BAB 2
LANDASAN TEORI

2.1. *Natural Language Processing*

Natural Language Processing (NLP) adalah sebuah sub-bidang studi dari ilmu komputer yang menjadi penghubung antara bahasa alami dengan komputer. NLP membantu memberdayakan mesin untuk dapat memahami, memproses, dan mengalisis bahasa manusia [14]. *Natural Language Processing* mendalami kegunaan komputer agar dapat memproses atau memahami bahasa manusia (bahasa alami) yang bertujuan untuk melakukan tugas-tugas yang bermanfaat. NLP adalah sebuah bidang disiplin ilmu yang menggabungkan antara ilmu linguistik komputasi, ilmu komputasi, ilmu kognitif dan kecerdasan buatan. Dari sudut pandang sains tujuan NLP adalah untuk memodelkan mekanisme kognitif yang mendasari pemahaman serta memproduksi bahasa manusia. Sementara dari sudut pandang teknik, NLP memperhatikan bagaimana cara untuk mengembangkan aplikasi baru yang praktis untuk memfasilitasi interaksi antara komputer dengan bahasa manusia. Jenis-jenis aplikasi dalam NLP contohnya adalah *speech recognition, spoken language understanding, dialogue systems, lexical analysis, parsing, machine translation, knowledge graph, information retrieval, question answering, sentiment analysis, social computing, natural language generation,* dan *natural language summarization* [15].

2.2. Analisis Sentimen

Analisis sentimen atau juga biasa disebut sebagai *opinion mining* merupakan bidang studi yang menganalisis pendapat, sentimen, penilaian, sikap dan emosi dari orang-orang terhadap entitas dan atributnya yang diekspresikan melalui teks. Entitas bisa saja berupa produk, pelayanan, organisasi, individual, acara, isu, maupun topik. Bidang ini merepresentasikan sebuah ruang permasalahan yang besar. Aplikasi dan penelitian dari analisis sentimen fokus pada teks tertulis, analisis sentimen telah menjadi penelitian aktif di bidang *natural language*

processing (NLP). Analisis sentimen utamanya berfokus pada opini-opini yang mengungkapkan atau menunjukkan sentimen yang positif atau negatif pada teks. Riset pada analisis sentimen utamanya dibagi menjadi tiga tingkat kedetailan, yaitu: *document level, sentence level, dan aspect level* [1]:

1. Level Dokumen (*Document Level)*

   Tugas pada level dokumen adalah untuk mengklasifikasi apakah keseluruhan opini dalam dokumen mengekspresikan sentimen positif atau negatif. Analisis level ini secara tersirat berasumsi bahwa setiap dokumen mengekspresikan pendapat pada satu entitas (misalnya produk atau isu). Jadi tidak dapat diterapkan pada dokumen yang mengevaluasi atau membandingkan entitas lebih dari satu.

2. Level Kalimat (*Sentence Level)*

   Level kalimat untuk menentukan apakah setiap kalimat mengekspresikan sebuah opini yang positif, negatif, atau netral. Level analisis ini sangat berhubungan dengan *subjectivity classification*, yang membedakan kalimat objektif dari sebuah kalimat yang diekspresikan secara subjektif berdasarkan pandangan dan opini pribadi yang mengutara pendapat tersebut. Meskipun subjektifitas tidak sama dengan sentimen, karena banyaknya kalimat objektif dapat menyiratkan sentimen atau pendapat.

3. Level Aspek (*Aspect Level)*

   Level dokumen maupun level kalimat tidak dapat menemukan apa yang orang sukai dan tidak sukai secara tepat. Mereka tidak menyebutkan apa target dari opini yang diungkapkan. Misalnya terdapat kalimat, “Saya menyukai Iphone 5” itu sentimennya positif, itu maknanya terbatas kecuali, jika pendapat positif Iphone 5 itu diketahui. Mungkin dapat dikatakan bahwa jika manusia dapat mengklasifikasi sebuah kalimat harus positif, semuanya dalam kalimat dapat disimpulkan sebagai opini positif. Namun dengan cara itu tidak berhasil, karena kalimat bisa mempunyai opini yang bernilai ganda. *Aspect-level analysis* ini yang dibutuhkan aplikasi dan hampir setiap semua kehidupan nyata sistem analisis sentimen yang ada di industri yaitu berdasarkan level analisis ini.

Tujuan level analisis ini adalah untuk menemukan sentimen pada entitas dan/atau aspeknya.

2.2.1. Analisis Sentimen Berbasis Aspek

Analisis Sentimen Berbasis Aspek (ABSA) membantu untuk memahami permasalahan pada Analisis sentimen lebih baik secara perbandingan dikarenakan ABSA secara langsung fokus terhadap sentimen daripada struktur bahasa. Dalam ABSA, sentimen merupakan kesadaran subjektif manusia mengenai suatu aspek [2]. Analisis Sentimen Berbasis Aspek pada awalnya disebut sebagai *feature-based opining mining*. Tujuan ABSA adalah untuk menemukan sentimen pada entitas dan/atau aspeknya. ABSA mempunyai dua tugas utama yaitu *Aspect exraction* dan *Aspect sentiment classification* [1].

1. *Aspect extraction*

   Aspect extraction mempunyai tugas untuk mengekstraksi aspek dan entitas yang telah dievaluasi, Misal pada kalimat "*The voice quality of this phone is amazing*", berarti harus mengekstrak *voice quality* sebagai aspek dari entitas yang direpresentasikan oleh "*this phone*". Simpelnya manusia sering mengabaikan diskusi yang dilakukan dan hanya berfokus pada aspek, namun setiap kali berbicara mengenai aspek, maka harus tahu entitas ini masuk dalam golongan yang mana, aspek tidaklah berarti. Karenanya *aspect extraction* mencakup *entity extraction*. Garis bawahi bahwa "*this phone*" tidak mengindikasikan aspek secara keseluruhan (*General*).

2. *Aspect sentiment classification*

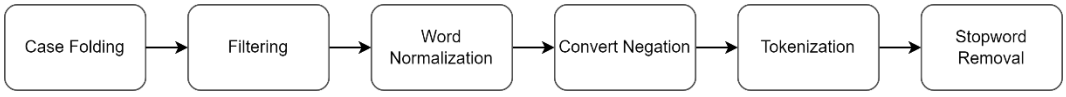
   Aspect sentiment classification tugasnya menentukan apakah opini pada aspek yang berbeda memiliki polaritas positif, negatif atau netral. Contohnya seperti kalimat "*The voice quality of this phone is amazing*", opini mengenai aspek *voice quality* bernilai positif, Lalu "*I love this phone"* opini ini mengenai aspek *general* (keseluruhan entitas) juga positif.

Pendekatan dalam level aspek klasifikasi sentimen ada dua, yaitu pendekatan *supervised learning* dan pendekatan *lexicon-based*. Untuk pendekatan

Supervised Learning menggunakan algoritma *machine learning* seperti, *Support Vector Machine* (SVM), Naïve bayes dan bahkan *Neural Network*, jenis pendekatan ini digunakan untuk klasifikasi sentimen level kalimat dan level klausa tidak lagi memadai. Kuncinya adalah fitur-fitur tersebut tidak memperhatikan dan tidak dapat menentukan manakah target opini (entitas dan/atau aspek) yang diacu. Lalu selanjutnya *Lexicon-Based Approach*. *Lexicon-Based Approach* pendekatannya tidak sama seperti klasifikasi sentiment level dokumen ataupun level kalimat. Kunci perbedaannya adalah butuh secara eksplisit mempertimbangkan target opini, yang mana tidak ada dalam *document-level* maupun *sentence-level*. Untuk memperhatikan target opini, kedua pendekatan yang telah disebutkan dapat digunakan [1].

2.3. *Preprocessing*

Preprocessing adalah tahapan yang dilakukan untuk memproses data teks agar menjadi lebih bersih, baik, dan terstruktur [4], [11]. *Preprocesing* teks merupakan tahap yang sangat penting khususnya dalam klasifikasi teks, umumnya *text mining*. *Preprocessing* teks merupakan teknik yang digunakan untuk mengurangi beberapa bentuk kata menjadi satu bentuk. Tujuan *preprocessing* teks adalah untuk merepresentasikan setiap dokumen sebagai sebuah vektor fitur yang memecah teks menjadi kata-kata terpisah [16]. *Preprocessing* juga digunakan untuk menghasilkan sentimen yang tepat agar efektif dalam pengambilan keputusan, dan juga diimplementasikan untuk menghilangkan sifat yang tidak terstruktur dari data yang diperoleh dari media sosial [17]. Tahapan pada *preprocessing* yang dilakukan terdiri dari *Case folding, filtering, word normalization, convert negation, tokenization,* dan *stopword removal*. Dapat dilihat pada gambar 2.1.

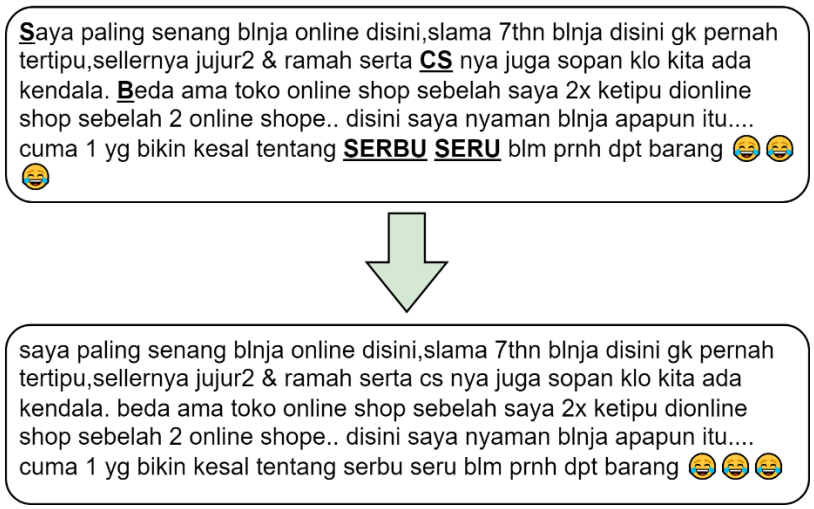


Gambar 2.1. Tahap *Preprocessing*

A. *Case Folding*

Case folding adalah tahapan yang dilakukan untuk menyeragamkan keseluruhan huruf menjadi huruf kecil atau huruf besar karena

beragamnya huruf yang ditulis dalam ulasan [18]. Contoh *case folding* dapat dilihat pada Gambar 2.2.

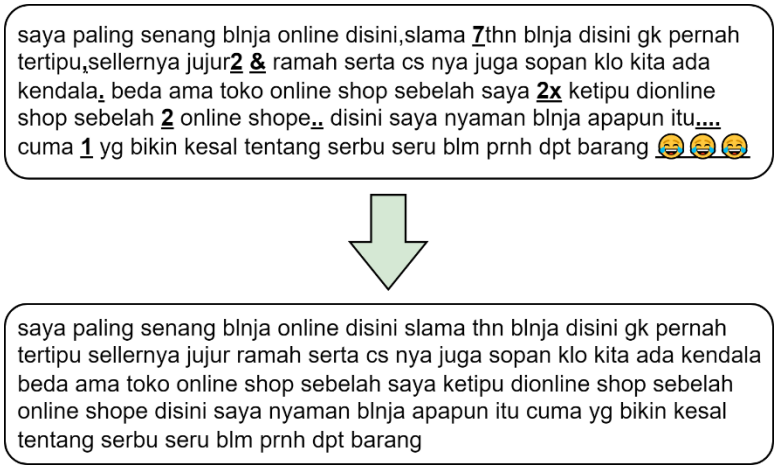


Gambar 2.2. Contoh Tahap *Case Folding*

Pada Gambar 2.2 tersebut kata “Saya”, “CS”, “Beda”, “SERBU”, dan “SERU” berubah menjadi huruf kecil pada proses *case folding* ini.

B. *Filtering*

Filtering adalah tahapan proses dalam menghapus simbol-simbol seperti tanda baca, angka, dan emoji yang tidak digunakan dalam kalimat. Proses ini dilakukan untuk menghilangkan gangguan (*noise*) dalam kalimat yang dapat mengganggu proses utama [3]. Contoh *filtering* dapat dilihat pada Gambar 2.3.

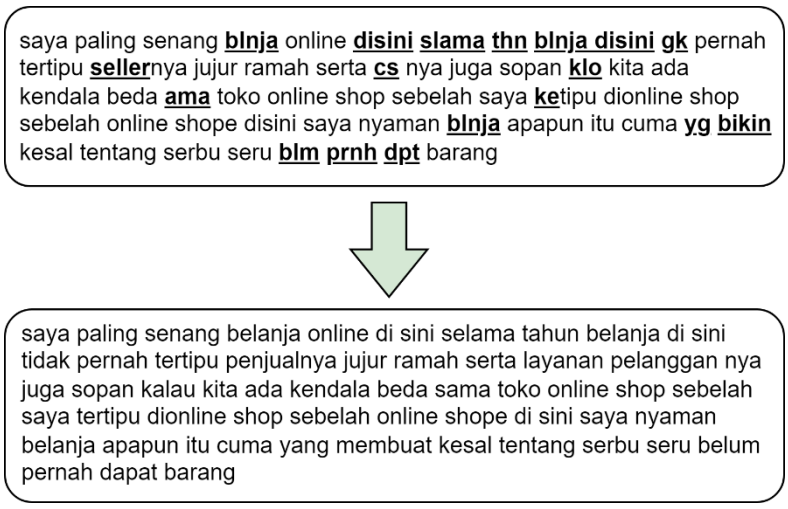


Gambar 2.3. Contoh Tahap *Filtering*

Pada Gambar 2.3 tanda baca seperti koma (,) tanda *ampersand* (&), dan titik (.) dibuang, angka (0-9) dibuang, emoji seperti diatas dihilangkan, dan satu karakter seperti “x” dihilangkan.

C. *Word Normalization*

Word normalization adalah tahap proses untuk mengubah kata-kata informal, kesalahan penulisan, kata singkatan, dan bahasa gaul menjadi bentuk kata formal. Sehingga dalam proses ini membutuhkan kamus [3], [6]. Contoh pada *word normalization* dapat dilihat pada Gambar 2.4.



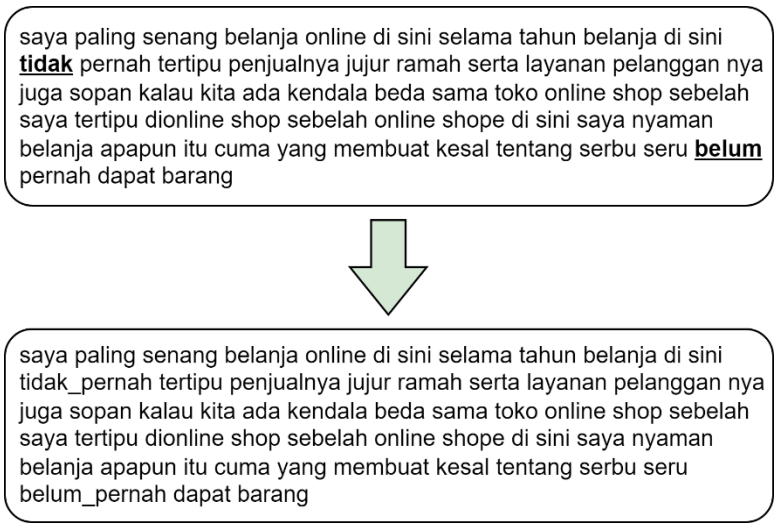
Gambar 2.4. Contoh *Word Normalization*

Pada Gambar 2.4 kata seperti “blnja”, “disini”, “slama”, “thn”, “gk”, “cs”, “klo”, “ama”, “yg”, “bikin”, “blm”, “prnh”, dan “dpt” diubah menjadi bentuk kata formal “belanja”, “di”, “sini” “selama”, “tahun”, “tidak”, “layanan pelanggan”, “kalau”, “sama”, “yang”, “membuat”, “belum”, “pernah”, dan “dapat” karena bentuknya informal atau bahasa gaul.

D. *Convert Negation*

Convert Negation adalah tahap proses menkonversi kata-kata bernilai negatif dalam sebuah teks seperti “tidak”, “bukan”, “belum”, “jangan”, “kurang”, dan lain-lain. Karena kata negasi dalam sebuah teks ataupun opini dapat merubah atau memutar maknanya, maka dari itu kata negasi

perlu digabungkan dengan kata berikutnya [18], [19]. Contoh pada *convert negation* dapat dilihat pada gambar 2.5.

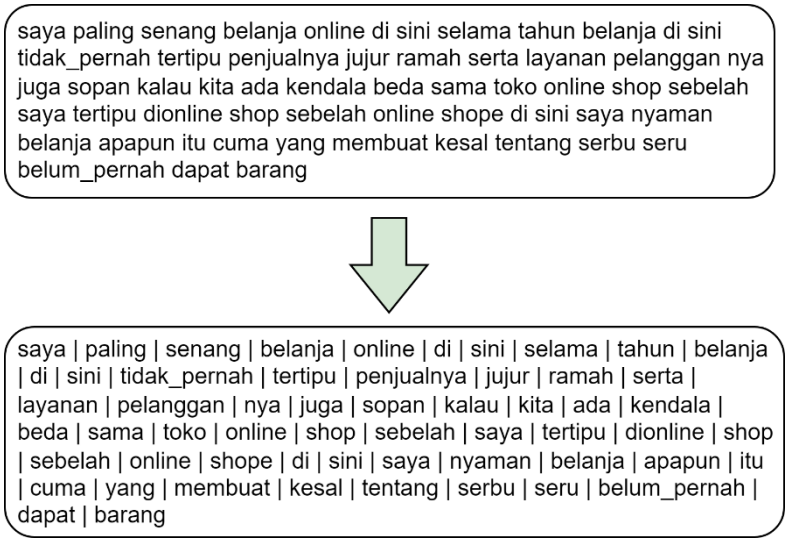


Gambar 2.5. Contoh *Convert Negation*

Pada gambar 2.5 kata “tidak” dan “belum” menjadi tergabung dengan kata setelahnya.

E. *Tokenization*

Tokenization adalah tahap proses memangkas kalimat menjadi kata demi kata, proses ini melakukan pemangkasan berdasarkan spasi yang ada pada kalimat [3], [17]. *Token* adalah sebutan dari hasil proses ini [18]. Contoh pada *tokenization* dapat dilihat pada Gambar 2.6.

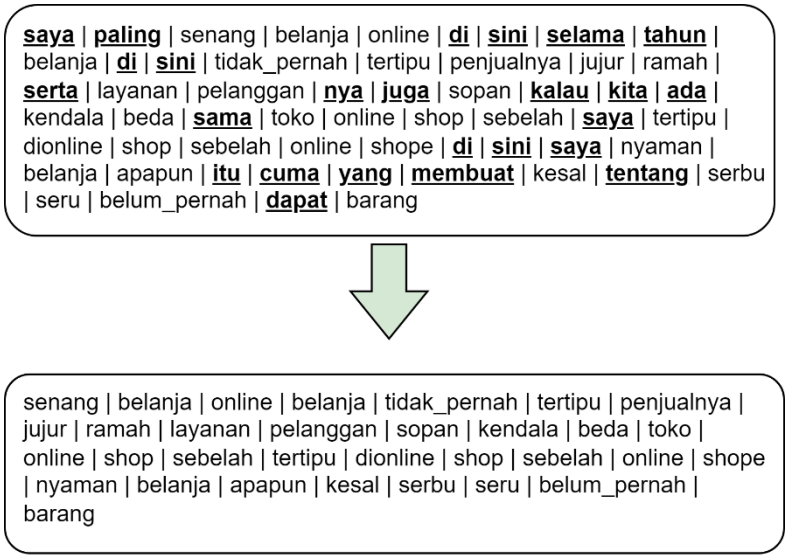


Gambar 2.6. Contoh Pada *Tokenization*

Pada Gambar 2.6 kalimat diatas diubah menjadi kata terpisah.

F. *Stopword Removal*

Stopword removal adalah tahapan proses dalam membuang kata yang terdapat dalam *stop word list*, biasanya mengacu pada kata-kata umum yang jarang digunakan seperti konjungsi misalnya seperti “atau”, “dan” dan “serta”. Tetapi kata ganti seperti “itu”, “dia”, dan lainnya juga dapat dibuang karena tidak berpengaruh dan tidak bermakna [3], [16], [17]. Contoh pada *stopword removal* dapat dilihat pada gambar 2.7.



Gambar 2.7. Contoh Pada *Stopword Removal*

Pada Gambar 2.7 membuang kata yang tidak berpengaruh dan bermakna seperti “saya”, “paling”, “di”, “sini”, “selama”, “tahun”, “serta”, “nya”, “juga”, “kalau”, “kita”, “ada”, “sama”, “itu”, “cuma”, “yang”, “tentang”, dan “dapat”.

2.4. *Word Embedding*

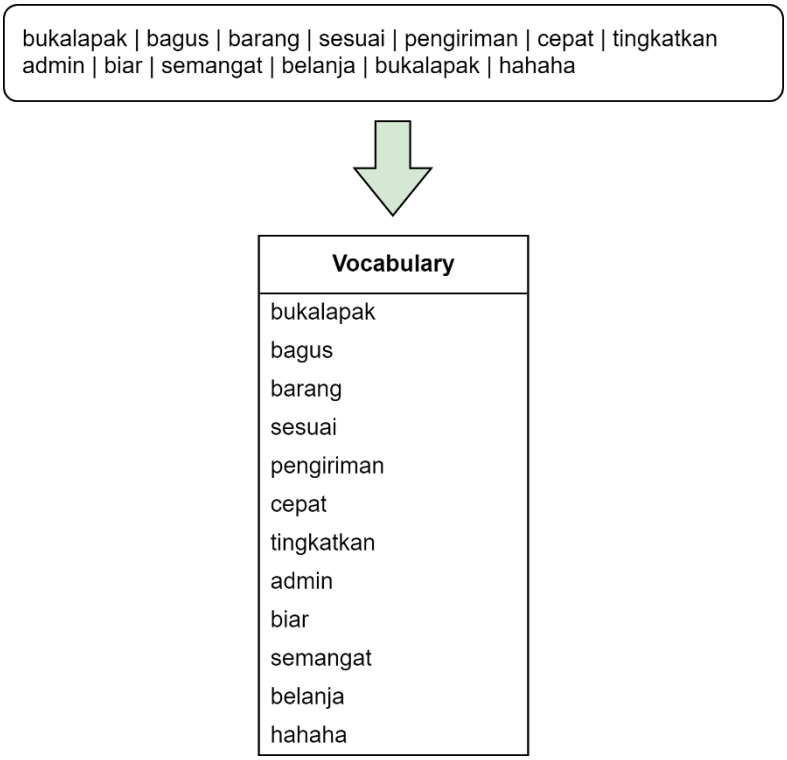
Word embedding merupakan teknik untuk pemodelan Bahasa (*language model*) dan fitur pembelajaran (*feature learning*) yang merubah kata menjadi nilai vektor bilangan riil [20], [21]. Dalam *Word Embedding* kata yang mirip memiliki representasi nilai vektor yang serupa, secara ideal kata yang secara semantik berhubungan menjadi erat kaitannya dalam ruang representasi [14], [21]. Teknik ini normalnya mencakup *embedding* dari sebuah ruang vektor renggang berdimensi

tinggi (*high-dimensional sparse vector space*) contohnya *one-hot encoding* menjadi ruang vektor padat berdimensi rendah (*lower-dimensional dense vector space*). Jaringan syaraf (*neural networks*) digunakan untuk melakukan pembelajaran *word embedding*, atau faktorisasi matriks (*matrix factorization*). Word2Vec adalah *Word embedding* yang paling umum dipakai, model prediksi *neural networks* yang efisien secara komputasi untuk *word embedding* [20].

2.4.1. *One-Hot Encoding*

One-hot encoding merupakan salah satu metode konversi dari kata menjadi vektor [20]. Pada *one-hot encoding*, setiap elemen yang unik perlu direpresentasikan sehignga memiliki dimensinya sendiri yang menghasilkan ruang vektor renggang berdimensi tinggi (*very high dimensional*, *very sparse representation*). Dalam bentuk representasinya, *one-hot encoding* tidak memiliki hubungan yang erat antara kata yang secara semantik, misalnya kata "air" dan "laut", padahal kata tersebut memiliki korelasi namun pada *one-hot encoding* tidak dapat menangkap korelasi kedua kata tersebut [14]. *One-hot encoding* memetakan setiap indeks menjadi sebuah unit vektor yang berbeda (*different unit vector*). Terdapat jumlah *token* yang berbeda atau *token* unik di dalam kosakata (*vocabulary*) dinotasikan menjadi N, dan indeks *token* dimulai dari angka 0 hingga N–1. Jika indeks *token* adalah *integer* i, semua vektor bernilai = 0 dengan panjang dari N, dan kecuali nilai elemen yang ada pada posisi ke-i nilainya = 1 [22].

Misalnya terdapat kata "*one-hot encoding* pilihan yang bagus" jumlah kosa kata = 5, maka indeks *token* nilainya = 5-1 = 4 jadi (0 1 2 3 4). Maka indeks dari *token* "*One-hot"* adalah indeks 0, maka dalam bentuk *one-hot encoding*nya adalah (1 0 0 0 0). *One-hot encoding* cenderung lebih mudah disusun, namun *one-hot endoding* tidak dapat secara tepat mengekspresikan kemiripan kata dengan kata lainnya [22]. Sebelum merubah menjadi *one-hot encoding* kata-kata dari kalimat ubah terlebih dahulu ke dalam bentuk *vocabulary*, jika terdapat kata yang sama lebih dari satu, maka tetap akan ditulis satu saja, contohnya dapat dilihat pada gambar 2.8.



Gambar 2.8. Contoh Pembentukan *Vocabulary* dari *Token*

Pada gambar 2.8. setiap *token* dibentuk ke dalam bentuk *vocabulary*, pada contoh ini *token* ”bukalapak” berjumlah sebanyak dua, namun di dalam *vocabulary* hanya terbentuk satu saja. Setelah dibentuk menjadi *vocabulary*, lalu bentuk ke dalam bentuk *one-hot encoding* contohnya pada *one-hot encoding* dapat dilihat pada gambar 2.9.

One-Hot Encoding													
Corpus	bukalapak	bagus	barang	sesuai	pengiriman	cepat	tingkatkan	admin	biar	semangat	belanja	bukalapak	hahaha
bukalapak	1	0	0	0	0	0	0	0	0	0	0	1	0
bagus	0	1	0	0	0	0	0	0	0	0	0	0	0
barang	0	0	1	0	0	0	0	0	0	0	0	0	0
sesuai	0	0	0	1	0	0	0	0	0	0	0	0	0
pengiriman	0	0	0	0	1	0	0	0	0	0	0	0	0
cepat	0	0	0	0	0	1	0	0	0	0	0	0	0
tingkatkan	0	0	0	0	0	0	1	0	0	0	0	0	0
admin	0	0	0	0	0	0	0	1	0	0	0	0	0
biar	0	0	0	0	0	0	0	0	1	0	0	0	0
semangat	0	0	0	0	0	0	0	0	0	1	0	0	0
belanja	0	0	0	0	0	0	0	0	0	0	1	0	1
hahaha	0	0	0	0	0	0	0	0	0	0	0	0	0

Gambar 2.9. Contoh *One-hot Encoding*

Pada proses *one-hot encoding* ini, jika terdapat *token* yang menempati posisi elemen dalam *vocabulary*, maka akan bernilai 1, dan sisanya akan bernilai 0

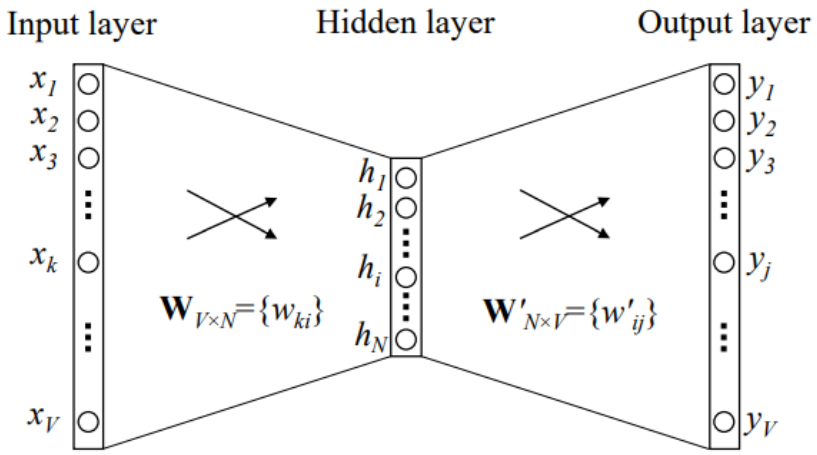
berdasarkan panjang dari kosa kata (*vocabulary*). Dalam contoh diatas terdapat dua *token* “bukalapak”, maka *token* “bukalapak” bernilai 1 jika menempati posisi elemen *token* yang terdapat dalam *vocabulary* dan seterusnya untuk masing-masing token.

2.4.2. Word2Vec Continuous Bag-of-Words

Word2Vec merupakan salah satu metode *word embedding*, yang diciptakan Mikolov dkk pada tahun 2013 [5]. Word2vec dapat lebih baik mengekspresikan kemiripan kata dengan kata lainnya [22]. Word2Vec banyak dipakai dalam penelitian NLP yang terdiri dari sebuah *hidden layer* dan *fully connected layer*. Dimensi dari matriks bobot pada masing-masing *layer* yaitu jumlah kata yang terdapat dalam *vocabulary* dikali dengan jumlah *hidden neuron* pada *hidden layer*. Matriks bobot yang terdapat pada *hidden layer* dari model yang sudah melakukan pelatihan dipakai untuk mengubah kata ke bentuk vektor. Matriks bobot ini ibaratnya seperti *lookup table*, di mana setiap kata direpresentasikan oleh setiap baris dan vektor dari kata tersebut direpresentasikan oleh kolom [23]. Word2Vec mempunyai dua model arsitektur yaitu *Continuous Bag-of-Words* (CBOW) dan *Continuous Skip-gram* (Skip-gram) [5]. Namun pada penelitian ini arsitektur yang digunakan yaitu *Continuous Bag-of-Words* (CBOW).

Continuous Bag-of-Words (CBOW) sering dipakai untuk aplikasi NLP yang mencoba untuk memprediksi sebuah target kata disekitar konteksnya biasanya terdiri dari beberapa kata yang berdekatan. CBOW tidak bergantung pada urutan kata-kata atau harus pada karakteristik probabilistiknya [14]. Kelebihannya memiliki waktu latih lebih singkat serta untuk kata yang sering muncul (*frequent words*) memiliki akurasi sedikit lebih baik [23].

Jika hanya satu kata yang dijadikan konteks [24], maka arsitektur CBOWnya dapat dilihat pada gambar 2.10.



Gambar 2.10. Arsitektur CBOW dengan Satu Kata Konteks [24]

Pada gambar terdapat simbol k adalah konteks kata, j adalah target kata, V yang berarti jumlah *vocabulary*, dan N adalah jumlah *hidden layer*, unit *layer* tersebut *fully connected*, dan *input layer* berupa vektor *one-hot encoding*, berarti untuk input kata konteks posisinya diisi nilai 1, dan sisanya 0 [24].

Dimulai dari perhitungan *hidden layer* (h), namun sebelumnya menentukan matriks bobot W dengan dimensi V × N. V adalah *vocabulary*, N adalah ukuran *hidden layer* Setiap baris dari matriks bobot W merepresentasikan vektor berukuran N-dimensi dari kata terkait di *input layer*. Jadi baris i dari matriks bobot W adalah v_w^T. Perhitungannya perkalian matriks antara matriks bobot W transpose dengan *input layer* yang berupa *one-hot encoding*. Nilai *hidden layer* didapatkan dengan persamaan (2.1) [24]:

$$h = W^T x := v_{w_I}^T \tag{2.1}$$

Dimana:

W^T : matriks bobot W transpose

x : *input layer*

v_{w_I} : representasi vektor dari *input* kata (w_I)

Lalu setelah *hidden layer* dihitung, hitung *output* u, namun matriks bobotnya berbeda dari yang sebelumnya, yaitu bobot matriksnya adalah $W' = \{w'_{ij}\}$ ukurannya adalah N × V. Perhitungannya perkalian matriks antara kolom j bobot matriks W' dengan *hidden layer* pada persamaan (2.2) [24].

$$u_j = {v'_{w_j}}^T h \tag{2.2}$$

Dimana:

${v'_{w_j}}^T$: kolom j dari matriks bobot W' transpose

h : *hidden layer*

Lalu menghitung *output* dari *unit* j di *output layer* dengan fungsi softmax untuk mendapatkan distribusi *posterior* kata-kata. Catatan bahwa v_w adalah representasi baris dari matriks bobot W', v'_w representasi kolom dari matriks bobot W' namun analisis berikutnya akan berubah. Nilai *output* softmax didapatkan dengan persamaan (2.3) [24].

$$p(w_j|w_I) = y_j = \frac{\exp(u_j)}{\sum_{j'=1}^{V} \exp(u_{j'})} \tag{2.3}$$

Dimana:

u_j : *output u* unit ke-j

y_j : *output* dari unit j di *output layer*

V : jumlah *Vocabulary*

Setelah didapatkan hasil keluarannya, *update* perhitungan *hidden* dan bobot *output*. Dengan menerapkan *backpropagation* dan melakukan turunan. Tujuan dari pelatihan adalah untuk maksimalkan persamaan (2.3) probabilitas bersyarat mengamati *output* kata sebenarnya dari j* mengingat *input* kata *context* w_I berkaitan dengan bobot. Namun sebelumnya menghitung *loss function* atau *error* yang tujuannya untuk meminimumkan nilai E dengan persamaan (2.4) [24]:

$$\max p(w_o|w_I) = u_{j^*} - \log \sum_{j'=1}^{V} exp(u_{j'}) := -E \tag{2.4}$$

Dimana:

w_I : *input* konteks kata

w_O : *output* kata sebenarnya atau target kata

u_{j^*} : *output u* target kata sebenarnya

$u_{j'}$: *output* u kata dalam vocabulary selain target kata

E : *loss function*

Lalu menghitung turunan bobot untuk memperbarui bobot pada *hidden* dan *output layer*. Turunan parsial E yang berkaitan dengan *input unit* ke-j dari u_j dapat dihitung dengan persamaan (2.5) [24].

$$\frac{\partial E}{\partial u_j} = y_j - t_j \coloneqq e_j \tag{2.5}$$

Dimana:

t_j : $1(j = j^*)$ akan bernilai 1 ketika posisi unit ke-j adalah nilai sebenarnya dari kata *output*, sebaliknya bernilai 0.

y_j : *output* dari unit j di *output layer*

e_j : prediksi *error* di *output layer*

Selanjutnya menghitung turunan parsial E pada w'_{ij} untuk memperoleh gradien yang terdapat pada bobot *hidden* dan *output*. Perhitungannya dengan persamaan (2.6) [24].

$$\frac{\partial E}{\partial w'_{ij}} = \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial w'_{ij}} = e_j \otimes h_i \tag{2.6}$$

Dimana:

e_j : prediksi *error*

h_i : unit ke-i pada *hidden layer*

$\otimes$: *outer product* atau *tensor product*

Menggunakan *stochastic gradient descent* untuk memperoleh bobot baru (W'). Sebagai catatan bahwa harus menulusuri setiap kemungkinan kata yang terdapat pada *vocabulary*. Periksalah *output* dari probabilitas y_j dan bandingkan dengan *output* yang diharapkan (t_j) 0 atau 1, jika $y_j > t_j$ maka akan melakukan pengurangan proporsi pada vektor *hidden layer* (h) dari v'_{wj}, sehingga membuat v'_{wj} lebih jauh dari v_{w_I}), jika $y_j < t_j$ yang mana benar hanya jika $t_j = 1$ ($w_j = w_O$), maka akan menambahkan beberapa h ke v'_{wO}, sehingga membuat v'_{wO} mendekat ke v_{w_I}. Dan jika nilai y_j terlalu dekat ke t_j, maka berdasarkan perhitungan baru, perubahan bobot baru akan sangat sedikit atau kecil [24]. Perhitungan memperbarui bobot W' dilakukan dengan persamaan (2.7).

$$w_{ij}'^{(baru)} = w_{ij}'^{(lama)} - \eta \cdot e_j \cdot h_i \quad (2.7)$$

Dimana:

η : *learning rate*

w_{ij}' : bobot W' dengan i baris dan j kolom

h_i : *hidden layer*

e_j : prediksi *error*

Selanjutnya melakukan pembaruan perhitungan bobot antara *input* dan *hidden layer* (W). Mengambil nilai turunan parsial dari E pada *output* dari *hidden layer*. Perhitungan turunan E terhadap *hidden layer* dapat dilakukan dengan persamaan (2.8) [24].

$$\frac{\partial E}{\partial h_i} = \sum_{j=1}^{V} \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial h_i} = \sum_{j=1}^{V} e_j \cdot w_{ij}' \coloneqq EH_i \quad (2.8)$$

Dimana:

h_i : *output* unit ke-i pada *hidden layer*

w_{ij}' : matriks bobot W'

e_j : prediksi *error* dari kata ke-j pada *output layer*

EH : sebuah vektor N-dim, jumlah vektor *output* dari semua kata yang ada pada *vocabulary*, dibobotkan oleh prediksi *errornya*.

Perhitungan dari turunan E terhadap bobot W sama dengan perhitungan produk tensor antara x dan EH dilakukan dengan persamaan (2.9) [24].

$$\frac{\partial E}{\partial W} = x \otimes EH = xEH^T \quad (2.9)$$

Dimana:

x : *input* vektor bukan nol

EH^T : sebuah vektor N-dim, jumlah vektor *output* dari semua kata yang ada pada *vocabulary*, dibobotkan oleh prediksi *errornya.*

$\otimes$: *outer product* atau *tensor product*

Lalu perhitungan untuk mendapatkan bobot baru v_{w_I}, dikarenakan hanya satu nilai x yang bukan nol dan nilai $\frac{\partial E}{\partial W}$ ini sama dengan nilai EH^T jadi menghitung bobot baru W dengan persamaan (2.10) [24].

$$v_{w_I}{}^{(baru)} = v_{w_I}{}^{(lama)} - \eta\, EH^T \quad (2.10)$$

Dimana:

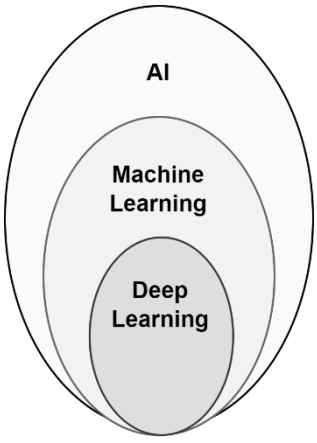
v_{w_I} : baris bobot W, *input* vektor hanya merupakan kata konteks dan hanya baris dari W yang turunannya bernilai bukan nol.

η : *learning rate*

EH^T : sebuah vektor N-dim, jumlah vektor *output* dari semua kata yang ada pada *vocabulary*, dibobotkan oleh prediksi *errornya* transpose.

2.5. *Deep Learning*

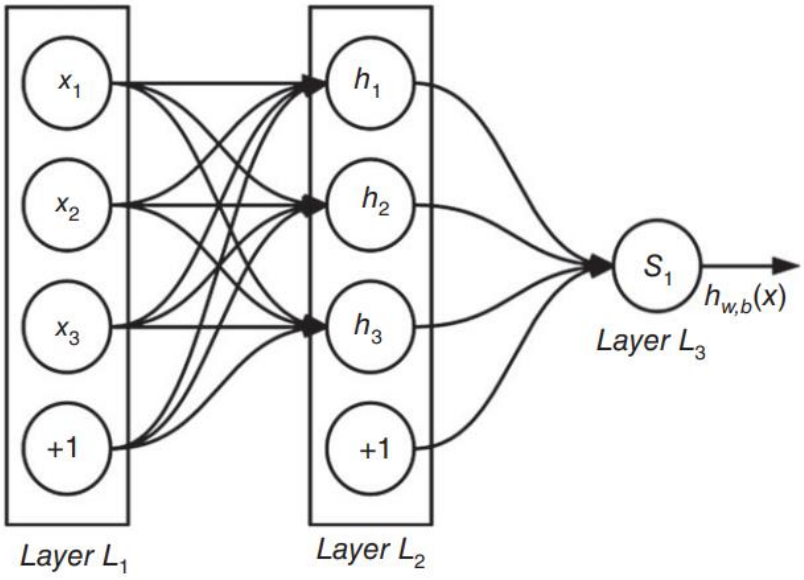
Deep Learning adalah himpunan bagian dari *Machine Learning* yang memanfaatkan banyak representasi lapisan (*layers*) dimana lapisan tersebut mengandung unit pemrosesan informasi (*neuron*) untuk ekstraksi fitur dan transformasi, serta menganalisis dan mengklasifikasikan pola. *Deep learning* meningkatkan berbagai tugas komputer yang berbeda termasuk *text mining, image processing, dan pattern recognition* [25]. Rina Dechter memperkenalkan *Deep Learning* pada tahun 1986, *Deep Learning* merupakan bagian dari *Artificial Neural Network* (ANN) karena *Deep learning* menggabungkan model komputasional dan algoritma yang meniru arsitektur jaringan saraf otak manusia [26], [27].



Gambar 2.11. Hubungan AI, *Machine Learning* & *Deep Learning*

Istilah *Deep* itu mengacu pada jumlah *layers* dalam ANN. Terdapat tiga jenis *layer* yaitu: *input layer* sebagai penerima data masukkan, *hidden layer* berperan untuk mengekstraksi pola di dalam data, dan *output layer* berperan memproduksi hasil pemrosesan data [27]. Berdasarkan topologi jaringannya, *neural networks*

umumnya dapat dikategorikan menjadi *feedforward neural networks* dan *recurrent/recursive neural networks* juga dapat dipadukan. Contoh untuk *feedforward neural network* divisualisasikan pada Gambar 2.12. Dimana terdiri dari tiga *input layer* (x_1, x_2, x_3) dan *bias term* (+1). Tiga *hidden layer* (h_1, h_2, h_3) dan *bias term* (+1) serta *output layer* (s_1). Lingkaran L_1 merepresentasikan sebuah elemen pada *input vector*, sementara lingkaran L_2 atau L_3 merepresentasikan sebuah *neuron*, elemen komputasi dasar pada sebuah *neural network* yang juga disebut *activation function*. Garis diantara dua *neuron* yang menggambarkan koneksi untuk mengalirkan informasi. Setiap koneksi dihubungkan dengan bobot (*weight*), nilai yang mengendalikan sinyal antara dua *neurons*. Pembelajaran jaringan saraf didapatkan dengan cara menyeseuaikan bobot antara *neuron* dengan informasi yang mengalir melaluinya. *Neuron* membaca *output* dari *neuron* lapisan sebelumnya, memproses informasi, dan kemudian menghasilkan *output* ke *neuron* di lapisan berikutnya [20].



Gambar 2.12. *Feedforward neural network* [20]

Contoh perhitungan sederhana $f(W'x) = f(\sum_i^3 W_i x_i + b)$, dimana W_i adalah koneksi bobot, b adalah *bias* dan f adalah fungsi aktivasi umumnya *nonlinear* [20]. Berikut ini akan dibahas beberapa fungsi aktivasi yang digunakan.

2.5.1. Fungsi Aktivasi

Fungsi Aktivasi memutuskan apakah *neuron* harus diaktifkan atau tidak dengan menghitung penjumlahan bobot dan selanjutnya menambah *bias* dengannya. Fungsi aktivasi adalah operator untuk mengubah sinyal *input* menjadi *output*, kebanyakan dari fungsi aktivasi menambahkan non-linearitas. Karena fungsi aktivasi sangat penting bagi *deep learning* [22]. Fungsi aktivasi yang umum dipakai adalah *Rectified Linear Unit* (RELU), *Sigmoid Function*, *Hyperbolic tangent* dan *Softmax Function* [22], [28]. Banyak macam fungsi aktivasi selain yang disebutkan diatas, namun yang akan digunakan pada penelitian ini adalah Sigmoid, tanh dan softmax yang akan dijelaskan berikut ini:

1. Fungsi Sigmoid

   Fungsi sigmoid mengubah inputnya, yang nilainya terletak pada domain $\mathbb{R}$, untuk *output* yang terletak pada interval (0,1). Sigmoid kerap kali disebut *squashing function*. Karena nilai *input* akan ditekan ke dalam rentang antara nilai 0 dan 1 [22]. Persamaan perhitungan sigmoid menggunakan persamaan ke (2.11).

   $$sigmoid(x) = \frac{1}{1 + \exp(-x)} \tag{2.11}$$

   Sigmoid masih banyak digunakan sebagai fungsi aktivasi pada unit *output*, dan probabilitas yang sangat cocok untuk permasalahan *binary classification*, sigmoid adalah *special case* dari softmax. Sebagai catatan bahwa ketika *input* mendekati 0, fungsi sigmoid mendekati transformasi linier [22]. Untuk menghitung turunan fungsi sigmoid dapat dilakukan dengan persamaan (2.12).

   $$\begin{aligned} \frac{d}{dx} sigmoid(x) &= \frac{\exp(-x)}{(1 + \exp(-x))^2} \\ &= sigmoid(x)\big(1 - sigmoid(x)\big) \end{aligned} \tag{2.12}$$

2. Fungsi *Hyperbolic Tangent* (tanh)

   Fungsi *Hyperbolic tangent* (tanh) seperti fungsi sigmoid, tanh juga menekan inputnya, nilai akan diubah menjadi elemen yang bernilai

antara –1 dan 1 [22]. Persamaan (2.13) adalah untuk menghitung fungsi aktivasi tanh.

$$\tanh(x) = \frac{1 - \exp(-2x)}{1 + \exp(-2x)} \tag{2.13}$$

Untuk turunan fungsi tanh menggunakan persamaan (2.14).

$$\frac{d}{dx}\tanh(x) = 1 - tanh^2(x) \tag{2.14}$$

3. Fungsi Softmax

   Fungsi aktivasi softmax adalah fungsi perhitungan probabilitas yang dipakai untuk permasalahan klasifikasi *multiclass* dengan *output* kelasnya yang memiliki nilai probabilitas tertinggi. Nilai rentang dari keluaran fungsi softmax adalah antara 0 dan 1 [28], [29]. Menghitung softmax dilakukan dengan persamaan (2.15).

$$softmax(X_i) = \frac{\exp(X_i)}{\sum_{j=1}^{k} \exp\,(X_i)}, i = 1, 2, .., k \tag{2.15}$$

Beberapa tipe dari *deep neural network* digunakan dalam NLP seperti *Feedforward Neural Network, Multi Layer Peceptron*, *Convolutional Neural Network* (CNN), *Recurrent Neural Network* (RNN) dan yang terbaru *Recursive Neural Network*. *Deep learning* mengungguli beberapa metode *machine learning* tradisional dalam beberapa tugas NLP seperti *machine translation* dan *named-entity recognition* [14], [25].

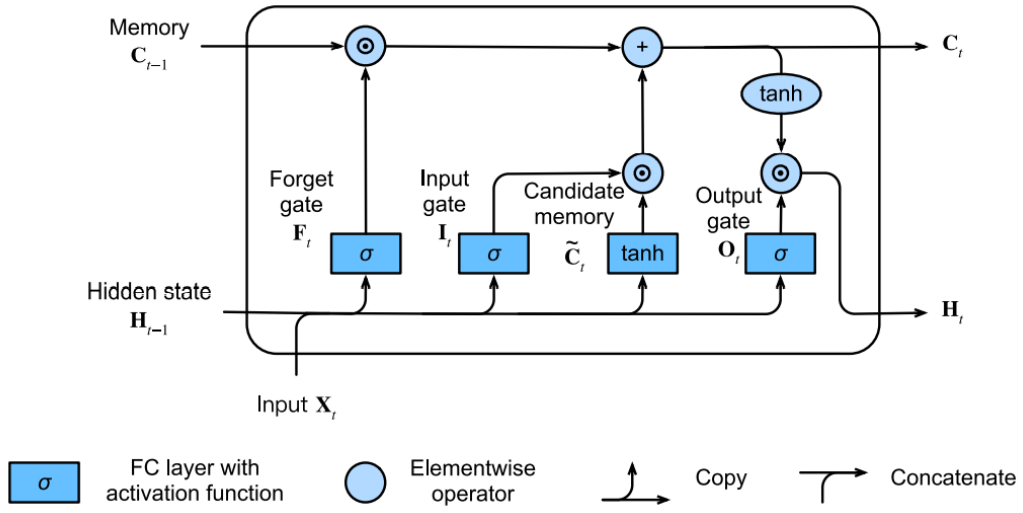
2.6. *Recurrent Neural Networks* (RNN)

Recurrent Neural Networks (RNN) merupakan pendekatan berbasis *neural network* yang efektif dalam memproses informasi yang sekuensial [25]. Berbeda dengan *feedforward neural network*. RNN dapat menggunakan "*Memory*" untuk memproses sebuah *input* yang sekuensial seperti teks. "*Memory*" dalam RNN berfungsi mengingat informasi yang telah diproses sebelumnya dan bergantung pada perhitungan sebelumnya [20]. Kelebihan dari RNN adalah menggunakan hasil dari perhitungan sebelumnya pada perhitungan berikutnya karena RNN memasukkan *input* kata secara berurutan. Tetapi kekurangan RNN masih mengalami permasalahan penurunan gradien atau kenaikan gradien yang sangat

ekstrim, menyebabkan sulit untuk belajar dan mengubah nilai parameter pada lapisan sebelumnya [25], [30]. *Layer* dalam RNN ada tiga kategori, yaitu *input, hidden*, dan *output layers* [14]. Terdapat beberapa macam RNN, seperti *Long Short-Term Memory* (LSTM), dan *Gated Recurrent Unit* (GRU) [25].

2.7. *Long Short-Term Memory*

Long Short-Term Memory (LSTM) merupakan tipe perkembangan spesial dari RNN yang diciptakan oleh Hochreiter & Schmidhuber pada tahun 1997. LSTM merupakan tipe algoritma RNN yang paling banyak digunakan. LSTM mempelajari cara untuk menangkap ketergantungan waktu yang lama (*long time dependencies*) antara *input* dari *time steps* yang berbeda [20], [25]. LSTM mampu mengatasi masalah hilangnya gradien (*vanishing gradient*) pada RNN [10]. LSTM adalah salah satu metode *deep neural network* paling berhasil dalam menangani data sekuensial yang panjang, juga baik dalam mempelajari pengetahuan secara tersirat. LSTM telah dikembangkan dan memperoleh performansi yang lebih baik [9]. LSTM memiliki 3 gerbang (*gates*), yaitu: *Input Gate, Forget Gate, dan Output Gate*, dan *memorycell* atau *cell state*. Dari ketiga *gate* berfungsi dalam menentukan apakah sebuah informasi mesti disimpan atau dilupakan, hasil nilai dari ketiga *gate* tersebut diantara 0 dan 1 [9], [22].



Gambar 2.13. Arsitektur *Long Short-Term Memory* [22]

Forget gate menentukan informasi mana yang dilupakan dalam *cell state* terakhir, *input*nya adalah h_{t-1} dan x_t, nilai *output* antara (0,1) [10]. Jika *forget gate* = 1 informasi akan disimpan, jika *forget gate* = 0 informasi akan dilupakan [12]. Hasil *forget gate* dihitung dengan persamaan ke (2.16).

$$f_t = \sigma\left(W_f x_t + U_f h_{t-1} + b_f\right) \tag{2.16}$$

Dimana:

W_f : koneksi bobot dari x_t dan f

x_t : *input time step* saat ini

U_f : koneksi bobot dari h_{t-1} dan f

h_{t-1} : hasil *hidden layer* sebelumnya

b_f : *bias term*

t : *time steps*

f : *forget gate*

σ : fungsi aktivasi sigmoid

Input gate menentukan informasi harus diperbarui dalam *cell state* pada waktu saat ini [10]. *Candidate cell state* nantinya akan dimasukkan ke dalam persamaan *Cell state*, nilai *candidate cell state* diaktivasi dengan fungsi hyperbolic tangent antara -1 dan 1 [20], [22]. *Cell state* tempat menyimpan informasi yang diberikan dari satu *time steps* ke *time steps* selanjutnya [11]. Perhitungan *input gate* dengan persamaan (2.17), perhitungan *Candidate cell state* menggunakan persamaan (2.18), dan perhitungan *cell state* menggunakan persamaan (2.19).

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{2.17}$$

$$\tilde{C}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c) \tag{2.18}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{C}_t \tag{2.19}$$

Dimana:

W_i : koneksi bobot dari x_t dan i_t

x_t : *input time step* saat ini

U_i : koneksi bobot dari h_{t-1} dan i_t

h_{t-1} : *hidden layer time step* sebelumnya

σ : fungsi aktivasi sigmoid

W_c : koneksi bobot dari x_t dan $\tilde{C}_t$

U_c : koneksi bobot $\tilde{C}_t$ dan h_{t-1}

tanh : fungsi aktivasi hyperbolic tangent

b_i, b_c : *bias term*

f_t : *forget gate*

i_t : *input gate*

$\odot$: operator *element-wise*

$\tilde{C}_t$: *candidate cell state*

c_t : *cell state time step* saat ini

c_{t-1} : *cell state time step* sebelumnya

Output gate menentukan nilai *output* unit LSTM [10] dihitung dengan persamaan (2.20), dan *hidden state* dihitung dari perkalian *elemen wise* antara *output gate* dan *cell state* nilai *hidden state* intervalnya -1 dan 1 dengan persamaan (2.21) [22].

$$o_t = \sigma(W_o\, x_t + U_o\, h_{t-1} + b_o) \tag{2.20}$$

$$h_t = o_t \odot \tanh(c_t) \tag{2.21}$$

Dimana:

σ : fungsi aktivasi sigmoid

W_o : koneksi bobot x_t dan o_t

x_t : *input time step* saat ini

U_o : koneksi bobot h_{t-1} dan o_t

h_{t-1} : *hidden layer time step* sebelumnya

b_o : *bias term*

o_t : *output gate*

c_t : *cell state time step* saat ini

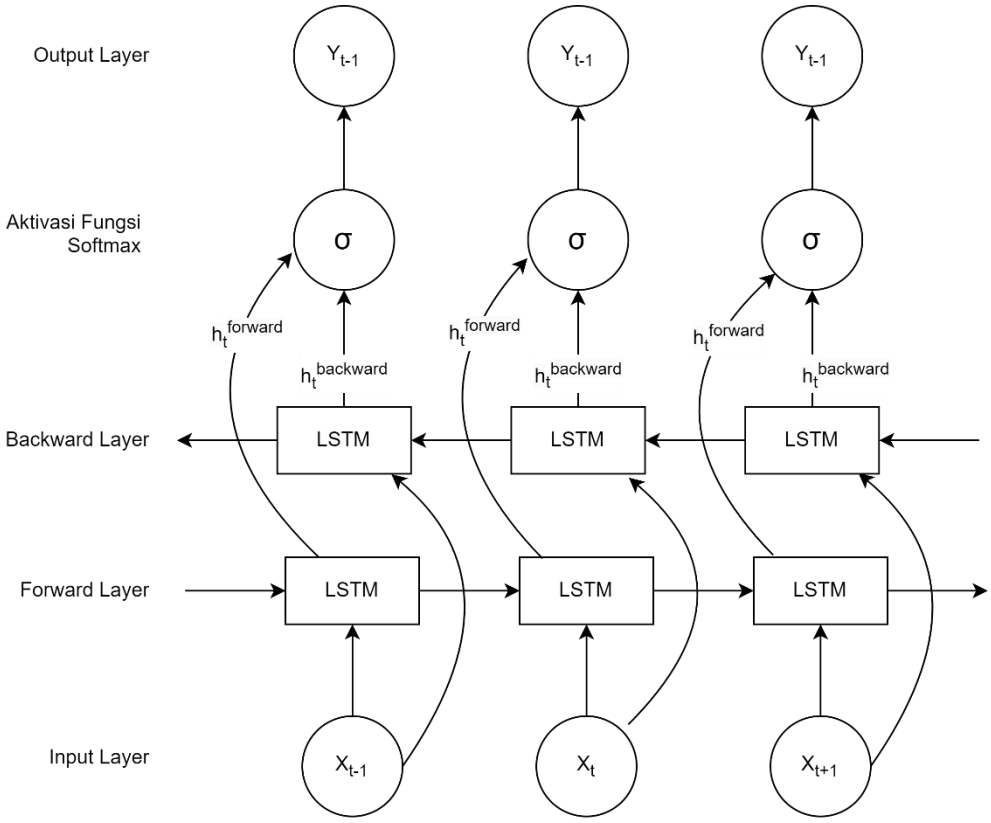
tanh : fungsi aktivasi *hyperbolic tangent*

2.8. *Bidirectional Long Short-Term Memory*

Bidirectional Long Short-Term Memory (Bi-LSTM) adalah *neural networks* LSTM yang terdiri dari dua *layer* LSTM, yakni *forward* LSTM *layer* untuk memodelkan konteks sebelumnya dan *backward* LSTM *layer* untuk memodelkan

setiap konteks berikutnya [4]. BiLSTM mengkoneksikan dua *hidden layer* dari arah yang berlawanan ke *output* yang sama. Sehingga *layers* dari *neurons* dapat memperoleh informasi dari kondisi masa lalu (*past*) dan masa depat (*future*) secara serentak [4]. Bi-LSTM sangat cocok untuk tugas pelabelan sekuensial yang. Bi-LSTM kerap digunakan untuk memodelkan informasi konteks (*context information*) dalam NLP. Menggunakan model LSTM bisa lebih baik menangkap jarak depedensi (*distance dependencies*) yang lebih jauh dikarenakan LSTM dapat mempelajari apa yang harus diingat dan apa yang harus dilupakan melalui proses pelatihan. Bi-LSTM mampu menutupi permasalahan LSTM dalam memodelkan kalimat, LSTM tidak dapat menyandikan informasi (*encode information*) dari belakang ke depan (*from back to front*) Bi-LSTM merupakan pilihan yang lebih baik dan dapat menangkap dependensi semantik (*semantic dependencies*) [5].

Bidirectional RNN diperkenalkan pada tahun 1997 oleh Schuster & Paliwal. Munculnya gagasan *Bidirectional* RNN karena *output* pada setiap *time step* tidak hanya bergantung pada elemen sebelumnya dalam suatu urutan. Tetapi juga bergantung pada elemen selanjutnya dalam suatu urutan. *Bidirectional* RNN terdiri dari dua RNN yang ditumpuk di atas satu sama lainnya [20]. Bi-LSTM menggabungkan *Bidirectional* RNN dan LSTM, Bi-LSTM memiliki kemampuan untuk mengumpulkan *context information* dari masa lampau dan masa depan (*past and future*) dan cocok untuk kalimat yang panjang [13]. Kelemahan LSTM adalah tidak cukup memperhitungkan informasi dari kata terakhir karena hanya membaca kalimat dari satu arah saja bergerak maju (*forward*). Bi-LSTM menerapkan dua LSTM terpisah yaitu dari arah depan (*past to future*) dan (*future to past*) lalu digabungkan [11], [12]. Arsitektur Bi-LSTM dapat dilihat pada gambar 2.14.



Gambar 2.14. Arsitektur Bi-LSTM

Perhitungan Bi-LSTM yaitu menggabungkan nilai *hidden state forward* LSTM *layer* dan *hidden state backward* LSTM *layer*. Persamaan gabungan *hidden state* Bi-LSTM menggunakan persamaan (2.22) [12], [13].

$$h_t{}^{BiLSTM} = \overrightarrow{h_t} \oplus \overleftarrow{h_t} = \left[\overrightarrow{h_t}, \overleftarrow{h_t}\right] \tag{2.22}$$

Dimana:

$\overrightarrow{h_t}$: *hidden states forward direction*

$\overleftarrow{h_t}$: *hidden states backward direction*

$\oplus$: operator *concatenation*

Setelah menggabungkan *hidden state* kedua *layer* yaitu *forward* dan *backward*, Karena h_t berjumlah dua, maka dilakukan perkalian matriks antara W_y dan h_t *forward layer* ditambah dengan perkalian matriks antara W_y dan h_t *backward layer* kemudian ditambah bias, akan menjadi hasil prediksi akhir yang

terdapat pada persamaan (2.23), fungsi aktivasi yang dipakai adalah softmax karena mengklasifikasikan *multi-class* [13], [31].

$$y_t = softmax(W_y\, h_t + b_y) \tag{2.23}$$

Dimana:

W_y : matriks bobot pada *output* y_t

h_t : *hidden state* Bi-LSTM

b_y : *bias* vektor pada *output* y_t

2.8.1. *Loss Function*

Loss function digunakan untuk mengetahui pengukuran dari model, seberapa baik atau buruk. Contoh beberapa *loss function* yang paling umum adalah *Root Mean Squared Error* (RMSE), dan *Cross-Entropy*. *Cross Entropy* digunakan untuk menghitung nilai *error* antara model dengan hasil keluaran prediksi model dan sering digunakan untuk kasus klasifikasi [22]. *Loss function* secara umum belajar untuk mengurangi *error* dalam prediksi [32]. Perhitungan *Cross Entropy* menggunakan persamaan (2.24).

$$LCE(y, \hat{y}) = -\sum_{i=1}^{N} y_i \log \hat{y}_i \tag{2.24}$$

Dimana:

y_i : *Output* label sebenarnya

$\hat{y}_i$: *Output* label prediksi

2.8.2. *Backpropagation Through Time* (BPTT)

Backpropagation sebagai teknik untuk menghitung gradien dari parameter *neural network*, metode ini melewati jaringan dengan cara berjalan mundur, dari lapisan *output* menuju lapisan *input* berdasarkan dengan aturan rantai (*chain rule*) yang ada pada kalkulus. *Backpropagation* menyimpan turunan parsial yang dibutuhkan ketika menghitung gradien terhadap beberapa parameter. Dalam pelatihan *deep learning*, *forward propagation* dan *backpropagation* saling berhubungan satu sama lain. *Backpropagation Through Time* (BPTT) adalah sebuah teknik untuk memperbarui parameter yang terdapat pada algoritma jenis

RNN [22]. Tujuan dari BPTT ini adalah menghitung dan menyimpan nilai gradien terhadap parameter suatu model dengan menggunakan teknik *chain rule* yang terdapat pada kalkulus [22], [31].

Dalam melakukan pelatihan Bi-LSTM, berarti membutuhkan *forward layer* dan *backward layer* LSTM, yang kemudian nilai *hidden state* keduanya digabungkan (*concatenate*) dan melakukan perhitungan dengan bobot W dan bias pada *output layer* kemudian diaktivasi dengan fungsi softmax. Sehingga untuk parameter model yang diperbarui adalah bobot W (W_f, W_i, W_c, W_o) pada *forward* dan *backward layer* LSTM, bobot W_y pada *output layer*, bobot U (U_f, U_i, U_c, U_o) pada *forward* dan *backward layer* LSTM, dan *bias* (b_f, b_i, b_c, b_o). Karena *bias* pada *output layer* tidak digunakan pada penelitian ini karena menyebabkan perubahan hasil klasifikasi. Sehingga jumlah parameter yang akan diperbarui berjumlah 25 parameter.

Dimulai dari BPTT menghitung turunan parsial nilai prediksi Bi-LSTM pada *output layer* dengan persamaan (2.25) dan turunan parsial bobot W pada *output layer* menggunakan persamaan (2.26) [31], [33].

$$dy_t = -(y_{rt} - y_t) \tag{2.25}$$

$$dW_y = \sum_{\mathrm{t}} h_t \otimes dy_t \tag{2.26}$$

Dimana:

y_{rt} : Label sebenarnya

y_t : *output* softmax Bi-LSTM

h_t : *hidden state* Bi-LSTM pada *timestep* ke t

dy_t : turunan parsial y_t *output layer*

Turunan parsial *loss* terhadap *hidden state* (h_t) pada *time step* terakhir, dilambangkan dalam persamaan (2.27) [31], [33].

$$\delta_{h_t} = \frac{\partial E}{\partial h_t} = W_y \, dy_t \tag{2.27}$$

Dimana:

W_y : Bobot matriks W pada *output layer*

dy_t : turunan parsial y_t *output layer*

Demikian pula turunan parsial *error* terhadap gerbangnya. Turunan parsial *output gate* (o_t) dapat dihitung dengan persamaan (2.28), turunan parsial *cell state* (c_t) dapat dihitung dengan persamaan (2.29), turunan parsial *forget gate* (f_t) dihitung dengan persamaan (2.30), turunan parsial *input gate* (i_t) dihitung dengan persamaan (2.31), dan turunan parsial *candidate cell state* ($\widetilde{C_t}$) dihitung dengan persamaan (2.32) [31].

$$do_t = \frac{\partial E}{\partial o_t} = \tanh(c_t)\, \delta_{h_t} \tag{2.28}$$

$$dc_t = (1 - \tanh(c_t)^2)\, o_t\, \delta_{h_t} + f_{t+1} \odot dc_{t+1} \tag{2.29}$$

$$df_t = \frac{\partial E}{\partial f_t} = c_{t-1}\, dc_t \tag{2.30}$$

$$di_t = \frac{\partial E}{\partial i_t} = \widetilde{C_t}\, dc_t \tag{2.31}$$

$$d\widetilde{C_t} = \frac{\partial E}{\partial \widetilde{C_t}} = i_t\, dc_t \tag{2.32}$$

Dimana:

c_t : *cell state*

$\widetilde{C_t}$: *candidate cell state*

f_t : *forget gate*

f_{t+1} : *forget gate time step* berikutnya

i_t : *input gate*

dc_{t+1} : turunan parsial *cell state time step* berikutnya

dc_t : turunan parsial *cell state time step* saat ini

δ_{h_t} : turunan parsial *hidden state*

Sambungan turunan parsial gerbang yaitu *forget gate* dikalikan dengan turunan sigmoid dari *forget gate* dengan persamaan (2.33), turunan parsial *input gate* dikalikan dengan turunan *sigmoid* dari *input gate* dengan persamaan (2.34), turunan parsial *candidate cell* dikalikan dengan turunan tanh dari *candidate cell state* dengan persamaan (2.35) dan turunan parsial *output gate* dikalikan dengan turunan sigmoid dari *output gate* dengan persamaan (2.36) [33].

$$\delta_{f_t} = \frac{\partial E}{\partial f_t} \odot f_t \odot (1 - f_t) \tag{2.33}$$

$$\delta_{i_t} = \frac{\partial E}{\partial i_t} \odot i_t \odot (1 - i_t) \tag{2.34}$$

$$\delta_{\tilde{C}_t} = \frac{\partial E}{\partial \widetilde{C}_t} \odot (1 - \tilde{C}_t^{\,2}) \tag{2.35}$$

$$\delta_{o_t} = \frac{\partial E}{\partial o_t} \odot o_t \odot (1 - o_t) \tag{2.36}$$

Untuk kesederhanaan fungsi, turunan parsial *forget gate* dapat diuraikan menjadi persamaan (2.37), turunan parsial *input gate* dapat diuraikan menjadi persamaan (2.38), turunan parsial *candidate cell state* dapat diuraikan menjadi persamaan (2.39) dan turunan parsial output gate dapat diuraikan menjadi persamaan (2.40).

$$\delta_{f_t} = c_{t-1} \odot\ dc_t \odot f_t \odot (1 - f_t) \tag{2.37}$$

$$\delta_{i_t} = \widetilde{C}_t \odot\ dc_t \odot i_t \odot (1 - i_t) \tag{2.38}$$

$$\delta_{\tilde{C}_t} = i_t \odot dc_t \odot (1 - \tilde{C}_t^{\,2}) \tag{2.39}$$

$$\delta_{o_t} = \tanh(c_t) \odot \delta_{h_t} \odot o_t \odot (1 - o_t) \tag{2.40}$$

Dimana:

c_t : *cell state*

c_{t-1} : *cell state time step* sebelumnya

dc_t : turunan parsial *cell state time step* saat ini

f_t : *forget gate*

$\widetilde{C}_t$: *candidate cell state*

i_t : *input gate*

o_t : *output gate*

δ_{h_t} : turunan parsial *hidden state time step* saat ini

Untuk turunan parsial *hidden pada time step* sebelumnya (t-1) berbeda dengan *time step* terakhir dalam sekuens dihitung dengan persamaan (2.41) yang kemudian dilanjutkan dengan persamaan (2.42) [31].

$$\delta_{h_{t-1}} = U_o\ \delta_{o_t} + U_i\ \delta_{i_t} + U_f\ \delta_{f_t} + U_c\ \delta_{\tilde{c}_t} \quad (2.41)$$

$$\delta_{h_{t-1}} = \delta_{h_{t-1}} + W_y\ dy_{t-1} \quad (2.42)$$

Dimana:

U : koneksi matriks bobot di *hidden state* pada masing-masing *gate*

W_y : Matriks bobot pada *output layer*

δ_{o_t} : turunan parsial *output gate*

δ_{i_t} : turunan parsial *input gate*

δ_{f_t} : turunan parsial *forget gate*

$\delta_{\tilde{c}_t}$: turunan parsial *candidate cell state*

Setelah menghitung turunan gradien dari setiap gerbang ($f, i, \tilde{C}, o$). Maka yang selanjutnya dilakukan adalah menghitung gradien *error*. Untuk turunan parsial *error* terhadap matriks bobot W dihitung dengan persamaan (2.43), untuk turunan parsial *error* terhadap matriks bobot U dihitung dengan persamaan (2.44), dan juga turunan parsial *error* terhadap bias dengan persamaan (2.45) [33].

$$\frac{\partial E}{\partial W_\theta} = \sum_t \delta_{\theta_t} \otimes x_t \quad (2.43)$$

$$\frac{\partial E}{\partial U_\theta} = \sum_t \delta_{\theta_t} \otimes h_{t-1} \quad (2.44)$$

$$\frac{\partial E}{\partial b_\theta} = \sum_t \delta_{\theta_t} \quad (2.45)$$

Dimana:

δ_{θ_t} : turunan parsial gerbang dalam LSTM ($f, i, \tilde{C}, o$)

h_{t-1} : *hidden state time step* sebelumnya

x_t : *input pada time step* saat ini

2.8.3. *Mini Batch Stochastic Gradient Descent*

Gradient descent adalah suatu teknik yang digunakan untuk optimisasi model *deep learning*, dengan cara memperbarui parameter dan secara bertahap menurunkan nilai *error* atau *loss*. Namun cara ini sangat lambat yang mengharuskan melewati iterasi keseluruhan dataset sebelum melakukan pembaruan

pada parameter. *Minibatch stochastic gradient descent* hanya cukup mengambil beberapa sampel acak yaitu beberapa contoh dari data pelatihan untuk menghitung *gradient descent* dan rata-rata *loss* pada *minibatch* terhadap parameter model. *Mini Batch Stochastic Gradient Descent* adalah alat yang biasa digunakan untuk optimisasi *neural network. Minibatch stochastic gradient descent* mampu menyesuaikan kecepatan konvergensi dan membuat komputasi menjadi lebih efisien. *Minibatch* lebih efisien dan cepat dibandingkan dengan *stochastic gradient descent* [22]. Perhitungan memperbarui bobot dengan *stochastic gradient descent* dilakukan pada persamaan (2.46) [31].

$$\theta_{(baru)} = \theta_{(lama)} - \eta \times \frac{\partial E}{\partial \theta} \tag{2.46}$$

Dimana:

$\theta_{(baru)}$: Nilai bobot baru

$\theta_{(lama)}$: Nilai bobot lama

η : Nilai *learning rate*

$\frac{\partial E}{\partial \theta}$: Nilai turunan parsial antara nilai *error* dan bobot

2.9. Python

Python merupakan bahasa pemrograman *interpreted* yang memiliki daya tarik yang kuat, Python pertama kali muncul pada tahun 1991 dibuat oleh Guido van Rossum. Python menjadi bahasa pemrograman *interpreted* terpopuler saat ini bersama dengan Perl, Ruby, dan lain-lain. Python bisa digunakan untuk membuat *website* dengan menggunakan berbagai *framework*, salah satu yang terkenal adalah Django. Semacam bahasa yang sering dipanggil *scripting languages*, dapat digunakan dengan cepat untuk menulis program yang kecil, atau *script* untuk mengotomasi tugas lainnya. Python telah berkembang secara luas dan aktif dalam komunitas *scientific computing* dan *data analysis*. Dalam sepuluh tahun terakhir Python telah menjadi salah satu bahasa yang paling penting untuk *data science, machine learning*, dan pengembangan perangkat lunak general di area akademik dan industri. Python memiliki berbagai *library* yang populer digunakan dalam tugas *data analysis* seperti numpy, matplotlib, pandas dan scikit-learn [34].

2.10. *Confusion Matrix*

Confusion matrix adalah informasi yang disimpan dalam bentuk matriks untuk mengetahui performansi model *Machine Learning* maupun *Deep Learning* yang dipakai, dan dapat digunakan sebagai tumpuan dari performansi klasifikasi algoritma yang dipakai pada tahap evaluasi [11]. Tugasnya adalah untuk memprediksi label dari data input menjadi kelas spesifik, yang dikodekan secara numerik. Nilai yang dipakai yaitu *True* Positif (TP), *True* Negatif (TN), *False* Positif (FP), dan *False* Negatif (FN) [32]. Dapat dilihat pada tabel 2.1.

Tabel 2.1. *Confusion Matrix*

Confusion Matrix			
Nilai Aktual		Nilai Prediksi	
		Positif	Negatif
	Positif	*True* Positif (TP)	*False* Negatif (FN)
	Negatif	*False* Positif (FP)	*True* Negatif (TN)

Keterangan:

True Positif (TP) : Nilai Prediksi dari model yang cocok dengan nilai aktual, model yang diprediksi mendapatkan nilai positif, dan nilai aktualnya bernilai positif.

False Negatif (FN) : Nilai Prediksi dari model adalah negatif, sementara nilai aktual adalah positif.

False Positif (FP) : Nilai Prediksi dari model adalah positif sementara nilai aktual nya negatif

True Negatif (TN) : Nilai prediksi dari model cocok dengan nilai aktual, model yang diprediksi mendapatkan nilai negatif, dan nilai aktualnya bernilai negatif.

Perhitungan performansi yang dipakai pada penelitian ini adalah *Accuracy, Precision, Recall,* dan *F1-Score*.

1. *Accuracy*

   Akurasi didefinisikan sebagai rasio dari total jumlah nilai yang diprediksi secara benar oleh model dengan jumlah total sampel [32]. Perhitungan akurasi dapat dilakukan dengan persamaan (2.47).

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{2.47}$$

2. *Precision*

   Precision memberitahu seberapa banyak sampel positif yang diprediksi secara benar untuk semua sampel yang diprediksi positif [32]. Perhitungan *precision* dapat dilakukan dengan persamaan (2.48).

$$Precision = \frac{TP}{TP + FP} \tag{2.48}$$

3. *Recall*

   Recall memberitahu seberapa banyak sampel yang diprediksi secara benar untuk semua sampel nilai aktual yang positif [32]. Perhitungan *recall* dapat dilakukan dengan persamaan (2.49).

$$Recall = \frac{TP}{TP + FN} \tag{2.49}$$

4. *F1-score*

   F1-score merupakan hasil kombinasi dari rumus *precision* dan *recall*. Nilai *F1-Score* adalah nilai rata-rata dari *precision* dan *recall* yang memiliki sebuah nilai maksimal ketika *precision* dan *recall* setara [32]. Perhitungan *F1-score* dapat dilakukan dengan persamaan (2.50).

$$F1 - Score = 2 \times \left(\frac{Precision * Recall}{Precision + Recall}\right) \tag{2.50}$$

2.11. *Random Over Sampling (ROS)*

Random Oversampling merupakan salah satu teknik untuk menangani data yang tidak seimbang dengan cara *oversampling* atau menambah jumlah kelas yang

minoritas agar sama dengan kelas mayoritasnya. Dalam *random oversampling*, kelas minoritas dipilih secara acak atau random sesuai dengan nama tekniknya '*random*' untuk mereplika kelas agar seimbang [35].

2.12. *Literature Review*

Literature review digunakan sebagai acuan, sumber informasi, penelitian terkait, dan penelitian sebelumnya guna untuk mendukung penelitian ini, pada penelitian ini menggunakan dua belas *literature review* yang diringkas dalam bentuk tabel dapat dilihat pada tabel 2.2.

Tabel 2.2. *Literatur Review*

Penulis	Judul	Metode	Hasil
Nelly Indriani Widiastuti, Maulvi Inayat Ali	Elman Recurrent Neural Network For Aspect Based Sentiment Analysis	Elman Recurrent Neural Network	Dengan menggunakan 1584 kalimat untuk data latih dan 422 kalimat untuk data uji, dengan aspek *Food, Price, Service,* dan *Ambience*. Mendapatkan hasil akurasi yaitu Aspek *Food* dengan akurasi sebesar 72.51%, aspek *Price* mendapatkan dengan akurasi sebesar 86.97%, aspek *Service* dengan akurasi sebesar 91.94%, dan aspek *Ambience* dengan akurasi sebesar 73.46%. Secara keseluruhan *F1 Score* mendapatkan nilai 82.78%. Elman RNN dalam menangani Analisis Sentimen berbasis Aspek mendapatkan hasil akurasi terbaik yaitu sebesar 82.78%. Untuk hasil terendah dalam memprediksi label Negatif, yaitu 21% dikarenakan kata yang digunakan yaitu kata informal, dengan campuran kata Indonesia dan bahasa asing sehingga menyebabkan kata “explain” diganti dengan UNKNOWN TOKEN

Penulis	Judul	Metode	Hasil
Auliya Rahman Isnain, Agus Sihabuddin, Yohanes Suyanto	Bidirectional Long Short-Term Memory Method and Word2vec Extraction Approach for Hate Speech Detection	Bidirectional Long Short-Term Memory and Word2vec	Kasus *hate speech detection* menggunakan Bi-LSTM dan Word2Vec CBOW dengan *epoch* 10, *learning rate* 0.001 dan jumlah *neuron* 200 pada *hidden layer* menghasilkan akurasi sebesar 94.66%, presisi 99.08%, *recall* 93.74%, dan *F-measure* 96.29%. Namun BiLSTM dengan tiga *layer* mendapatkan akurasi tertinggi, yaitu sebesar 96.93% meskipun menambahkan *layer* dapat membutuhkan proses yang panjang. Dalam menguji arsitektur Word2vec CBOW mendapatkan akurasi terbesar yaitu 93.00% dibandingkan *Skip-gram* dan *One hot encoding.*
Wang Yue and Lei Li	Sentiment Analysis using Word2vec-CNN-BiLSTM Classification	Convolutional Neural Network-Bidirectional Long Short-Term Memory (CNN-BiLSTM) dengan Word2vec	Model Word2Vec-CNN-BiLSTM mendapatkan akurasi yang terbesar yaitu 91.48%. Penelitian ini membandingkan LSTM, CNN, BiLSTM, CNN-LSTM, dan CNN-LSTM dengan *word embedding* Word2Vec, akurasi terkecil didapatkan oleh Word2Vec-LSTM taitu 79.83%. Kombinasi CNN-BiLSTM memberikan hasil yang baik, dengan kemampuan CNN dalam mengekstrasi fitur, dan karakteristik BiLSTM untuk mempelajari *short-term bidirectional* teks depedensi. Kekurangannya yaitu membutuhkan lebih banyak data dan waktu dalam pelatihannya, Meskipun dengan keterbatasan itu, dapat

Penulis	Judul	Metode	Hasil
			lebih efektif dalam menklasifikasi yang membutuhkan banyak data latih. Dengan menggunakan word2vec untuk menempatkan kata dengan makna yang sama di tempat terdekat, serta dengan besarnya jumlah *corpus* untuk memperoleh *word vector* permasalah *polysemy* dapat diselesaikan. BiLSTM juga digunakan dalam menangkap *long-distance bidirectional dependencies* dan me-enkode informasi dari belakang ke depan.
Mhd. Theo Ari Bangsa, Sigit Priyanta, Yohanes Suyanto	Aspect-Based Sentiment Analysis of Online Marketplace Reviews Using Convolutional Neural Network	Convolutional Neural Network (CNN) dan Word2Vec	Penelitian ini membuktikan bahwa CNN dengan Word2Vec mendapatkan nilai skor lebih baik dibandingkan Naïve Bayes dengan rata-rata nilai akurasi 85.54%, presisi 96.12%, *recall* 88.39%, dan *f-measure* 92.02%. Klasifikasi tanpa *stemming* pada dataset meningkatkan akurasi sebesar 2.77%. Dataset yang digunakan adalah dari *web* menggunakan Selenium sebanyak 7500 data ulasan dengan Aspek *Accuracy, Quality, Service, Price, Packaging,* dan *Delivery*, dan aspek dengan akurasi tertinggi yaitu price sebesar 89.86%
Bayu Anggara Putra, Yosi Kristian, Esther Irawati Setiawan, dan Joan Santoso	Aspect based Sentiment Analysis Aduan Mahasiswa UMSIDA Dimasa Pandemi Menggunakan LSTM	Long Short-Term Memory dan GloVe	ABSA Aduan Mahasiswa UMSIDA Dimasa Pandemi, dengan menggunakan 3000 data, 3 aspek (ekonomi, Pendidikan, dan kesehatan), dengan 2 sentimen (positif dan negatif). Word Embedding yang digunakan adalah GloVe dengan dimensi

Penulis	Judul	Metode	Hasil
			50, Untuk metode LSTM menggunakan *epoch* 50-100 dan *learning rate* 0.01-0.02, menunjukkan bahwa LSTM mendapatkan akurasi 82% pada prediksi tiga aspek, 80% pada prediksi sentimen dan akurasi rata-rata 81%.
Lamei Xu, Jin Liu, Lina Wan dan Chunyong Yin	Aspect Based Sentiment Analysis for Online Reviews	Convolution Neural Network & Conditional Random Fields	Penelitian ini dengan menggunakan data ulasan dari Yelp dengan domain leptop dan restoran, *word embedding* dengan 300 dimensi menggunakan Word2Vec, ukuran *mini-batch*nya 10, *dropout rate*nya 0.5 dan nilai maksimal l2 nya 3, polaritas positif dan negatif. Metode CNN digunakan untuk ekstraksi aspek dan CRF digunakan untuk deteksi target opini mendapatkan hasil akurasi 68.34% pada domain restoran, dan 70.90% untuk domain laptop. Dimana dibandingkan dengan metode Feature+SVM, LSTM, dan RNN. Feature+SVM masih mendapatkan akurasi lebih baik dibandingkan metode yang diusulkan (CNN-CRF), RNN, dan LSTM. Rinciannya yaitu Feature+SVM akurasinya 72.10% (restoran) dan 80.89% (laptop), LSTM akurasinya 66.45% (restoran) dan 74.28% (laptop), dan RNN akurasinya 63.00% (restoran) dan 64.56% (laptop) 64.56%.
Win Lei Kay Khine, Nyein Thwet Thwet Aung	Applying Deep Learning Approach to Targeted Aspect-based	Long Short-Term Memory, SenticNet	Hasil penelitian dengan menggunakan data ulasan Yangon yang diambil dari tripadvisor dataset mengandung lebih dari 20

Penulis	Judul	Metode	Hasil
	Sentiment Analysis for Restaurant Domain	Multi-Attentive LSTM	ribu kalimat berformat XML dengan 12 aspek (AMBIENCE#GENERAL, DRINKS#QUALITY, #DRINKS#PRICES, DRINKS#STYLE_OPTIONS, FOOD#PRICES, FOOD#QUALITY, FOOD#STYLE_OPTIONS, LOCATION#GENERAL, RESTAURANT#GENERAL, RESTAURANT#PRICES, RESTAURAN#MISCELLANEOUS, dan SERVICE#GENERAL) dengan membandingkan 7 metode LSTM berbeda yaitu LSTM, TD-LSTM, TC-LSTM, AE-LSTM, AT-LSTM, ATAE-LSTM, dan SenticNetMA-LSTM. Dimana keseluruhan akurasi dari masing-masing metode adalah LSTM dengan akurasi 82.0%, TD-LSTM dengan akurasi 82.6%, TC-LSTM dengan akurasi 82.9%, AE-LSTM dengan akurasi 82.5%, AT-LSTM dengan akurasi 83.1, ATAE-LSTM dengan akurasi 84.0 dan SenticNet MA-LSTM 87.2%. Yang mana metode yang diusulkan oleh peneliti (SenticNet MA-LSTM) mendapatkan akurasi tertinggi.
Guixian Xu, Yueting Meng, Xiaoyu Qiu, Ziheng Yu, And Xu Wu	Sentiment Analysis of Comment Texts Based on BiLSTM	Bidirectional Long Short-Term Memory	Paper ini membuktikan BiLSTM adalah metode yang efektif dengan akurasi yang tinggi dibandingkan dengan RNN, CNN, LSTM dan *Naïve Bayes* dan *Word Representation* SenInfo+TF-IDF mempunyai nilai tertinggi yaitu dengan *precision*

Penulis	Judul	Metode	Hasil
			91.50%, *Recall* 92.87%, dan F1 score 92.18% dibandingkan Seninfo, TF-IDF, dan vec. BiLSTM mendapatkan *precision* sebesar 91.54%, *recall* 92.82%, dan *F1 score* 92.18% dan BILSTM mampu memecahkan masalah pada *gradient of appearance* sampai batas tertentu. BiLSTM sepenuhnya memperhatikan konteks informasi dan dapat memperoleh representasi teks dari komentar lebih baik. Namun, membutuhkan waktu yang lama dalam membentuk model *training*.
Habib Faizal Fadli, Ahmad Fathan Hidayatullah	Identifikasi Cyberbullying pada Media Sosial Twitter Menggunakan Metode LSTM dan BiLSTM	Long Short-Term Memory Dan Bidirectional Long Short-Term Memory	Metode LSTM mendapatkan akurasi 93.77%, *f1-score* 92.02% dan untuk metode Bi-LSTM mendapatkan akurasi 95.24%, *f1-score*nya 93.84% dimana Bi-LSTM lebih unggul dibandingkan LSTM.
Kuncahyo Setyo Nugroho, Ismail Akbar, Affi Nizar Suksmawati, Istiadi	Deteksi Depresi dan Kecemasan Pengguna Twitter Menggunakan Bidirectional LSTM	Long Short-Term Memory dan Bidirectional Long Short-Term Memory	Metode LSTM mendapatkan akurasi 84.91%, *precision* 76.59% dan *recall* 76.73% sementara BiLSTM mendapatkan akurasi 94.12%, *precision* 97.59% dan *recall* 83.86% untuk mendeteksi depresi dan kecemasan pengguna Twitter.
Yunxiang Zhang, Zhuyi Rao	n-BiLSTM: BiLSTM with n-gram Features for Text Classification	Bidirectional Long Short-Term Memory dengan	n-BiLSTM (BiLSTM with n-gram) untuk mengklasifikasi teks mendapatkan hasil yang lebih unggul dibandingkan LSTM atau BiLSTM biasa. Meskipun demikin BiLSTM

Penulis	Judul	Metode	Hasil
		n—gram feature	menunjukkan bawah lebih baik dari LSTM ketika mempertimbangkan informasi semantik dua arah. Pada data IMDB CORPUS hasil LSTM akurasinya 85.9, BiLSTM akurasinya 87.5, uni-BiLSTM akurasinya 88.5%, bi-BiLSTM akurasinya 88.8%, tri-BiLSTM akurasinya 88.4%. Pada Data SSTB *Corpus One vs one*, uni-BiLSTM akurasinya 89.2%, bi-BiLSTM 88.6%, tri-BiLSTM 88.5%, LSTM 85.2%, dan BiLSTM 87.3%. Dalam data IMDB *Corpus* bigram-BiLSTM mendapatkan akurasi terbesar dalam *one-vs-one* yaitu 88.8 sementara untuk data SSTb Corpus pada *one-vs-rest* (5 kelas) trigram-BiLSTM mendapatkan akurasi terbesar 48.5, di sisi lain *One-vs-one* model uni-BiLSTM mendapatkan hasil akurasi terbesar yaitu 89.2.