BAB 2
LANDASAN TEORI

2.1 Natural Language Processing

Natural Language Processing (NLP) merupakan cabang ilmu - ilmu komputer yang menjelaskan hubungan antara bahasa alami dengan komputer. NLP dapat membantu mesin untuk memahami, memproses, dan menganalisis bahasa manusia [8]. *Natural Language Processing* atau Pemrosesan bahasa alami merupakan studi yang mempelajari tentang bagaimana komputer digunakan untuk memproses atau memahami bahasa manusia (bahasa alami) untuk tujuan melakukan tugas - tugas yang berguna.

Natural Language Processing dapat membantu untuk memahami data yang dihasilkan oleh manusia dengan memahami konteksnya, memfasilitasi analisis dan penambangan pada teks. Kemajuan terbaru dalam daya komputasi dan data besar telah menyebabkan pergeseran paradigma ke arah pendekatan berbasis data, karena pendekatan ini lebih menjanjikan dan lebih mudah direkayasa. NLP ini merupakan cabang ilmu yang menyatukan linguistik komputasional, ilmu komputer, ilmu kognitif, dan kecerdasan buatan.

Secara ilmiah, tujuan NLP adalah memahami dan mereplikasi mekanisme kognitif yang mendasari bahasa manusia. Dari segi teknis, NLP bertujuan untuk menciptakan aplikasi praktis yang memungkinkan komunikasi efektif antara manusia dan komputer. Contoh aplikasi dalam NLP mencakup *Machine Translation, Question Answering and Information Retrieval, Chatbots & Dialogue System, Automatic Speech Recognition and Text-to-Speech* [9]

2.2 Analisis Sentimen

Analisis sentimen atau biasa dikenal sebagai *opinion mining* merupakan salah satu bidang studi yang dapat menganalisis pendapat seseorang, sentimen, sikap, penilaian dan emosi dari seseorang terhadap entitas dan atributnya yang

diekspresikan melalui teks. Entitas yang dimaksud dapat berupa produk, pelayanan, organisasi, individual, acara, isu, maupun jasa. Aplikasi analisis sentimen dan penelitian berfokus pada teks tertulis.

Analisis sentimen telah menjadi bidang penelitian aktif di bidang *Natural Language Processing* (NLP). Analisis tersebut berfokus pada opini yang mengekspresikan atau menunjukkan emosi positif atau negatif dalam sebuah teks. Penelitian analisis sentimen terutama dibagi menjadi tiga tingkat klasifikasi utama, yaitu: level dokumen, level kalimat dan level aspek. [10]:

1. Level Dokumen *(Document Level)*

   Tujuan dari analisis sentimen pada level dokumen adalah untuk mengklasifikasikan keseluruhan opini pada dokumen yang mengekspresikan sentimen berupa positif atau negatif. Pada level ini akan berasumsi bahwa dokumen ini hanya memiliki satu entitas atau satu topik saja.

2. Level Kalimat *(Sentence Level)*

   Pada level kalimat akan mengklasifikasikan setiap kalimat, apakah setiap kalimat mengekspresikan opini positif atau negatif. Tingkat analisis ini erat kaitannya dengan klasifikasi subjektivitas, yang membedakan antara teks objektif dan teks yang diungkapkan secara subjektif berdasarkan pandangan pribadi dan pendapat orang yang mengungkapkan pandangan tersebut.

3. Level Aspek *(Aspect Level)*

   Pada level ini, analisis sentimen bertujuan untuk mengklasifikasikan sentimen berdasarkan aspek tertentu dari suatu entitas. Pemberian opini seseorang dapat berbeda untuk entitas yang sama pada aspek yang berbeda seperti “Skill dalam bermain sepak bola dapat di dipertaruhkan, tetapi kebugaran fisik masih kurang kompetitif”

2.3 Analisis Sentimen Berbasis Aspek

Analisis Sentimen Berbasis Aspek (ASBA) merupakan bagian dari bidang *opinion mining* yang bertujuan untuk mengidentifikasi suatu polaritas teks tertulis berdasarkan aspek tertentu. Pengklasifikasian opini pada level dokumen atau kalimat seringkali tidak memadai. Dikarenakan hanya dapat mampu mengenali sentimen secara umum saja.

Analisis sentimen lebih baik secara perbandingan, dikarenakan ABSA secara langsung berfokus pada sentimen daripada struktur bahasa. ASBA bertujuan untuk menemukan sentimen terkait dengan entitas dan/atau aspeknya. ASBA memiliki dua tugas utama, yaitu ekstraksi aspek (*Aspect Extraction*) dan klasifikasi sentimen aspek (*Aspect Sentiment Classification*) [10].

1. Ekstraksi Aspek (*Aspect Extraction*)

   Aspect Extraction memiliki tugas untuk mengidentifikasi aspek yang telah dievaluasi. Misalnya, dalam kalimat, "Kualitas suara ponsel ini luar biasa" maka aspeknya adalah "kualitas suara" dari entitas yang diwakili oleh "ponsel ini." Setiap kali kita membicarakan sebuah aspek, hal yang harus diperhatikan yaitu mengetahui aspek tersebut milik entitas yang mana.

   Dalam sebuah diskusi, seringkali hanya menyebutkan aspek saja untuk kemudahan penyajian. Karena itu, ekstraksi aspek mencakup juga ekstraksi entitas. Penting untuk dicatat bahwa "ponsel ini" tidak menunjukkan aspek secara keseluruhan (Umum)
2. Klasifikasi Sentimen Aspek (*Aspect Sentiment Classification*)

   Aspect Sentiment Classification mempunyai tugas dalam menentukan apakah pendapat tentang berbagai aspek tersebut positif, negatif, atau netral. Contoh pertama pada kalimat “Kualitas suara ponsel ini luar biasa”, opini tentang aspek "kualitas suara" adalah positif. Pada contoh kalimat kedua “Saya menyukai ponsel ini”, opini ini mengenai aspek general (keseluruhan entitas) juga positif.

Tujuan dari analisis sentimen berdasarkan aspek yaitu untuk dapat mengidentifikasi aspek dari suatu entitas, dan sentimen yang diungkapkan oleh penulis komentar tentang aspek tersebut. Dalam analisis sentimen berbasis aspek, terdapat dua pendekatan. Pertama adalah pendekatan *Supervised Learning* dan Kedua adalah pendekatan *Lexicon-Based*.

Pendekatan *Supervised Learning* menggunakan algoritma pembelajaran mesin seperti *Support Vector Machine* (SVM), *Naïve Bayes*, dan *Neural Network*. Jenis pendekatan ini diterapkan untuk klasifikasi sentimen pada tingkat kalimat dan klausa karena pendekatan pada tingkat ini tidak lagi memadai. Masalah utamanya adalah bahwa fitur-fitur yang digunakan tidak mempertimbangkan dan tidak dapat menentukan entitas dan/atau aspek mana yang menjadi target dari opini tersebut.

Pendekatan *Lexicon-Based*, di sisi lain pendekatan tersebut memiliki pendekatan yang berbeda dari klasifikasi sentimen pada tingkat dokumen atau kalimat. Perbedaan kunci adalah bahwa pendekatan ini memerlukan pertimbangan eksplisit terhadap target opini, yang tidak ada dalam klasifikasi pada tingkat dokumen atau kalimat. Untuk memperhatikan target opini, kedua pendekatan tersebut dapat digunakan.

2.4 Preprocessing

Preprocessing merupakan salah satu tahap proses penting dalam memproses data teks agar lebih bersih, rapi, dan terstruktur. Tahap *preprocessing* ini sangat krusial terutama dalam klasifikasi teks dan umumnya dalam text mining. Setiap kalimat ulasan akan diberi label secara manual berdasarkan aspek dan sentimen yang terdapat dalam kalimat tersebut.

Pada tahap preprocessing akan dilakukan *case folding, cleaning, word normalization, convert negation, tokenization* dan *stopwords removal*. [11], [12]

2.4.1 Case Folding

Case Folding adalah sebuah proses yang dilakukan untuk membuat semua huruf dalam teks menjadi seragam, baik berupa huruf kapital *(uppercase)* maupun huruf kecil *(lowercase)*. Dalam penelitian ini, teks akan diubah menjadi huruf kecil *(lowercase)*.

2.4.2 Cleaning

Proses *cleaning* merupakan langkah penting dalam preprocessing yang bertujuan untuk membersihkan teks dari karakter atau entitas yang tidak relevan atau tidak diinginkan. Beberapa tindakan yang mungkin dilakukan dalam tahap cleaning antara lain menghapus karakter khusus, seperti tanda baca, simbol, dan karakter non-alfanumerik lainnya yang tidak memberikan informasi penting. Menangani URL, email, dan tautan, lalu menangani karakter berulang, seperti "hahahahaha" menjadi "haha". Terakhir yaitu menangani singkatan dan akronim dengan mengkonversinya ke bentuk lengkapnya

2.4.3 Word Normalization

Word Normalization adalah proses untuk mengubah kata-kata slang dan singkatan menjadi kata yang sebenarnya. Proses ini dilakukan karena banyaknya kata slang yang beredar, yang akan menyebabkan ketidakakuratan dalam pemrosesan teks jika tidak ditangani dengan benar. Kata-kata slang dan singkatan sering digunakan dalam komunikasi sehari-hari, terutama dalam media sosial dan pesan instan, sehingga menghambat analisis data teks jika tidak dinormalisasi.

2.4.4 Convert Negation

Convert Negation adalah proses yang dilakukan untuk menangani kata-kata negatif dalam teks agar dapat diinterpretasikan dengan benar oleh model analisis

sentimen. Dalam teks, negasi dapat mengubah makna dari kata-kata yang mengikutinya, sehingga penting untuk menangani pola negasi dengan tepat. Proses convert negation ini melibatkan identifikasi kata-kata negatif seperti "tidak", "bukan", "tak", dan sebagainya, lalu mengubah kata yang mengikutinya menjadi bentuk negatif. Contohnya seperti "tidak bahagia" menjadi "tidak_bahagia", dengan demikian model dapat mengenali frasa tersebut sebagai satu kesatuan yang memiliki makna negatif.

2.4.5 Tokenization

Tokenization adalah sebuah proses untuk dapat mengubah struktur teks atau kalimat menjadi suatu bentuk token atau potongan kata. Proses ini dilakukan dengan memisahkan setiap kata dalam kalimat berdasarkan spasi di antara mereka

2.4.6 Stopwords Removal

Stopwords dalam penelitian ini digunakan untuk menghapus kata - kata yang tidak memberikan makna signifikan dalam kalimat. Umumnya, kata-kata ini termasuk dalam kategori *stop-word*. Istilah "stop-word" biasanya merujuk pada kata-kata yang paling umum seperti "dan", "dengan", "di", dan sebagainya.

2.5 Word Embedding

Word embedding merupakan teknik dalam pemodelan Bahasa dan pembelajaran fitur yang mengubah kata-kata menjadi nilai vektor bilangan riil. Dalam *word embedding*, kata-kata yang mirip memiliki representasi nilai vektor yang serupa, sehingga kata-kata yang memiliki hubungan semantik erat akan memiliki representasi yang berdekatan dalam ruang representasi. Biasanya, teknik ini melibatkan transformasi dari ruang vektor renggang berdimensi tinggi, seperti one-hot encoding, menjadi ruang vektor padat berdimensi rendah.[13]

Metode yang umum digunakan dalam *word embedding* adalah Word2Vec. Metode tersebut merupakan model prediksi neural networks yang efisien secara komputasi untuk menghasilkan *word embedding.* Teknik ini dapat dilakukan melalui jaringan syaraf atau faktorisasi matriks untuk menghasilkan representasi kata-kata dalam bentuk vektor yang mengandung informasi semantik.[13], [14]

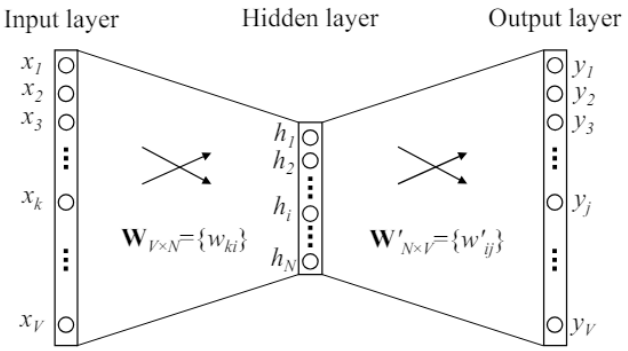
2.5.1 Word2vec Continuous Bag-of-Words

Word2Vec merupakan salah satu metode *word embedding* yang diciptakan oleh Mikolov et al. pada tahun 2013. [13] Word2Vec memiliki kemampuan untuk lebih baik mengekspresikan kemiripan antara kata-kata. Word2Vec banyak digunakan dalam penelitian NLP, terdiri dari sebuah *hidden layer* dan *fully connected layer*.

Dimensi dari matriks bobot pada masing-masing *layer* adalah jumlah kata yang terdapat dalam *vocabulary* dikalikan dengan jumlah *hidden neuron* pada *hidden layer*. Matriks bobot yang terdapat pada *hidden layer* dari model yang sudah dilatih digunakan untuk mengubah kata ke dalam bentuk vektor. Matriks bobot ini berfungsi seperti tabel pencarian, di mana setiap kata direpresentasikan oleh setiap baris, dan vektor dari kata tersebut direpresentasikan oleh kolom.

Word2Vec memiliki dua model arsitektur, yaitu *Continuous Bag-of-Words* (CBOW) dan *Continuous Skip-gram* (Skip-gram). [11]Namun, dalam penelitian ini, arsitektur yang digunakan adalah *Continuous Bag-of-Words* (CBOW). *Continuous Bag-of-Words* (CBOW) sering digunakan dalam aplikasi NLP yang berusaha memprediksi sebuah kata target berdasarkan konteks di sekitarnya.

Biasanya terdiri dari beberapa kata yang berdekatan. [13]CBOW tidak bergantung pada urutan kata atau karakteristik probabilitasnya. Keunggulannya terletak pada waktu pelatihan yang lebih singkat dan sedikit lebih akurat untuk kata-kata yang sering muncul [14]. Apabila terdapat hanya satu kata yang diambil sebagai konteks, maka arsitektur CBOW dapat dilihat pada



Gambar 2.1 Sebuah model CBOW sederhana dengan hanya satu kata dalam konteks

Dalam gambar tersebut, simbol k mewakili kata konteks, j mewakili kata target, V menunjukkan jumlah kosakata, dan N adalah jumlah lapisan tersembunyi. *Unit-layer* tersebut terhubung penuh *(fully connected)*, dan lapisan input terdiri dari vektor one-hot encoding. Ini berarti untuk kata konteks sebagai input, posisinya diisi dengan nilai 1, dan yang lainnya 0.

2.6 Oversampling

Oversampling adalah teknik yang digunakan untuk menangani ketidakseimbangan kelas dengan membuat salinan data dari kelas minoritas sehingga proporsi antara kelas minoritas dan mayoritas menjadi seimbang. Teknik ini dapat membantu mencegah model menjadi bias terhadap kelas mayoritas dan meningkatkan kinerja model dalam memprediksi kelas minoritas. Beberapa metode *oversampling* yang umum digunakan antara lain adalah *Random Over Sampling* (ROS), *Synthetic Minority Over-sampling Technique* (SMOTE), dan *Adaptive Synthetic* (ADASYN).

2.6.1 SMOTE (Synthetic Minority Over-sampling Technique)

Synthetic Minority Over-sampling Technique (SMOTE) menghasilkan sampel sintetik baru dengan melakukan interpolasi antara sampel minoritas yang ada. Teknik ini membantu mengurangi kemungkinan overfitting karena sampel baru yang dihasilkan lebih bervariasi.

Proses SMOTE dapat digambarkan sebagai berikut:

1. Memilih sampel dari kelas minoritas.
2. Untuk setiap sampel yang dipilih, pilih tetangga terdekat berdasarkan metrik jarak (misalnya, Euclidean distance).
3. Buat sampel sintetik baru dengan interpolasi antara sampel yang dipilih dan tetangganya.

SMOTE mudah diimplementasikan pada berbagai algoritma machine learning dibandingkan dengan metode yang lainnya.[15] oleh sebab itu penelitian ini akan menggunakan metode *Synthetic Minority Over-sampling Technique (*SMOTE).

2.7 Deep Learning

Deep learning merupakan salah satu cabang dari *machine learning* yang menggunakan *neural networks* dengan banyak lapisan *(deep neural networks)* untuk memodelkan hubungan kompleks antara *input* dan *output*. Terdapat keunggulan utamanya yaitu kemampuannya untuk secara otomatis dapat mengekstrak fitur-fitur yang berguna dari data yang kompleks dan besar, tanpa memerlukan proses manual pembuatan fitur yang rumit. Dengan arsitektur yang mendalam, *deep learning* mampu mempelajari representasi data secara hierarkis, di mana lapisan-lapisan yang lebih dalam akan mempelajari fitur-fitur yang semakin abstrak dan kompleks dari data.

Deep learning menjadi sangat populer dan banyak digunakan dalam berbagai aplikasi. Contohnya adalah pengenalan wajah, pengenalan ucapan, pemrosesan bahasa alami, penglihatan komputer, dan sebagainya.[10] Keberhasilan *deep learning* dalam memodelkan data yang kompleks dan *non-linear* telah mengubah cara kita menyelesaikan banyak masalah di berbagai bidang, dan terus menjadi fokus utama penelitian di dunia AI dan machine learning. Dengan kemajuan teknologi komputasi yang terus meningkat dan tersedianya sumber daya komputasi

yang kuat, *deep learning* memiliki potensi besar untuk terus berkembang dan memberikan dampak positif dalam berbagai industri dan kehidupan sehari-hari.

2.8 Artificial Neural Network (ANN)

Artificial Neural Network (ANN) merupakan sebuah model yang terinspirasi dari struktur jaringan saraf manusia. ANN terdiri dari unit-unit pemrosesan yang disebut neuron, yang terhubung dalam lapisan-lapisan. Setiap neuron menerima input, melakukan operasi matematika pada input tersebut, dan menghasilkan output yang kemudian disampaikan ke neuron berikutnya. ANN dapat digunakan untuk memodelkan hubungan kompleks antara input dan output, dan telah digunakan dalam berbagai aplikasi seperti klasifikasi, regresi, pengenalan pola, dan lainnya.

2.9 Recurrent Neural Network (RNN)

Recurrent Neural Network (RNN) merupakan salah satu jenis *neural network* dimana terdapat hubungan antar neuron yang membentuk siklus. Hal tersebut dapat memungkinkan model untuk memproses urutan data seperti teks atau waktu. RNN memiliki kemampuan untuk "mengingat" informasi dari urutan sebelumnya dalam pemrosesan.

Oleh karena itu, RNN cocok untuk tugas-tugas yang melibatkan data berurutan seperti pemrosesan bahasa alami, pemodelan bahasa, dan pengenalan ucapan. Tetapi, RNN juga memiliki beberapa tantangan, seperti masalah vanishing gradient, yang dapat menghambat kinerja model pada urutan yang panjang.

2.10 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) adalah suatu teori yang memiliki jenis pengembangan khusus dari Recurrent Neural Network (RNN). Teori tersebut

dirancang oleh Hochreiter & Schmidhuber pada tahun 1997.[11] LSTM merupakan jenis algoritma RNN yang paling populer digunakan.

LSTM dapat belajar untuk menangkap ketergantungan waktu yang panjang antara input dari langkah-langkah waktu yang berbeda. Hal tersebut dinilai efektif karena dapat mengatasi masalah hilangnya gradien yang sering terjadi pada RNN. Teori ini dianggap sebagai salah satu metode deep neural network yang paling berhasil dalam menangani data sekuensial yang panjang, dan mampu memahami pengetahuan yang tersirat.

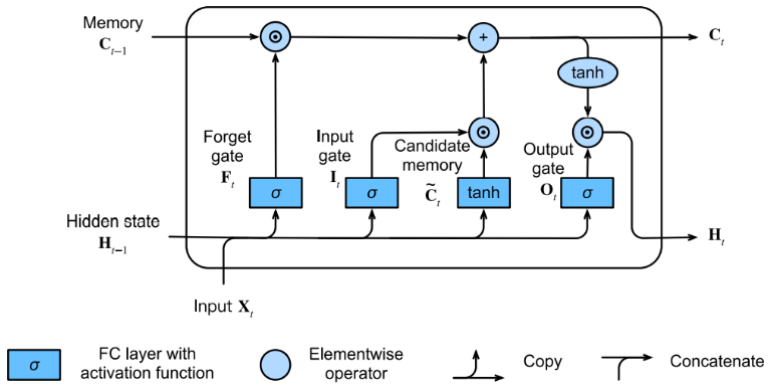
LSTM terus berkembang dan mencapai kinerja yang lebih baik dari waktu ke waktu. LSTM memiliki tiga gerbang yang memiliki fungsi untuk dapat menentukan apakah suatu informasi harus disimpan atau dilupakan dan nilai-nilai hasilnya biasanya berkisar antara 0 dan 1.[11], [16]

2.10.1 Forward Propagation

Forward Propagation pada LSTM melibatkan beberapa langkah kunci dengan tiga gerbang utama: Input Gate, Forget Gate, dan Output Gate. Proses ini memungkinkan LSTM untuk memproses informasi dari input saat ini serta mempertahankan informasi dari langkah waktu sebelumnya.

Tiga gerbang LSTM terdiri dari :

1. *Input Gate,*
2. *Forget Gate,*
3. *Output Gate,* serta sebuah *memory cell* atau *cell state.*



Gambar 2.2 Gambaran Long Short-Term Memory

Gerbang pertama yaitu Forget gate yang dimana memiliki peran untuk memutuskan informasi mana yang akan dihapus dari cell state sebelumnya. Dengan menerima input dari $t-1$ dan xt, dan menghasilkan nilai output di antara 0 dan 1. Ketika nilai forget gate adalah 1, itu menunjukkan bahwa informasi akan tetap disimpan, sedangkan ketika nilainya adalah 0, informasi akan dihapus.

Inisialisasi bobot dilakukan untuk menghitung bobot awal yang akan digunakan pada perhitungan Long Short-Term Memory (LSTM) menggunakan persamaan yang akan digunakan pada penelitian kali ini dapat dilihat pada persamaan 2.1 dengan tujuan sebagai batas keputusan yang memisahkan kelas positif dan negatif dengan margin maksimal

$$W = \left(-\frac{1}{\sqrt{d}}, \frac{1}{\sqrt{d}}\right) \quad 2.1$$

Perhitungan nilai dari forget gate didasarkan pada persamaan (2.2).[11]

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \quad 2.2$$

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*

Gerbang kedua yaitu *Input gate* yang bertanggung jawab untuk menentukan informasi mana yang perlu diperbarui dalam *cell state* pada waktu saat ini. Selanjutnya, *Candidate cell state* kemudian akan dihitung dan diintegrasikan ke dalam persamaan *Cell state*. Nilai *candidate cell state* dapat diaktivasi dengan menggunakan fungsi hiperbolik tangen yang menghasilkan nilai antara -1 dan 1.

Cell state memiliki fungsi sebagai tempat penyimpanan informasi yang diteruskan dari satu *time step* ke *time step* berikutnya. Perhitungan *input gate* dilakukan menggunakan persamaan (2.3), perhitungan *Candidate cell state* menggunakan persamaan (2.4), dan perhitungan *cell state* menggunakan persamaan (2.5).[11], [16]

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \quad 2.3$$

$$\widetilde{C_t} = tanh(W_c x_t + U_c h_{t-1} + b_c) \quad 2.4$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \widetilde{C_t} \quad 2.5$$

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} : *hasil hidden layer* sebelumnya

σ : fungsi aktivasi *sigmoid*

W_c : koneksi bobot dari x_t dan C_t

U_c : koneksi bobot dari 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*

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

c_t : *cell state time step* saat ini

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

Gerbang terakhir yaitu *Output gate* yang berfungsi untuk menentukan nilai *output* dari unit LSTM yang dihitung menggunakan persamaan (2.6). Selanjutnya *Hidden state* yang kemudian akan dihitung dari perkalian *elemen-wise* antara *output gate* dan *cell state.* Nilai dari *hidden state* biasanya berada dalam rentang -1 hingga 1 dan dihitung menggunakan persamaan (2.7).[11], [16]

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \quad (2.6)$$

$$h_t = o_t \odot tanh\,(C_t) \quad (2.7)$$

Dimana:

W_o : koneksi bobot dari x_t dan o_t

x_t : *input time step* saat ini

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

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

σ : fungsi aktivasi *sigmoid*

$tanh$: fungsi aktivasi *hyperbolic tangent*

b_o : *bias term*

o_t : *output gate*

C_t : *cell state time step* saat ini

$\odot$: operator *element-wise*

2.10.2 Backward Propagation

Backward Propagation dalam LSTM melibatkan penghitungan gradien dari loss terhadap bobot dan bias di setiap gerbang, serta memperbarui bobot dan bias menggunakan gradien ini. Proses ini dimulai dari menghitung turunan parsial nilai prediksi lstm pada output layer dengan persamaan 2.8 dan turunan parsial bobot w pada output layer menggunakan persamaan 2.9. [11], [16]

$$dy_t = -(y_{rt} - y_t) \quad 2.8$$

$$dW_y = \sum_t dy_t \otimes dy_t \quad 2.9$$

Dimana:

y_{rt} : Label sebenarnya

y_t : *output softmax* atau *sigmoid* lstm

h_t : *hidden state* lstm pada *timestep* ke t

dy_t : turunan parsial y_t output layer

Dilanjutkan dengan turunan parsial terhadap gerbangnya. Turunan parsial *output gate* (O_t) dapat dihitung menggunakan persamaan 2.10, turunan parsial *cell state* (c_t) dapat dihitung menggunakan persamaan 2.11, turunan parsial *candidate cell state* ($\tilde{C}_t$) dapat dihitung menggunakan persamaan 2.12, turunan parsial *input gate* (i_t) dapat dihitung menggunakan persamaan 2.13, turunan parsial *forgate gate* (f_t) dapat dihitung menggunakan persamaan 2.14[16]

1. Gradien Output Gate

Dimulai dengan menghitung gradien dari loss terhadap output gate $\frac{\partial L}{\partial O_t}$, dengan menggunakan persamaan 2.10

$$\frac{\partial L}{\partial o_t} = \frac{\partial L}{\partial h_t} \odot \tanh(c_t) \odot o_t \odot (1 - o_t) \quad 2.10$$

Dimana:

$\frac{\partial L}{\partial h_t}$: gradien dari loss terhadap hidden state

$\tanh(c_t)$: output cell state menggunakan tanh

$o_t \odot (1 - o_t)$: gradien dari fungsi sigmoid output gate

2. Gradien Cell State

Dilanjutkan dengan menghitung gradien dari loss terhadap cell state $\frac{\partial L}{\partial c_t}$, dengan menggunakan persamaan 2.11

$$\frac{\partial L}{\partial c_t} = \frac{\partial L}{\partial h_t} \odot o_t \odot (1 - tanh^2(c_t)) \quad 2.11$$

Dimana:

$\frac{\partial L}{\partial h_t}$: gradien dari loss terhadap hidden state

$o_t \odot (1 - tanh^2(c_t)$: gradien dari cell state dengan mempertimbangkan efek output gate

3. Gradien Input Gate

Sebelum menghitung input gate diperlukan candidate cell state dan berikut ini merupakan persamaan 2.12 yang digunakan untuk menghitung candidate cell state

$$\frac{\partial L}{\partial \tilde{C}_t} = \frac{\partial L}{\partial c_t} \odot i_t \odot (1 - \tilde{C}_t^{\,2}) \quad 2.12$$

Dimana:

$\frac{\partial L}{\partial c_t}$: gradien dari loss terhadap cell state

i_t : nilai *input gate* pada timestep t

$\tilde{C}_t$: nilai candidate cell state pada timestep t

Dilanjutkan dengan menghitung gradien dari loss terhadap input gate $\frac{\partial L}{\partial i_t}$, dengan menggunakan persamaan 2.13 berikut

$$\frac{\partial L}{\partial i_t} = \frac{\partial L}{\partial c_t} \odot \tilde{C}_t \odot i_t \odot (1 - i_t) \quad 2.13$$

Dimana:

$\frac{\partial L}{\partial c_t}$: gradien dari loss terhadap cell state

$i_t \odot (1 - i_t)$: gradien dari fungsi sigmoid input gate

4. Gradien Forget Gate

Dilanjutkan dengan menghitung gradien dari loss terhadap forgate gate $\frac{\partial L}{\partial f_t}$, dengan menggunakan persamaan 2.14 berikut

$$\frac{\partial L}{\partial f_t} = \frac{\partial L}{\partial c_t} \odot c_{t-1} \odot f_t \odot (1 - f_t) \quad 2.14$$

Dimana:

$\frac{\partial L}{\partial c_t}$: gradien dari loss terhadap cell state

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

$f_t \odot (1 - f_t)$: gradien dari fungsi sigmoid forget gate

2.11 Dropout Layer

Dropout adalah teknik regularisasi yang digunakan untuk mengurangi overfitting dalam jaringan saraf, termasuk dalam LSTM. Teknik ini bekerja dengan "menjatuhkan" (atau menonaktifkan) sejumlah neuron secara acak selama fase pelatihan. Ini mencegah neuron dari saling bergantung terlalu banyak pada satu sama lain dan memaksa jaringan untuk belajar representasi yang lebih umum dari data.[11], [16]

Selama pelatihan, output dari neuron yang tidak dijatuhkan akan diskalakan dengan faktor $\frac{1}{p}$ untuk menjaga skala keseluruhan tetap konsisten saat inferensi. Probabilitas dropout p mewakili proporsi neuron yang akan dijatuhkan.

Persamaan yang digunakan pada dropout dapat dilihat pada persamaan 2.15a

$$O_i = \frac{h_t}{1 - p} \qquad 2.15a$$

Dimana:

h_t= Output dari time step terakhir forward propagation lstm

p = Probabilitas dropout yang digunakan

Dropout layer pada backpropagation digunakan untuk memperbaharui bobot yang akan digunakan. Persamaan yang digunakan untuk dropout layer pada backpropagation dapat dilihat pada persamaan 2.15b

$$dO_i = \frac{h_t}{p} \qquad 2.15b$$

Dimana:

h_t= Output dari time step terakhir backpropagation lstm

p = Probabilitas dropout yang digunakan

Probabilitas dropout yang sering digunakan adalah 0.2 hingga 0.5, tergantung pada kompleksitas model dan ukuran dataset. Dalam penelitian ini, probabilitas dropout yang digunakan adalah 0.5, yang berarti 50% dari neuron akan dijatuhkan selama pelatihan. Pemilihan nilai ini bertujuan untuk mencapai keseimbangan antara meminimalkan overfitting dan mempertahankan kemampuan model untuk mempelajari pola yang relevan dalam data

2.12 Dense Layer

Dense layer, atau fully connected layer, adalah komponen fundamental dari jaringan saraf dalam model deep learning. Dalam lapisan ini, setiap neuron terhubung ke semua neuron di lapisan sebelumnya, sehingga menciptakan koneksi yang penuh antara lapisan-lapisan. Dense layer berfungsi untuk menggabungkan fitur-fitur yang telah diekstraksi dari lapisan sebelumnya dan menghasilkan output berdasarkan bobot dan bias yang terhubung dengan setiap neuron.[11], [16]

Persamaan yang digunakan dalam dense layer untuk memprediksi output y dari input x adalah persamaan 2.16

$$y_i = f(W_i \cdot x + b_i) \qquad 2.16$$

Dimana:

y_i = Output dari neuron ke-i dalam dense layer

W_i = Bobot yang terhubung dengan neuron ke-i

x = Input dari lapisan sebelumnya

b_i = Bias dari neuron ke-i

f = Fungsi aktivasi yang digunakan

Dalam konteks klasifikasi multi-aspek, fungsi aktivasi yang digunakan pada dense layer biasanya adalah softmax, yang mengkonversi nilai output menjadi probabilitas untuk setiap kelas. Fungsi ini memastikan bahwa probabilitas yang dihasilkan berjumlah total 1, sehingga output dapat diinterpretasikan sebagai probabilitas dari setiap kelas

2.13 Fungsi Aktivasi

Fungsi aktivasi merupakan sebuah elemen penting dalam *neural network* karena dapat membantu jaringan tersebut mengenali pola data yang bersifat *non-linear*. Karena hal tersebut *output* yang dihasilkan oleh *neural network* jarang memiliki sifat *linear*. Selain itu, fungsi aktivasi digunakan untuk mengontrol kapan neuron akan diaktifkan atau dinonaktifkan. Memungkinkan model untuk belajar dan merepresentasikan data yang kompleks.

Sifat-sifat dari jaringan syaraf tiruan ditentukan oleh bobotnya serta *input* dan *output* dari fungsi aktivasi yang diterapkan. Fungsi aktivasi secara umum yang digunakan dalam jaringan saraf termasuk ReLU (Rectified Linear Unit), sigmoid, tanh, dan softmax. Tetapi dalam konteks penelitian ini, fungsi aktivasi yang utama adalah softmax karena menggunakan klasifikasi multi kelas.[16], [17]

1. Sigmoid

Fungsi sigmoid mengubah nilai input menjadi output antara 0 dan 1, yang dapat diinterpretasikan sebagai probabilitas. Fungsi ini sangat berguna dalam kasus klasifikasi biner. Rumus fungsi sigmoid dapat dilihat pada persamaan 2.17

$$\sigma(x) = \frac{1}{1 + e^{-x}} \qquad 2.17$$

di mana:

σ(x) = Output dari fungsi sigmoid

x = input

e = eksponensial

2. Tanh

Fungsi aktivasi tanh (*tangens hiperbolik*) mengubah nilai input menjadi output dalam rentang -1 hingga 1. Fungsi ini mirip dengan fungsi sigmoid, tetapi simetris terhadap asal (0,0), yang membuatnya lebih baik dalam mengatasi data dengan distribusi negatif dan positif yang seimbang. Rumus fungsi tanh menggunakan persamaan 2.18

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \qquad 2.18$$

di mana:

x = input

Fungsi tanh memiliki keuntungan dibandingkan fungsi sigmoid karena outputnya berpusat di sekitar nol, yang sering kali mempercepat konvergensi dalam pelatihan jaringan saraf.

3. Softmax

Softmax merupakan fungsi aktivasi yang sering digunakan dalam lapisan output dari jaringan saraf untuk klasifikasi multi kelas. Fungsi ini mengambil vektor nilai real sebagai input dan mengkonversinya menjadi distribusi probabilitas. Setiap elemen dalam output softmax merupakan probabilitas dari masing-masing kelas, dan jumlah seluruh probabilitas ini adalah 2.19

Rumus dari fungsi softmax adalah sebagai berikut:

$$Softmax(Z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \quad 2.19$$

di mana:

z = vektor input dan

K = jumlah kelas

Softmax memastikan bahwa outputnya dapat diinterpretasikan sebagai probabilitas, yang berguna dalam mengklasifikasikan seperti analisis sentimen berbasis aspek.

2.14 Optimasi

Optimasi adalah proses penting dalam pelatihan model machine learning dan deep learning. Tujuan utama dari optimasi adalah untuk menemukan nilai parameter model yang meminimalkan fungsi kerugian atau cost function, sehingga model dapat melakukan prediksi yang akurat. Berbagai teknik optimasi digunakan untuk mencapai tujuan ini, dan setiap teknik memiliki kelebihan dan kekurangan tergantung pada jenis model dan data yang digunakan

2.14.1 Adaptive Moment Estimation (ADAM)

Adaptive Moment Estimation (Adam) adalah teknik optimasi yang banyak digunakan dalam pelatihan model deep learning, menggabungkan keuntungan dari dua algoritma optimasi lainnya, yaitu Momentum dan RMSProp. Adam mengestimasi momentum (rata-rata gradien) dan rata-rata kuadrat gradien untuk memperbarui bobot model, menjadikannya sangat efisien dan adaptif. Keunggulan utama Adam adalah kemampuannya untuk menyesuaikan laju pembelajaran untuk setiap parameter secara individual, yang membantu mengatasi masalah skala parameter yang berbeda dan sering mempercepat konvergensi pelatihan. Adam juga menggunakan koreksi bias untuk estimasi momentum yang memastikan bahwa

estimasi tersebut stabil sejak awal pelatihan, meningkatkan efektivitas algoritma pada iterasi awal. Perhitungan pembaharuan bobot dengan adaptive moment estimation dilakukan untuk estimasi momentum dengan persamaan 2.20[17], [18]

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot \nabla_\theta E(\theta) \quad (2.20)$$

Dimana:

m_t : estimasi momen pertama (momentum)

β_1 : parameter momen pertama

$\nabla_\theta E(\theta)$: gradien dari fungsi loss E terhadap bobot θ

Selanjutnya menghitung pembaharuan bobot dengan adaptive moment estimation dilakukan untuk rata-rata kuadrat gradien dengan persamaan 2.21

$$v_t = \beta_2 \cdot m_{t-1} + (1 - \beta_2) \cdot (\nabla_\theta E(\theta))^2 \quad (2.21)$$

Dimana:

v_t : estimasi momen kedua (rata-rata kuadrat gradien)

β_2 : parameter untuk momen kedua

$(\nabla_\theta E(\theta))^2$: kuadrat dari gradien fungsi loss

Setelah mendapatkan nilai estimasi moment selanjutnya yaitu menghitung koreksi untuk estimasi momen pertama dan kedua dengan persamaan 2.22 dan 2.23

$$\widehat{m_t} = \frac{m_t}{1 - \beta_1^t} \quad (2.22)$$

$$\widehat{v_t} = \frac{v_t}{1 - \beta_2^t} \quad (2.23)$$

Dimana:

$\widehat{m_t}, \widehat{v_t}$: estimasi momen yang telah dikoreksi bias

β_1^t, β_2^t : faktor koreksi bias

Perhitungan diakhiri dengan pembaharuan bobot dengan persamaan 2.24

$$\theta_{(baru)} = \theta_{(lama)} - \frac{\eta \cdot \widehat{m_t}}{\sqrt{\widehat{v_t}} + \epsilon} \quad 2.24$$

Dimana:

$\theta_{(baru)}$: nilai bobot yang diperbarui

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

η : nilai learning rate

ϵ : nilai kecil untuk mencegah pembagian oleh nol

2.14.2 RMSprop (Root Mean Square Propagation)

RMSprop adalah algoritma optimasi yang dirancang untuk mengatasi beberapa masalah yang sering muncul dalam teknik optimasi klasik seperti Stochastic Gradient Descent (SGD). Algoritma ini melakukan penyesuaian laju pembelajaran berdasarkan rata-rata kuadrat gradien, yang membantu menangani masalah skala parameter yang berbeda dan membuat proses pelatihan lebih stabil. Dengan menggunakan rata-rata kuadrat gradien untuk memperbarui bobot, RMSprop dapat mengurangi fluktuasi besar dalam pembaruan bobot, sehingga membuat pelatihan menjadi lebih stabil dan efisien. Penyesuaian laju pembelajaran yang dilakukan RMSprop memungkinkan algoritma ini untuk menyesuaikan laju pembelajaran untuk setiap parameter secara individu, sehingga dapat mengatasi masalah yang timbul dari skala parameter yang berbeda dan mempercepat proses konvergensi pelatihan. [19]

2.15 Loss Function

Loss function dapat digunakan untuk mengukur seberapa baik atau buruk performa sebuah model. Contoh dari beberapa *loss function* yang umum digunakan adalah *Binary Cross-Entropy* dan *Categorical Cross-Entropy*. [20]

1. ***Binary Cross-Entropy***

Binary Cross-Entropy biasa digunakan untuk mengukur kesalahan dalam kasus klasifikasi biner, yang dimana akan mengevaluasi seberapa baik probabilitas yang diprediksi model sesuai dengan label yang sebenarnya. Ini mengukur seberapa jauh prediksi probabilitas model dari nilai yang sebenarnya, baik dalam hal kelas yang benar maupun ketepatan probabilitas. Perhitungan Binary Cross-Entropy ini menggunakan persamaan 2.25

$$L_{BCE}(y, \hat{y}) = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})] \quad 2.25$$

Dimana:

y : label sebenarnya

$\hat{y}$: probabilitas prediksi dari model untuk kelas positif

2. ***Categorical Cross-Entropy***

*Categorical Cross-Entropy s*ering digunakan dalam kasus klasifikasi untuk menghitung *error* antara *output* prediksi model dengan nilai sebenarnya dari data. Secara umum tujuan *loss function* adalah untuk mengurangi *error* dalam prediksi model. Perhitungan *Categorical Cross-Entropy* ini dilakukan dengan menggunakan persamaan (2.26).

$$L_{CCE}(y, \hat{y}) = -\sum_{i=1}^{C} y_i log \hat{y}_i \quad 2.26$$

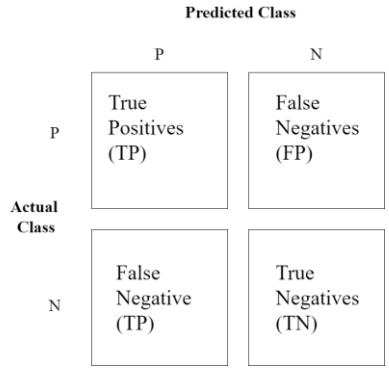
Dimana:

C : Jumlah kelas

y_i : *Output* label sebenarnya

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

2.16 Confusion Matrix



Gambar 2.3 Gambaran Confusion Matrix

Confusion matrix merupakan tabel yang digunakan untuk mengevaluasi kinerja pada suatu model klasifikasi. *Matrix* ini memiliki empat sel yang mewakili jumlah prediksi yang benar dan salah dari masing-masing kelas. Sel-sel ini biasanya diberi label sebagai *True Positive* (TP), *True Negative* (TN), *False Positive* (FP), dan *False Negative* (FN). *Confusion matrix* membantu kita memahami seberapa baik model dapat memprediksi kelas yang benar dan seberapa sering model melakukan kesalahan.[11], [10], [16]

1. Akurasi

Akurasi merupakan ukuran umum yang digunakan untuk mengevaluasi kinerja model klasifikasi. Ini mengukur seberapa sering model tersebut membuat prediksi yang benar, dihitung sebagai rasio prediksi yang benar (TP dan TN) terhadap total jumlah data. Namun, akurasi tidak selalu menjadi ukuran yang baik, terutama jika kelas target tidak seimbang. Adapun rumus dalam perhitungan akurasi dapat dilihat pada persamaan 2.28

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \times 100\% \qquad 2.28$$

2. Presisi

Presisi mengukur seberapa akurat model dalam memprediksi kelas positif, dihitung sebagai rasio *True Positive* (TP) terhadap total prediksi positif (TP dan FP). Presisi memberikan informasi tentang seberapa sedikit

prediksi positif yang salah dilakukan oleh model. Adapun rumus dalam perhitungan presisi dapat dilihat pada persamaan 2.29

$$Precision = \frac{TP}{TP + FP} \times 100\% \quad (2.29)$$

3. Recall

Recall, juga dikenal sebagai sensitivitas, mengukur seberapa baik model dalam menemukan semua *instance* yang benar dari kelas yang positif, dihitung sebagai rasio *True Positive* (TP) terhadap total jumlah instance yang seharusnya positif (TP dan FN). *Recall* memberikan informasi tentang seberapa sedikit instance positif yang terlewatkan oleh model. Adapun rumus dalam perhitungan recall dapat dilihat pada persamaan 2.30

$$Recall = \frac{TP}{TP + FN} \times 100\% \quad (2.30)$$

4. F1-Score

F1-Score adalah rata-rata harmonik dari presisi dan *recall*, memberikan keseimbangan antara kedua metrik tersebut. *F1 Score* dihitung sebagai 2 * (Presisi * *Recall*) / (Presisi + *Recall*). Hal ini memberikan pengukuran yang baik tentang kinerja model dalam memprediksi kelas positif dan meminimalkan jumlah *false negatives* dan *false positives* secara bersamaan. Adapun rumus dalam perhitungan akurasi dapat dilihat pada persamaan 2.31

$$F1\ Score = \frac{2 * (Recall * Precision)}{Recall + Precision} \quad (2.31)$$

2.17 Python

Python merupakan sebuah bahasa pemrograman tingkat tinggi yang sangat populer dalam dunia ilmu data dan kecerdasan buatan (AI). *Python* dikenal karena sintaksisnya yang mudah dipahami dan mudah digunakan. Selain itu, *Python*

membuatnya menjadi pilihan utama bagi para peneliti, pengembang perangkat lunak, dan data scientist.

Python memiliki berbagai keunggulan, termasuk dukungan untuk paradigma pemrograman yang beragam seperti pemrograman berorientasi objek, pemrograman fungsional, dan pemrograman prosedural. Salah satu alasan utama popularitas *Python* dalam ilmu data yaitu karena adanya berbagai pustaka dan framework yang kuat seperti :

1. *NumPy*, yang menyediakan dukungan untuk operasi matematika dan *array multidimention.*
2. *Pandas*, yang menyediakan struktur data dengan tingkat tinggi seperti *Data Frame* guna mempermudah manipulasi data.
3. *Matplotlib*, yang digunakan untuk memvisualisasi data.
4. *Scikit-learn,* yang menyediakan algoritma *machine learning* yang lengkap dan mudah digunakan.
5. *TensorFlow*: Library open-source untuk machine learning dan deep learning. TensorFlow menyediakan dukungan untuk membangun dan melatih model neural network yang kompleks dengan efisiensi tinggi.
6. *Keras*: API tingkat tinggi untuk membangun dan melatih model deep learning. Keras berfungsi sebagai antarmuka yang lebih user-friendly di atas TensorFlow, memudahkan pengguna untuk merancang dan eksperimen dengan model deep learning.
7. *Imbalanced-learn*: Library untuk menangani dataset yang tidak seimbang dalam machine learning. Menyediakan metode untuk oversampling dan undersampling, seperti SMOTE, untuk mengatasi ketidakseimbangan kelas dalam dataset.
8. *Gensim*: Library untuk pemrosesan bahasa alami yang mendukung model embedding teks seperti Word2Vec. Gensim memfasilitasi analisis semantik dan representasi teks dalam bentuk vektor.

9. Natural Language Toolkit (nltk): Paket untuk pemrosesan bahasa alami (NLP) dengan berbagai alat dan sumber daya untuk memanipulasi teks, termasuk tokenisasi, stemming, dan parsing.
10. Regular Expression Operations (re): Modul bawaan Python untuk operasi regular expression, yang digunakan untuk pencocokan pola dan manipulasi teks.
11. Common String Operation (string): Modul bawaan Python yang menyediakan konstanta dan fungsi untuk manipulasi string.
12. Seaborn: Library visualisasi data berbasis Matplotlib yang menyediakan antarmuka yang lebih menarik dan fungsional untuk membuat grafik statistik yang kompleks dengan mudah.
13. Sastrawi: Library khusus untuk bahasa Indonesia yang menyediakan fungsi stemming untuk memproses kata-kata dalam bahasa Indonesia.
14. Emoji: Library untuk menangani emoji dalam teks, berguna untuk analisis data teks yang mengandung simbol emoji.