR corpus mendapatkan *precision* 0.67, *recall* 0.12 dan *f-measure* 0.20 meningkat 60% dibandingkan dengan *baseline*.

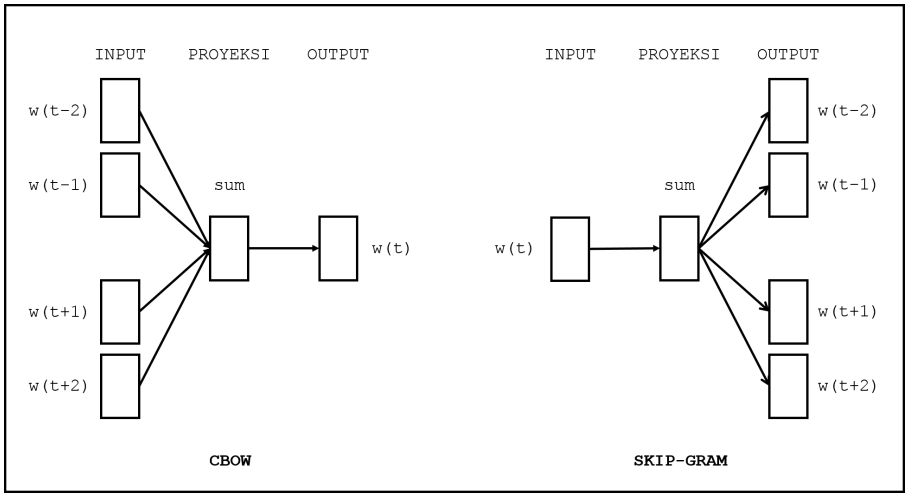
2.7 Word Embedding

Pada umumnya, pendekatan yang digunakan untuk merepresentasikan sebuah kata sebagai *input* model adalah dengan menggunakan *one-hot-vetor* (Turian et al., 2010). Panjang dari sebuah vektor kata ini bergantung dari banyaknya kata unik di dalam sebuah korpus. Ada beberapa cara untuk mengubahnya menjadi *one-hot-vector*, seperti mengumpulkan semua kata unik kemudian mngurutkannya secara alfabetis. Vektor *one-hot* tersebut bernilai 1 pada indeks kata yang bersesuaian. Misalnya kata "obat" berada di indeks ke 25 pada kumpulan kata unik, maka representasi vektornya elemen ke 21 di vektor "obat" adalah 1 sedangkan yang lainnya 0.

Dari ilustrasi singkat tersebut, representasi *one-hot-vector* memiliki kelemahan yaitu besar vektor yang tergantung jumlah kata unik di dalam korpus. Selain itu, jika terdapat sebuah kata yang muncul di korpus namun tidak muncul di *training* ataupun *testing data*, kata tersebut tidak dapat diproses. Selain itu, sangat susah untuk mencari hubungan baik sintak maupun semantik dari representasi kata ini, karena antar kata hanya dibedakan indeks yang berisi angka 1 saja.

Dari kelemahan di atas, terdapat sebuah representasi vektor lain dari kata yang lebih baik, yaitu dengan menggunakan *word embedding*. *Word embedding* adalah salah satu jenis dari representasi kata yang memiliki kelebihan yaitu padat, berdimensi rendah, dan memiliki nilai yang real. *Word embedding* memetakan kata dengan vektor berisi bilangan *real*, misalkan W("obat") = [0.4, -0.9, 0.1, ..., 0.9], dimana W adalah fungsi yang memetakan suatu kata menuju representasi vektor dan W("obat") merupakan *word embedding* dari kata "obat". *Word embedding* dapat meningkatkan performa dari *tasks* dalam NLP dengan cara mengelompokkan kata-kata yang mirip, karena kata yang mirip memiliki vektor yang mirip pula. Ada beberapa metode *word embedding* yang banyak digunakan dalam beberapa *task* di NLP, seperti Glove (Pennington et al., 2014) dan Word2Vec (Mikolov et al., 2014). Pada pembahasan ini, we hanya menuliskan mengenai Word2Vec.

Word2Vec merupakan model linguistik yang dikembangkan oleh Mikolov et al. (2014) dan berdasarkan pada *neural networks*. Word2Vec mempelajari *embedding* dari setiap kata untuk dipetakan ke masing-masing vektor yang berdimensi rendah dari sifat distribusinya pada korpus yang diberikan. Dari situ, Word2Vec mampu mengelompokkan kata berdasarkan kemiripannya di dalam *vector space*.



Gambar 2.5: Arsitektur Word2Vec

Ada dua arsitektur Word2Vec yang dikembangkan oleh Mikolov et al. (2014), yaitu arsitektur *skip-gram* dan arsitektur *continuous bag-of-words* (CBOW). Dari gambar 2.5, dapat dilihat bahwa arsitektur CBOW memprediksi masing-masing kata berdasarkan kata di sekelilingnya. *Input layer* dalam arsitektur ini direpresentasikan dengan *bag-of-words*. CBOW sendiri dapat mempelajari data dengan ukuran yang sangat besar yang tidak dapat dilakukan oleh model *neural network* yang lain. Sedangkan arsitektur *skip-gram* memprediksi kata-kata di sekeliling dan konteksnya berdasarkan sebuah kata yang diberikan (gambar 2.5). *Skip-gram* mampu menangkap *co-occurance* rata-rata dari dua buah kata di dalam *training set*.

BAB 3
METHODOLOGY

In this chapter, we describe the methodology used in this research. It consists of data gathering, data pre-processing, data annotation, experiment, and evaluation. Before we describe each part in details, we explain the big picture of the methodology in a form of pipeline.

3.1 Pipeline

The goal of this research is to build a model that predicts the semantic roles of each Indonesian sentence. In other languages such as English, there are already semantic role corpus from which the SRL system is built. Unfortunately, that is not the case for Indonesian, since there is no annotated corpus available yet. We therefore create our own with the annotation guideline crafted for Indonesian conversational language. In this research, we focus on building SRL system for the conversational Indonesian language.

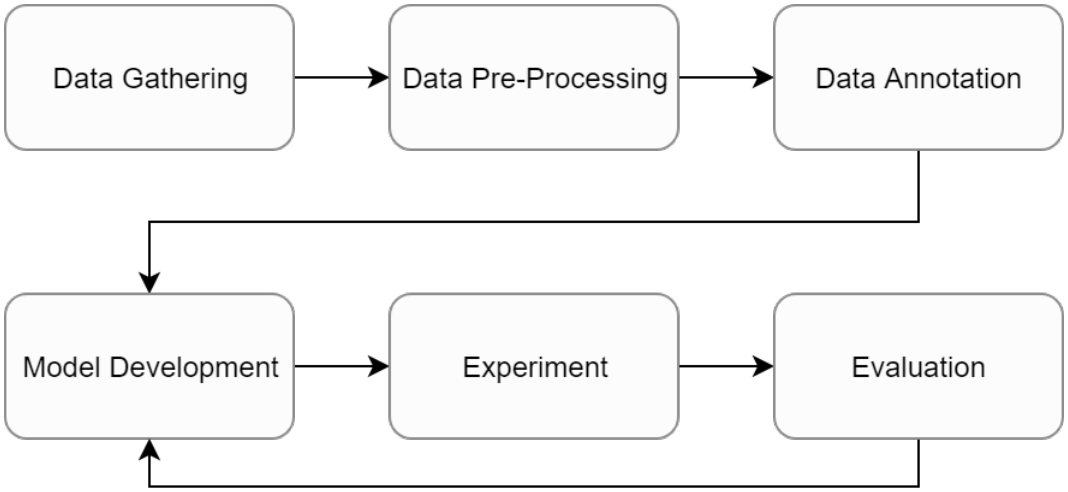
We view SRL as a sequence labeling problem. Suppose that we have an input of n words $w = (w_1, w_2, ..., w_n)$, the goal is to find the best label sequence $y = (y_1, y_2, ..., y_n)$, with y_i representing the semantic roles. The probabilities of the label in each time step i is described as follows.

$$P(y_i | w_{i-l}, ..., w_{i+l}, y_{i-l}, ..., y_{i+l}) \tag{3.1}$$

whereby l is a small number.

In this section, we explain our research methodology including the data annotation, features used, and the proposed model architecture. Figure 3.1 shows the pipeline of this research.

This research uses the data from one of Kata.ai's chat bots. Firstly, the data is pre-processed before going into the next steps. After that, the data is then annotated with semantic roles based on the annotation guideline proposed by us for Indonesian conversational language.



Gambar 3.1: Methodology Pipeline

There are 2 main scenarios for the experiment. The first scenario aims for finding the best set of features that output the best performance. The goal of the second scenario is to find which deep learning model architecture has the best result. In the first scenario, we have three features, which are word embedding, POS tag, and neighboring word embeddings. In the second scenario, we compare four model architectures, namely vanilla Long Short-Term Memories (LSTM), Bi-Directional LSTM (BLSTM), CNN-BLSTM, and Context-Aware BLSTM (CA-BLSTM).

We use 5-fold cross validation for every experiment. Each experiment is evaluated based on precision, recall, and F1 of each semantic roles with partial match approach. The performance of a model is retrieved by averaging precision, refall, and F1 of every semantic role. We then analyze and explain the results of each experiment scenario.

3.2 Data Gathering

In this research we use real-world data from one of Kata.ai chat bots. We firstly retrieved data consisting of 40.000 instances of text chats. We then manually deleted junk chats which contain, for example, only laugh or greeting. After that, we run a script to delete duplicate chats. The deletion process outputs a clean data with 30.000 instances in total. Finally, we randomly selected 9.000 out of 30.000 instances as the data to be annotated. This data set will be the one which is trained and tested to build the SRL system.

It is worth to note that conversational language has unique characteristics. First, they use slangs and abbreviations. For example, one might use *"u"* instead of *"you"* in *"I brought u a present"*. The grammars are often unstructured and thus, one

cannot rely on syntactic parsers to build SRL system for conversational language. The sentences are also filled with interjections such as *"haha"* and *"lol"*. Lastly, since conversational sentences are really short, averaging around 5-7 words per sentence, it sometimes contains incomplete information. These are the interesting challenges the SRL system should learn and tackle.

3.3 Data Pre-Processing

After the data has been gathered, the next step is to pre-process the data so that it could be fed into the machine learning model later. In this step, each sentence is going through a process called tokenization. Tokenization splits sentence into its individual words. Traditionally, one can split sentence by *space*, however, it does not work for sentences in conversational language since number or symbol are often concatenated with words, such as "makan2". It thus needs further tokenization technique, that is, splitting alphabet with number and symbol tokens. This way, "makan2" will be tokenized as "makan 2". In addition to tokenization by *space*, the rules are listed as follows:

1. <alphabet><numeric> to be <alphabet><space><numeric>
2. <numeric><alphabet> to be <numeric><space><alphabet>
3. <alphabet><symbol> to be <alphabet><space><symbol>
4. <symbol><alphabet> to be <symbol><space> <alphabet>

3.4 Data Annotation

In this work, we create a new set of semantic roles mainly crafted for informal conversational language. The summary of semantic roles proposed with its examples are presented in Table 3.1.

Tabel 3.1: Set of Semantic Roles for Conversational Language

Semantic Roles	*Example*
AGENT	Aku beliin kamu kado
PATIENT	Aku beliin kamu kado
BENEFICIARY	Aku beliin kamu kado
GREET	Hai Budi! Aku beliin kamu kado
MODAL	Aku bisa makan di rumah besok
LOCATION	Aku bisa makan di rumah besok
TIME	Aku bisa makan di rumah besok

These semantic roles are mainly inspired by the work of Saeed (1997), except that AGENT and PATIENT refer to PROTO-AGENT and PROTO-PATIENT as explained by Dowty (1991). The main difference of this set of semantic roles compared to the previous ones is GREET.

The center of all semantic roles is the PREDICATE. As in English, PREDICATEin Indonesian is usually in a form of *verb*, as illustrated by the examples below:

- "Kemarin aku makan di rumah"
- "Aku ada ujian nih hari Senin"

In Indonesian, however, predicate can also be an adjective. Some examples are presented as follows:

- "Kamu cantik deh"
- "Aku lagi sedih nih"

In this section, we briefly explain each semantic roles with their respective examples.

1. AGENT

   An entity is called as AGENT if one of this following properties is fulfilled:

   Volitional involvement in event or state

   Sentience (and/or perception)

   Causing an event or change of state in another participant

   Movement (relative to position of another participant) (exists independently of event named)

   The examples of AGENT in a sentence are given as follows:

- "Aku makan ayam dulu ya"
- "Kamu gak tidur?"
- "Kamu mau beliin aku pulsa?"

2. PATIENT
   An entity is called as PATIENT if one of this following properties is fulfilled:

   Volitional involvement in event or state

   Sentience (and/or perception)

   Causing an event or change of state in another participant

   Movement (relative to position of another participant) (exists independently of event named)

   The examples of PATIENT in a sentence are given as follows:

   - "Aku makan ayam dulu ya"
   - "Kamu mau beliin aku pulsa?"
   - "Aku lagi sedih nih.."

3. BENEFICIARY
   BENEFICIARY is an entity which gets benefit of the predicate. It is usually in a form of indirect object. The examples of BENEFICIARY in a sentence are given as follows:

   - "Kamu mau beliin aku pulsa?"
   - "Aku pengen ngobrol sama kamu"

4. GREET
   GREET is the main difference of this set of semantic roles for conversational language. GREET refers to an animate object, usually a person, which is being greeted in a chat. In conversational language, one often calls the name of person it is talking to. This information is useful, for instance, we can derive that *"you"* refers to *"Budi"* in *"Halo Budi! Aku beliin kamu kado loh"*. The examples of GREET in a sentence are given as follows:

   - "Hai rizky! kamu udah makan belum?"
   - "aku ga bisa tidur nih Val"

5. MODAL
   MODAL refers to *modal verb* of a predicate. The word examples are *"boleh, harus, pernah, sudah, udah, mesti, perlu, akan, lagi, bisa, mau, ingin, pengen, pingin"*. The examples of MODAL in a sentence are given as follows:
   - "Aku mau makan dulu ya!"
   - "Kamu udah tidur belum?"
6. LOCATION
   LOCATION refers to the location of a predicate. The examples of LOCATION in a sentence are given as follows:
   - "Aku mau makan di rumah ya!"
   - "Kamu gak pergi ke sekolah?"
7. TIME
   TIME refers to the time of a predicate. The examples of TIME in a sentence are given as follows:
   - "Kemarin aku makan di rumah"
   - "Aku ada ujian nih hari Senin"

Following Collobert et al., all the labels are tagged using BIO (Begin Inside Outside) tagging Collobert et al. (2011). Suppose that a label PATIENTconsists of more than one word, such as *"ayam goreng"* in "Aku makan *ayam goreng*", "ayam" and "goreng" are tagged as "B-Patient" and "I-Patient", respectively. If the label has only one word, than it is tagged as "B-Patient". Word that does not have any label is thus tagged as "O" which means "Others".

After the data has been labeled, the labels need to be encoded in a way the deep learning model understands. To do so, the labels are then transformed into *one-hot-vector*. Each label is mapped into a unique one-hot-vector, hence the relation is 1-to-1.

Tabel 3.2: Set of Semantic Roles for Conversational Language

Sentence	Aku	pengen	makan	ayam
BIO-Label	B-AGEN	B-MD	B-PRED	B-PATIENT
One-Hot-Vector	[1, 0, 0, .., 0]	[0, 1, 0, .., 0]	[0, 0, 1, .., 0]	[0, .., 1, 0, 0]

Table 3.2 shows an example of how sentence is labeled with the one-hot vectors representations of BIO format.

3.5 Model Development

In this section, the features and model architecture are described. We firstly describe word embedding and POS Tag as the feature candidates. Following after that, the four model architectures, one of which is the new architecture we propose, are explained.

3.5.1 Feature Extraction

In this step, we extract features from the data that has been annotated with semantic roles. We propose three features which will be combined later to find the best feature selection that outputs the best result. Those features are word embedding, POS-Tag, and neighboring word embeddings.

3.5.1.1 Word Embedding

Word embedding represents word as a vector. Word embedding has proved to be one of the most contributing features by a lot of deep learning research, such as for SRL system proposed by Zhou dan Xu (2015) and Collobert et al. (2011). The interesting characteristic of word embedding is that similar words have proved to have similar vectors. This is very important when dealing with conversational language which has a lot of slang words. For instance, pronoun "Aku" will have similar vector with its slang form, "Gue". We believe that this feature will contribute greatly to the model performance. Therefore, we utilized our embedding as one of our feature candidates.

In order to utilize word embedding as our features, we conduct three steps which consist of: 1.) data gathering for building word embedding model, 2.) training the word embedding model, and 3.) converting words into vectors with the trained word embedding model.

1. Data gathering for building word embedding model
   We first gather an unlabeled dataset in order to build the word embedding model. The dataset is retrieved from 1.300.000 sentences of chats from Kata.ai.

2. Training the word embedding model
   The word embedding model is trained using the afore-mentioned unlabeled data set. This model is used to transform words into their respective vector representations.

3. Converting words into vectors with the trained word embedding model
   The trained word embedding model is then used to convert words into vectors. The example of this conversion can be seen in Table 3.3

Tabel 3.3: An example of word embedding vector representation with dimension of 3

Sentence	Aku	pengen	makan	ayam
Word Vector	[0.2, -0.4, 0.9]	[0.7, 0.1, 0.2]	[0.1, 0.6, -0.5]	[0.9, 0.1, 0.8]

In Table 3.3 provides an example assuming that the vector dimension is 3. This means that every word is mapped into a vector with a length of 3.

3.5.1.2 Part-of-Speech Tag (POS-Tag)

POS Tag is a feature to represent the class of each word. In this research, we use POS Tag annotation which is mainly inspired by the work of Ruli Manurung (20XX). The POS Tags that we use are: Verb (V), Noun (NN), Adjective (ADJ), Adverb (ADV), Coordinative Conjunction (CC), Subordinative Conjunction (SC), Interjection (INTJ), Question (WH), Preposition (PREP), and Negation (NEG). Before feeding the feature to the deep learning model, we encode the POS tag features as a one-hot vector. Each one-hot vector represents each POS tag uniquely.

The example of POS Tag features is presented in Table 3.4

Tabel 3.4: An example of POS Tag feature and its respective one-hot-vector

Sentence	Aku	pengen	makan	ayam
POS Tag	NN	ADV	V	NN
One-Hot-Vector	[1, 0, 0, .., 0]	[0, 1, 0, .., 0]	[0, 0, 1, .., 0]	[0, .., 1, 0, 0]

While most of the deep learning research aims for not using such feature, we argue that POS tag is still important for building a robust model in our case. This is because of the fact that size of our corpus is relatively small compared to the huge English-based SRL corpus such as ConLL 2015 by Carreras dan Màrquez (2005). We argue that having Verb as one of our POS Tag will help to determine which one is the predicate, since predicate is usually in a form of verb, though some can also be adjectives. As the arguments having semantic role are mostly in a form of Noun, POS Tag Noun will be a helpful information as well. Other POS tags will help to determine which word that obviously does not have any semantic role.

In this work, we use gold-standard POS tag on our data as the features to prevent propagation errors from the POS tag model. This way, we can focus on analyzing errors resulting from the SRL model later in chapter 5.

3.5.1.3 Neighboring Word Embeddings

Neighboring word embeddings are the vector representations of words located before and after the word being processed. We use specifically one word before and after the word being processed. Suppose that we are processing the word w_t at time step t, the neighboring words embeddings are the vector representations of the word w_{t-1} and w_{t+1}. We argue that this feature can be useful for capturing the context of the word by looking at the surrounding words. Suppose that the machine is processing the word *"rumah"* in *"aku tidur di rumah"*. By looking at the previous word, which is the preposition *"di"*, it gives a hint that *"rumah"* might have the semantic role LOCATION.

Tabel 3.5: An example of neighboring word embedding vectors of every time step

Sentence	Aku	pengen	makan	ayam
Word Vector	[0.2, -0.4, 0.9]	[0.7, 0.1, 0.2]	[0.1, 0.6, -0.5]	[0.9, 0.1, 0.8]
Neighbor Vector	[0.0, 0.0, 0.0]	[0.2, -0.4, 0.9]	[0.7, 0.1, 0.2]	[0.1, 0.6, -0.5]
	[0.7, 0.1, 0.2]	[0.1, 0.6, -0.5]	[0.9, 0.1, 0.8]	[0.0, 0.0, 0.0]

Table 3.5 illustrates the neighboring word embeddings of every time step. The table shows that in every time step, it adds the word vector information from the left and right. Since the first word does not have any previous word, its left neighbor is a vector $\vec{0}$. This is also the case for the last word which does not have any subsequent word. Hence, the right neighbor of the last word is also a vector $\vec{0}$.

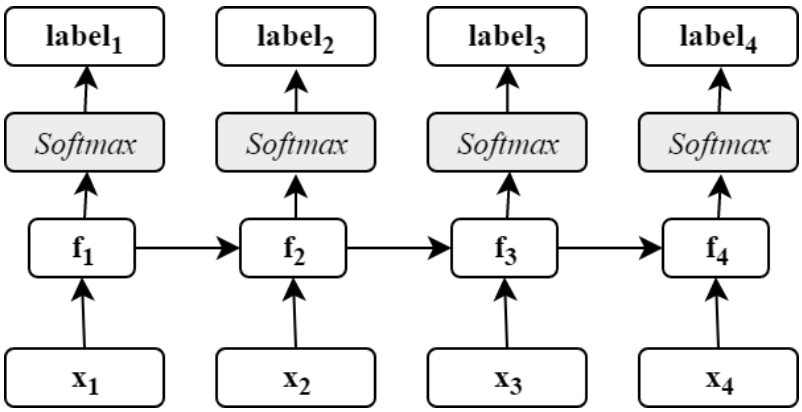
3.5.2 Model Architecture

Recurrent Neural Networks (RNN) has a nature advantage for solving sequence labeling problem Zhou dan Xu (2015). Hochreiter dan Schmidhuber (1997) proposed Long Short-Term Memories (LSTM) as the specific version of RNN designed to overcome vanishing and exploding gradient problem. In this research, we experiment on various LSTM architectures, namely vanilla Long Short-Term Memories (LSTM), Bi-Directional LSTM (BLSTM), Convolutional Neural Network BLSTM (CNN-BLSTM), and Context-Aware BLSTM (CA-BLSTM). Each architecture is built incrementally upon the previous architectures, starting

from the vanilla LSTM. The last architecture, CA-BLSTM, is the one we propose in this work.

3.5.2.1 Vanilla LSTM

Vanilla LSTM is the basic, one-directional LSTM networks designed to overcome vanishing and exploding gradient problem found in RNN. To do so, it has forget gates, a gate to open or close incoming information from the previous time steps. We firstly explain the LSTM networks as the big picture for solving sequence labeling problem, followed by the details of every LSTM unit.



Gambar 3.2: An architecture of Context-Aware Bi-Directional Long Short Term Memories with total time step of 4

Figure 3.2 illustrates the big picture of the vanilla LSTM networks used in this research. Suppose that we have sequence of input tensors $[\mathbf{x_1}; \mathbf{x_2}; ...; \mathbf{x_n}]$, with n denotes the number of time steps. Each input tensor $\mathbf{x_t}$ represents features of the word in time step t. For every time step t, each input tensor $\mathbf{x_t}$ is fed into the LSTM layer, resulting tensor $\mathbf{f_t}$, as shown in Equation 3.2.

$$\mathbf{f_t} = LSTM(\mathbf{x_t}) \tag{3.2}$$

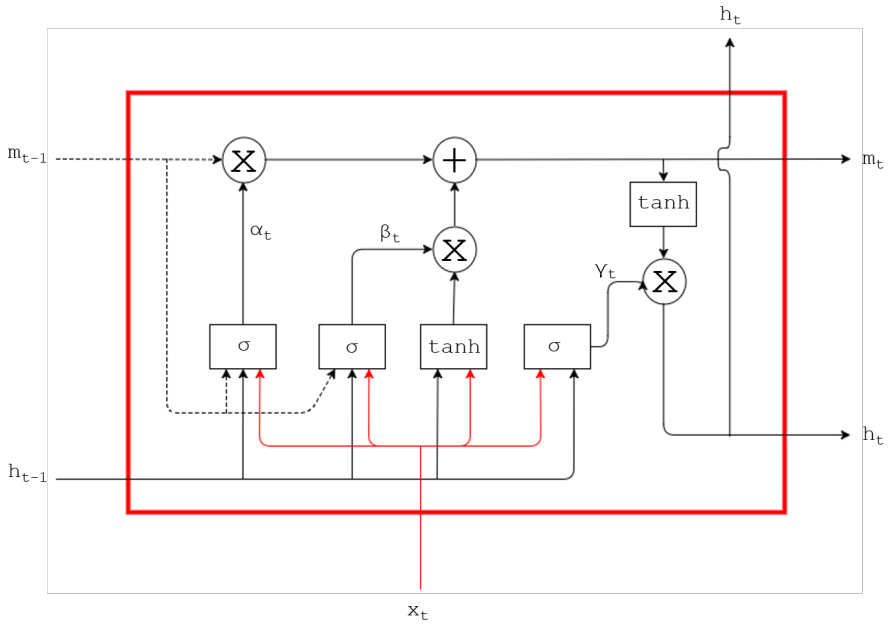
Equation 3.3 shows that each LSTM output from every time step is then processed by the time-distributed softmax layer to produce the probabilities of every possible label.

$$label_t = Softmax(\mathbf{f_t}) \tag{3.3}$$

In each time step, label with highest probability among others will be the final output. This way, we have all the predicted labels for every time step.

After explaining the big picture of LSTM networks, we describe the details of

every LSTM unit in time step t shown in Equation 3.2. That is, given an input tensor $\mathbf{x_t}$, each LSTM unit will output tensor $\mathbf{f_t}$. Figure 3.3 illustrates the architecture of one LSTM unit in time step t.



Gambar 3.3: An LSTM unit in time step t

The LSTM unit in time step t requires an input tensor a_t. As explained in subchapter X, the equations to produce output tensor $\mathbf{f_t}$ are presented as follows:

$$\mathbf{m_t} = \alpha_t \cdot \mathbf{m_{t-1}} + \beta_t \cdot tanh(W_{xm} \cdot \mathbf{x_t} + W_{fm} \cdot \mathbf{f_{t-1}}) \tag{3.4}$$

$$\mathbf{f_t} = \gamma_t \cdot tanh(\mathbf{m_t}) \tag{3.5}$$

where α_t, β_t, and γ_t are the gates:

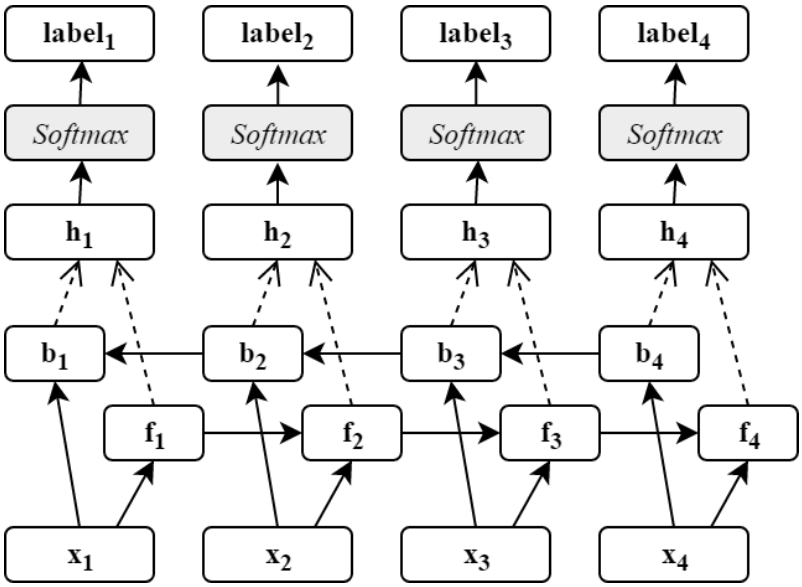
1. *Forget gates*: $\alpha_t = \sigma(W_{x\alpha} \cdot \mathbf{x_t} + W_{f\alpha} \cdot \mathbf{f_{t-1}} + W_{m\alpha} \cdot \mathbf{m_{t-1}})$
2. *Input gates*: $\beta_t = \sigma(W_{x\beta} \cdot \mathbf{x_t} + W_{f\beta} \cdot \mathbf{f_{t-1}} + W_{m\beta} \cdot \mathbf{m_{t-1}})$
3. *Output gates*: $\gamma_t = \sigma(W_{x\gamma} \cdot \mathbf{x_t} + W_{f\gamma} \cdot \mathbf{f_{t-1}} + W_{m\gamma} \cdot \mathbf{m_{t-1}})$

It is worth noting that the LSTM layer is recursive, meaning that one of the inputs comes from the output of previous time step. This way, the result of each time step also depends on the previous ones.

3.5.2.2 Bi-Directional LSTM (BLSTM)

Bi-Directional LSTM (BLSTM) is a modification of LSTM networks. While vanilla LSTM only goes one direction, BLSTM goes both ways in order to capture context

information from the past and future, as explained by Zhou dan Xu (2015). This characteristic is suitable for SRL task, since information from the right (future) might be useful to determine the label of current time step, as explained by Zhou dan Xu (2015). The idea is to have two LSTM layers, one for going forward and another for going backward, as illustrated in Figure 3.4



Gambar 3.4: An architecture of Context-Aware Bi-Directional Long Short Term Memories with total time step of 4

Figure 3.4 shows that the input $\mathbf{x_t}$ in every time step t is fed into two LSTM layers, the first one for going forward and the second one for going backward. These are illustrated in Equation 3.6 and Equation 3.7

$$\mathbf{f_t} = ForwardLSTM(\mathbf{x_t}) \tag{3.6}$$

$$\mathbf{b_t} = BackwardLSTM(\mathbf{x_t}) \tag{3.7}$$

In each time step, the result tensors $mathbff_t$ and $mathbfb_t$ are then concatenated to be one tensor output $\mathbf{h_t}$, as shown in Equation 3.8

$$\mathbf{h_t} = Concatenate(\mathbf{f_t}, \mathbf{g_t}) \tag{3.8}$$

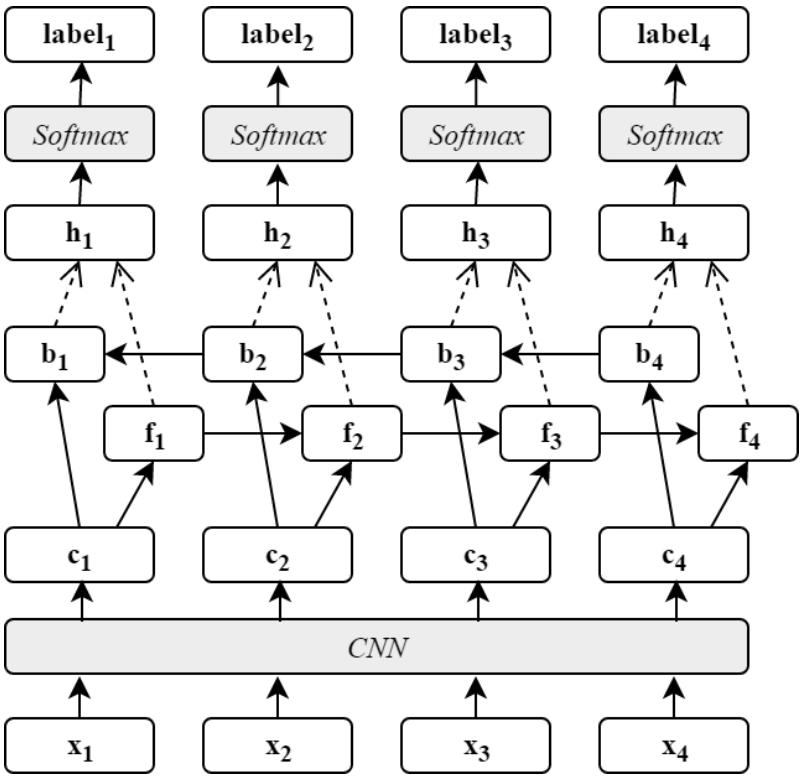
Likewise in vanilla LSTM architecture, the output tensor $\mathbf{h_t}$ is then fed into softmax layer, as shown in Equation 3.9

$$label_t = Softmax(\mathbf{h_t}) \tag{3.9}$$

After which, the output of the softmax layer will determine the final label for each time step.

3.5.2.3 CNN-BLSTM

In addition to the BLSTM architecture, we also experiment on adding Convolutional Neural Networks (CNN) layer underneath the BLSTM layer. The rationale is to capture raw context from the neighboring input tensors. This way, CNN can implicitly extract meaningful context information. Figure 3.5 illustrates the big picture of the CNN-BLSTM architecture.



Gambar 3.5: An architecture of Context-Aware Bi-Directional Long Short Term Memories with total time step of 4

Instead of feeding the BLSTM with raw input tensor $\mathbf{x_t}$, CNN-BLSTM firstly processes it through CNN layer, resulting output tensor $\mathbf{c_t}$, as shown in Equation **??**.

$$[\mathbf{c_1};\mathbf{c_2};...;\mathbf{c_n}] = CNN([\mathbf{x_1};\mathbf{x_2};...;\mathbf{x_n}]) \tag{3.10}$$

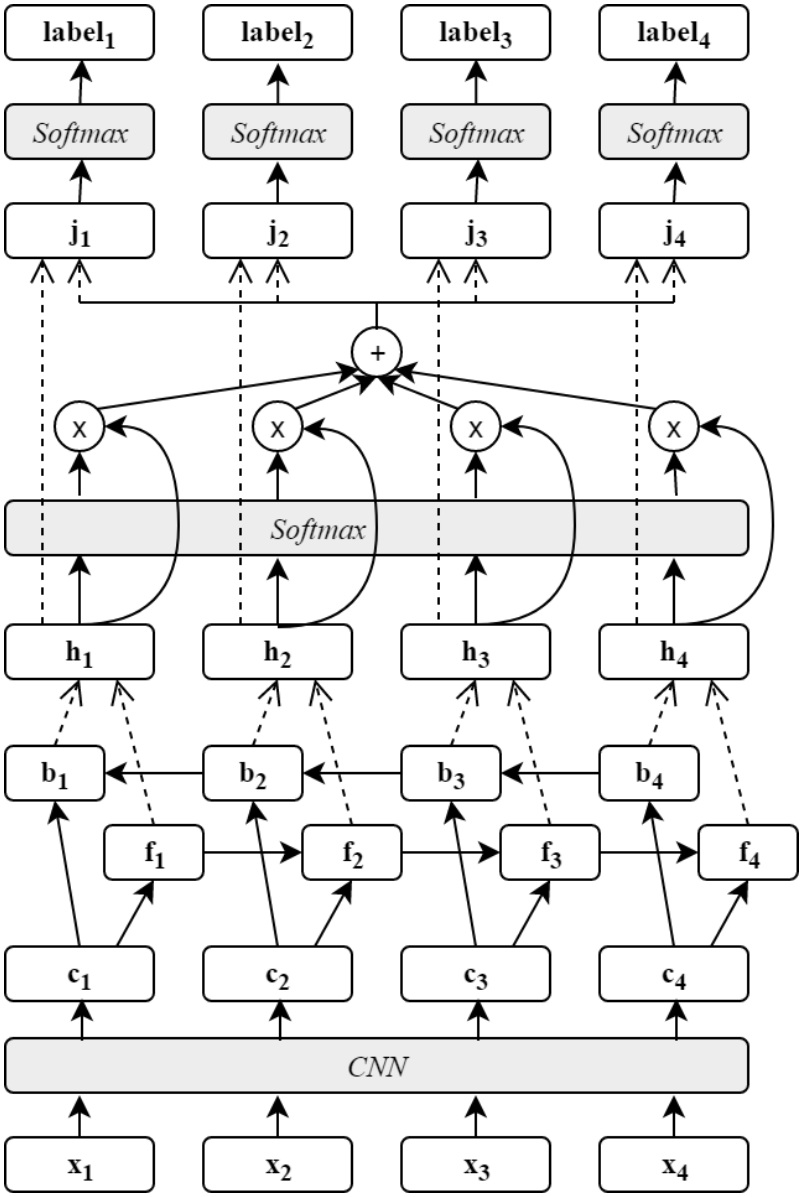
The result tensor $\mathbf{c_t}$ is then fed into the BLSTM layer, as explained in section 3.5.2.2. To simplify the notation, the BLSTM layer equation is summarized as follows.

$$\mathbf{h_i} = BLSTM(\mathbf{c_i}) \tag{3.11}$$

As explained in section 3.5.2.2, the BLSTM's output tensor $\mathbf{h_i}$ is then processed by the time-distributed softmax layer to determine the final output label.

3.5.2.4 Context-Aware BLSTM (CA-BLSTM)

In this work, we propose a new LSTM architecture named Context-Aware Bi-Directional Long Short-Term Memory Networks (CA-BLSTM). The rationale is to add a dense yet useful high-level information containing a sentence context to every time step in order to help the machine to decide semantic roles better. With this in mind, we design an attention mechanism on top of the LSTM networks layers, as illustrated in Figure 3.6



Gambar 3.6: An architecture of Context-Aware Bi-Directional Long Short Term Memories with total time step of 4

The attention mechanism firstly collects the context information by multiplying trainable weights with all the vectors from every time step of the last LSTM output. We sum each element for each weighted vectors to reduce the dimension. The results are then fed into a hidden softmax layer which outputs weights with a total of 1. The original output vectors of the last LSTM output are multiplied by these distributed weights respectively. We then sum all the multiplication results as the final context information. The original LSTM outputs are concatenated with this context information before going to the last softmax layer to predict the semantic roles.

We describe the formulation of the networks from the first layer all the way to

the top. Like the previous architectures, it begins with these two equations:

$$[\mathbf{c_1};\mathbf{c_2};...;\mathbf{c_n}] = CNN([\mathbf{x_1};\mathbf{x_2};...;\mathbf{x_n}]) \tag{3.12}$$

$$\mathbf{h_i} = BLSTM(\mathbf{c_i}) \tag{3.13}$$

$\mathbf{x_i}$, $\mathbf{c_i}$, and $\mathbf{h_i}$ are the input tensors, the output of CNN layer, and the output of BLSTM layer respectively, with i indicating the time step.

$$g(\mathbf{h_i}) = Sum(\mathbf{W}.\mathbf{h_i}) \tag{3.14}$$

$\mathbf{h_i}$ is then fed into a differentiable neural networks function $g(\mathbf{h_i})$ in which it is multiplied by the time-distributed matrix $\mathbf{W} \in \mathbb{R}^{H\times K}$ and all the elements in it are summed. H is the vector dimension of $\mathbf{h_i}$, meanwhile K is the dimension size that we want as an output when we multiply W with $\mathbf{h_i}$.

$$[\alpha_1, \alpha_2, ..., \alpha_n] = Softmax([g(\mathbf{h_1}); g(\mathbf{h_2}); ..., g(\mathbf{h_n})]) \tag{3.15}$$

$$\mathbf{r_i} = \alpha_i.\mathbf{h_i} \tag{3.16}$$

Once we have all the values of $[g(\mathbf{h_1}); g(\mathbf{h_2}); ...; g(\mathbf{h_n})]$, we make it as an input for the softmax layer, resulting weights $[\alpha_1;\alpha_2;...;\alpha_n]$. All the original LSTM outputs $[\mathbf{h_1};\mathbf{h_2};...;\mathbf{h_n}]$ are multiplied by these weights, with the results of $[\mathbf{r_1};\mathbf{r_2};...;\mathbf{r_n}]$

$$\mathbf{z} = \mathbf{r_1} + \mathbf{r_2} + ... + \mathbf{r_n} \tag{3.17}$$

$$\mathbf{j_i} = Concatenate(\mathbf{h_i}, z) \tag{3.18}$$

We then sum all these vectors element-wise to have a context tensor $\mathbf{z}$. All the original LSTM outputs are thus concatenated with tensor $\mathbf{z}$ as the additional information to predict the semantic roles.

$$label_i = Softmax(\mathbf{j_i}) \tag{3.19}$$

Lastly, the time-distributed softmax layer produces the final semantic roles label.

3.6 Experiment

We use 5-fold cross validation for out experiments and thus, the data set is firstly split into 5 sets. These 5 sets are divided into training and testing sets with the ratio

of 4:1. After that, we train the model using the training set and evaluate it using the testing set. This process is done 5 times until every set of data is tested.

3.7 Evaluation

In each scenario, we evaluate the trained model in order to see how good it predicts the semantic roles as expected. The metrics for our evaluation are precision, recall, and F1. These metrics is applied to all semantic role labels. We then average each metrics from all semantic role labels to get the average precision, recall, and F1 of a model. The evaluation approach used is partial match in which a set of predicted labels is counted right if there is an intersection with the gold-standard (Seki dan Mostafa, 2003).

Suppose that we are evaluating the semantic role PATIENT. The rules for evaluation using partial match for semantic role PATIENTare explained as follows.

1. Counting *True Positive* (TP)
   For every gold standard label that has intersection with the predicted label, the value of True Positive (TP) is added by 1.

Gold-standard: Aku pengen makan <Patient>**ayam goreng**</Patient> deh
Predicted.1: Aku pengen makan <Patient>**ayam goreng**</Patient> deh
Predicted.2: Aku pengen makan <Patient>**ayam**</Patient> goreng deh
Predicted.3: Aku pengen makan <Patient>**ayam goreng deh** </Patient>
Predicted.4: Aku pengen makan ayam goreng deh

   The examples above illustrate four scenarios of possible predicted results, denoted as Predicted.1, Predicted.2, Predicted.3, and Predicted.4, given a gold-standard called Expected which has "ayam goreng" as the PATIENT.

   Predicted.1 predicts exactly the same as the Expected, hence the value of TP is added by 1. The result of Predicted.2 has an intersection with the gold-standard, which is "ayam", the value of TP is then added by 1 as well. Although Predicted.3 predicts too much as it includes "deh" as part of PATIENT, it also has an intersection with the gold-standard, which is "ayam goreng". The Predicted.3 therefore also adds the value of TP by 1. Meanwhile, Predicted.4 does not predict anything. In this case, the value of TP is not added at all.

2. Counting *False Positive* (FP)
   For every predicted label that should not be predicted according to gold-standard, the value of False Positive (FP) is added by 1.

Gold-standard: Aku pengen makan <Patient>**ayam goreng**</Patient> deh Predicted.1: <Patient>Aku</Patient> pengen makan ayam goreng deh

   From the example above, "Aku" is predicted as PATIENT, while it should not be predicted as PATIENTaccording to the gold-standard. This will add the value of FP by 1.

3. Counting *False Negative* (FN)
   For every gold-standard label that is either not predicted or predicted with wrong label, the value of False Negative (FN) is added by 1.

Gold-standard: Aku pengen makan <Patient>**ayam goreng**</Patient> deh Predicted.1: Aku pengen makan <Agent>**ayam goreng**</Agent> deh Predicted.2: Aku pengen makan ayam goreng deh

   From the example above, Predicted.1 predicts "ayam goreng" as AGENT, while it should be predicted as PATIENT. In this case, the value of FP is added by 1. Predicted.2 illustrates an example which does not predict "ayam goreng" with any label, while it should be predicted as PATIENT. In this case, the value of FP is added by 1 as well.

After we have the value of TP, FP, and FN for the semantic role PATIENT, we then count the precision, recall, and F1 with following equations:

$$Precision = \frac{TP}{TP+FP} \tag{3.20}$$

$$Recall = \frac{TP}{TP+FN} \tag{3.21}$$

$$F-Measeure = 2 \cdot \frac{Precission \cdot Recall}{Precission + Recall} \tag{3.22}$$

Other semantic roles, such as AGENTand BENEFICIARY, are also evaluated by following these rules. After we have the value of precision, recall, and F1 for every semantic role, we average them to get the average precision, recall, and F1 for the model being evaluated.

BAB 4
IMPLEMENTATION

This chapter explains the implementations of the methodology explained in chapter 3. It includes the implementation of data annotation and pre-processing, model development, experiment, as well as the evaluation.

4.1 Computer Specification

For every experiment, we use GPU-based virtual server provided by Kata.ai. The specifications of the server are explained as follows.

Tabel 4.1: Server Specifications

Processor	i7-4770S
Number of Cores	8 core
Processor Frequency	3.1 GHz per core
RAM	8 GB
Operating System	Ubuntu 14

The server uses 8-core i7 processor with 3.1 GHz per core frequency. The size of the RAM is 8 GB. We use Ubuntu 14 distribution as the operating system.

4.2 Data Annotation and Pre-processing

For data annotation, we use an in-house tool provided by Kata.ai, named *kata-annotator*. The total amount of data to be annotated was 9000 sentences. The data was annotated by three linguists with each of them annotating different set of data containing 3000 sentences for 8 weeks. In order to align the annotation understanding between them, the three linguists annotated the same trial set consisting of 100 sentences before starting to annotate the real one. The annotation differences found are then discussed in order to align the understanding between them.

After 8 weeks of annotation, the total amount of data that has been annotated is 8000 sentences. The other 1000 sentences missing is due to one annotator that could not complete the annotation on time. After finish annotating, the tool outputs the tokenized annotation result as JSON in BIO format. An example is given as follows:

```
[
{
"data": ["Aku", "pengen", "makan", "ayam", "goreng", "dong"],
"label": ["B-AGENT", "B-MD", "B-PRED", "B-PATIENT", "I-PATIENT", "O"]
},
{
"data": ["Kamu", "gak", "tidur", ",", "Andi", "?"],
"label": ["B-AGENT", "O", "B-PRED", "O", "B-GREET", "O"]
}
]
```


The label is then converted into a one-hot-vector representation which is presented in Pseudocode 4.1

Kode 4.1: A pseudocode for converting labels of a sentence into one-hot-vectors

```
1 Function convertLabelToOneHotVector(arrLabel) is
      Input  : array of labels of a sentence
      Output: array of one hot vectors
2     oneHotVectorLabel = [];
3     foreach label in arrLabel do
4         oneHotVectorLabel.append(label.convertToOneHotVector())
5     return oneHotVectorLabel;
```

It turns out there are only 5000 sentences which contain predicate in it. These 5000 sentences are the main data set to be trained and tested.

4.3 Model Development

This section explains the implementation of model development, including the feature extraction and model architecture. We use Python as our main programming language for all implementations.

4.3.1 Feature Extraction

The features to be extracted are word embedding, POS tag, and neighboring word embedding.

4.3.1.1 Word Embedding

We use Gensim's Word2Vec as the library for training the word embedding model as well as converting words into vectors. Pseudocode 4.2 explains on how to train the word embedding model.

Kode 4.2: A pseudocode to train word embedding model using Word2Vec

```
1 Function trainWordEmbeddingModel(corpus, contextWindow,
    vectorDimension) is
      Input  : training corpus, context window, vector dimension
      Output: Word2Vec model
2     model = Word2Vec.createModel(corpus, contextWindow,
        vectorDimension)
3     return model;
```

There are two parameters, which are context window and vector dimension. Context window determines the area of interest in building the word embedding model. Vector dimension represents the length of the output vector. In this work, the context window and vector dimension used are 5 and 32, respectively.

Kode 4.3: A pseudocode to transform words into vectors by word embedding model

```
1 Function wordToVector(model, arrWord) is
      Input  : trained word embedding model, array of words of a sentence
      Output: array of word vectors
2     arrVector = [];
3     foreach word in arrWord do
4         arrVector.append(model.getVector(word))
5     return arrVector;
```

From the Pseudocode 4.2, the output is the trained word embedding model. 4.3 shows how to use the model to convert words into vectors.

4.3.1.2 POS Tag

For POS Tag feature, we use the gold-standard POS tag annotated by the three linguists. The annotation tool *kata-annotator* is also used for annotating the POS tag. The output example of the POS tag from the tool in a form of JSON is given as follows:

```
[
{
"data": ["Aku", "pengen", "makan", "ayam", "goreng", "dong"],
"label": ["NN", "ADV", "V", "NN", "V", "INTJ"]
},
{
"data": ["Kamu", "gak", "tidur", ",", "Andi", "?"],
"label": ["NN", "NEG", "V", "O", "NN", "O"]
}
]
```

The JSON file consists of an array of sentences. Each word in a sentence is labeled with the POS tag accordingly.

Kode 4.4: A pseudocode for converting POS tags of a sentence into one hot vectors

```
1 Function convertPOSTagToOneHotVector(arrPOS) is
      Input  : array of POS tags of a sentence
      Output: array of one hot vector
2     posTagFeature = [];
3     foreach pos in arrPOS do
4         posTagFeature.append(pos.convertToOneHotVector())
5     return posTagFeature;
```

Each of the POS tag in a sentence is then converted into one-hot-vector. The implementation is presented in Pseudocode 4.4.

4.3.1.3 Neighboring Word Embeddings

For neighboring word embeddings, we extract one vector on the left and one on the right of the word being processed. Pseudocode 4.5 shows the implementation of extracting neighboring word embeddings.

The parameter here is the window. Since we only extract one word on the left and another on the right of the word being processed, the value of parameter window is 1.

4.3.2 Model Architecture

As explained in chapter 3, we experiment on four model architectures, namely vanilla LSTM (LSTM), Bi-Directional LSTM (BLSTM), CNN-BLSTM, and

Kode 4.5: A pseudocode to extract neighboring word embeddings

```
1  Function extractNeighboringWordEmbedding(sentenceVector) is
       Input  : array of word embedding vectors of a sentence
       Output: array of neighboring word embedding vectors
2      window = 1 vectorDimension = getVectorDimension(sentenceVector);
3      padded = window * [vectorDimension*[0.]] + sequence + window *
         [vectorDimension*[0.]];
4      neighboringVectors = [];
5      for i in 0...sentenceVector.length - 1 do
6          left = [item for sublist in padded[i:(i + window)] for item in sublist];
7          right = [item for sublist in padded[(i+ window + 1):(i + nbContexts)]
             for item in sublist];
8          concate = left + right;
9          neighboringVectors.append(concate);
10     end
11     return neighboringVectors;
12 end
```

Context-Aware BLSTM (CA-BLSTM). In this section, each implementation of the model architecture is explained. We use Keras 2.0 (Chollet, 2015) as our deep learning library with Tensorflow 1.0 backend for all the architectures. We use the *functional model* of Keras. Keras model only receives input data with a fixed number of time steps for all sentences. Suppose that the maximum number of time steps in our data is l. Thus, sentences in our data whose number of time steps is lower than l have to be padded with vector $\vec{0}$ in order to get a fixed number of time steps of l. To do the padding, we use *padsequences* function available from Keras.

4.3.2.1 Vanilla LSTM (LSTM)

Vanilla LSTM (LSTM) consists of only one layer of forward LSTM. Pseudocode **??** shows the implementation of the LSTM architecture.

The Pseudocode 4.6 takes x train, y train, number of time steps, number of features, x test and y test as the inputs. The model starts with the defining the input layer, with the input shape of (timesteps, features). The input layer is then connected to the LSTM layer thas has 128 hidden units. These hidden units are the output of the LSTM layer. We use recurrent dropout in LSTM, as recommended by He et al. (2017). The recurrent dropout used is 0.2. We also use dropout layer on top of the LSTM layer by the value of 0.2. The output of the dropout layer is connected to the time-distributed dense layer with softmax activation function. These last layer produces the labels of semantic roles. The model is trained with

Kode 4.6: A pseudocode for building and training vanilla LSTM architecture

```
1  Function lstm(xTrain, yTrain, timesteps, features, xTest, yTest) is
       Input  : x train, y train, number of time steps, number of features, x test,
                y test
       Output: trained model, testing prediction result
2      inputLayer = Input(shape=(timesteps, features));
3      forwardLayer = LSTM(units=128, returnSequences=True,
         recurrentDropout=0.2)(inputLayer);
4      dropoutLayer = Dropout(0.2)(forwardLayer);
5      outputLayer = TimeDistributed(Dense(units=17,
         activation='softmax'))(dropoutLayer);
6      model = Model(inputs=[inputLayer], outputs=[outputLayer]);
7      model.compile(loss='categoricalCrossentropy', optimizer='adam');
8      model.fit(xTrain, yTrain, epochs=50, batchSize=50);
9      prediction = model.predict(xTest);
10     return model, prediction;
```

categorical crossentropy loss function and Adam optimizer. The number of epochs and batch size used are both 50. After the model has been trained, we use it to predict the semantic roles the x test data which later will be evaluated. The function returns the trained model as well as the prediction result of the test data.

4.3.2.2 Bi-Directional LSTM (BLSTM)

Bi-Directional LSTM (BLSTM) consists of two layers of LSTM. The first layer is moving forward and the other is moving backward. Pseudocode 4.7 shows the implementation of the BLSTM architecture.

The Pseudocode 4.7 takes x train, y train, number of time steps, number of features, x test and y test as the inputs. The model starts with the defining the input layer, with the input shape of (timesteps, features). The input layer is then connected to two LSTM layers: forward layer and backward layer. Both forward and backward LSTM layers have 128 hidden units. The recurrent dropout used is 0.2. The output of both forward and backward layers are concatenated, resulting a vector whose length is 256. We also use dropout layer on top of the concatenated layer by the value of 0.2. The output of the dropout layer is connected to the time-distributed dense layer with softmax activation function. This last layer produces the labels of semantic roles. The model is trained with categorical crossentropy loss function and Adam optimizer. The number of epochs and batch size used are both 50. After the model has been trained, we use it to predict the semantic roles the x

Kode 4.7: A pseudocode for building and training BLSTM architecture

```
1  Function blstm(xTrain, yTrain, timesteps, features, xTest, yTest) is
       Input  : x train, y train, number of time steps, number of features, x test,
                y test
       Output: trained model, testing prediction result
2      inputLayer = Input(shape=(timesteps, features));
3      forwardLayer = LSTM(units=128, returnSequences=True,
         recurrentDropout=0.2)(inputLayer);
4      backwardLayer = LSTM(units=128, returnSequences=True,
         goBackwards=True, recurrentDropout=0.2)(inputLayer);
5      mergedLayer = Concatenate([forwardLayer, backwardLayer]);
6      dropoutLayer = Dropout(0.2)(mergedLayer);
7      outputLayer = TimeDistributed(Dense(units=17,
         activation='softmax'))(dropoutLayer);
8      model = Model(inputs=[inputLayer], outputs=[outputLayer]);
9      model.compile(loss='categoricalCrossentropy', optimizer='adam');
10     model.fit(xTrain, yTrain, epochs=50, batchSize=50);
11     prediction = model.predict(xTest);
12     return model, prediction;
```

test data which later will be evaluated. The function returns the trained model as well as the prediction result of the test data.

4.3.2.3 CNN-BLSTM

CNN-BLSTM adds a CNN layer underneath the BLSTM layers. Pseudocode 4.8 shows the implementation of the CNN-BLSTM architecture.

In this architecture, the CNN layer is added before the BLSTM layers. Thus, the input layer is connected with the CNN layer. The parameters of the CNN layer are filters, kernel size, and strides. Filters represent the number of output hidden layers. Kernel size defines the context window of the CNN layer. Strides define the amount of slide for every time step. The value of filters, kernel size, and strides parameters are 128, 3, and 1, respectively. CNN layer is then connected to the BLSTM layers as explained in the previous section.

4.3.2.4 Context-Aware BLSTM (CA-BLSTM)

CA-BLSTM adds an attention mechanism on top of the BLSTM layers. Pseudocode 4.9 shows the implementation of the CA-BLSTM architecture.

The output of the dropout layer from BLSTM is then fed into a time-distributed,

Kode 4.8: A pseudocode for building and training CNN-BLSTM architecture

```
1  Function cnnblstm(xTrain, yTrain, timesteps, features, xTest, yTest) is
       Input  : x train, y train, number of time steps, number of features, x test,
                y test
       Output: trained model, testing prediction result
2      inputLayer = Input(shape=(timesteps, features));
3      cnnLayer = Conv1D(filters=128, kernelSize=3, padding='same',
         activation='relu', strides=1)(inputLayer);
4      forwardLayer = LSTM(units=128, returnSequences=True,
         recurrentDropout=0.2)(cnnLayer);
5      backwardLayer = LSTM(units=128, returnSequences=True,
         goBackwards=True, recurrentDropout=0.2)(cnnLayer);
6      mergedLayer = Concatenate([forwardLayer, backwardLayer]);
7      dropoutLayer = Dropout(0.2)(mergedLayer);
8      outputLayer = TimeDistributed(Dense(units=17,
         activation='softmax'))(dropoutLayer);
9      model = Model(inputs=[inputLayer], outputs=[outputLayer]);
10     model.compile(loss='categoricalCrossentropy', optimizer='adam');
11     model.fit(xTrain, yTrain, epochs=50, batchSize=50);
12     prediction = model.predict(xTest);
13     return model, prediction;
```

raw dense layer. The output dimension of this dense layer is set to 512. The output of this layer is then summed for each time step. The result of this sum is then fed into the dense layer with softmax activation function. This dense layer outputs the weights alpha for each time step. Lambda function *kaliAlphaSum* firstly multiplies the original dropout output of the BLSTM with the alpha values. The function then sums all the result element-wise in order to get a context vector z. Vector z is then duplicated by the number of time steps before it is concatenated with all the original dropout output of the BLSTM layers. These concatenated vectors are then fed into the last softmax layer to produce the output labels.

4.4 Experiment

4.5 Evaluation

Kode 4.9: A pseudocode for building and training CA-BLSTM architecture

```
1  Function cablstm(xTrain, yTrain, timesteps, features, xTest, yTest) is
       Input  : x train, y train, number of time steps, number of features, x test,
                y test
       Output: trained model, testing prediction result
2      inputLayer = Input(shape=(timesteps, features));
3      cnnLayer = Conv1D(filters=filters, kernelSize=3, padding='same',
         activation='relu', strides=128)(inputLayer);
4      forwardLayer = LSTM(units=128, returnSequences=True,
         recurrentDropout=0.2)(cnnLayer);
5      backwardLayer = LSTM(units=128, returnSequences=True,
         goBackwards=True, recurrentDropout=0.2)(cnnLayer);
6      mergedLayer = Concatenate([forwardLayer, backwardLayer]);
7      dropoutLayer = Dropout(0.2)(mergedLayer);
8      outs1 = TimeDistributed(RawDense(outputDim=512))(dropoutLayer);
9      m = Lambda(sum)(outs1);
10     alpha = Dense(timesteps, activation="softmax")(m);
11     z = Lambda(kaliAlphaSum)([alpha, dropoutLayer]);
12     dropoutZ = Dropout(0.2)(z);
13     repeatedZ = RepeatVector(timesteps)(dropoutZ);
14     outFinal = concatenate([dropoutLayer, repeatedZ]);
15     outputLayer = TimeDistributed(Dense(units=17,
         activation='softmax'))(outFinal);
16     model = Model(inputs=[inputLayer], outputs=[outputLayer]);
17     model.compile(loss='categoricalCrossentropy', optimizer='adam');
18     model.fit(xTrain, yTrain, epochs=50, batchSize=50);
19     prediction = model.predict(xTest);
20     return model, prediction;
```

BAB 5
EXPERIMENTS

In this chapter, we describe the data statistics followed by experiment scenarios, results, and analyses.

5.1 Evaluation Metrics

5.2 Data Statistics

5.3 Experiment Design

In this section, we present our experiment results and the analysis accordingly. There are two set of scenarios. The first scenario set aims to find the best combination set of features. This scenario consists of four combinations of three features, namely feature Word Embedding (WE), POS Tag (POS), and Word Embedding of Neighbors (WE-N) as presented in Table 5.1.

The second scenario set evaluates two different architectures, which are the original BLSTM and CA-BLSTM. Underlying both architectures is a Convolutional Neural Network (CNN) layer, in order to catch more information surrounding each time step. This second scenario also aims to see the effect of hyper-parameter tuning for the CA-BLSTM.

As for evaluation, we use precision, recall and F1 metrics for all scenarios. The results are evaluated with partial match approach. To illustrate, suppose that the expected label consists of two words or more and the predicted contains only one word of it, partial match will still count it as a true positive.

5.4 Scenario 1: Feature Selection

Table 5.1 shows the scenario results of four feature combinations. The highest result is achieved with combination of WE + POS, followed by WE + WE-N + POS, WE, and WE + WE-N, with F1 scores of 72.29%, 72.22%, 62.00%, and 61.92% respectively. From these results, we can see the big impact POS Tag contributes for the performance. Using POS Tag enhances the result up to 10.29%, when we compare WE + POS and WE combinations. The explanation would be the fact

that POS Tag contains meaningful information such as Noun and Adjective which describes the word, it thus supports the word embedding feature when the training data size is relatively low.

Surprisingly, when we combine the neighboring words of the argument as in WE + WE-N, the result slightly decreases by 0.08%, compared to only using WE feature. This is also the case when we compare WE + WE-N with WE + WE-N + POS scenarios, which decreases by 0.07% when we used neighboring words. It shows us that neighboring words do not improve the performance at all. We suggest that this is because the CNN layer already extracts these information implicitly, by capturing surrounding information and compressed it into one vector. Hence, we do not need explicit neighboring words as part of our features.

Tabel 5.1: Results of Feature Combination Scenario

Features	Precision	Recall	F1
WE	64.68%	60.25%	62.00%
WE + POS	74.24%	**71.26%**	**72.29%**
WE + WE-N	64.17%	60.29%	61.92%
WE + POS + WE-N	**74.51%**	70.69%	72.23%

Since WE + POS outputs the best result in terms of F1 score in this scenario set, we will use it for the next set of scenarios.

5.4.1 Scenario 2: Model Selection

The experiment results of the second scenario set are presented in Table 5.2. First we show the result using the original BLSTM with F1 score of 72.29%. On the next experiment, we added the attention mechanism on top of the BLSTM layer, with three different dimensions K of weight matrix $W \in \mathbb{R}^{H \times K}$ from the Equation 3.14. The three dimensions of K are 64, 128, and 256. When we experimented on the first dimension size of K, 64, it is notable that the F1 score increases by 0.76%. The performance keeps increasing as we add the dimension size of K until 256 with F1 score of 74.78% which increased by 2.49% compared to the original BLSTM. We also obtained the highest precision and recall with dimension size K of 256. However, when we added more dimension above 256, the model seems to be overfitting.

The results show that the CA-BLSTM architecture outperforms the original BLSTM architecture. We suggest that our proposed model can capture context information at abstract level. We believe that the attention mechanism plays an

essential role to extract which word has the most significant value as a hint to predict the semantic roles of each time step.

Tabel 5.2: Results of Model Architecture Scenario

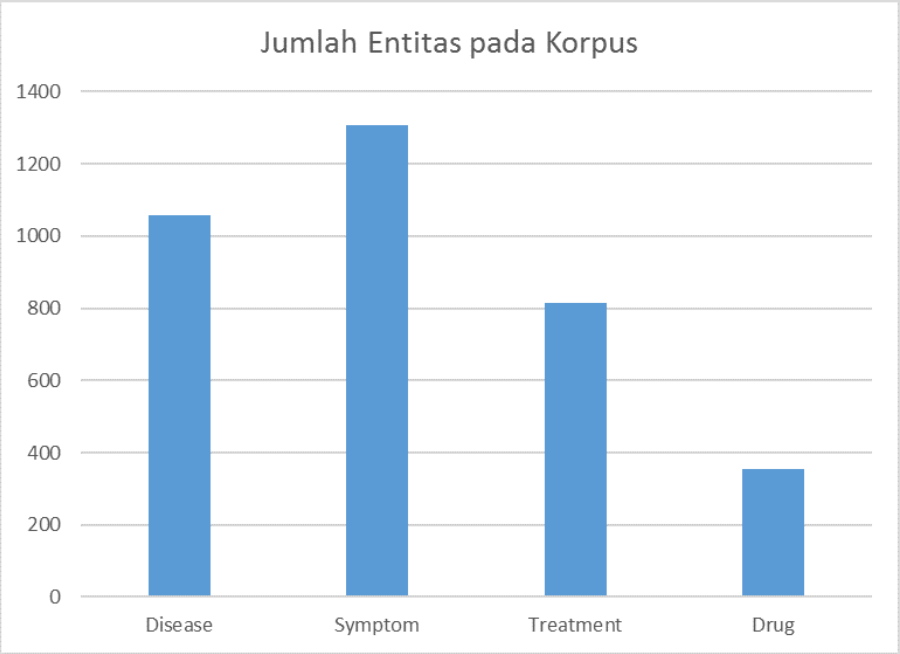
Name	Precision	Recall	F1
BLSTM	74.24%	71.26%	72.29%
CA-BLSTM-64	73.37%	73.25%	73.05%
CA-BLSTM-128	76.25%	73.52%	74.05%
CA-BLSTM-256	**77.03%**	**73.55%**	**74.78%**

5.5 Metrik Evaluasi

Pada eksperimen ini, untuk mendapatkan nilai akurasi dari masing-masing eksperimen we menggunakan *precision*, *recall* dan *f-measure*. We menggunakan *10-cross fold validation* dalam menjalankan eksperimen. Terkait dengan penjelasan mengenai cara penghitungan dan evaluasi sudah dijelaskan pada Bab 3.

5.6 Visualisasi Data

Berikut merupakan visualisasi data dari korpus yang we miliki. Dapat dilihat dari grafik 5.1, bahwa persebaran jumlah entitas tidak seimbang. Hal ini menjadi kendala we dalam melakukan penelitian ini, karena keterbatasan *resource* dan tenaga untuk melakukan pelabelan dokumen secara manual.



Gambar 5.1: Histogram Jumlah Entitas pada Korpus

Tabel 5.3 menunjukkan 39 daftar entitas *disease* teratas yang terdapat di dalam korpus.

Tabel 5.3: Tabel Beberapa Entitas *Disease* pada Korpus dan Jumlahnya

asma (20)	isk (20)	tbc (18)
infeksi (17)	sinusitis (16)	jerawat (15)
alergi (14)	oligomenorea (14)	maag (12)
fam (11)	flu (11)	kanker serviks (11)
hiv (11)	penyakit jantung (11)	varikokel (10)
wasir (10)	infeksi saluran kemih (9)	obesitas (9)
cacar air (9)	diabetes (9)	albino (9)
gerd (8)	hepatitis b (8)	tth (8)
liver (8)	sifilis (8)	tuberkulosis (7)
stroke (7)	sakit maag (7)	anemia aplastik (7)
alergi susu sapi (7)	hepatitis (7)	diare (6)
scabies (6)	kanker otak (6)	jengger ayam (6)
pid (6)	osteoporosis (5)	tifus (5)

Tabel 5.4 menunjukkan 39 daftar entitas *symptom* teratas yang terdapat di dalam korpus.

Tabel 5.4: Tabel Beberapa Entitas *Symptom* pada Korpus dan Jumlahnya

nyeri (60)	sakit kepala (51)	demam (50)
mual (40)	gatal (30)	batuk (27)
pusing (24)	muntah (23)	diare (16)
keputihan (15)	nyeri dada (14)	sesak nafas (14)
kejang (13)	keringat dingin (13)	pilek (12)
nyeri kepala (10)	stres (10)	stress (10)
sesak (8)	jantung berdebar - debar (7)	flu (7)
rasa nyeri (7)	lemas (7)	kesemutan (7)
bersin (6)	sakit gigi (6)	mimisan (6)
nyeri pinggang (6)	depresi (5)	perih (5)
sakit perut (5)	anyang - anyangan (4)	bintilan (4)
kram perut (4)	pingsan (4)	susah tidur (4)
rasa gatal (4)	perubahan mood (4)	kelelahan (4)

Tabel 5.5 menunjukkan 39 daftar entitas *treatment* teratas yang terdapat di dalam korpus.

Tabel 5.5: Tabel Beberapa Entitas *Treatment* pada Korpus dan Jumlahnya

operasi (40)	pemeriksaan fisik (34)	terapi (21)
usg (15)	pemeriksaan penunjang (11)	tes darah (10)
istirahat (9)	pemeriksaan darah (8)	pemeriksaan jantung (7)
ct scan (7)	ekg (6)	x - ray (6)
foto rontgen (5)	biopsi (5)	imunisasi (5)
tes pap (5)	operasi sesar (4)	mri (4)
istirahat cukup (4)	kontrasepsi (4)	dinebu (3)
tes lbc (3)	minum air putih (3)	rontgen (3)
rontgen dada (3)	susu (3)	fisioterapi (3)
tes iva (3)	berolahraga (3)	tes penunjang (3)
hindari memencet (2)	test pack (2)	terapi obat - obatan (2)
makan makanan bergizi (2)	pasang iud (2)	cek darah (2)
kompres dingin (2)	tes hpv (2)	gunakan kondom (2)

Tabel 5.6 menunjukkan 39 daftar entitas *drug* teratas yang terdapat di dalam korpus.

Tabel 5.6: Tabel Beberapa Entitas *Drug* pada Korpus dan Jumlahnya

antibiotik (31)	paracetamol (8)	ibuprofen (7)
parasetamol (5)	whey protein (5)	obat tetes mata (5)
obat batuk (5)	vitamin c (4)	obat herbal (4)
nebulizer (4)	kiranti (4)	salbutamol (4)
salep (4)	obat antibiotik (4)	amoxilin (3)
metronidazol (3)	obat pelangsing (3)	nitrokaf (3)
kortikosteroid (3)	asam mefenamat (3)	tetrahydrolipstatin (3)
steroid (3)	ctm (3)	asam valproat (3)
rifampicin (3)	obat pereda nyeri (2)	habbatussauda (2)
obat depaken (2)	glucovance (2)	obat pereda rasa nyeri (2)
obat nyeri (2)	obat penurun panas (2)	probiotik (2)
sunblock (2)	obat jerawat (2)	bicrolid (2)
plembab (2)	aspirin (2)	acyclovir (2)

5.7 Desain Eksperimen

Pada penelitian ini, we melakukan 3 buah skenario utama, yaitu skenario untuk melakukan implementasi dan eksperimen ulang pada *baseline* (penelitian sebelumnya), skenario untuk menguji fitur yang memiliki kontribusi untuk meningkatkan akurasi dari setiap eksperimen dan skenario untuk menguji arsitektur RNNs yang we usulkan. Berikut merupakan skenario yang we rancang dalam penelitian ini:

1. Skenario 1: Skenario untuk mengimplementasikan ulang eksperimen sebelumnya
   Skenario ini bertujuan untuk mengimplementasikan ulang model yang diusulkan pada penelitian Herwando (2016), yaitu dengan menggunakan model *Conditional Random Fields* (CRFs). Fitur yang digunakan merupakan fitur yang memberikan hasil terbaik pada penelitian sebelumnya, yaitu fitur kata, fitur kamus, fitur frasa, fitur panjang kata dan fitur kata sebelum. Tujuan dari skenario ini yaitu sebagai *baseline* dan pembanding pada penelitian ini.

2. Skenario 2: Skenario untuk menguji fitur
   Skenario ini bertujuan untuk mendapatkan kombinasi fitur terbaik sehingga memberikan akurasi terbaik. We mencoba masing-masing fitur dengan

menggunakan model arsitektur LSTMs 1 layer. Apabila penggunaan fitur memberikan hasil yang lebih dari pada hasil eksperimen sebelumnya, fitur tersebut akan dipertahankan untuk eksperimen yang selanjutnya. Skenario ini memiliki 9 sub-skenario, yaitu:

(a) Sub-skenario 2.1 Fitur kata

(b) Sub-skenario 2.2 Fitur kata dan kamus

(c) Sub-skenario 2.3 Fitur kata, kamus dan *stopword*

(d) Sub-skenario 2.4 Fitur kata, kamus, *stopword* dan POS-Tag

(e) Sub-skenario 2.5 Fitur kata, kamus, *stopword*, POS-Tag dan frasa kata

(f) Sub-skenario 2.5 Fitur kata, kamus, *stopword* dan frasa kata

(g) Sub-skenario 2.5 Fitur kata, kamus, *stopword*, frasa kata dan kata sebelum

(h) Sub-skenario 2.5 Fitur kata, kamus, *stopword*, frasa kata, kata sebelum dan kata sesudah

3. Skenario 3: Skenario untuk menguji arsitektur RNNs
   Skenario ini bertujuan untuk melihat pengaruh arsitektur RNNs pada penelitian ini. We mencoba kedua arsitektur RNNs yang telah diusulkan sebelumnya dengan menggunakan kombinasi fitur terbaik dari eksperimen di skenario pengujian fitur di atas. Pada skenario ini, terdapat 2 sub-skenario, yaitu:

   (a) Sub-skenario 3.1: LSTMs 1 layer

   (b) Sub-skenario 3.2: LSTMs 2 layer multi-*input*

5.8 Skenario 1: *Baseline* Eksperimen Herwando (2016)

Pada penelitian ini, we mencoba melakukan implementasi ulang penelitian yang dilakukan oleh Herwando (2016). Data yang digunakan adalah data yang we gunakan dalam penelitian ini supaya perbandingan yang didapatkan *apple-to-apple*. Implementasi dan eksperimen ini bertujuan sebagai *baselaine* eksperimen dan penelitian yang we lakukan. Selain itu, juga untuk mengetahui secara singkat fitur yang diskriminatif dalam melakukan *sequence labeling* pada MER ini. Model yang digunakan sama dengan penelitian Herwando (2016), yaitu CRFs dengan penggunaan fitur kata, fitur kamus, fitur frasa, fitur panjang kata dan fitur kata sebelum.

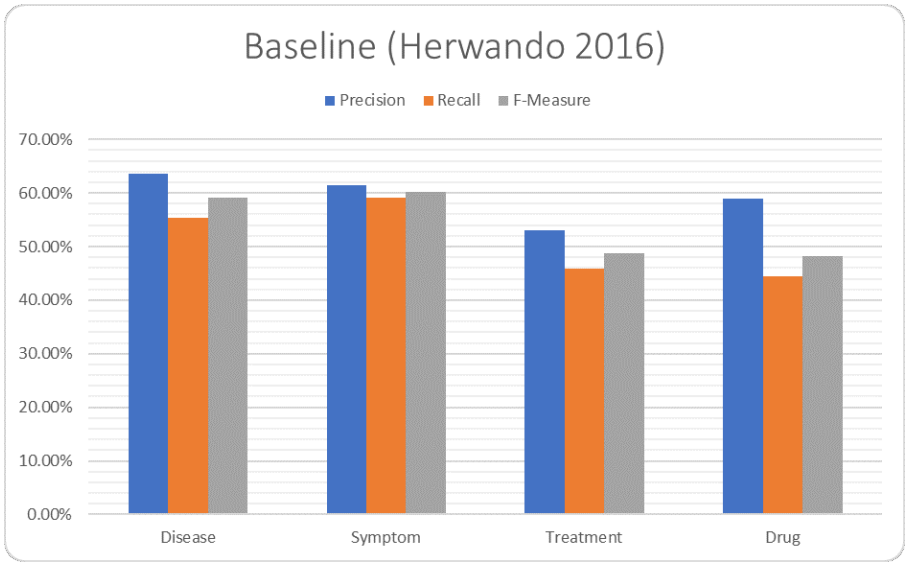
5.8.1 Hasil Eksperimen

Waktu komputasi: 6093.0 detik.

Berikut merupakan hasil implementasi ulang penelitian yang dilakukan oleh Herwando (2016).

Tabel 5.7: Tabel Hasil Eksperimen dari Penelitian Herwando (2016) (*Baseline*)

	Precission	*Recall*	*F-Measure*
Disease	63.68%	55.45%	59.13%
Symptom	61.43%	59.21%	60.18%
Treatment	53.10%	45.97%	48.82%
Drug	58.99%	44.46%	48.23%
Overall	**59.03%**	**51.27%**	**54.09%**



Gambar 5.2: Histogram Metrik Evaluasi dari Penelitian Herwando (2016) (*Baseline*)

5.9 Skenario 2: Skenario Pengujian Fitur

5.9.1 Eksperimen 2.1: Fitur Kata

Merujuk pada penelitian Mujiono et al. (2016), penelitian tersebut bertujuan untuk mendapatkan *non-handcrafted feature*, yaitu fitur kata itu sendiri dengan menggunakan *tools Word Embedding*. Oleh karena itu, we menguji fitur ini untuk mengetahui pengaruhnya pada program MER di penelitian ini.

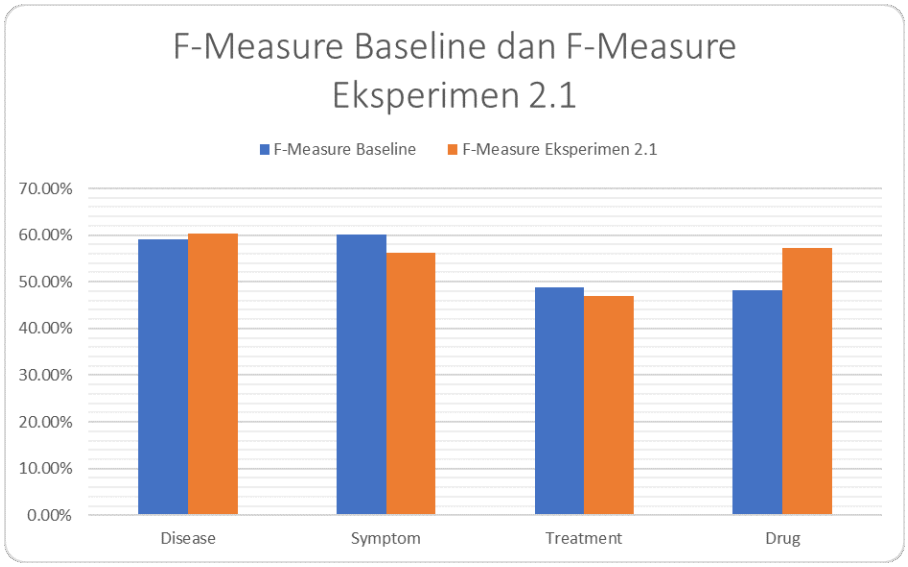
5.9.1.1 Hasil Eksperimen

Waktu komputasi: 5191.0 detik.

Tabel 5.8 menampilkan hasil pelabelan otomatis dengan menggunakan fitur kata itu sendiri yang direpresentasikan dengan menggunakan vektor *word embedding*.

Tabel 5.8: Tabel Hasil Eksperimen 2.1 dibandigkan dengan *Baseline*

Entitas	*Baseline (Herwando 2016)*			Eksperimen 2.1		
	Precision	*Recall*	*F-Measure*	*Precision*	*Recall*	*F-Measure*
Disease	63.68%	55.45%	59.13%	61.38%	60.42%	60.37%
Symptom	61.43%	59.21%	60.18%	57.05%	56.13%	56.19%
Treatment	53.10%	45.97%	48.82%	49.92%	47.17%	46.96%
Drug	58.99%	44.46%	48.23%	62.86%	53.32%	57.28%
Overall	**59.30%**	**51.27%**	**54.09%**	**57.80%**	**54.26%**	**55.20%**



Gambar 5.3: Histogram Perbandingan *F-measure Baseline* dengan Eksperimen 2.1

5.9.1.2 Analisis

Pada tabel 5.8, dapat dilihat bahwa secara umum *recall* dan *F-measure* yang dihasilkan lebih baik dibandingkan dengan *baseline*, walaupun untuk beberapa entitas nilainya lebih rendah (entitas *symptom* dan *treatment*). Selain itu, apabila dilihat pada grafik 5.3, secara umum model ini memberikan hasil yang lebih baik pada entitas *disease* dan *drug*. Kemudian rata-rata *F-measure* yang didapatkan yaitu 55.20%, lebih tinggi dibandingkan *baseline* yang we gunakan yaitu 54.09%.

Hal ini sangat menarik karena hanya dengan penggunaan fitur kata saja, hasil yang diberikan secara umum lebih baik dibandingkan *baseline*.

Pada eksperimen ini, ada beberapa entitas masih memiliki nilai *precision, recall* dan *f-measure* lebih kecil apabila dibandingkan dengan hasil yang dicapai Herwando (2016). Setelah we melakukan analisis terhadap penggunaan penggunaan *tools Word Embedding*, ternyata terdapat 429 kata unik yang tidak terdapat dalam model *word embedding*. Hal ini disebabkan oleh beberapa hal, yaitu:

1. Terdapat kata di dalam korpus yang tidak baku atau salah eja
   Korpus yang didapatkan berasal dari forum kesehatan *online* yang bersifat non-formal. Oleh karena itu baik pasien maupun dokter bebas mengutarakan pendapatnya tanpa adanya aturan bahasa formal. Oleh karena itu banyak ditemukan adanya kata yang tidak baku atau salah eja, misalnya:

   - dllsebaiknya,
   - sekatrang,
   - infeksiny,
   - kliengan.

2. Terdapat istilah sulit yang tidak terkandung di dalam model
   Terdapat beberapa istilah kesehatan pada korpus yang tidak ada di dalam model, hal ini disebabkan karena data untuk *training* model *word embedding* terbatas (model sangat tergantung pada data *training*). Oleh karena terdapat beberapa kata yang tidak terdaftar di dalam model. Contoh beberapa istilah sulit tersebut yaitu:

   - microdermabrians,
   - flixotide,
   - bimaflox,
   - polysiloxanes,
   - scizophrenia.

3. Terdapat kata yang merupakan nama orang
   Adanya kata yang merupakan nama orang tidak bisa dihindari di dalam forum kesehatan *online*. Selain itu, nama orang berbeda untuk setiap orang sehingga sulit mendapatkan vektor kata nama orang tersebut di dalam *word embedding*. Contoh dari kata yang merupakan nama orang di dalam korpus yaitu:

   - novira,

- risma,
- oktavia,
- sudianto.

Dari beberapa kasus di atas, we mengusulkan menambahkan fitur yang memperkaya informasi dari fitur kata itu sendiri, misalnya seperti apakah suatu kata terdapat dalam sebuah kamus kesehatan, informasi POS-Tag atau informasi yang lain. Oleh karena itu, we mencoba menggunakan tambahan fitur lain untuk meningkatkan akurasi pada penelitian ini, yaitu pada sub-eksperimen 5.9.2.

5.9.2 Eksperimen 2.2: Fitur Kata dan Kamus Kesehatan (*Disease, Symptom, Treatment* dan *Drug*)

Pada sub-eksperimen ini, we menggunakan tambahan fitur Kamus Kesehatan karena berdasarkan penelitian Herwando (2016) fitur ini memiliki konribusi untuk menambah akurasi pada sistem MER. Selain itu, menurut we, informasi suatu kata terdapat dalam sebuah kamus kesehatan mungkin akan memberikan kontribusi untuk meningkatkan akurasi. Oleh karena itu, we mencoba untuk menambahkan fitur ini ke dalam model RNNs.

5.9.2.1 Hasil Eksperimen

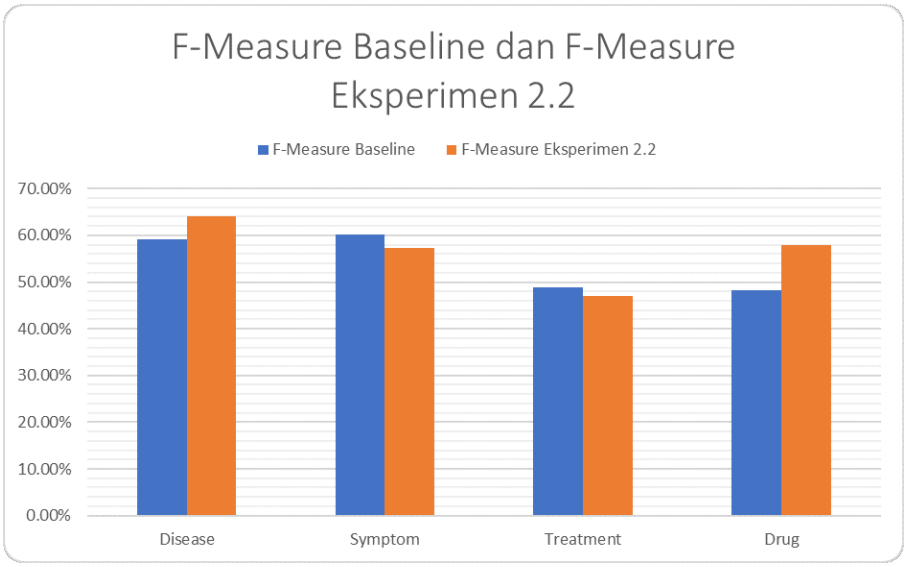
Waktu komputasi: 5658.0 detik.

Tabel 5.9 merupakan tabel hasil eksperimen yang didapatkan dengan menggunakan fitur ini.

Tabel 5.9: Tabel Hasil Eksperimen 2.2 dibandigkan dengan *Baseline*

Entitas	Baseline (Herwando 2016)			Eksperimen 2.2		
	Precision	*Recal*	*F-Measure*	*Precision*	*Recal*	*F-Measure*
Disease	63.68%	55.45%	59.13%	67.32%	61.78%	64.10%
Symptom	61.43%	59.21%	60.18%	60.55%	55.12%	57.41%
Treatment	53.10%	45.97%	48.82%	52.21%	44.18%	47.02%
Drug	58.99%	44.46%	48.23%	59.42%	59.71%	57.90%
Overall	**59.30%**	**51.27%**	**54.09%**	**59.88%**	**55.20%**	**56.61%**

Berikut merupakan grafik yang menunjukkan perbandingan *F-Measure* eksperimen 5.9 dengan *baseline* dalam bentuk histogram.



Gambar 5.4: Histogram Perbandingan *F-measure Baseline* dengan Eksperimen 2.2

5.9.2.2 Analisis

Dari tabel dan grafik 5.4, didapatkan informasi bahwa dengan menggunakan tambahan fitur kamus kesehatan terlihat bahwa entitas *Disease* mengalami kenaikan nilai *precision*, *recall*, dan *f-measure*. Selain itu, entitas *symptom* dan *tratment* mengalami kenaikan nilai *precision* dan *f-measure*. Entitas *drug* mengalami penurunan pada nilai *precision* namun mengalami kenaikan pada nilai *recall* dan *f-measure*-nya. Secara keseluruhan, Sedangkan entitas *drug* memiliki *precission* tertinggi, yaitu 62.86%. Grafik 5.4 menunjukkan perbandingan *precision*, *recall* dan *f-measure* untuk masing-masing entitas.

Dari analisis yang we lakukan terhadap korpus dan kamus kesehatan, terdapat beberapa entitas pada korpus yang tidak terdapat pada kamus sehingga mengakibatkan kenaikan hasil tidak besar. Hal ini karena terdapat beberapa penyebab, yaitu:

1. Ada entitas yang merupakan kombinasi atau gabungan dari entitas lain yang dihubungkan dengan kata penghubung. Hal ini banyak we temukan pada entitas *treatment* dan *symptom*. Contoh dari kasus ini yaitu:

    - nyeri hebat dibagian ulu hati dan pinggang belakang (gabungan dari "nyeri hebat dibagian ulu hati" dan "nyeri hebat pinggang belakang")
    - kondisi fases berampas , kuning , sedikit berlendir (gabungan dari "kondisi fases berampas", "kondisi fases kuning" dan "kondisi fases sedikit berlendir")

- alis atas dan bibir tidak bisa digerakkan (gabungan dari "alis atas tidak bisa digerakkan" dan "bibir tidak bisa digerakkan")

2. Penggunaan kata ganti orang di dalam entitas. Contoh dari kasus ini yaitu:

   - suara **saya** hilang
   - gusi **saya** berdarah
   - pinggang **saya** sakit

3. Kesalahan eja pada entitas atau penggunaan kata yang tidak baku, misalnya:

   - radang paru 2
   - butawarna
   - jrawatan
   - kanker darah setadium 1

Dibandingkan dengan hasil eksperimen Herwando (2016), hasil yang dicapai pada eksperimen ini masih lebih rendah pada entitas *symptom* dan *treatment*. Menurut we perlu ada informasi tambahan untuk meningkatkan akurasi. Seperti yang kita ketahui bahwa eksperimen Herwando (2016) tidak hanya menggunakan fitur kata itu sendiri dan kamus kesehatan saja. Oleh karena itu, we mencoba melakukan eksperimen kembali dengan menggunakan tambahan fitur lain pada sub-eksperimen 5.9.3.

5.9.3 Eksperimen 2.3: Fitur Kata, Kamus Kesehatan dan *Stopword*

Pada sub-eksperimen ini. we mencoba menambahkan informasi lain berupa fitur yang berisi sebuah kata apakah terdapat di dalam kamus *stop word* atau tidak. We berpendapat bahwa dengan adanya informasi *stop word*, adanya kesalahan suatu kata tidak berentitas yang dilabeli sebagai kata berentitas oleh model dapat dikurangi.

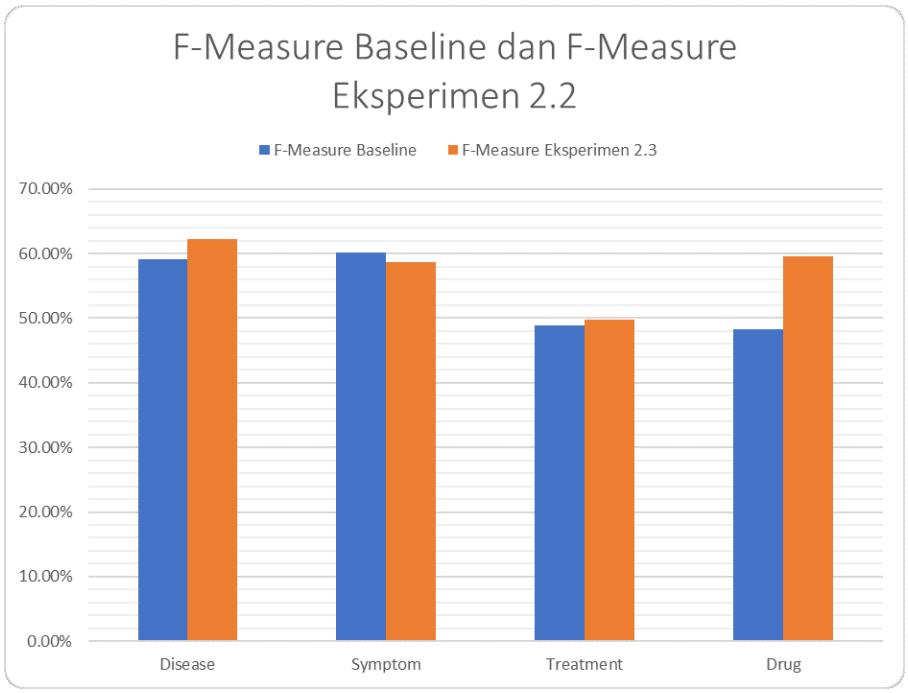
5.9.3.1 Hasil Eksperimen

Waktu komputasi: 6019.5 detik.

Rangkuman hasil sub-eksperimen ini dapat dilihat di Tabel 5.10 dan Gambar 5.5.

Tabel 5.10: Tabel Hasil Eksperimen 2.3 dibandigkan dengan *Baseline*

Entitas	Baseline (Herwando 2016)			Eksperimen 2.3		
	Precision	*Recall*	*F-Measure*	*Precision*	*Recall*	*F-Measure*
Disease	63.68%	55.45%	59.13%	65.97%	59.81%	62.28%
Symptom	61.43%	59.21%	60.18%	63.08%	55.20%	58.68%
Treatment	53.10%	45.97%	48.82%	54.73%	46.27%	49.69%
Drug	58.99%	44.46%	48.23%	61.88%	58.99%	59.57%
Overall	**59.30%**	**51.27%**	**54.09%**	**61.42%**	**55.07%**	**57.56%**



Gambar 5.5: Histogram Perbandingan *F-measure Baseline* dengan Eksperimen 2.3

5.9.3.2 Analisis

Dari Tabel 5.10 dan Gambar 5.5 dapat diamati bahwa secara umum, penggunaan fitur kamus *stop word* dapat meningkatkan *precision, recall,* dan *f-measure*. Untuk lebih detailnya, entitas *disease* mengalami penurunan nilai *precision* dan *f-measure* tetapi mengalami kenaikan nilai *recall*. Entitas *symptom* dan *treatment* mengalami kenaikan untuk nilai *precision, recall* dan *f-measure*. Sedangkan entitas *drug* mengalami kenaikan pada nilai *precission* dan *f-measure* teteapi mengalami penurunan pada nilai *recall*.

Pada sub-eksperimen ini, walaupun secara umum akurasi lebih baik dibandingkan dengan sub-eksperimen sebelumnya, hasil sub-eksperimen ini masih lebih

rendah pada entitas *treatment* apabila dibandingkan dengan hasil eksperimen Herwando (2016). Oleh karena we mengusulkan fitur tambahan lain yaitu fitur POS-Tag yang akan dijelaskan pada sub-eksperimen 5.9.4.

5.9.4 Eksperimen 2.4: Fitur Kata, Kamus Kesehatan, *Stopword* dan POS-Tag

Pada sub-eksperimen ini, we menambahkan informasi baru pada *resource* yang akan digunakan untuk *training* model yang berupa fitur POS-Tag. Sebelumnya fitur ini telah digunakan pada penelitian Abacha dan Zweigenbaum (2011) pada dokumen berbahasa Inggris dan memberikan kontribusi meningkatkan akurasi dari model MER yang dibangun. Oleh karena itu pada eksperimen ini we mencoba menggunakan fitur tersebut dan ingin mengetahui apakah fitur POS-Tag memiliki kontribusi untuk meningkatkan akurasi pada MER dengan dokumen berbahasa Indonesia.

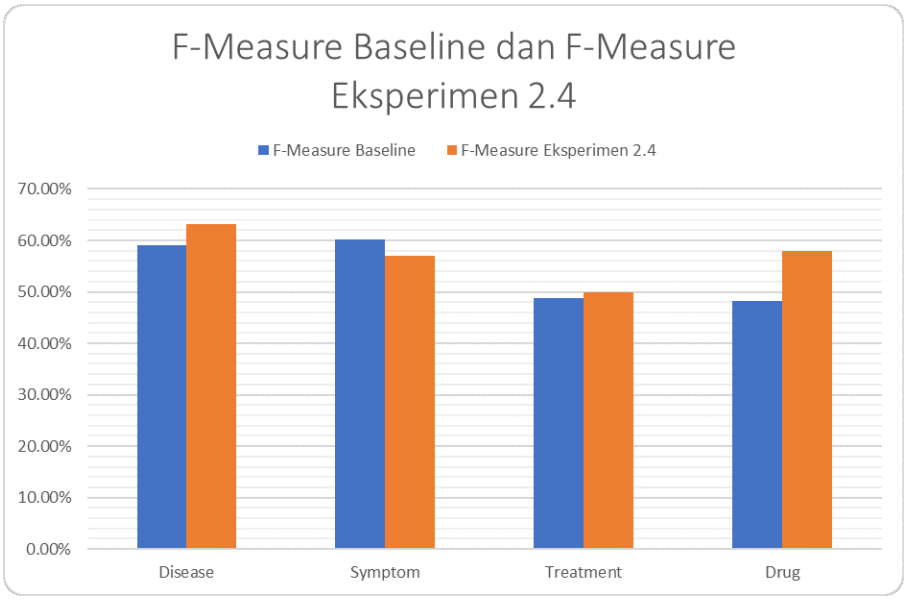
5.9.4.1 Hasil Eksperimen

Waktu komputasi: 6952.0 detik.

Rangkuman hasil sub-eksperimen ini dapat dilihat pada Tabel 5.11 dan Gambar 5.6.

Tabel 5.11: Tabel Hasil Eksperimen 2.4 dibandigkan dengan *Baseline*

Entitas	Baseline (Herwando 2016)			Eksperimen 2.4		
	Precision	*Recall*	*F-Measure*	*Precision*	*Recall*	*F-Measure*
Disease	63.68%	55.45%	59.13%	69.10%	58.67%	63.22%
Symptom	61.43%	59.21%	60.18%	61.09%	54.43%	57.00%
Treatment	53.10%	45.97%	48.82%	59.73%	44.10%	49.87%
Drug	58.99%	44.46%	48.23%	62.00%	55.74%	57.87%
Overall	**59.30%**	**51.27%**	**54.09%**	**62.98%**	**53.24%**	**56.99%**



Gambar 5.6: Histogram Perbandingan *F-measure Baseline* dengan Eksperimen 2.4

5.9.4.2 Analisis

Dari tabel dan grafik di atas, entitas *disease* dan *treatment* memiliki nilai *precision* dan *f-measure* yang meningkat, tetapi dengan nilai *recall* yang turun. Untuk entitas *symptom*, nilai *precision, recall*, dan *f-measure* mengalami penurunan. Sedangkan entitas *drug* mengalami kenaikan hanya pada *precision*-nya saja.

Hasil dari penggunaan fitur ini kurang baik, karena beberapa hal, yaitu:

1. Tag yang dihasilkan tidak konsisten. Pada beberapa entitas, terkadang suatu kata mendapatkan tag "A", namun di entitas yang lain untuk kata yang sama mendapatkan tag yang berbeda. Contoh dari kasus ini yaitu:
    - "antibiotik" memiliki tag "vb", sedangkan pada entitas lain "antibiotik" memiliki tag "NN"
    - "sakit kepala" memiliki beberapa tag pada entitas berbeda, yaitu "CD NN", "JJ NN", dan "NN NN"
    - "nyeri" memiliki beberapa tag pada entitas berbeda, yaitu "NN", "VB", "IN", "WH", dan "IN".
2. Tidak ada perbedaan tag antara kata berentitas dengan tidak, misalnya nama orang mendapatkan tag "NN" (intan_NN lusia_NN), namun nama penyakit juga mendapatkan tag "NN" (Kanker_NN Otak_NN).
3. Model POS-Tagger yang digunakan merupakan model untuk kalimat dengan topik umum, tidak dikhususkan pada topik kesehatan.

Oleh karena itu pada sub-eksperimen selanjutnya, we mencoba menambahkan fitur lain yang lebih spesifik dibandingkan dengan fitur POS-Tag, yaitu fitur Frasa Kata. Penjelasan lebih lanjut akan dibahas pada sub-eksperimen 5.9.5.

5.9.5 Eksperimen 2.5: Fitur Kata, Kamus Kesehatan, *Stopword*, POS-Tag dan Frasa Kata

Pada sub-eksperimen ini we menambahkan fitur baru yaitu fitur Frasa Kata. Seperti yang telah dijelaskan pada Bab 3, entitas *symptom* dan *treatment* diharapkan akan lebih mudah dikenali karena pada umumnya merupakan frasa kata kerja. Sedangkan entitas *disease* dan *drug* diharapkan juga akan lebih mudah dikenali karena pada umumnya merupakan frasa kata benda.

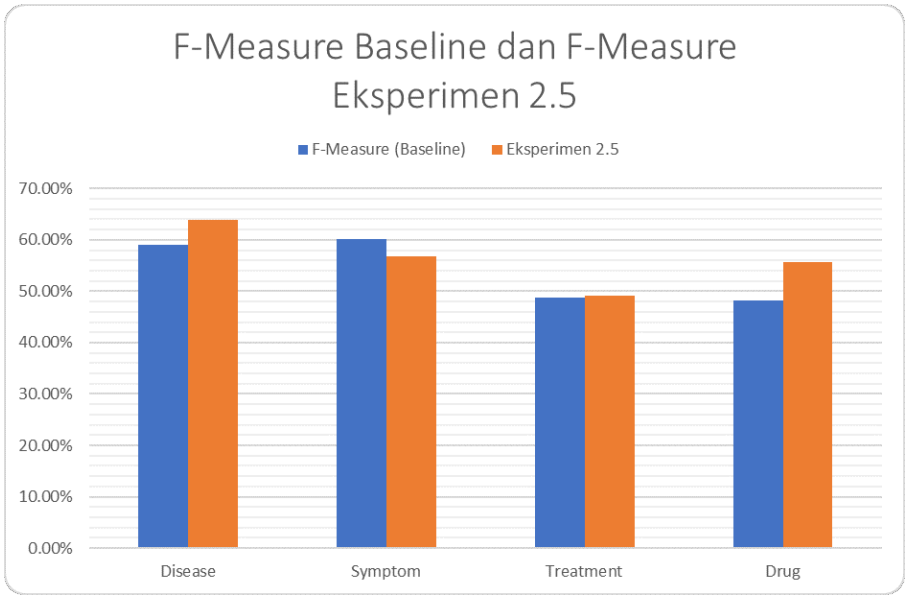
5.9.5.1 Hasil Eksperimen

Waktu komputasi: 7528.5 detik.

Rangkuman hasil sub-eksperimen ini dapat dilihat pada Tabel 5.11 dan Gambar 5.6.

Tabel 5.12: Tabel Hasil Eksperimen 2.5 dibandigkan dengan *Baseline*

Entitas	Baseline (Herwando 2016)			Eksperimen 2.5		
	Precision	*Recall*	*F-Measure*	*Precision*	*Recall*	*F-Measure*
Disease	63.68%	55.45%	59.13%	67.49%	61.56%	63.81%
Symptom	61.43%	59.21%	60.18%	62.89%	52.27%	56.72%
Treatment	53.10%	45.97%	48.82%	54.87%	44.92%	49.06%
Drug	58.99%	44.46%	48.23%	59.77%	53.37%	55.66%
Overall	**59.30%**	**51.27%**	**54.09%**	**61.26%**	**53.03%**	**56.31%**



Gambar 5.7: Histogram Perbandingan *F-measure Baseline* dengan Eksperimen 2.5

5.9.5.2 Analisis

Dari tabel dan grafik di atas, entitas *drug* mengalami penurunan untuk nilai *precission, recall* dan *f-measure*. Selain itu, entitas *disease* mengalami penurunan pada nilai *precission* tetapi mengalami kenaikan pada nilai *recall* dan *f-measure*. Entitas *symptom* mengalami kenaikan pada nilai *precision* tetapi mengalami penurunan pada nilai *recall* dan *f-measure*. Sedangkan pada entitas *treatment*, terjadi kenaikan nilai *recall* tetapi nilai *precision* dan *f-measure* mengalami penurunan.

Dari korpus yang we miliki, berikut merupakan informasi statistik dari penggunaan fitur frasa kata kerja:

1. Untuk entitas *disease*, sebanyak 442 entitas merupakan frasa kata benda, 31 entitas merupakan bagian dari frasa kata benda dan 583 entitas bukan merupakan frasa.

2. Untuk entitas *symptom*, sebanyak 486 entitas merupakan frasa kata benda, 194 entitas merupakan bagian dari frasa kata benda dan 626 entitas bukan merupakan frasa.

3. Untuk entitas *treatment*, sebanyak 338 entitas merupakan frasa kata benda, 76 entitas merupakan bagian dari frasa kata benda dan 401 entitas bukan merupakan frasa.

4. Untuk entitas *drug*, sebanyak 152 entitas merupakan frasa kata benda,

8 entitas merupakan bagian dari frasa kata benda dan 194 entitas bukan merupakan frasa.

Sedangkan berikut merupakan informasi statistik dari penggunaan fitur frasa kata benda:

1. Untuk entitas *disease*, sebanyak 943 entitas merupakan frasa kata benda, 43 entitas merupakan bagian dari frasa kata benda dan 70 entitas bukan merupakan frasa.

2. Untuk entitas *symptom*, sebanyak 842 entitas merupakan frasa kata benda, 363 entitas merupakan bagian dari frasa kata benda dan 101 entitas bukan merupakan frasa.

3. Untuk entitas *treatment*, sebanyak 561 entitas merupakan frasa kata benda, 201 entitas merupakan bagian dari frasa kata benda dan 53 entitas bukan merupakan frasa.

4. Untuk entitas *drug*, sebanyak 318 entitas merupakan frasa kata benda, 14 entitas merupakan bagian dari frasa kata benda dan 22 entitas bukan merupakan frasa.

Dari 2 informasi di atas, dapat diambil informasi bahwa sebagian besar entitas *disease* dan *drug* merupakan frasa kata benda dan entitas *symptom* dan *treatment* merupakan frasa kata kerja. Hal ini menjadi seharusnya menjadi informasi pembeda dengan kata yang bukan merupakan entitas. Namun apabila dilihat dari hasil eksperimen ini, performa penggunaan fitur ini tidak terlalu bagus atau bahkan turun di entitas *disease* dan *symptom* tersebut. We berpendapat hal ini terjadi karena penggunaan fitur ini sudah cukup mewakili informasi fitur POS-Tag, akrena untuk menentukan suatu kata atau kumpulan kata merupakan frasa adalah dengan menggunakan POS-Tag. Selain itu, pada fitur POS-Tag, tidak ada perbedaan antara kata yang merupakan frasa maupun kata yang bukan frasa. Padahal, mayoritas entitas seperti yang telah dijelaskan di atas merupakan frasa. Oleh karena itu, pada sub-eksperimen 5.9.6, penulis menghilangkan fitur POS-Tag dan tetap mempertahankan fitur frasa kata untuk mengetahui hal tersebut.

5.9.6 Eksperimen 2.6: Fitur Kata, Kamus Kesehatan, *Stopword* dan Frasa Kata

Pada sub-eksperimen ini we menghilangkan fitur POS-Tag berdasarkan hasil dan analisis pada sub-eksperimen 5.9.5.

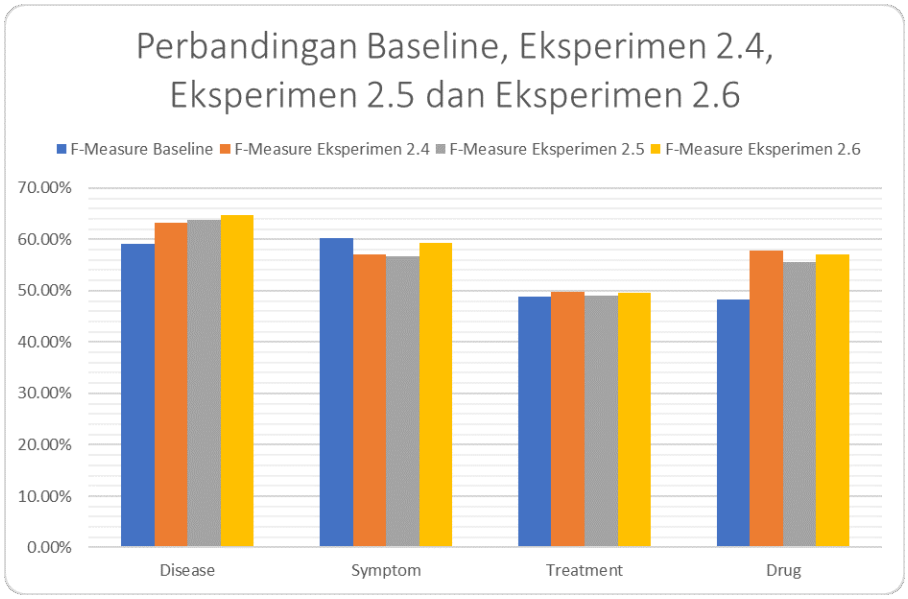
5.9.6.1 Hasil Eksperimen

Waktu komputasi: 6636.5 detik.

Rangkuman hasil sub-eksperimen ini dapat dilihat pada Tabel 5.13 dan Gambar 5.8.

Tabel 5.13: Tabel Hasil Eksperimen 2.6 dibandigkan dengan *Baseline*

Entitas	Baseline (Herwando 2016)			Eksperimen 2.6		
	Precision	*Recall*	*F-Measure*	*Precision*	*Recall*	*F-Measure*
Disease	63.68%	55.45%	59.13%	68.67%	61.80%	64.78%
Symptom	61.43%	59.21%	60.18%	63.79%	56.10%	59.23%
Treatment	53.10%	45.97%	48.82%	54.47%	46.72%	49.58%
Drug	58.99%	44.46%	48.23%	60.08%	56.70%	57.00%
Overall	**59.30%**	**51.27%**	**54.09%**	**61.75%**	**55.33%**	**57.65%**



Gambar 5.8: Histogram Perbandingan *F-measure Baseline* dengan Eksperimen 2.4, 2.5 dan 2.6

5.9.6.2 Analisis

Dari Tabel 5.13 dan Gambar 5.8, terlihat bahwa semua entitas (*disease, symptom, treatment,*) dan *drug* mengalami kenaikan pada nilai *precision, recall,* dan *f-measure*. Seperti yang telah dijelaskan pada sub-eksperimen 5.9.5, penggabungan fitur POS-Tag dan frasa akan memberikan hasil yang lebih rendah. Oleh karena itu, sebaiknya fitur POS-Tag tidak digabung dengan fitur frasa.

Untuk mempermudah dalam membandingkan hasil eksperimen 2.4, 2.5 dan 2.6, we menyajikan grafik 5.8 untuk membandingkan nilai *F-Measure* pada

masing-masing eksperimen. Dapat dilihat bahwa apabila fitur POS-Tag dan Frasa digunakan secara bersama-sama, hasil yang diberikan lebih rendah apabila kedua fitur tersebut dipisah. Namun, apabila dengan menggunakan fitur POS-Tag tanpa Frasa, hasil pada entitas *treatment* dan *drug* lebih bagus. Sedangkan apabila dengan menggunakan fitur Frasa tanpa POS-Tag, hasil pada entitas *disease* dan *symptom* lebih baik. Oleh karena itu we memilih salah satu dari kedua fitur tersebut. Apabila dilihat dari rata-rata *F-Measure*, penggunaan fitur Frasa tanpa POS-Tag memberikan hasil yang paling tinggi (57.65%) apabila dibandingkan dengan penggunaan fitur POS-Tag tanpa Frasa (56.99%). Oleh karena itu, we mempertahankan fitur Frasa dan tidak menggunakan fitur POS-tag pada eksperimen selanjutnya.

Walaupun pada sub-eksperimen ini hasil yang dicapai lebih baik dari sub-eksperimen sebelumnya, hasilnya tetap lebih rendah dari hasil eksperimen Herwando (2016) pada nilai *recall* dan *f-measure* pada entitas *symptom*. Oleh karena itu we mencoba fitur yang lain, yaitu fitur Kata Sebelum. Untuk penjelasan lebih lanjut akan dibahas pada sub-eksperimen 5.9.7.

5.9.7 Eksperimen 2.7: Fitur Kata, Kamus Kesehatan, *Stopword*, Frasa Kata dan Kata Sebelum

Pada sub-eksperimen ini we menambahkan fitur baru yaitu fitur 1 kata sebelum. Fitur ini digunakan pada penelitian Herwando (2016) yang juga berkontribusi memberikan hasil terbaik pada penelitiannya. Menurut we, ada beberapa entitas yang akan lebih mudah diketahui apabila diketahui kata sebelumnya. Misalnya kata "masuk angin", apabila hanya diberikan informasi kata "angin" tanpa kata "masuk", akan lebih sulit menentukan kata tersebut bagian dari suatu entitas *disease* atau bukan. Oleh karena itu, pada sub-eksperimen ini we mencoba menambahkan fitur tersebut.

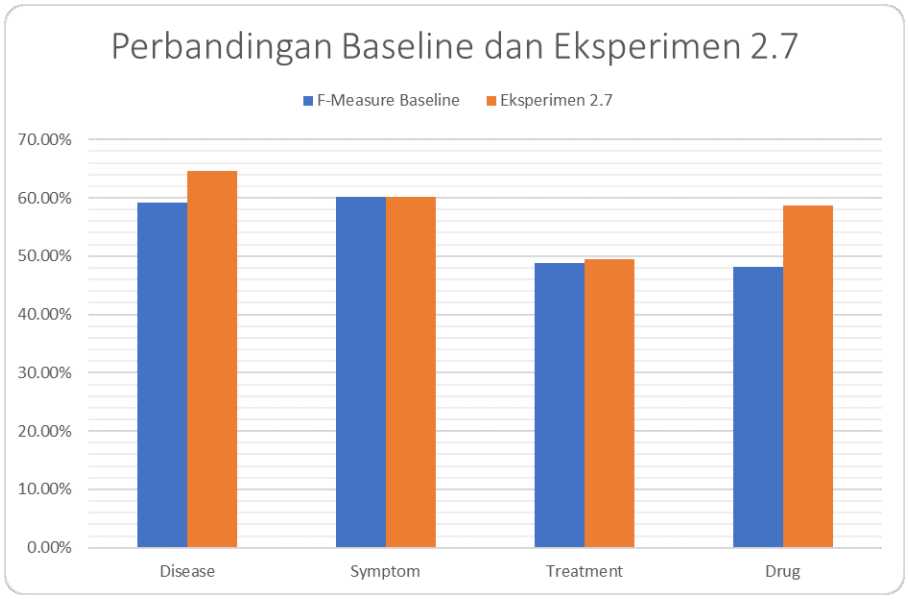
5.9.7.1 Hasil Eksperimen

Waktu komputasi: 9275.5 detik.

Rangkuman hasil sub-eksperimen ini dapat dilihat pada Tabel 5.14 dan Gambar 5.9.

Tabel 5.14: Tabel Hasil Eksperimen 2.7 dibandigkan dengan *Baseline*

Entitas	Baseline (Herwando 2016)			Eksperimen 2.7		
	Precision	*Recall*	*F-Measure*	*Precision*	*Recall*	*F-Measure*
Disease	63.68%	55.45%	59.13%	69.49%	61.60%	64.68%
Symptom	61.43%	59.21%	60.18%	64.78%	57.15%	60.23%
Treatment	53.10%	45.97%	48.82%	56.58%	44.71%	49.54%
Drug	58.99%	44.46%	48.23%	62.22%	57.28%	58.76%
Overall	**59.30%**	**51.27%**	**54.09%**	**63.27%**	**55.19%**	**58.30%**



Gambar 5.9: Histogram Perbandingan *F-measure Baseline* dengan Eksperimen 2.7

5.9.7.2 Analisis

Melihat pada Tabel 5.14 dan Gambar 5.9, dapat diketahui bahwa entitas *disease* dan *treatment* mengalami kenaikan pada nilai *precission*, tetapi mengalami penurunan pada nilai *recall* dan *f-measure*. Sedangkan entitas *symptom* dan *drug* mengalami kenaikan pada nilai *precision, recall,* dan *f-measure*.

Seperti pada penelitian Herwando (2016), fitur ini berhasil meningkatkan performa dari beberapa entitas, karena fitur ini memberikan informasi tambahan kata sebelumnya, misalnya:

- "penyakit", "penderita", "mengalami" dan "mengalami" dapat memberikan informasi mengenai entitas *disease*
- "mengandung", "minum", "pemberian", "obat", "menggunakan" dapat memberikan informasi mengenai entitas *drug*

- "dengan", "melakukan", "dilakukan" dapat memberikan informasi mengenai entitas *treatment*
- "mengalami", "disertai", "sering", "keluhan", "penyebab" dapat memberikan informasi mengenai entitas *symptom*

Hasil sub-eksperimen ini masih lebih rendah dibandingkan dengan hasil eksperimen Herwando (2016) pada *recall* dan *f-measure* entitas *treatment*. Oleh karena itu, we mencoba menambahkan fitur yang lain yaitu fitur 1 Kata sesudah, yang akan dibahas lebih lanjut pada sub-eksperimen 5.9.8.

5.9.8 Eksperimen 2.8: Fitur Kata, Kamus Kesehatan, *Stopword*, Frasa Kata, Kata Sebelum dan Kata Sesudah

Pada sub-eksperimen ini we menambahkan fitur lain yaitu fitur 1 Kata Setelah. Hal ini karena ada beberapa kasus yang mana apabila suatu kata merupakan sebuah entitas, akan lebih mudah dikenali apabila melihat kata atau konteks setelahnya. Sama seperti contoh pada fitur 1 kata sebelum, misal diberikan kata "masuk angin", apabila hanya diberikan informasi "masuk" tanpa "angin", akan lebih sulit mengenali apakah kata tersebut termasuk entitas *disease* atau bukan. Selain itu, fitur ini juga dapat membedakan kata berentitas dengan kata yang bukan, misalnya kata "masuk angin" dengan "masuk rumah". Apabila informasi pada saat tersebut hanya diberikan kata "masuk" saja tanpa kata setelahnya, akan lebih sulit mengenali kata tersebut termasuk kata berentitas atau bukan.

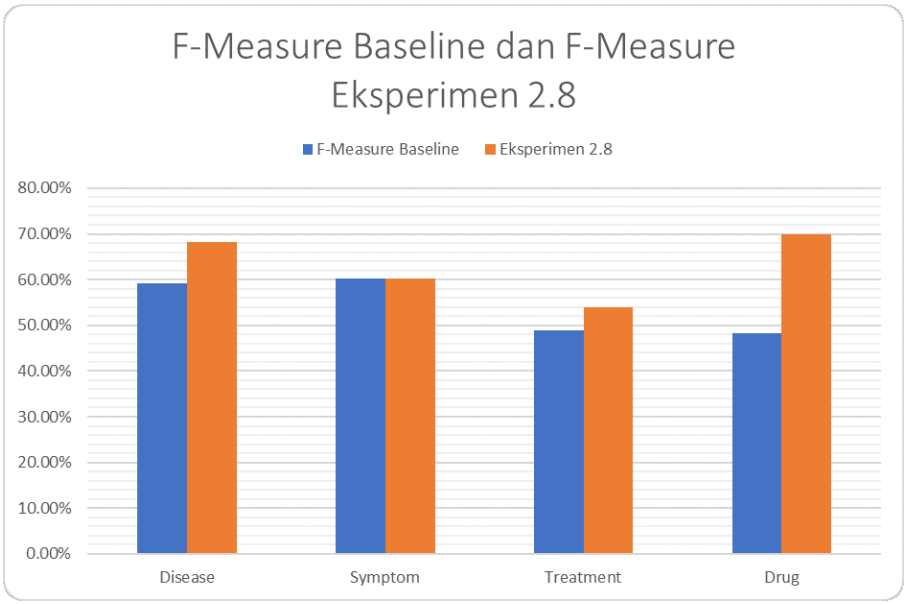
5.9.8.1 Hasil Eksperimen

Waktu komputasi: 14031.5 detik.

Rangkuman hasil sub-eksperimen ini dapat dilihat pada Tabel 5.15 dan Gambar 5.10.

Tabel 5.15: Tabel Hasil Eksperimen 2.8 dibandigkan dengan *Baseline*

Entitas	Baseline (Herwando 2016)			Eksperimen 2.7		
	Precision	*Recall*	*F-Measure*	*Precision*	*Recall*	*F-Measure*
Disease	63.68%	55.45%	59.13%	70.68%	66.18%	68.17%
Symptom	61.43%	59.21%	60.18%	64.16%	59.55%	60.23%
Treatment	53.10%	45.97%	48.82%	61.02%	51.13%	54.03%
Drug	58.99%	44.46%	48.23%	70.85.%	70.33%	69.82%
Overall	**59.30%**	**51.27%**	**54.09%**	**65.29%**	**61.80%**	**63.06%**



Gambar 5.10: Histogram Perbandingan *F-measure Baseline* dengan Eksperimen 2.8

5.9.8.2 Analisis

Melihat pada Tabel 5.15 dan Gambar 5.10, dapat diketahui bahwa hanya entitas *symptom* yang mengalami penurunan nilai pada *precision*, tetapi nilai *recall* dan *f-measure*-nya naik. Sedangkan entitas lain mengalami kenaikan pada nilai *precision, recall* dan *f-measure*. Oleh karena itu, setelah we mencoba kemungkinan fitur yang memberikan kontribusi dalam penelitian ini, we mencoba arsitektur untuk model RNNs yang lain. Penjelasan lebih lanjut akan dibahas pada eksperimen 5.10.

5.10 Skenario 3: Skenario Pengujian Arsitektur RNNs

Pada eksperimen ini, we mencoba dua buah arsitektur RNNs yang telah we usulkan pada Bab 3 yaitu RNNs dengan 1 layer dan RNNs dengan 2 layer. Fitur yang digunakan dalam pengujian ini yaitu kombinasi fitur yang menghasilkan akurasi terbaik pada eksperimen pertama, yaitu fitur kata itu sendiri, kamus kesehatan, *stop word*, frasa kata, 1 kata sebelum dan 1 kata sesudah.

5.10.1 Eksperimen 3.1: Menguji Arsitektur LSTMs 1 Layer

Pada sub-eksperimen ini, we menggunakan struktur RNNs yang mana semua fitur digabung menjadi satu dalam sebuah *timestep*. Artinya fitur-fitur yang berbeda tersebut akan digabung atau di-*concat* menjadi sebuah vektor yang akan menjadi

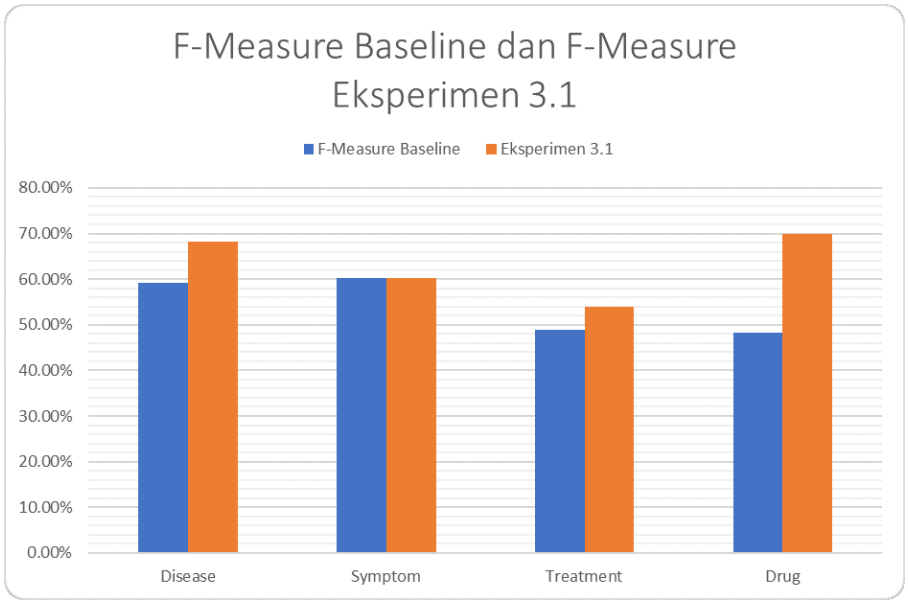
input bagi LSTMs ini. LSTMs inilah yang digunakan pada eksperimen pertama, sehingga hasilnya sama dengan sub-eksperimen 5.9.8.

5.10.1.1 Hasil Eksperimen

Waktu komputasi: 14031.5 detik.

Tabel 5.16: Tabel Hasil Eksperimen 3.1 dibandigkan dengan *Baseline*

Entitas	Baseline (Herwando 2016)			Eksperimen 2.7		
	Precision	*Recall*	*F-Measure*	*Precision*	*Recall*	*F-Measure*
Disease	63.68%	55.45%	59.13%	70.68%	66.18%	68.17%
Symptom	61.43%	59.21%	60.18%	64.16%	59.55%	60.23%
Treatment	53.10%	45.97%	48.82%	61.02%	51.13%	54.03%
Drug	58.99%	44.46%	48.23%	70.85.%	70.33%	69.82%
Overall	**59.30%**	**51.27%**	**54.09%**	**65.29%**	**61.80%**	**63.06%**



Gambar 5.11: Histogram Perbandingan *F-measure Baseline* dengan Eksperimen 3.1

5.10.1.2 Analisis

Pada eksperimen ini, hasil yang sudah lebih baik apabila dibandingkan dengan hasil yang dicapai Herwando (2016) di entitas. Namun, dari eksperimen sebelumnya, terdapat akurasi yang turun, yaitu nilai *precision* untuk entitas *symptom*. Menurut we hal ini terjadi karena informasi fitur yang berbeda-beda dijadikan satu, sehingga ada kemungkinan hilangnya informasi dari masing-masing fitur tersebut. Oleh

karena itu, untuk mengatasi permasalahan tersebut we mengusulkan arsitektur yang mana masing-masng kelompok fitur yang berbeda dipisahkan dan menjadi *input* bagi masing-masing LSTMs. Untuk penjelasan eksperimen ini akan dijelaskan pada sub-eksperimen 5.10.2

5.10.2 Eksperimen 3.2: Menguji Arsitektur LSTMs 2 Layer Multi-Input

Pada sub-eksperimen sebelumnya, fitur-fitur yang berbeda digabung menjadi satu, sehingga ada kemungkinan hilangnya informasi dari fitur tersebut. Oleh karena itu, we mengusulkan adanya layer tambahan setelah masing-masing fitur tersebut masuk ke dalam model. We mengusulkan bahwa masing-masing kelompok fitur menjadi *input* LSTMs secara terpisah. Setelah masuk di RNNs, *output* dari masing-masing LSTMs tersebut di-*merge* ke dalam sebuah layer, lalu masuk kembali ke LSTMs untuk melihat konteks fitur-fitur sebelumnya. Dengan diusulkannya arsitektur RNNs ini we berharap bahwa masing-masing fitur terjaga informasinya dan tidak terganggu dengan informasi lain.

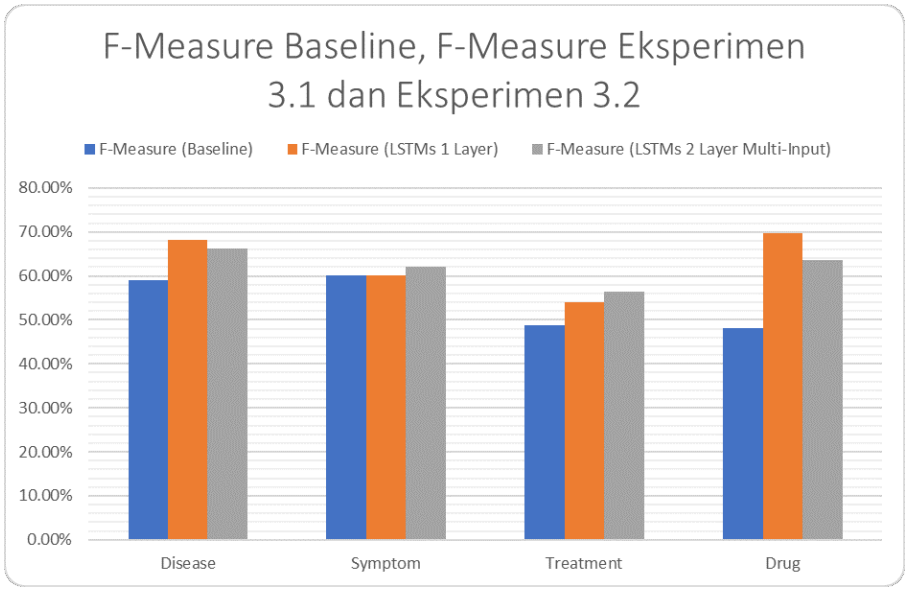
5.10.2.1 Hasil Eksperimen

Waktu komputasi: 20362.5 detik.

Rangkuman hasil sub-eksperimen ini dapat dilihat pada Tabel 5.17 dan Gambar 5.12.

Tabel 5.17: Tabel Hasil Eksperimen 3.2 dibandigkan dengan *Baseline*

	Precision	*Recall*	*F-Measure*
Disease	67.47%	67.19%	66.31%
Symptom	64.90%	60.63%	62.13%
Treatment	63.92%	53.13%	56.51%
Drug	66.39%	62.33%	63.61%
Overall	**65.67%**	**60.82%**	**62.14%**



Gambar 5.12: Histogram Perbandingan *F-measure Baseline* dengan Eksperimen 3.1, dan 3.2

5.10.2.2 Analisis

Pada eksperimen ini, hasil yang sudah lebih baik apabila dibandingkan dengan hasil yang dicapai Herwando (2016) di masing-masing entitas dan lebih baik dibandingkan eksperimem 5.10.1 pada identifikasi entitas *symptom* dan *treatment*.

BAB 6
CONCLUSIONS

6.1 Kesimpulan

Semantic Role Labeling (SRL) is an integral part of understanding semantic information of a text. One of its applications is to make chat bots understand user's chat better and thus, it can provide more engaging answers. Even though the SRL on formal language has been widely studied, the conversational language used on chatting platform is barely tapped. In this work, we introduce a new set of semantic roles for conversational language and propose a new model architecture called Context-Aware Bi-Directional Long Short-Term Memories (CA-BLSTM). CA-BLSTM adds an attention mechanism on top of the BLSTM layer, by collecting context information from all the words in a sentence and representing it as a vector that is concatenated to all the output of BLSTM.

We conducted two set of experiment scenarios, which are evaluating feature combinations and architectures. Our experiments on feature combination show that when the size of training data is relatively small, one still needs to use traditional feature such as POS tag in addition to word embedding. For the experiment on architecture, the results show that our proposed architecture, CA-BLSTM, can outperforms the original BLSTM. Based on the increasing result of all precision, recall, and F1, we suggest that our architecture successfully extracts context information at higher level. Since it becomes more context-aware, our analysis shows that CA-BLSTM can predict the labels more carefully.

For future works, once the SRL system is established, one can focus on building the Natural Language Generation (NLG) system for chat bots based on the semantic roles of the conversational language. This way, we can create more intelligent chat bots which understand deeper on conversational language. Another interesting work would be integrating coreference resolution on the SRL system knowing that conversational language is usually in a form of dialogues.

Terkait dengan rumusan masalah pertama, setelah dilakukan penelitian secara garis besar didapatkan kesimpulan bahwa model RNNs yang dihasilkan mampu memberikan performa yang lebih baik dibandingkan dengan model CRF (*baseline*) pada penelitian Herwando (2016). Dari penggunaan fitur kata itu sendiri saja, model RNNs sudah memberikan performa yang lebih baik dengan nilai *F-Measure* dan

recall yang lebih tinggi.

Terkait dengan rumusan masalah kedua, didapatkan kesimpulan lain bahwa fitur kata itu sendiri, kamus kesehatan, *stop word*, frasa Kata, 1 kata sebelum dan 1 kata sesudah memberikan hasil yang terbaik, yaitu dengan rata-rata *f-measure* 63.06% (*disease* 68.17%, *symptom* 61.42%, *treatment* 68.17% dan *drug* 68.17%).

Dua arsitektur yang diusulkan memiliki kelebihan masing-masing. Untuk arsitektur LSTMs dengan 1 layer, *f-measure* sama dengan percobaan untuk mendapatkan fitur terbaik, karena eksperimen tersebut menggunakn arsitektur LSTMs yang sama. Sedangkan untuk arsitektur kedua (LSTMs 2 layer), rata-rata *f-measure* yang didapatkan adalah 62.14%. LSTMs pertama memiliki nilai *f-measure* pada entitas *disease* dan *drug* yang lebih bagus, yaitu masing-masing 68.17% dan 69.82%. Sedangkan LSTMs kedua memiliki nilai *f-measure* pada entitas *symptom* 62.13% dan *treatment* 56.51%. Namun, apabila dilihat dari waktu komputasi, LSTMs pertama lebih baik dibandingkan LSTMs kedua.

LSTMs pertama tidak bisa dikatakan lebih baik dibandingkan LSTMs kedua dan begitu pula sebaliknya, karena hasil yang diperoleh mengatakan bahwa masing-masing arsitektur memiliki hasil yang lebih baik di beberapa macam entitas. Namun, arsitektur ini mampu memberikan hasil yang lebih baik dari hasil penelitian Herwando (2016). Hal ini akan menarik apabila *resource* semakin diperbesar, apakah tetap LSTMs 1 layer lebih baik, karena LSTMs 2 layer memiliki parameter lebih banyak, sehingga mampu menyimpan informasi yang lebih besar.

6.2 Saran

Setelah melakukan eksperimen dan menganalisis hasilnya, ada beberapa saran untuk penelitian selanjutnya, antara lain sebagai berikut.

1. Penelitian ini hanya menggunakan 309 *post* forum kesehatan *online* sehingga perlu penambahan data *training* dan *testing* mengingat *deep learning* membutuhkan data yang besar dalam melakukan *training* untuk mendapatkan model yang baik.

2. Terdapat beberapa parameter bebas seperti dalam pembuatan model *word embedding* yaitu panjang *window* dan *vector*. Hal ini bisa menjadi bahan penelitian lanjutan untuk mendapatkan panjang *window* dan *vector* yang tepat supaya model mampu memberikan akurasi yang lebih baik.

3. Dalam menentukan label entitas, we tidak mempertimbangkan konteks kalimat yang berada di sekitarnya. Padahal kalimat di sekitarnya akan

memberikan informasi lebih terkait hubungan antar entitas. Misalnya pada kalimat pertama dokter menjelaskan penyakit yang dialami. Pada kalimat selanjutnya dokter tersebut menjelaskan cara penyembuhan dari penyakit tersebut. Oleh karena itu, we menyarankan untuk mempertimbangkan fitur konteks kalimat pada penelitian selanjutnya.

4. Perlu dibuat korpus dengan jumlah masing-masing entitas yang seimbang, sehingga hasil yang diberikan tidak bias.

5. Sebaiknya, pelabelan dokumen secara manual melibatkan pihak yang ahli di bidangnya (dalam hal ini dokter, perawat, apoteker, atau mahasiswa di bidang kesehatan) supaya label yang diberikan lebih tepat.

6. Sama seperti pada penelitian Herwando (2016), sebaiknya dibuat model POS-Tagger yang khusus di bidang kesehatan, sehingga pemberian tag pada dokumen kesehatan lebih tepat.

DAFTAR REFERENSI

Abacha, A. B. dan Zweigenbaum, P. (2011). Medical entity recognition: A comparison of semantic and statistical methods. In *Proceedings of BioNLP 2011 Workshop*, pages 56–64. Association for Computational Linguistics.

Almgren, S., Pavlov, S., dan Mogren, O. (2016). Named entity recognition in swedish health records with character-based deep bidirectional lstms. *BioTxtM 2016*, page 30.

Bakker, B. (2001). Reinforcement learning with long short-term memory. In *NIPS*, pages 1475–1482.

Bengio, Y., LeCun, Y., et al. (2007). Scaling learning algorithms towards ai. *Large-scale kernel machines*, 34(5).

Bengio, Y., Simard, P., dan Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.

Carreras, X. dan Màrquez, L. (2005). Introduction to the conll-2005 shared task: Semantic role labeling. In *Proceedings of the Ninth Conference on Computational Natural Language Learning*, pages 152–164. Association for Computational Linguistics.

Chollet, F. (2015). keras. https://github.com/fchollet/keras.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., dan Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537.

Dowty, D. (1991). Thematic proto-roles and argument selection. *language*, pages 547–619.

Elman, J. L. (1990). Finding structure in time. *Cognitive science*, 14(2):179–211.

Emanuele, B., Castellucci, G., Croce, D., dan Basili, R. (2013). Textual inference and meaning representation in human robot interaction. In *Joint Symposium on Semantic Processing.*, page 65.

Gildea, D. dan Jurafsky, D. (2002). Automatic labeling of semantic roles. *Computational linguistics*, 28(3):245–288.

Gildea, D. dan Palmer, M. (2002). The necessity of parsing for predicate argument recognition. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 239–246. Association for Computational Linguistics.

Goodfellow, I., Bengio, Y., dan Courville, A. (2016). Deep learning. Book in preparation for MIT Press.

Graves, A. (2012). Neural networks. In *Supervised Sequence Labelling with Recurrent Neural Networks*, pages 15–35. Springer.

Graves, A., Mohamed, A.-r., dan Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE.

Hachey, B., Radford, W., Nothman, J., Honnibal, M., dan Curran, J. R. (2013). Evaluating entity linking with wikipedia. *Artificial intelligence*, 194:130–150.

Haykin, S. S., Haykin, S. S., Haykin, S. S., dan Haykin, S. S. (2009). *Neural networks and learning machines*, volume 3. Pearson Upper Saddle River, NJ, USA:.

He, L., Lee, K., Lewis, M., dan Zettlemoyer, L. (2017). Deep semantic role labeling: What works and whatâĂŹs next. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Herwando, R. (2016). Pengenalan entitas kesehatan pada forum kesehatan online berbahasa indonesia menggunakan algoritma conditional random fields. Bachelor's thesis, Universitas Indonesia, Kampus UI Depok.

Hinton, G. E., Osindero, S., dan Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554.

Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, page 91.

Hochreiter, S., Bengio, Y., Frasconi, P., dan Schmidhuber, J. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.

Hochreiter, S. dan Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Jagannatha, A. N. dan Yu, H. (2016). Bidirectional rnn for medical event detection in electronic health records. In *Proceedings of the conference. Association for Computational Linguistics. North American Chapter. Meeting*, volume 2016, page 473. NIH Public Access.

Jordan, M. I. (1986). Attractor dynamics and parallellism in a connectionist sequential machine.

Lang, K. J., Waibel, A. H., dan Hinton, G. E. (1990). A time-delay neural network architecture for isolated word recognition. *Neural networks*, 3(1):23–43.

LeCun, Y., Bengio, Y., dan Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

Limsopatham, N. dan Collier, N. (2016). Learning orthographic features in bi-directional lstm for biomedical named entity recognition. *BioTxtM 2016*, page 10.

Liu, D. dan Gildea, D. (2010). Semantic role features for machine translation. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 716–724. Association for Computational Linguistics.

Lo, C.-k., Addanki, K., Saers, M., dan Wu, D. (2013). Improving machine translation by training against an automatic semantic frame based evaluation metric. In *ACL (2)*, pages 375–381.

Mikolov, T., Chen, K., Corrado, G., dan Dean, J. (2014). word2vec.

Mikolov, T., Karafiát, M., Burget, L., Cernockỳ, J., dan Khudanpur, S. (2010). Recurrent neural network based language model. In *Interspeech*, volume 2, page 3.

Moschitti, A., Morarescu, P., dan Harabagiu, S. M. (2003). Open domain information extraction via automatic semantic labeling. In *FLAIRS conference*, pages 397–401.

Mozer, M. C., Jordan, M. I., dan Petsche, T. (1997). *Advances in Neural Information Processing Systems 9: Proceedings of the 1996 Conference*. Mit Press.

Mujiono, S., Fanany, M. I., dan Basaruddin, C. (2016). A new data representation based on training data characteristics to extract drug named-entity in medical text. *arXiv preprint arXiv:1610.01891*.

Pennington, J., Socher, R., dan Manning, C. D. (2014). Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–43.

Pradhan, S., Ward, W., Hacioglu, K., Martin, J. H., dan Jurafsky, D. (2005). Semantic role labeling using different syntactic views. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 581–588. Association for Computational Linguistics.

Punyakanok, V., Roth, D., dan Yih, W.-t. (2008). The importance of syntactic parsing and inference in semantic role labeling. *Computational Linguistics*, 34(2):257–287.

Saeed, J. (1997). I. 2003. semantics. *GB: Blackwell Publishing*.

Schmidhuber, J., Wierstra, D., Gagliolo, M., dan Gomez, F. (2007). Training recurrent networks by evolino. *Neural computation*, 19(3):757–779.

Seki, K. dan Mostafa, J. (2003). A probabilistic model for identifying protein names and their name boundaries. In *Bioinformatics Conference, 2003. CSB 2003. Proceedings of the 2003 IEEE*, pages 251–258. IEEE.

Shen, D. dan Lapata, M. (2007). Using semantic roles to improve question answering. In *EMNLP-CoNLL*, pages 12–21.

Surdeanu, M., Harabagiu, S., Williams, J., dan Aarseth, P. (2003). Using predicate-argument structures for information extraction. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 8–15. Association for Computational Linguistics.

Suwarningsih, W., Supriana, I., dan Purwarianti, A. (2014). Imner indonesian medical named entity recognition. In *Technology, Informatics, Management, Engineering, and Environment (TIME-E), 2014 2nd International Conference on*, pages 184–188. IEEE.

Turian, J., Ratinov, L., dan Bengio, Y. (2010). Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 384–394. Association for Computational Linguistics.

Zhou, J. dan Xu, W. (2015). End-to-end learning of semantic role labeling using recurrent neural networks. In *ACL (1)*, pages 1127–1137.

LAMPIRAN