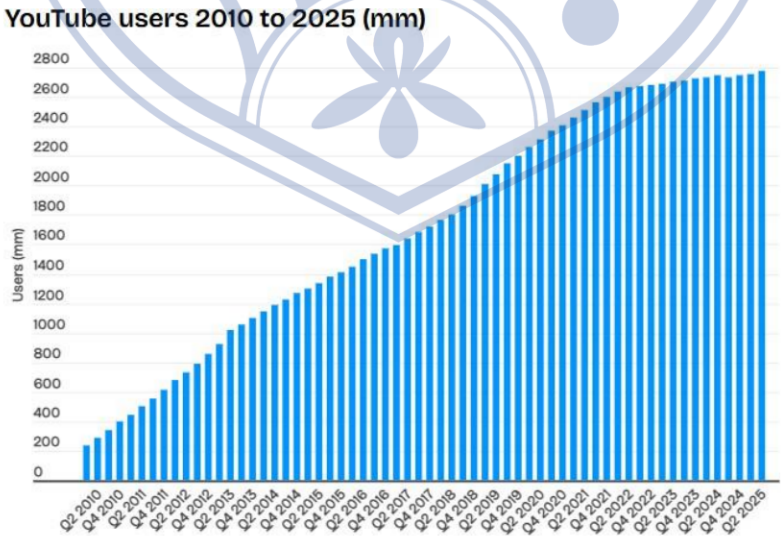
BAB II
KAJIAN LITERATUR

2.1 YouTube Sebagai Platform Media Sosial

YouTube merupakan salah satu platform media sosial terbesar dan paling populer di dunia yang memiliki berbagai keunggulan dibandingkan platform lainnya. Platform ini memungkinkan pengguna untuk menonton video tanpa harus memiliki akun, sementara pengguna yang terdaftar dapat mengunggah dan membagikan konten mereka sendiri. Selain itu, video dari YouTube juga dapat diunduh melalui layanan pihak ketiga dengan menyalin tautan video ke situs tertentu [25]. Seiring dengan perkembangan teknologi internet, YouTube telah bertransformasi dari sekadar repositori video menjadi media *edutainment* interaktif yang terbukti mampu memberikan dampak positif, seperti menumbuhkan motivasi belajar siswa melalui tayangan yang bernilai edukatif [25][26].

Seiring meningkatnya popularitas platform media sosial YouTube, analisis sentimen menjadi metode yang penting bagi berbagai organisasi dan perusahaan untuk memahami serta mengukur opini dan persepsi publik terhadap suatu merek maupun produk [25]. Fitur ini berfungsi sebagai ruang publik digital di mana audiens dapat mengekspresikan opini, memberikan kritik, saran, maupun dukungan emosional terhadap video yang ditonton melalui pesan teks maupun fitur like dan dislike [27]. Komentar-komentar ini menghasilkan tumpukan data teks tak terstruktur yang kaya akan informasi mengenai persepsi, preferensi, dan sentimen masyarakat terhadap suatu topik atau tayangan secara luas [27][28].



Gambar 2.1 Distribusi Pengguna Youtube [29]

© Karya Dilindungi UU Hak Cipta
1. Dilarang menyebarluaskan dokumen tanpa izin.
2. Dilarang melakukan plagiasi.
3. Pelanggaran diberlakukan sanksi sesuai peraturan UU Hak Cipta.

Namun, karakteristik teks pada komentar YouTube memiliki tantangan tersendiri dalam pemrosesan bahasa alami (*Natural Language Processing* / NLP). Sebagai sumber komunikasi informal, teks media sosial cenderung tidak konsisten dan rentan terhadap variasi tekstual yang menghasilkan fenomena *Out-of-Vocabulary* (OOV), di mana kata-kata yang digunakan tidak terdaftar dalam kamus standar bahasa [25]. Audiens sering kali menggunakan bahasa yang tidak baku, mengandung *slang* (bahasa gaul), singkatan, hingga kesalahan ejaan dan tata bahasa seperti *typo* [28]. Meskipun demikian, melalui metode *preprocessing* yang tepat serta penggunaan arsitektur *Deep learning* yang dikombinasikan dengan teknik normalisasi teks, ekstraksi informasi dari komentar YouTube dapat memberikan *insight* berharga bagi penyelenggara acara, seperti Ruangguru. Hal ini memungkinkan evaluasi kualitas tayangan dan pengukuran tingkat penerimaan masyarakat secara lebih objektif [28].

2.2 Clash of Champions Ruangguru

Ruangguru didirikan pada tahun 2014, Ruang Guru juga telah memperoleh berbagai penghargaan nasional dan internasional, yang menyatakan posisinya sebagai penyedia layanan pendidikan digital terkemuka di Indonesia [30]. Penelitian ini menyoroti salah satu program Ruang Guru yang sempat *viral* di media sosial khusus nya di kalangan anak muda, yaitu *Clash of Champions* (COC). *Clash of Champions* merupakan *game show* yang diselenggarakan oleh Ruangguru dengan penayangan perdananya season 1 dengan 11 episode pada 29 Juni 2024 dan penayangan season 2 dengan 16 episode di tahun 2025 yang dapat di tonton di akun YouTube Official Ruangguru [31].

Clash of Champion dibangun dengan konsep *edutainment*, yaitu penggabungan antara unsur pendidikan dan hiburan [28]. Program ini melibatkan mahasiswa berprestasi dari berbagai universitas ternama dunia seperti NUS, NTU, Yale, dan KAIST, serta perguruan tinggi unggulan di Indonesia seperti UGM, ITB, UI, BINUS, ITS, dan UNAIR. *Clash of Champions* mulai ditayangkan melalui aplikasi Ruangguru dan kanal YouTube sehingga dapat menjangkau *audiens* yang lebih luas. Setiap episode menghadirkan berbagai tantangan berbasis logika, matematika, memori, dan strategi, seperti *Extreme Addition*, *Cryptarithm War*, dan *Memory Madness*, yang dirancang untuk menguji kemampuan berpikir serta ketahanan mental para peserta [32].

Tidak hanya fokus pada pengujian kemampuan akademik. *Clash of Champions* juga berperan sebagai wadah pengembangan karakter dengan menanamkan nilai-nilai penting

© Karya Dilindungi UU Hak Cipta
1. Dilarang menyebarluaskan dokumen tanpa izin.
2. Dilarang melakukan plagiasi.
3. Pelanggaran diberlakukan sanksi sesuai peraturan UU Hak Cipta.

seperti kerja sama tim, keberanian dalam menghadapi tantangan, serta kedisiplinan. Konsep ini menempatkan kompetisi intelektual sebagai inti utama, di mana peserta diuji kemampuan berpikir logika, penghafalan, analitis, perhitungan dan pemecahan masalah dalam suasana yang kompetitif dan dinamis [33]. COC ini menarik perhatian publik secara luas dan menunjukkan tingkat keterlibatan yang tinggi di berbagai media sosial terutama di platform YouTube [1]. Popularitas tersebut menjadikan COC sebagai fenomena *edutainment* yang menarik untuk dikaji.

2.3 Analisis Sentimen

Analisis sentimen merupakan salah satu cabang penelitian dari *text mining* yang memanfaatkan teknik penggalian informasi secara otomatis untuk mengidentifikasi opini dan emosi dalam teks [34]. Metode ini secara intensif digunakan untuk menganalisis berbagai aspek subjektif seperti penilaian, sikap, serta perasaan seseorang terhadap suatu entitas seperti layanan, produk, atau fenomena sosial tertentu [35]. Dalam konteks ulasan di media sosial, *model* yang digunakan dituntut untuk mampu memahami konteks serta makna yang tersembunyi [36]. Pernyataan fakta adalah pernyataan apa adanya tentang suatu kejadian atau ciri-ciri sesuatu, tanpa dicampuri pendapat atau perasaan pribadi [35]. Sebaliknya, opini merupakan ekspresi subjektif yang mencerminkan sentimen dan penilaian individu terhadap suatu hal yang sedang dibahas [35]. Implementasi teknologi ini sangat krusial untuk menentukan tingkat kepuasan serta persepsi audiens terhadap suatu topik secara menyeluruh di media sosial [34][36]. Dalam praktiknya, klasifikasi analisis sentimen umumnya dibagi ke dalam tiga kategori utama[34][37]:

1. Sentimen Positif: Merupakan klasifikasi teks yang terdiri dari kumpulan kata-kata bermakna baik, menguntungkan, atau mendukung subjek yang dibahas [35]. Jika sebuah teks memiliki dominasi kata positif yang lebih tinggi dibandingkan kata negatif, maka teks tersebut ditetapkan sebagai sentimen positif [34].
2. Sentimen Negatif: Merupakan klasifikasi yang menggunakan kumpulan kata-kata bersifat buruk, kritis, atau menunjukkan ketidakpuasan [35]. Suatu ulasan akan dikategorikan sebagai sentimen negatif apabila kuantitas kata bermakna negatif di dalamnya jauh lebih besar [34].
3. Sentimen Netral: Merupakan klasifikasi yang diberikan ketika suatu teks tidak menunjukkan kecenderungan emosional yang kuat, baik ke arah positif maupun negatif [37]. Kategori ini biasanya berisi pernyataan umum atau informasi faktual yang tidak

© Karya Dilindungi UU Hak Cipta
1. Dilarang menyebarluaskan dokumen tanpa izin.
2. Dilarang melakukan plagiasi.
3. Pelanggaran diberlakukan sanksi sesuai peraturan UU Hak Cipta.

mengandung unsur penilaian rasa [35].

Analisis sentimen pada teks berbahasa Indonesia memiliki berbagai tantangan yang lebih kompleks dibandingkan teks formal pada umumnya. Salah satu tantangan utama adalah penggunaan bahasa informal pada media sosial. Pengguna media sosial sering menulis komentar menggunakan bahasa sehari-hari yang tidak sesuai dengan kaidah bahasa baku, seperti penggunaan singkatan, slang, emoji, campuran bahasa asing, serta kesalahan penulisan [38]. Kondisi tersebut menyebabkan proses pemahaman konteks oleh sistem menjadi lebih sulit. Selain itu, keberadaan singkatan dan slang juga menjadi kendala dalam proses analisis sentimen. Kata-kata seperti "gk", "btw", "mantul", "bjir", atau "kereeen" memiliki makna tertentu yang sering kali tidak terdapat dalam kamus bahasa formal [39]. Variasi penulisan yang tidak konsisten menyebabkan model analisis sentimen perlu memahami makna kontekstual agar dapat mengklasifikasikan sentimen secara tepat. Tantangan lain dalam analisis sentimen bahasa Indonesia adalah keberadaan *noisy text* atau teks tidak terstruktur. *Noisy text* dapat berupa pengulangan huruf, penggunaan simbol berlebihan, tautan, tagar, maupun kombinasi angka dan huruf dalam suatu kalimat [35]. Contoh *noisy text* seperti "baguuuusss!!!", "wkwkwk", atau "mantapppp" dapat menurunkan kualitas hasil klasifikasi apabila tidak dilakukan *preprocessing* secara tepat.

Selain permasalahan bahasa informal, analisis sentimen juga sering menghadapi masalah ketidakseimbangan data (*class imbalance*), yaitu kondisi ketika jumlah data pada satu kategori sentimen jauh lebih dominan dibandingkan kategori lainnya [40]. Ketidakseimbangan tersebut dapat menyebabkan model cenderung memprediksi kelas mayoritas dan mengabaikan kelas minoritas sehingga performa klasifikasi menjadi kurang optimal. Analisis sentimen bahasa Indonesia juga memiliki tantangan dalam memahami konteks kalimat. Sebuah kata dapat memiliki makna berbeda tergantung pada konteks penggunaannya dalam kalimat [41]. Oleh karena itu, diperlukan metode *machine learning* maupun *deep learning* yang mampu memahami hubungan kontekstual antar-kata agar proses klasifikasi sentimen dapat dilakukan secara lebih akurat.

2.4 Pre-processing

Pre-processing teks merupakan tahapan awal dalam pengolahan data yang bertujuan untuk membersihkan, menormalisasi, dan mentransformasikan data teks mentah menjadi bentuk yang lebih terstruktur sehingga dapat diproses oleh sistem machine learning maupun deep learning [38]. Tahapan ini sangat penting dalam analisis sentimen karena data teks pada

© Karya Dilindungi UU Hak Cipta
1. Dilarang menyebarluaskan dokumen tanpa izin.
2. Dilarang melakukan plagiasi.
3. Pelanggaran diberlakukan sanksi sesuai peraturan UU Hak Cipta.

media sosial umumnya mengandung *noise*, seperti simbol, URL, emoji, kata tidak baku, serta kesalahan penulisan yang dapat memengaruhi performa model .

Secara umum, *preprocessing* bertujuan untuk meningkatkan kualitas data dengan mengurangi informasi yang tidak relevan dan mempertahankan informasi penting yang berkaitan dengan makna teks [42]. Tanpa *preprocessing*, model akan kesulitan memahami pola bahasa sehingga akurasi klasifikasi dapat menurun [38]. Dalam penelitian analisis sentimen, *preprocessing* teks umumnya meliputi beberapa tahapan, yaitu *text cleaning, case folding, tokenizing, stopword removal, stemming,* dan normalisasi kata slang. Dalam praktiknya, terdapat beberapa tahapan spesifik yang dilakukan selama proses *preprocessing* teks [8]:

2.4.1 Case Folding

Case Folding merupakan tahapan yang bertujuan untuk menyeragamkan seluruh karakter teks menjadi huruf kecil (*lowercase*).

$$Char_{lower} = Char_{upper} + 32 \quad (2.1)$$

Di mana**:**

$Char_{lower}$ = Karakter dalam format huruf kecil (*lowercase*).

$Char_{upper}$ = Karakter dalam format huruf kapital (*uppercase*).

32 = Nilai selisih (*offset*) desimal antara huruf kapital dan huruf kecil pada tabel ASCII.

2.4.2 Text Cleaning

Data Cleansing merupakan proses pembersihan teks dari karakter yang tidak diperlukan seperti *hashtag*, *URL*, *mention*, dan simbol-simbol non-alfabet lainnya.

$$S_{\{clean\}} = \{c \in S \mid c \in A\} \quad (2.2)$$

Di mana:

$S_{\{clean\}}$ = Rangkaian teks hasil pembersihan.

S = Rangkaian teks asli (*input*).

c = Karakter individu dalam teks.

A: Himpunan karakter yang diizinkan (misalnya alfabet $\boldsymbol{a}$ ... $\boldsymbol{z}$ dan spasi), sehingga karakter di luar himpunan tersebut (seperti simbol, *hashtag*, atau URL) akan dibuang.

© Karya Dilindungi UU Hak Cipta
1. Dilarang menyebarluaskan dokumen tanpa izin.
2. Dilarang melakukan plagiasi.
3. Pelanggaran diberlakukan sanksi sesuai peraturan UU Hak Cipta.

2.4.3 Word Splitting

Word Splitting atau biasa dikenal dengan istilah *Tokenizing* merupakan proses memecah rangkaian teks menjadi satuan kata tunggal (*token*) sekaligus menghapus tanda baca spasi yang digunakan sebagai pemisah kata dalam kalimat.

$$T = f_{tok}(S) \rightarrow \{w_1, w_2, \dots, w_n\} \quad (2.3)$$

Di mana:

S = String tunggal

f_{tok} = Fungsi *Tokenisasi*

T = Himpunan token (hasil akhir)

$w_1, w_2, \dots, w_n$ = Token ke –n

2.4.4 Stopword Removal

Stopword Removal merupakan tahap pengambilan kata-kata bermakna penting dengan cara membuang kata sambung atau kata tugas yang sering muncul namun tidak memiliki bobot sentimen.

$$T_{\{clean\}} = \{w \in T \mid w \notin L_{\{stopword\}}\} \quad (2.4)$$

Di mana:

$T_{\{clean\}}$ = Himpunan kata setelah proses *stopword removal*.

w = Kata individu dalam rangkaian teks.

T = Himpunan kata hasil *tokenizing*.

$L_{\{stopword\}}$ = Daftar *stopword* (kata yang akan dihapus).

2.4.5 Normalisasi Kata Slang

Remove Slang merupakan proses konversi kata tidak baku, bahasa gaul, atau singkatan menjadi kata baku sesuai dengan kamus bahasa Indonesia agar makna teks tetap konsisten. Proses normalisasi teks atau *remove slang* dilakukan dengan menggunakan kamus kata alay (*colloquial Indonesian lexicon*) untuk menangani kata-kata tidak baku.

$$w_{baku}\begin{cases} f_{map(w_{slang})}, & jika\ w_{slang} \in D \\ w_{slang}, & jika\ w_{slang} \notin D \end{cases} \quad (2.5)$$

Di mana:

$w_{\{baku\}}$ = Kata hasil normalisasi (format baku).

$w_{\{slang\}}$ = Kata *input* yang terindikasi tidak baku atau singkatan.

© Karya Dilindungi UU Hak Cipta
1. Dilarang menyebarluaskan dokumen tanpa izin.
2. Dilarang melakukan plagiasi.
3. Pelanggaran diberlakukan sanksi sesuai peraturan UU Hak Cipta.

$f_{\{map\}}$ = Fungsi pemetaan (*mapping*) yang mengubah kata gaul menjadi kata baku.

D = Kamus *slang* (*slang dictionary*) yang berisi pasangan kata tidak baku dan padanannya.

2.4.6 Stemming

Stemming merupakan tahapan untuk mengubah suatu kata yang berimbuhan menjadi kata dasar dengan menghilangkan prefiks, konfiks, maupun sufiks.

$$Stem(w) = w - (Prefix(w) + Infix(w)\, Suffix(w)) \quad \text{.....(2.6)}$$

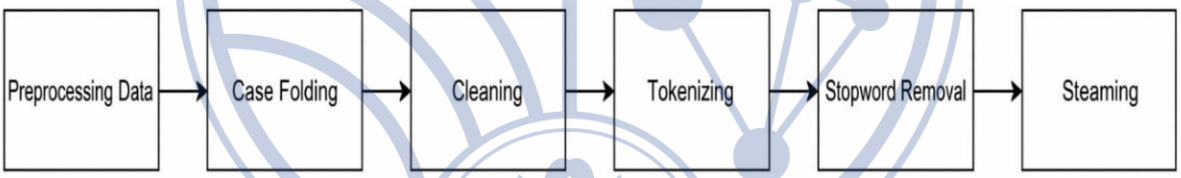
Di mana:

w = Kata berimbuhan,

$Pre\,fix(w),\ Infix(w),\ Suffix(w)$ = Masing-masing merupakan awalan, sisipan, dan akhiran yang terdeteksi pada kata w,

Hasil $Stem(w)$ = Kata dasar dari w.

Normalisasi kata slang membantu menjaga konsistensi makna kata sehingga model dapat memahami konteks teks dengan lebih baik. Gambar 2.2 menunjukkan alur umum *preprocessing teks* dalam analisis sentimen, mulai dari proses pembersihan data hingga transformasi teks menjadi bentuk yang siap digunakan pada tahap pemodelan.



Gambar 2.2 Preprocessing Dataset [35]

2.4.7 Pelabelan Data

Pelabelan data merupakan tahapan krusial untuk menentukan kelas sentimen (target) pada setiap baris komentar sebelum digunakan dalam proses pelatihan model. Dalam penelitian ini, proses pelabelan dilakukan secara otomatis menggunakan metode *lexicon-based* dengan kamus InSet (*Indonesian Sentiment Lexicon*). InSet dipilih karena merupakan leksikon yang dikembangkan khusus untuk bahasa Indonesia yang terdiri dari ribuan entitas kata dengan bobot skor sentimen yang telah ditentukan.[43][44].

Proses pelabelan dilakukan dengan menghitung nilai total sentimen dari setiap kata

© Karya Dilindungi UU Hak Cipta
1. Dilarang menyebarluaskan dokumen tanpa izin.
2. Dilarang melakukan plagiasi.
3. Pelanggaran diberlakukan sanksi sesuai peraturan UU Hak Cipta.

dalam sebuah komentar. Setiap kata yang teridentifikasi dalam kamus InSet akan memberikan kontribusi skor positif atau negatif. Hasil akhir sentimen ditentukan berdasarkan akumulasi skor tersebut dengan ketentuan sebagai berikut:

1. Sentimen Positif: Jika total skor sentimen bernilai lebih besar dari 0 (skor > 0).
2. Sentimen Negatif: Jika total skor sentimen bernilai kurang dari 0 (skor < 0).
3. Sentimen Netral: Jika total skor sentimen bernilai sama dengan 0 (skor = 0).

Penggunaan InSet Lexicon dalam penelitian ini sangat efektif untuk menangani dataset berskala besar, seperti 202.558 komentar YouTube, karena memungkinkan proses pemberian label dilakukan secara konsisten dan efisien dibandingkan pelabelan manual oleh manusia (*human labeling*) [45]. Pembobotan secara manual dilakukan oleh dua penutur asli bahasa Indonesia yang telah diberikan instruksi yang sama sebelum melakukan penilaian [sitasi]. Hasil dari pelabelan otomatis ini kemudian menjadi landasan bagi model *hybrid* LSTM dan GRU untuk mempelajari pola fitur teks berdasarkan label yang telah dihasilkan.

2.4.8 Ketidakseimbangan Data (*Imbalance*)

Ketidakseimbangan kelas (*class imbalance*) adalah kondisi di mana jumlah sampel antar kelas dalam suatu dataset berbeda secara signifikan. Dalam konteks analisis sentimen media sosial, kondisi ini kerap terjadi karena opini pengguna cenderung didominasi oleh sentimen tertentu, sehingga distribusi data menjadi tidak proporsional. Akibatnya, model pembelajaran mesin cenderung lebih banyak mempelajari pola dari kelas mayoritas dan mengabaikan karakteristik kelas minoritas. Hal ini berdampak pada menurunnya kemampuan model dalam mengklasifikasikan data dari kelas minoritas, yang umumnya tercermin dari rendahnya nilai *recall* dan *F1-score* pada kelas tersebut [46].

Salah satu pendekatan yang umum digunakan untuk menangani permasalahan ini adalah *Class Weight* dan penelitian ini memilih menggunakan metode *Class Weight*. Alasan utamanya adalah untuk menjaga otentisitas dan integritas semantik dari data teks asli.[22]. Selain itu, volume kelas minoritas dalam penelitian ini (Netral: 58.145; Negatif: 50.470) dinilai sudah memadai secara kuantitas untuk melatih model mengeksplorasi variasi linguistik tanpa memerlukan *oversampling*. Penggunaan *Class Weight* juga meminimalkan risiko amplifikasi kesalahan label (*mislabeling*) bawaan dari kamus yang terdeteksi pada dataset asli sebesar 0,41% pada ketiga kelas, sekaligus menjaga efisiensi komputasi selama proses pelatihan model.

© Karya Dilindungi UU Hak Cipta
1. Dilarang menyebarluaskan dokumen tanpa izin.
2. Dilarang melakukan plagiasi.
3. Pelanggaran diberlakukan sanksi sesuai peraturan UU Hak Cipta.

Class Weight bekerja dengan memberikan bobot yang lebih besar pada kelas minoritas dan bobot yang lebih kecil pada kelas mayoritas selama proses pelatihan [22]. Dengan cara ini, kesalahan prediksi pada kelas minoritas akan menghasilkan penalti yang lebih besar pada fungsi *loss*, sehingga model didorong untuk memberikan perhatian yang lebih merata terhadap seluruh kelas [23]. Penerapan *Class Weight* banyak digunakan pada model *deep learning* karena dapat mengatasi ketidakseimbangan data tanpa perlu menambahkan sampel baru, sehingga risiko *overfitting* akibat proses *oversampling* dapat diminimalkan. Sejumlah penelitian menunjukkan bahwa penggunaan *Class Weight* mampu meningkatkan kemampuan model dalam mengenali kelas minoritas, serta berkontribusi pada peningkatan nilai *Recall*, *Precision*, dan *F1-Score* pada tugas klasifikasi teks dengan distribusi kelas yang tidak seimbang [48].

Secara matematis, bobot untuk setiap kelas dihitung menggunakan persamaan berikut:

$$\omega_i = \frac{N}{K \times n_i} \quad \text{.....(2.7)}$$

Keterangan:

ω_i = Bobot untuk kelas ke-i.

N = Jumlah seluruh sampel pada dataset.

K= Jumlah kelas.

n_i = Jumlah sampel pada kelas ke-i.

Berdasarkan rumus tersebut, bobot yang diberikan kepada setiap kelas berbanding terbalik dengan jumlah sampel pada kelas tersebut [22]. Artinya, semakin sedikit jumlah sampel suatu kelas, semakin besar bobot yang diperolehnya dan sebaliknya.

2.5 Representasi Teks

Representasi teks adalah proses transformasi data tekstual yang bersifat tidak terstruktur ke dalam format numerik agar dapat diproses oleh algoritma *Machine Learning* atau *Deep learning* [34][35][37]. Mengingat komputer tidak dapat memahami makna semantik kata secara langsung, sehingga diperlukan proses pemetaan setiap kata ke dalam bentuk angka (*integer)* [34]. Dalam arsitektur jaringan saraf tiruan seperti LSTM dan GRU, representasi teks bertujuan untuk mempertahankan hubungan antar kata dan konteks informasi dalam sebuah kalimat [38]. Teknik *Word Embedding* sering digunakan sebagai standar representasi teks berbasis semantik selain itu representasi teks juga dapat dilakukan

© Karya Dilindungi UU Hak Cipta
1. Dilarang menyebarluaskan dokumen tanpa izin.
2. Dilarang melakukan plagiasi.
3. Pelanggaran diberlakukan sanksi sesuai peraturan UU Hak Cipta.

melalui tiga tahapan utama: pertama, melakukan *Tokenization* untuk memberikan indeks numerik pada setiap kata unik; kedua, menyusun indeks tersebut menjadi *Sequence* berdasarkan urutan kalimat; dan ketiga, menerapkan *Padding* untuk menyeimbangkan panjang data agar konsisten saat masuk ke tahap pemodelan.

2.5.1 Word Embedding

Word Embedding merupakan teknik representasi kata di mana kata-kata yang memiliki kemiripan makna akan dipetakan ke dalam titik-titik koordinat yang berdekatan dalam ruang vektor berdimensi rendah [38]. Berbeda dengan teknik *One-Hot Encoding* yang menghasilkan matriks jarang (*sparse*) dan besar, *Word Embedding* menghasilkan vektor padat (*dense vector*) yang dimensinya dapat ditentukan (misal: 100 atau 300 dimensi) [38]. Dalam praktiknya, terdapat dua pendekatan utama dalam mengimplementasikan teknik ini, yaitu menggunakan vektor yang telah dilatih sebelumnya (*pre-trained*) atau membangun lapisan vektor yang dilatih bersamaan dengan model (*trainable*). Keunggulan dari pendekatan *trainable embedding* dibandingkan *pre-trained embedding* (seperti Word2Vec atau FastText) terletak pada kemampuannya untuk menyesuaikan representasi kata secara spesifik terhadap konteks dan karakteristik *dataset* yang digunakan. Hal ini membuat representasi yang dihasilkan menjadi lebih adaptif terhadap domain penelitian. Dengan memanfaatkan *Embedding Layer* sebagai bagian dari arsitektur model maka model akan mempelajari bobot vektor setiap kata secara otomatis selama proses pelatihan (*training*) berlangsung guna menangkap fitur-fitur penting dari komentar YouTube secara optimal[38]. Secara matematis, proses ini memetakan setiap kata W ke dalam sebuah vektor berdimensi tinggi [38] :

$$v = W \cdot x \quad (2.8)$$

Di mana :

W = matriks bobot *embedding*

x = *one-hot vector* dari kata tersebut

2.5.2 Integer Encoding

Dalam pemrosesan bahasa alami (*Natural Language Processing*), teks harus melalui tahap standarisasi agar dapat diproses oleh model secara optimal. Tahapan ini dimulai dengan proses *Integer Encoding* menggunakan pustaka Keras. Pada tahap ini, setiap kata unik dari hasil *pra-pemrosesan* (yang telah dipisahkan berdasarkan karakter spasi) akan

© Karya Dilindungi UU Hak Cipta
1. Dilarang menyebarluaskan dokumen tanpa izin.
2. Dilarang melakukan plagiasi.
3. Pelanggaran diberlakukan sanksi sesuai peraturan UU Hak Cipta.

dipetakan ke dalam indeks bilangan bulat (*integer*) tertentu berdasarkan daftar kosakata (*vocabulary*) yang telah dibentuk [34]. Namun, pemberian indeks saja belum cukup karena arsitektur *Deep Learning* memerlukan input dengan dimensi matriks yang konsisten. Mengingat setiap komentar memiliki jumlah kata yang bervariasi, diperlukan standarisasi panjang urutan data agar seragam [38]. Oleh karena itu, proses representasi ini melibatkan tahapan teknis sebagai berikut:

1. *Tokenizer*: Tahap ini bertujuan untuk memetakan setiap kata unik ke dalam sebuah indeks integer tunggal untuk membentuk kamus atau *vocabulary* (V).

$$V = \{w_1: 1, w_2: 2, \dots, w_m: m\} \quad \text{.....(2.9)}$$

Di mana:

V = Kamus kosakata (*vocabulary*).

w = Kata unik dalam korpus data.

m = Jumlah total kata unik dalam kamus.

2. *Text-to-Sequence*: Proses mengubah kalimat (S) yang terdiri dari rangkaian kata menjadi daftar urutan angka (T) berdasarkan kamus V yang telah dibentuk sebelumnya.

$$T = f_{\{seq\}(S,V)} \rightarrow \{v_1, v_2, \dots, v_n\} \quad \text{.....(2.10)}$$

Di mana:

T = Rangkaian angka (*sequence*).

$f_{\{seq\}}$ = Fungsi transformasi teks ke angka.

v_i = Indeks integer dari kata ke-i dalam kalimat berdasarkan kamus V.

n = Jumlah kata dalam satu kalimat.

3. *Padding*: Teknik untuk menyamakan panjang setiap *sequence* dengan menambahkan nilai nol (*zero-Padding*) pada kalimat yang lebih pendek dari panjang maksimum (*max_length*) atau memotong kalimat yang terlalu panjang (*truncating*).

$$P = \begin{cases} k\,[v_1, v_2, \dots, v_n, 0_{\{n+1\}}, \dots, 0_L], & jika\ n < L \\ [v_1, v_2, \dots, v_L], & jika\ n > L \end{cases} \quad \text{....(2.11)}$$

Di mana:

P = *Sequence* hasil *Padding* atau *truncating*.

T = *Sequence* awal yang memiliki panjang n.

L = Panjang maksimal yang ditentukan (*max_length*).

© Karya Dilindungi UU Hak Cipta
1. Dilarang menyebarluaskan dokumen tanpa izin.
2. Dilarang melakukan plagiasi.
3. Pelanggaran diberlakukan sanksi sesuai peraturan UU Hak Cipta.

$\boldsymbol{v}$ = Nilai indeks kata di dalam *sequence*.

0 = Nilai pengisi (*zero-Padding*) yang ditambahkan jika panjang kalimat kurang dari *L*.

Data akan di proses transformasi dari teks asli hingga menjadi data yang siap diolah.

2.6 Deep learning

Deep learning (DL) merupakan subbidang dari *machine learning* yang memanfaatkan jaringan saraf tiruan berlapis (*deep neural networks*) untuk melakukan ekstraksi fitur secara hierarkis. Berbeda dengan metode *machine learning* konvensional yang membutuhkan rekayasa fitur manual, DL mampu mempelajari representasi data secara otomatis dari tingkat rendah hingga tinggi [49]. Kemajuan DL saat ini didorong oleh tiga faktor utama: ketersediaan data skala besar (*big data*), peningkatan kapasitas komputasi (GPU), serta inovasi algoritma yang memungkinkan pelatihan model yang lebih dalam dan kompleks [50].

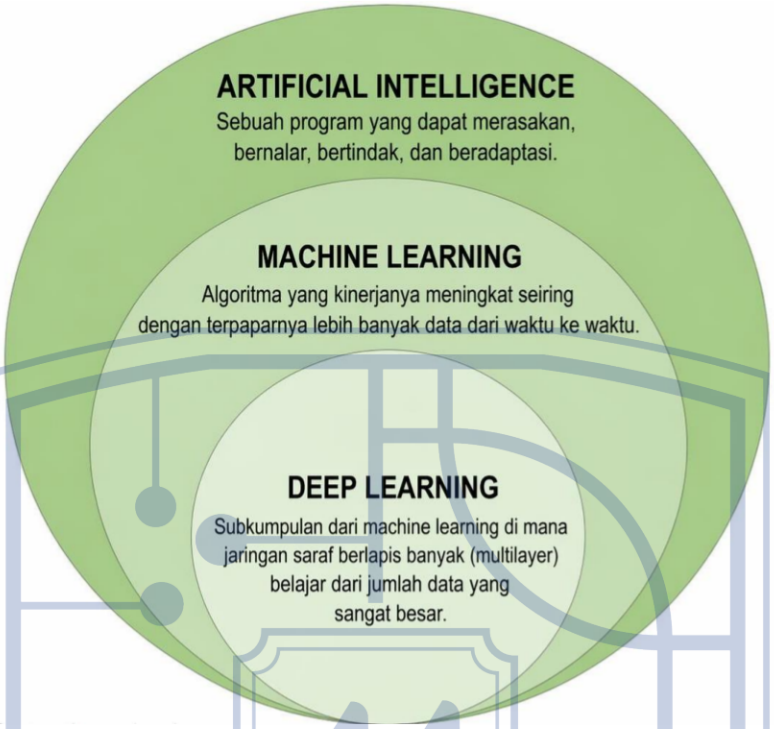
Secara arsitektural, model *deep learning* terdiri atas satu *input layer*, banyak *hidden layer* (lapisan tersembunyi), serta satu *output layer*. Mekanisme kerjanya didasarkan pada proses matematis berikut:

1. *Forward Propagation*: Setiap *neuron* menerima masukan yang dikalikan dengan bobot (*weights*), ditambahkan *bias*, kemudian diproses melalui fungsi aktivasi *non-linear* seperti *ReLU*, *Sigmoid*, *Tanh* dan lainnya untuk menghasilkan keluaran.
2. *Loss Function*: Menghitung selisih antara prediksi model dengan target aktual untuk mengukur tingkat kesalahan.
3. *Backpropagation & Optimization*: Model memperbarui bobot berdasarkan nilai kesalahan menggunakan algoritma optimisasi seperti *Gradient Descent*. Proses ini dilakukan secara berulang *(iterasi)* hingga model mencapai tingkat akurasi yang optimal [50][51].

Struktur berlapis ini memberikan kemampuan bagi DL untuk menangkap hubungan *non-linear* yang sangat kompleks dan berdimensi tinggi, yang sulit dicapai oleh model *shallow network*. Dalam praktiknya, DL telah menunjukkan performa superior pada berbagai domain, termasuk visi komputer dan pengolahan bahasa alami [52]. Dalam konteks pengolahan teks (NLP), *deep learning* menjadi sangat relevan karena kemampuannya memodelkan data sekuensial dan menangkap ketergantungan antar-kata dalam kalimat. Untuk menangani karakteristik data berurutan tersebut, dikembangkan arsitektur khusus

© Karya Dilindungi UU Hak Cipta
1. Dilarang menyebarluaskan dokumen tanpa izin.
2. Dilarang melakukan plagiasi.
3. Pelanggaran diberlakukan sanksi sesuai peraturan UU Hak Cipta.

seperti *Recurrent Neural Network* (RNN).



Gambar 2.3 Hierarki AI [52]

2.6.1 Recurrent Neural Network (RNN)

Recurrent Neural Network (RNN) merupakan arsitektur jaringan saraf tiruan yang secara khusus dirancang untuk memproses data yang memiliki struktur *sekuensial* dan dependensi temporal. Berbeda dengan jaringan saraf umpan maju (*feed forward neural network*) yang memproses setiap *input* secara independen satu sama lain, RNN memiliki mekanisme koneksi berulang yang memungkinkan informasi dari langkah waktu sebelumnya untuk dilanjutkan ke langkah waktu berikutnya melalui suatu kondisi internal yang tersembunyi. Karakteristik inilah yang menjadikan RNN mampu secara teoritis memodelkan ketergantungan jangka panjang dalam data deret waktu, teks, maupun sinyal sekuensial lainnya [40]. Secara matematis, komputasi pada RNN dapat diformulasikan sebagai :

$$h_t = \tanh(W_h \cdot h_{\{t-1\}} + W_x \cdot x_t + b_h) \quad (2.12)$$

Di mana :

h_t = merepresentasikan keadaan tersembunyi pada langkah waktu ke-*t*,

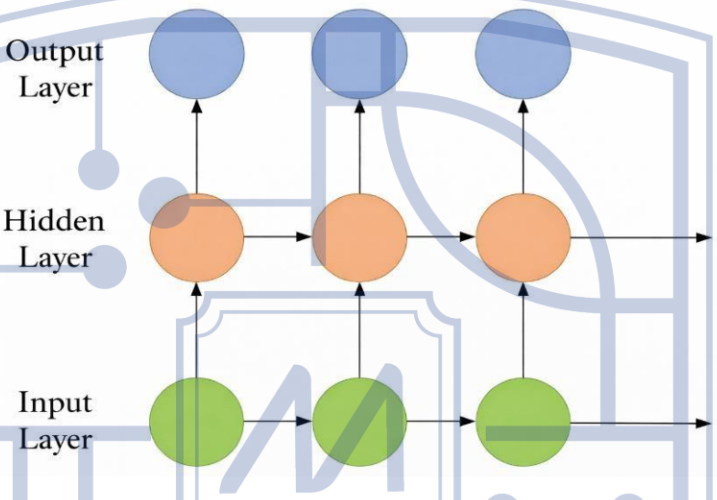
x_t = adalah vektor *input* saat ini

W_h & W_x = merupakan matriks bobot yang dapat dipelajari

© Karya Dilindungi UU Hak Cipta
1. Dilarang menyebarluaskan dokumen tanpa izin.
2. Dilarang melakukan plagiasi.
3. Pelanggaran diberlakukan sanksi sesuai peraturan UU Hak Cipta.

b_h = adalah vektor bias.

Keluaran model pada tiap langkah waktu selanjutnya dihitung sebagai fungsi dari keadaan tersembunyi tersebut. Proses pembaruan keadaan tersembunyi secara rekuren inilah yang secara konseptual membentuk semacam memori kontekstual di dalam jaringan, memungkinkan model untuk mempertimbangkan informasi historis dalam setiap prediksinya [52].



Gambar 2.4 Diagram RNN [53]

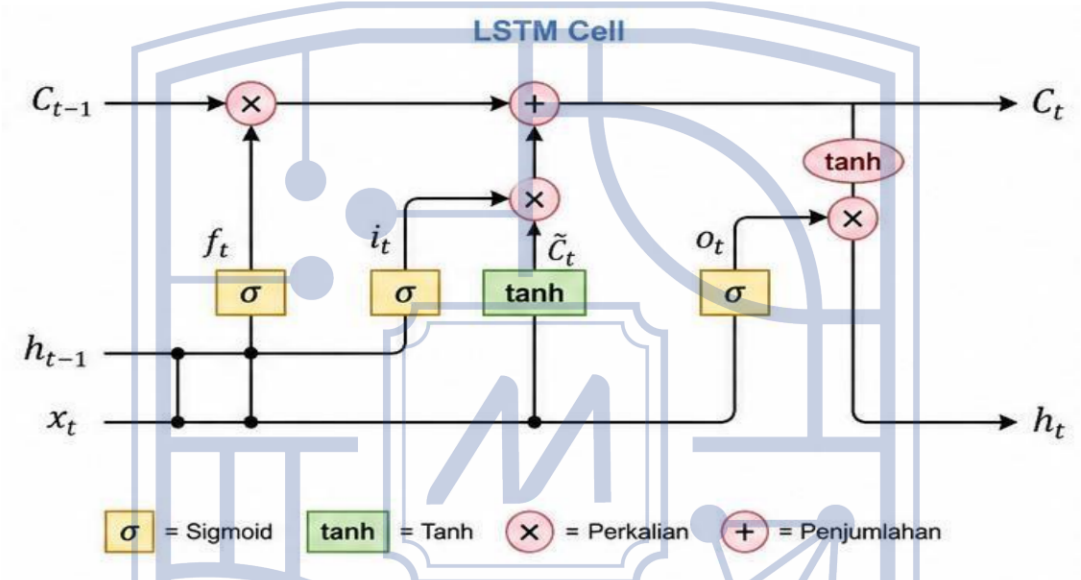
Meskipun secara konseptual RNN mampu menangkap ketergantungan *temporal*, dalam praktik implementasinya arsitektur ini menangani permasalahan mendasar yang dikenal sebagai *vanishing gradient* dan *exploding gradient*. Fenomena *vanishing gradien* terjadi ketika *gradien* yang dipropagasikan mundur melalui banyak langkah waktu menjadi sangat kecil mendekati nol, sehingga bobot pada lapisan awal hampir tidak memperoleh pembaruan yang berarti dan model gagal mempelajari ketergantungan jangka panjang. Sebaliknya, ledakan *gradien* terjadi ketika *gradien* tumbuh secara eksponensial dan mengakibatkan ketidakstabilan numerik selama pelatihan. Berbagai kajian literatur kontemporer menegaskan bahwa keterbatasan bawaan ini menjadi motivasi utama pengembangan arsitektur rekuren yang lebih canggih seperti LSTM dan GRU [54].

2.6.2 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) adalah arsitektur jaringan saraf *recurrent* yang dikembangkan untuk mengatasi kelemahan dasar RNN konvensional, khususnya

© Karya Dilindungi UU Hak Cipta
1. Dilarang menyebarluaskan dokumen tanpa izin.
2. Dilarang melakukan plagiasi.
3. Pelanggaran diberlakukan sanksi sesuai peraturan UU Hak Cipta.

permasalahan *vanishing gradien* yang menghalangi pembelajaran dependensi jangka panjang. LSTM memperkenalkan sebuah *cell state* sebagai komponen memori eksplisit jangka panjang yang aliran informasinya dikendalikan ol ehtiga mekanisme *gating*, yaitu *forget gate, input gate, dan output gate*. Keberadaan *cell state* yang terpisah dari *Hidden state* inilah yang memungkinkan informasi relevan dipertahankan selama ratusan bahkan ribuan langkah waktu tanpa mengalami *Vanishing Gradient* yang signifikan [40] .



Gambar 2.5 Arsitektur LSTM [55]

LSTM merupakan salah satu varian *Recurrent Neural Network (RNN)* yang dirancang untuk mengatasi permasalahan *vanishing gradient* serta mampu menangkap dependensi jangka panjang dalam data sekuensial. LSTM memiliki komponen utama berupa *cell state* (C_t) dan *hidden state* (h_t), serta tiga mekanisme gerbang (*gates*) yaitu *forget gate*, *input gate*, dan *output gate*. Alur Perhitungan LSTM pada setiap waktu ke-t, menerima input berupa:

x_t = input saat ini

h_{t-1} = *hidden state* sebelumnya

C_{t-1} = *cell state* sebelumnya

Kemudian dilakukan proses sebagai berikut:

1. *Forget Gate* (Gerbang Lupa)

   Forget gate berfungsi untuk menentukan informasi mana dari *cell state* sebelumnya yang perlu dipertahankan atau dihapus.

© Karya Dilindungi UU Hak Cipta
1. Dilarang menyebarluaskan dokumen tanpa izin.
2. Dilarang melakukan plagiasi.
3. Pelanggaran diberlakukan sanksi sesuai peraturan UU Hak Cipta.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad (2.13)$$

Di mana:

f_t= nilai forget gate

σ= fungsi aktivasi sigmoid (bernilai antara 0 – 1 (0 = informasi dibuang, 1 = informasi dipertahankan))

W_f = Matriks Bobot (*Weight*) untuk gerbang *forget*.

h_t = *Hidden state* pada waktu t.

x_t = *Input* pada langkah waktu saat ini.

b_f = Vektor Bias untuk gerbang *forget*.

2. *Input Gate* (Gerbang Input)

Input gate menentukan informasi baru yang akan disimpan ke dalam *cell state*. Dua tahap pada input gate:

Menentukan bagian yang diperbarui:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad (2.14)$$

Di mana:

i_t = *Input Gate* (Menentukan informasi baru apa yang akan disimpan ke *cell state*).

σ= fungsi aktivasi sigmoid

W_i = Matriks Bobot (*Weight*) untuk gerbang *input*.

h_t = *Hidden state* pada waktu t.

x_t = *Input* pada langkah waktu saat ini.

b_i = Vektor Bias untuk gerbang *input*.

3. Membuat kandidat nilai baru:

$$C_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \quad (2.15)$$

Di mana:

C_t = *Candidate Cell State* informasi baru yang potensial untuk ditambahkan ke dalam *cell state* pada langkah waktu t.

$tanh$ = Fungsi aktivasi *hyperbolic tangent* yang memetakan output ke dalam rentang [-1, 1] untuk mengatur aliran informasi.

W_c = Matriks Bobot (*Weight*) untuk *cell state*.

h_t = *Hidden state* pada waktu t.

x_t = *Input* pada langkah waktu saat ini.

© Karya Dilindungi UU Hak Cipta
1. Dilarang menyebarluaskan dokumen tanpa izin.
2. Dilarang melakukan plagiasi.
3. Pelanggaran diberlakukan sanksi sesuai peraturan UU Hak Cipta.

b_c = Vektor Bias

4. *Update Cell State* (Memori Utama)

   Cell state diperbarui dengan menggabungkan informasi lama dan informasi baru:

$$C_t = f_t \cdot C_{t-1} + i_t \cdot C_t \quad (2.16)$$

   Di mana:

   C_t = *Cell State* pada waktu t (Memori jangka panjang unit tersebut).

   Bagian lama dikontrol oleh f_t

   Informasi baru dikontrol oleh i_t

5. *Output Gate* (Gerbang Output)

   Output gate menentukan informasi yang akan dikeluarkan sebagai hidden state.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad (2.17)$$

   Di mana:

   o_t = *Output Gate* (Menentukan bagian mana dari *cell state* yang akan menjadi *output*).

   σ = Fungsi Aktivasi Sigmoid (Menghasilkan nilai antara 0 dan 1).

   W_o = Matriks Bobot (*Weight*) untuk *output gate*.

   h_t = *Hidden state* pada waktu t.

   x_t = *Input* pada langkah waktu saat ini.

   b_o = Vektor Bias

6. *Hidden State* (Output Akhir)

   Hidden state diperoleh dari cell state yang telah difilter:

$$h_t = o_t \cdot \tanh(C_t) \quad (2.18)$$

   Di mana:

   h_t = *Hidden state* pada waktu t.

   o_t = *Output Gate* (Menentukan bagian mana dari *cell state* yang akan menjadi *output*).

   b_c = Vektor Bias

   $tanh$ = Fungsi aktivasi *hyperbolic tangent* yang memetakan output ke dalam rentang [-1, 1] untuk mengatur aliran informasi.

   C_t = *Cell State* pada waktu t (Memori jangka panjang unit tersebut).

© Karya Dilindungi UU Hak Cipta
1. Dilarang menyebarluaskan dokumen tanpa izin.
2. Dilarang melakukan plagiasi.
3. Pelanggaran diberlakukan sanksi sesuai peraturan UU Hak Cipta.

LSTM mampu mempertahankan informasi jangka panjang melalui mekanisme *cell state* yang dikontrol oleh tiga gerbang. Struktur ini membuat LSTM efektif dalam memahami konteks yang kompleks dalam data teks. *Forget gate* pada LSTM bertanggung jawab untuk secara selektif menghapus informasi yang tidak lagi relevan dari *cell state*, dengan menghasilkan nilai vektor antara 0 dan 1 untuk setiap elemen *cell state* berdasarkan keadaan tersembunyi sebelumnya dan *input* saat ini. Gerbang masukan kemudian menentukan informasi baru apa yang akan ditambahkan ke dalam keadaan sel, sedangkan gerbang keluaran mengontrol bagian dari keadaan sel yang akan diubah menjadi keadaan tersembunyi dan mengeluarkan jaringan pada langkah waktu tersebut. Interaksi sinergis antara gerbang ketiga ini memberikan LSTM kemampuan adaptif untuk mengatur memorinya secara dinamis sesuai dengan konteks data yang sedang diproses [56].

Dalam ranah prediksi waktu tertentu, LSTM telah menunjukkan kinerja yang luar biasa dalam berbagai domain aplikasi. Penelitian yang dilakukan oleh Liu dkk memperkenalkan model DSTP-RNN berbasis LSTM dengan mekanisme perhatian dua tahap, yang mampu menghasilkan akurasi prediksi unggul pada dataset multivariat dibandingkan model kompetitor [56]. Mengembangkan *autoencoder* LSTM bertumpuk yang dilatih tanpa pengawasan untuk pemodelan deret waktu multivariat, dan menunjukkan bahwa representasi fitur yang dipelajari tanpa pengawasan oleh LSTM mampu secara signifikan meningkatkan kualitas prediksi [38]. Hasil-hasil penelitian tersebut secara konsisten menunjukkan bahwa LSTM merupakan komponen arsitektural yang sangat kuat untuk pemodelan data sekuensial dengan pola temporal yang kompleks dan *non-linear* [56].

Keunggulan LSTM komparatif dibandingkan metode statistik tradisional seperti ARIMA dan *Exponential Smoothing* juga telah dikonfirmasi dalam literatur kontemporer. Tidak seperti metode statistik yang mensyaratkan asumsi *stasioneritas* dan *linearitas*, LSTM mampu secara langsung memodelkan pola *non-stasioner*, musiman ganda, dan hubungan *non-linear* yang kompleks tanpa memerlukan transformasi data yang ekstensif. Fleksibilitas ini menjadikan LSTM sangat relevan untuk aplikasi prediksi pada domain seperti keuangan, energi, cuaca, dan kesehatan yang datanya memiliki waktu yang memiliki karakteristik dinamis dan *heterogen*.

© Karya Dilindungi UU Hak Cipta
1. Dilarang menyebarluaskan dokumen tanpa izin.
2. Dilarang melakukan plagiasi.
3. Pelanggaran diberlakukan sanksi sesuai peraturan UU Hak Cipta.

Inisialisasi Parameter pada LSTM dilakukan sebelum proses pelatihan dimulai, seluruh parameter yang dapat dipelajari pada lapisan LSTM harus diinisialisasi ke nilai awal tertentu. Parameter tersebut meliputi matriks bobot (W), vektor bias (b), dan *hidden state* awal (h_0). Inisialisasi yang tepat sangat berpengaruh terhadap kecepatan konvergensi dan kestabilan proses pelatihan.

1. Matriks Bobot (W)

   Setiap *gate* pada LSTM memiliki dua matriks bobot yang perlu diinisialisasi, yaitu matriks bobot untuk input (Wx) dan matriks bobot untuk *hidden state* sebelumnya (Wh). Secara spesifik, terdapat empat pasang matriks bobot yang bersesuaian dengan *forget gate, input gate, candidate cell state*, dan *output gate*. Metode inisialisasi yang digunakan dalam penelitian ini adalah *Glorot Uniform Initialization* (*Xavier Initialization*), yaitu metode default pada pustaka Keras/TensorFlow untuk lapisan rekuren. Matriks bobot diinisialisasi dengan mengambil sampel secara acak dari distribusi seragam (*uniform distribution*) dalam rentang:

   $$W \sim Uniform(-\sqrt{(6 / (n_{in} + n_{out}))}, +\sqrt{(6 / (n_{in} + n_{out}))}) \quad (2.19)$$

   Di mana:

   n_{in} = jumlah *neuron* pada lapisan sebelumnya (input dimensi).

   n_{out} = jumlah unit/*neuron* pada lapisan LSTM saat ini.

   Pendekatan ini bertujuan untuk menjaga variansi sinyal tetap stabil saat propagasi maju (*forward pass*) maupun mundur (*backpropagation*), sehingga mencegah masalah *vanishing gradient* atau *exploding gradient* sejak awal pelatihan. Selain *Glorot Uniform*, matriks bobot rekuren (W_h) pada LSTM juga dapat diinisialisasi menggunakan *Orthogonal Initialization*, yang menghasilkan matriks *ortogonal* sehingga norma vektor *hidden state* tetap terjaga. Namun, dalam implementasi standar penelitian ini, *Glorot Uniform* digunakan secara konsisten untuk semua *gate*.

2. Vektor Bias (b)

   Vektor bias pada setiap *gate* LSTM diinisialisasi dengan nilai nol (b = 0) sebagai nilai default. Hal ini merupakan praktik umum karena bobot jaringan (W) sudah mengandung variasi acak dari proses inisialisasi Glorot. Satu-satunya pengecualian yang umum diterapkan adalah pada *forget gate*:

   $b_f = 1 \quad (Forget\ Gate\ Bias)$ ..... (2.20)

© Karya Dilindungi UU Hak Cipta
1. Dilarang menyebarluaskan dokumen tanpa izin.
2. Dilarang melakukan plagiasi.
3. Pelanggaran diberlakukan sanksi sesuai peraturan UU Hak Cipta.

Inisialisasi $b_f = 1$ dilakukan karena pada awal pelatihan, model sebaiknya mempertahankan sebagian besar informasi dari *cell state* sebelumnya ($C_t - 1$). Dengan $b_f = 1$, *forget gate* menghasilkan nilai mendekati $\sigma(1) \approx 0{,}73$, artinya sekitar 73% informasi masa lalu dipertahankan. Ini mencegah model lupa terlalu cepat di awal pelatihan sebelum pola temporal sempat dipelajari. Rangkuman inisialisasi bias untuk setiap *gate* LSTM adalah sebagai berikut:

Tabel 2.1 Inisialisasi bias untuk setiap *gate*

Gate	**Inisialisasi Bias**	**Alasan**
Forget Gate (b_f)	$b_f = 1$	Agar model mempertahankan memori di awal pelatihan
Input Gate (b_i)	$b_i = 0$	Tidak ada informasi baru yang diprioritas di awal
Candidate Cell (b_c)	$b_c = 0$	Nilai kandidat diinisialisasi netral
Output Gate (b_o)	$b_o = 0$	Output tidak diaktivasi secara khusus di awal

3. *Hidden State* Awal (h_o) dan *Cell State* Awal (C_o)

   Pada awal sekuens data (*timestep* $t = 0$), jaringan LSTM membutuhkan nilai awal untuk *hidden state* (h_o) dan *cell state* (C_o). Kedua nilai ini diinisialisasi sebagai vektor nol:

   $h_0 = 0$ *(vektor nol berukuran n_units × 1)*

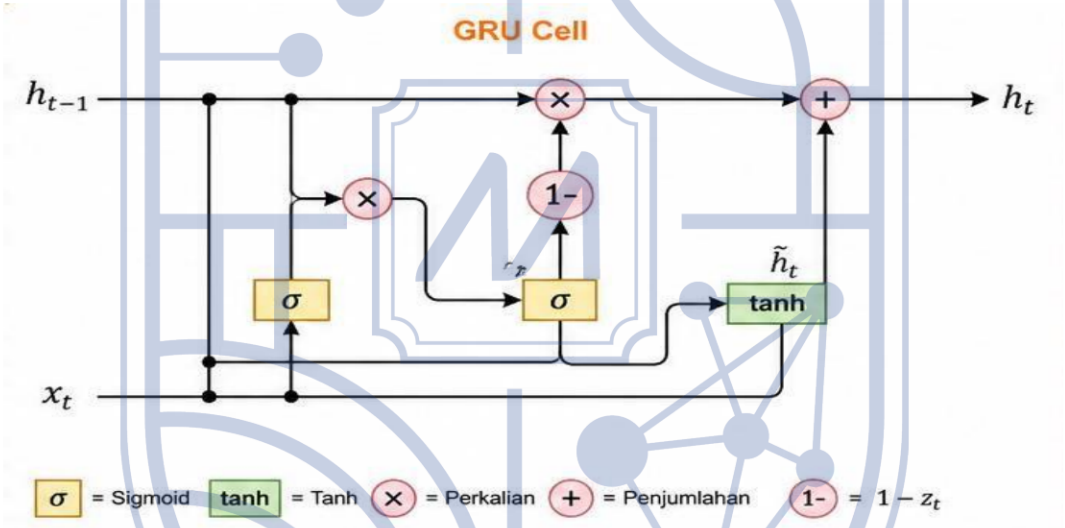
   $C_0 = 0$ *(vektor nol berukuran n_units × 1)*

   Di mana *n_units* adalah jumlah unit LSTM yang dikonfigurasi (dalam penelitian ini, 64 unit untuk lapisan LSTM). Inisialisasi nol ini mengandung asumsi bahwa pada awal setiap urutan (*sequence*), model belum memiliki konteks atau informasi historis yang relevan. Keras/TensorFlow secara otomatis melakukan inisialisasi ini pada setiap awal batch selama pelatihan.

© Karya Dilindungi UU Hak Cipta
1. Dilarang menyebarluaskan dokumen tanpa izin.
2. Dilarang melakukan plagiasi.
3. Pelanggaran diberlakukan sanksi sesuai peraturan UU Hak Cipta.

2.6.3 Gated Recurrent Unit (GRU)

Gated Recurrent Unit (GRU) adalah arsitektur jaringan saraf *recurrent* yang dikembangkan sebagai alternatif yang lebih efisien secara komputasional terhadap LSTM. GRU mengaktifkan mekanisme *gating* LSTM dengan menggabungkan *forget gate* dan *input gate* menjadi satu komponen yang disebut *Update Gate*, serta menghilangkan komunikasi antara *cell state* dan *Hidden state* yang ada pada LSTM. Selain *Update Gate*, GRU juga memiliki *reset gate* yang mengontrol seberapa besar pengaruh *Hidden state* sebelumnya terhadap perhitungan kandidat *Hidden state* yang baru. Penyederhanaan struktural ini menghasilkan arsitektur dengan jumlah parameter yang lebih sedikit tanpa mengorbankan kemampuan modelnya secara substansial [57].



Gambar 2.6 Arsitektur GRU [57]

Update Gate pada GRU menentukan proporsi informasi dari keadaan tersembunyi sebelumnya yang akan dipertahankan atau digantikan oleh kandidat keadaan tersembunyi baru yang dihitung berdasarkan *input* saat ini. Ketika gerbang pembaruan mendekati nilai 1, model mempertahankan hampir seluruh informasi dari langkah waktu sebelumnya sehingga mencerminkan memori jangka panjang, sedangkan ketika mendekati 0 model sepenuhnya menggantikan representasi lamanya dengan informasi baru. *Reset gate* bekerja secara komplementer dengan menentukan relevansi keadaan tersembunyi masa lalu dalam menghitung kandidat representasi konteks saat ini [58].

Efisiensi komputasi pada GRU bukan berarti LSTM tidak layak digunakan, melainkan menunjukkan bahwa GRU mampu mencapai performa yang sebanding dengan

© Karya Dilindungi UU Hak Cipta
1. Dilarang menyebarluaskan dokumen tanpa izin.
2. Dilarang melakukan plagiasi.
3. Pelanggaran diberlakukan sanksi sesuai peraturan UU Hak Cipta.

jumlah parameter yang lebih sedikit. Pada dataset berukuran besar, pengurangan jumlah parameter tersebut berpotensi mempercepat proses pelatihan, mengurangi penggunaan memori, serta menurunkan kompleksitas komputasi. Oleh karena itu, penelitian ini tidak menggunakan GRU sebagai pengganti LSTM, tetapi mengombinasikan keduanya agar kemampuan LSTM dalam menangkap dependensi jangka panjang tetap dipertahankan, sementara GRU membantu menghasilkan representasi fitur secara lebih efisien.

Alur Perhitungan GRU Cell dapat dilihat pada prose dibawah.

Input yang digunakan:

x_t = input saat ini

h_{t-1} = *hidden state* sebelumnya

1. *Update Gate* (Gerbang Update)

   Update gate menentukan seberapa besar informasi lama dipertahankan.

   $$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z) \quad (2.21)$$

   Di mana:

   z_t = *Update Gate* (Menentukan seberapa banyak informasi masa lalu yang harus diteruskan ke masa depan).

   h_t = *Final Hidden state* (*Output* akhir unit GRU pada waktu t yang menggabungkan informasi lama dan kandidat baru).

   x_t = *Input* pada langkah waktu saat ini.

   σ = Fungsi Aktivasi *Sigmoid* (Menghasilkan *output* dalam rentang 0 - 1).

   W_z = Matriks Bobot (*Weights*) yang dipelajari model selama pelatihan.

   b_z = Vektor Bias yang dipelajari model.

2. *Reset Gate* (Gerbang Reset)

   Reset gate menentukan seberapa besar informasi lama dilupakan.

   $$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r) \quad (2.22)$$

   Di mana:

   r_t = *Reset gate* (Menentukan seberapa banyak informasi masa lalu yang harus dilupakan).

   h_t = *Final Hidden state* (*Output* akhir unit GRU pada waktu t yang menggabungkan informasi lama dan kandidat baru).

   x_t = *Input* pada langkah waktu saat ini.

   σ = Fungsi Aktivasi *Sigmoid* (Menghasilkan *output* dalam rentang 0 hingga 1).

© Karya Dilindungi UU Hak Cipta
1. Dilarang menyebarluaskan dokumen tanpa izin.
2. Dilarang melakukan plagiasi.
3. Pelanggaran diberlakukan sanksi sesuai peraturan UU Hak Cipta.

W_r = Matriks Bobot (*Weights*) yang dipelajari model selama pelatihan.

b_r = Vektor Bias yang dipelajari model.

3. Kandidat *Hidden State*

Menghitung kandidat nilai *hidden state* baru:

$$\tilde{h}_t = \tanh(W_h \cdot [r_t \cdot h_{t-1}, x_t] + b_h \quad (2.23)$$

Di mana:

r_t = mengontrol seberapa besar informasi masa lalu digunakan.

$\tilde{h}_t$ = *Candidate Hidden state* (Kandidat nilai baru yang dihitung dari *input* saat ini dan pengaruh *reset gate* terhadap informasi masa lalu).

x_t = *Input* pada langkah waktu saat ini.

$tanh$ = Fungsi Aktivasi *Tanh* (Menghasilkan *Output* dalam rentang -1 - 1).

W_h = Matriks Bobot (*Weights*) yang dipelajari model selama pelatihan.

b_h = Vektor Bias yang dipelajari model.

4. *Hidden State* Akhir

Hidden state diperbarui dengan menggabungkan informasi lama dan baru:

$$h_t = (1 - z_t) \cdot h_{t-1} + z_t \cdot \tilde{h}_t \quad (2.24)$$

Di mana:

Jika $z_t$besar maka lebih banyak informasi baru.

Jika $z_t$kecil maka lebih banyak mempertahankan informasi lama.

GRU memiliki struktur yang lebih sederhana dibandingkan LSTM karena menggabungkan fungsi *forget* dan *input gate* menjadi satu *update gate*. Hal ini membuat GRU lebih efisien secara komputasi, namun tetap mampu menangkap dependensi dalam data sekuensial. Berbagai penelitian kontemporer telah mengkaji kinerja GRU dalam konteks prediksi waktu dan mengkonfirmasinya sebagai alternatif yang kompetitif terhadap LSTM. Kim dan Sohn melakukan evaluasi komprehensif terhadap GRU untuk prediksi deret waktu finansial dan menemukan bahwa GRU menghasilkan akurasi yang setara dengan LSTM namun dengan waktu komputasi yang lebih singkat, menjadikannya pilihan yang lebih praktis dalam skenario di mana efisiensi komputasi menjadi pertimbangan penting [57]. Pengtao Jia dkk juga memvalidasi efektivitas GRU dalam konteks prediksi beban jangka pendek, menunjukkan bahwa kemampuan adaptif GRU dalam mengelola informasi temporal mampu menghasilkan kesalahan prediksi yang rendah secara konsisten pada berbagai kondisi beban [58].

© Karya Dilindungi UU Hak Cipta
1. Dilarang menyebarluaskan dokumen tanpa izin.
2. Dilarang melakukan plagiasi.
3. Pelanggaran diberlakukan sanksi sesuai peraturan UU Hak Cipta.

Satu keunggulan praktis GRU yang sering dikutip dalam literatur adalah kemampuannya untuk memberikan kinerja yang baik bahkan pada dataset dengan ukuran yang relatif terbatas. Dengan jumlah parameter yang lebih sedikit, GRU memiliki risiko *overfitting* yang lebih rendah pada kondisi data pelatihan yang terbatas, sehingga menghasilkan generalisasi yang lebih baik pada pengujian data. Karakteristik ini menjadikan GRU pilihan yang sangat relevan dalam skenario penelitian di mana ketersediaan data historis terbatas atau sumber daya komputasi tidak memadai untuk pelatihan model LSTM yang lebih kompleks [57].

Inisialisasi parameter pada GRU sama seperti LSTM, parameter GRU juga memerlukan inisialisasi sebelum pelatihan dimulai. GRU memiliki jumlah parameter yang lebih sedikit karena menggabungkan *forget gate* dan *input gate* menjadi *update gate*, serta menghapus *cell state* terpisah. Hal ini membuat proses inisialisasi GRU relatif lebih sederhana.

1. Matriks Bobot (W) pada GRU

   Setiap *gate* GRU (*update gate, reset gate*, dan *candidate hidden state*) memiliki dua matriks bobot: matriks bobot untuk input (W_x) dan matriks bobot untuk *hidden state* sebelumnya (W_h). Total terdapat tiga pasang matriks bobot yang diinisialisasi menggunakan *Glorot Uniform Initialization*, sama seperti pada LSTM:

   $$Wz, Wr, W \sim Glorot\ Uniform\left(-\sqrt{\frac{6}{n_{in}+n_{out}}}, +\sqrt{\frac{6}{n_{in}+n_{out}}}\right) \quad (2.25)$$

   Di mana W_z adalah matriks bobot *update gate*, W_r adalah matriks bobot *reset gate*, dan W adalah matriks bobot *candidate hidden state*.

2. Vektor Bias (b) pada GRU

   Seluruh vektor bias pada GRU diinisialisasi dengan nilai nol (b = 0). Berbeda dengan LSTM yang merekomendasikan $b_f = 1$ untuk forget gate, pada GRU hal ini tidak diperlukan karena mekanisme *update gate* sudah dirancang untuk secara adaptif mempertahankan informasi historis melalui persamaan:

   $$h_t = (1 - z_t) \odot h_t - 1 + z_t \odot \tilde{h}_t \quad (2.26)$$

   Ketika z_t mendekati 0, hidden state lama ($h_t - 1$) dipertahankan secara penuh tanpa memerlukan bias khusus. Oleh karena itu, semua bias GRU diinisialisasi dengan $b_z = b_r = b = 0$.

© Karya Dilindungi UU Hak Cipta
1. Dilarang menyebarluaskan dokumen tanpa izin.
2. Dilarang melakukan plagiasi.
3. Pelanggaran diberlakukan sanksi sesuai peraturan UU Hak Cipta.

3. *Hidden State* Awal (h_o) pada GRU

   GRU hanya memiliki satu state tersembunyi (*hidden state*) dan tidak memiliki *cell state* terpisah seperti LSTM. Nilai awal *hidden state* diinisialisasi sebagai vektor nol:

   $h_o = 0 \ (vektor\ nol\ berukuran\ n_units \times 1)$ .....(2.27)

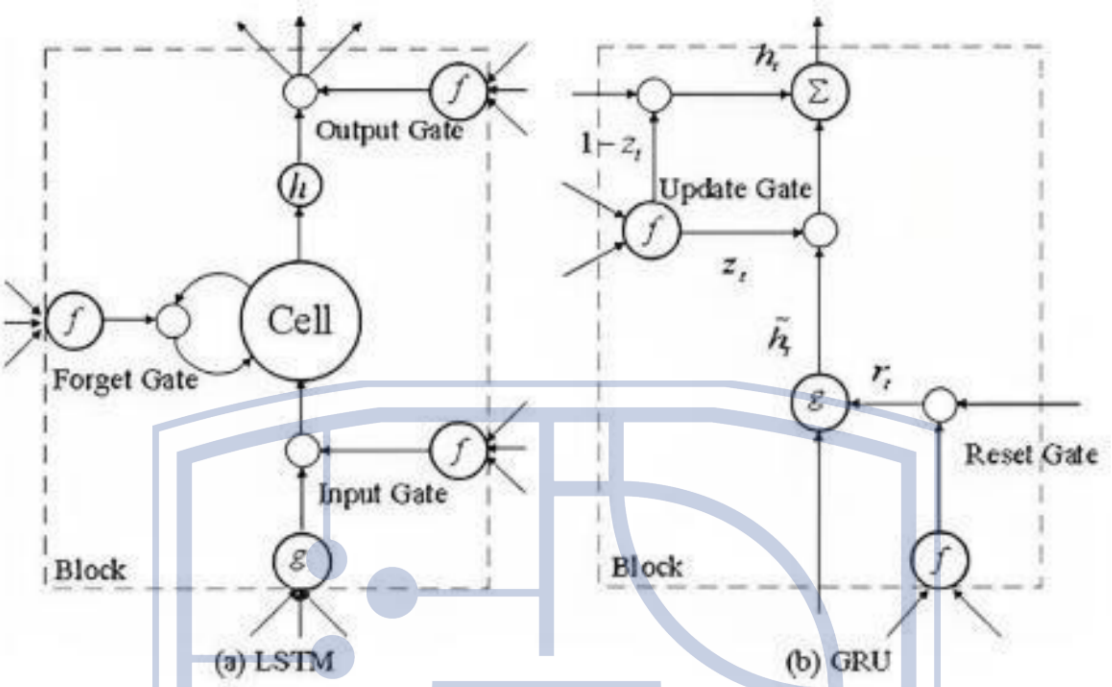
   Di mana $n_{units} = 32$ unit GRU sesuai konfigurasi arsitektur model dalam penelitian ini. Inisialisasi $h_o = 0$ mengandung asumsi bahwa tidak ada informasi historis yang tersedia sebelum timestep pertama dari setiap sekuens input.

2.6.4 Arsitektur *Hybrid* LSTM dan GRU

Pengembangan model yang mengintegrasikan arsitektur LSTM dan GRU dalam satu kerangka jaringan tunggal telah menjadi fokus penelitian yang semakin intensif dalam beberapa tahun terakhir. Motivasi mendasar di balik pendekatan ini adalah untuk memanfaatkan komplementaritas antara kedua arsitektur: kemampuan LSTM dalam mempertahankan dan mengelola memori jangka panjang melalui mekanisme *cell state* eksplisitnya, serta efisiensi komputasi dan kemampuan adaptasi jangka pendek GRU yang berasal dari struktur *gating*-nya yang lebih ringkas. Dengan mengintegrasikan kedua kekuatan tersebut, model diharapkan mampu menangkap dinamika data temporal pada berbagai skala secara *paralel* [59].

Dalam implementasinya, arsitektur LSTM dan GRU umumnya diterapkan dalam konfigurasi bertumpuk (*Hybrid*), di mana lapisan LSTM diposisikan pada lapisan awal jaringan untuk mengekstraksi representasi fitur temporal jangka panjang yang mendalam dari *input* data. Representasi yang dihasilkan oleh lapisan LSTM diteruskan sebagai *input* ke lapisan GRU, yang bertugas melakukan penajaman (*refinement*) terhadap representasi tersebut. Fokus utama pada tahap ini adalah menangkap dinamika jangka pendek serta pola transisi lokal yang relevan untuk proses prediksi. Konfigurasi hirarkis ini memungkinkan model membangun pemahaman *multi-level* terhadap struktur data temporal secara lebih mendalam [59][60]. Contoh dari arsitektur LSTM yang ditumpuk dengan GRU dapat dilihat pada gambar di bawah.

© Karya Dilindungi UU Hak Cipta
1. Dilarang menyebarluaskan dokumen tanpa izin.
2. Dilarang melakukan plagiasi.
3. Pelanggaran diberlakukan sanksi sesuai peraturan UU Hak Cipta.



Gambar 2.7 Arsitektur Hybrid LSTM dan GRU [55]

Efektivitas arsitektur *Hybrid* LSTM dan GRU telah teruji secara *empiris* melalui berbagai studi lintas domain. Amirfarhad dkk melakukan evaluasi pada prediksi beban listrik jangka pendek dan menemukan bahwa model LSTM dan GRU yang digabung secara konsisten menghasilkan nilai RMSE dan MAE yang lebih rendah dibandingkan model LSTM tunggal maupun GRU tunggal, mengkonfirmasi adanya efek sinergis dari penumpukan kedua arsitektur [61]. Aakash dkk Melaporkan temuan serupa pada domain prediksi harga emas, di mana arsitektur *hybrid* yang mengintegrasikan lapisan konvolusi, LSTM, dan GRU menghasilkan peningkatan akurasi prediksi yang signifikan secara statistik [60]. Penelitian terbaru yang dilakukan oleh H. Farhadi dkk lebih lanjut mengeksplorasi penerapan model *hybrid* LSTM dan GRU untuk prediksi sisa umur pakai komponen industri, dan hasilnya menunjukkan keunggulan arsitektur *hybrid* dalam menangkap baik tren degradasi jangka panjang maupun jangka pendek yang karakteristik dari kondisi data mesin [61]. Temuan-temuan ini secara kolektif mempertegas posisi arsitektur *hybrid* LSTM dan GRU sebagai salah satu pendekatan paling menjanjikan dalam pemodelan prediktif deret waktu yang memiliki karakteristik multi-skala dan kompleksitas temporal yang tinggi [61].

© Karya Dilindungi UU Hak Cipta
1. Dilarang menyebarluaskan dokumen tanpa izin.
2. Dilarang melakukan plagiasi.
3. Pelanggaran diberlakukan sanksi sesuai peraturan UU Hak Cipta.

2.7 Teknik Optimasi dan Pelatihan Model

Proses pelatihan model *Deep learning* memerlukan pengaturan parameter dan *teknik optimasi* yang tepat untuk memastikan model dapat mencapai titik *konvergensi* secara efisien. Penggunaan arsitektur hybrid seperti LSTM dan GRU pada dataset yang sangat besar (lebih dari 200.000 baris data) menuntut kontrol yang ketat terhadap proses pembelajaran agar model tidak hanya menghafal data (*overfitting*) tetapi juga mampu melakukan generalisasi terhadap data baru. Oleh karena itu, diperlukan serangkaian mekanisme mulai dari penghentian dini (*early stopping*), hingga pemilihan fungsi kerugian (*loss function*) yang sesuai dengan karakteristik data sentimen yang bersifat multikelas.

2.7.1 Early Stopping

Early stopping merupakan salah satu teknik *regularisasi* yang digunakan dalam proses pelatihan *model deep learning* untuk mencegah terjadinya *overfitting* [62][63]. Teknik ini bekerja dengan cara memantau performa *model* terhadap data validasi selama proses *iterasi epoch* berlangsung [63]. Secara konseptual, *early stopping* akan menghentikan proses pelatihan secara otomatis apabila performa pada set validasi tersebut mulai menurun atau terdegradasi [63]. Penurunan performa ini sering ditandai dengan nilai *loss* yang tidak lagi mengalami penurunan yang signifikan atau justru mulai meningkat setelah sejumlah iterasi tertentu yang dikenal sebagai parameter *patience*[62] Dalam implementasinya, penghentian pelatihan didasarkan pada pencarian nilai kesalahan minimum pada data validasi, yang dirumuskan sebagai berikut:

$$E_{\{opt\}} = \arg\backslash min_t(L_{\{val\}(t)}) \quad (2.28)$$

Di mana:

$L_{\{val\}(t)}$ = adalah nilai *loss* pada data validasi pada *epoch* ke-*t*.

$E_{\{opt\}}$ = adalah nilai *loss* minimum yang diperoleh selama proses pelatihan.

t = adalah indeks *epoch*.

© Karya Dilindungi UU Hak Cipta
1. Dilarang menyebarluaskan dokumen tanpa izin.
2. Dilarang melakukan plagiasi.
3. Pelanggaran diberlakukan sanksi sesuai peraturan UU Hak Cipta.

2.7.2 Optimizer

Optimizer adalah algoritma yang digunakan untuk mengatur pembaruan bobot dan *bias model* dengan tujuan meminimalkan fungsi *loss* selama proses pelatihan [64]. Pemilihan *optimizer* yang tepat sangat berpengaruh terhadap kecepatan *konvergensi* dan kualitas akhir *model deep learning* [64]. Beberapa jenis *optimizer* yang umum digunakan dalam *deep learning* adalah sebagai berikut:

a. *Stochastic Gradient Descent* (SGD) adalah *optimizer* yang bersifat sederhana dan ringan secara komputasi, namun sering kali membutuhkan iterasi lebih banyak untuk mencapai konvergensi [64]. SGD melakukan pembaruan bobot berdasarkan *gradien* saat ini tanpa mempertimbangkan informasi dari iterasi sebelumnya, sehingga rentan terhadap osilasi pada proses pelatihan [64].

$$m_t = \beta_1 m_{\{t-1\}} + (1 - \beta_1) g_t \quad (2.29)$$

Di mana:

m_t = *Momentum* atau estimasi arah *gradien* pada langkah *t*.

β_1 = Koefisien peluruhan (*exponential decay*), biasanya bernilai 0.9.

g_t = *Gradien* dari fungsi objektif terhadap parameter.

b. *Root Mean Square Propagation* (RMSProp) adalah *optimizer* yang menyesuaikan laju pembelajaran secara adaptif berdasarkan rata-rata kuadrat *gradien* terbaru [64]. RMSProp menawarkan keunggulan dalam kecepatan konvergensi dibandingkan SGD melalui pendekatan adaptif dalam menangani *gradien* selama proses pelatihanRata-rata bergerak *gradien* kuadrat:

$$v_t = \beta_2 v_{\{t-1\}} + (1 - \beta_2) g_t^2 \quad (2.30)$$

Di mana:

v = Moving average dari kuadrat *gradien* pada langkah waktu t.

β_2 = Koefisien peluruhan untuk momen kedua.

g_t^2 = Kuadrat dari *gradien* pada langkah waktu t.

Parameter update:

$$\theta_t = \theta_{\{t-1\}} - \frac{\alpha \cdot g_t}{\sqrt{\{v_t\}} + \epsilon} \quad (2.31)$$

© Karya Dilindungi UU Hak Cipta
1. Dilarang menyebarluaskan dokumen tanpa izin.
2. Dilarang melakukan plagiasi.
3. Pelanggaran diberlakukan sanksi sesuai peraturan UU Hak Cipta.

Di mana:

θ_t = Parameter atau bobot model yang baru.

α = *Learning rate*

g_t = *Gradien* pada waktu langkah t.

v_t = Nilai *moving average* kuadrat *gradien* yang digunakan untuk penskalaan.

ϵ = Konstanta kecil untuk menjaga stabilitas numerik agar tidak terjadi pembagian dengan nol.

c. *Adaptive Moment Estimation* (Adam) adalah salah satu *algoritma optimasi* yang digunakan secara luas dalam pelatihan *model deep learning* [64]. *Adam* menggabungkan keunggulan dari *adaptive gradient* (AdaGrad) dan RMSProp dengan mengatur secara dinamis laju *eksponensial* untuk estimasi momen pertama dan kedua dari *gradien* [64]. Selain itu, *Adam* juga menjadi *optimizer default* yang paling banyak direkomendasikan pada *model* berbasis teks seperti LSTM dan GRU karena kemampuannya dalam menangani data sekuensial dan *konvergensi* yang lebih stabil [65]. *Optimizer Adam* terbukti mencapai akurasi lebih tinggi dibandingkan SGD dan RMSProp pada berbagai tugas klasifikasi citra [66].

a. Estimasi Momen :

$$m_t = \beta_1 m_{\{t-1\}} + (1 - \beta_1) g_t \quad (2.32)$$

Di mana:

m_t = Estimasi momen pertama (rata-rata *gradien*).

g_t = Estimasi momen kedua (*off-center variance* atau rata-rata kuadrat *gradient*)

β_1 = Koefisien peluruhan eksponensial untuk estimasi momen.

b. Koreksi Bias (*Bias-Correction*):

$$\hat{m_t} = \frac{m_t}{1 - \beta_1^t} \quad (2.33)$$

Di mana:

$\hat{m_t}$ = Estimasi momen pertama yang telah dikoreksi biasnya.

m_t = Estimasi momen pertama (rata-rata *gradien*t)

β_1^t = Nilai koefisien peluruhan pertama pangkat t (iterasi saat ini).

© Karya Dilindungi UU Hak Cipta
1. Dilarang menyebarluaskan dokumen tanpa izin.
2. Dilarang melakukan plagiasi.
3. Pelanggaran diberlakukan sanksi sesuai peraturan UU Hak Cipta.

c. Pembaruan Parameter (Parameter Update):

$$\theta_t = \theta_{\{t-1\}} - \frac{\alpha \cdot \hat{m_t}}{\sqrt{\hat{v_t}} + \epsilon} \quad (2.34)$$

Di mana :

θ_t = Parameter atau bobot model yang baru diperbarui.

α = *Learning rate*.

ϵ = Konstanta kecil untuk menjaga stabilitas numerik agar tidak terjadi pembagian dengan nol.

2.7.3 Loss Function

Loss function adalah fungsi yang digunakan untuk mengukur seberapa besar kesalahan prediksi *model* terhadap nilai sebenarnya selama proses pelatihan [61][64]. Nilai *loss* yang semakin kecil mengindikasikan bahwa model semakin baik dalam mempelajari pola dari data latih [66]. *Loss function* atau fungsi kerugian adalah *algoritma fundamental* dalam *machine learning* yang menghitung kuantitas selisih atau kesalahan (*error*) antara nilai *aktual* (sebenarnya) dengan nilai hasil *prediksi model*. Selama fase pelatihan (*training loop*), jaringan model selalu berupaya meminimalkan angka kerugian ini. Pada kasus klasifikasi sentimen berbasis arsitektur *deep learning* yang menggunakan antarmuka *Keras/TensorFlow*, pemilihan fungsi kerugian terpusat pada himpunan *Probabilistic Losses* (kerugian probabilistik). Pendekatan *loss* bervariasi bergantung pada struktur data target (*label*) dari sentimen:

1. *Binary Crossentropy*: Digunakan secara ekstensif saat pengklasifikasian target bersifat *biner* (seperti sentimen Positif vs Negatif; 1 dan 0). Fungsi ini mengevaluasi seberapa jauh probabilitas yang dihasilkan *model* bergeser dari kelas aktual yang diharapkan.
   Rumus Matematis:

$$L_{BCE} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \, log(\hat{y}_i) + (1 - y_i) \, log(1 - \hat{y}_i)] \quad (2.35)$$

Di mana:

L_{BCE} = Nilai total *loss* atau kerugian rata-rata.

N = Jumlah total sampel data atau observasi.

y_i = Nilai label sebenarnya (*ground truth*) untuk sampel ke-i. Nilainya 0 atau 1.

$\hat{y}_i$ = Nilai prediksi (probabilitas) hasil keluaran model untuk sampel ke-i. Nilainya berada di rentang 0 hingga 1.

log = Fungsi logaritma (biasanya menggunakan logaritma natural atau basis e).

© Karya Dilindungi UU Hak Cipta
1. Dilarang menyebarluaskan dokumen tanpa izin.
2. Dilarang melakukan plagiasi.
3. Pelanggaran diberlakukan sanksi sesuai peraturan UU Hak Cipta.

$\sum_{i=1}^{N}$ = Simbol penjumlahan dari data pertama sampai data ke-N.

2. *Categorical Crossentropy*: Diimplementasikan apabila penelitian melibatkan klasifikasi sentimen *multikelas* (contoh: Positif, Netral, Negatif) di mana *label* data telah di-*encode* dengan format *one-hot representation*.

   Rumus Matematis:

   $$L_{CCE} = -\sum_{c=1}^{C} y_c \log(\hat{y}_c) \quad (2.36)$$

   Di mana:

   L_{CCE} = Nilai *loss* atau kerugian untuk satu sampel data dalam klasifikasi *multi-kelas*.

   C = Jumlah total kelas atau kategori yang ada.

   y_c = Indikator *biner* (0 atau 1) yang menyatakan apakah kelas c adalah *label* yang benar untuk sampel tersebut (*one-hot encoded*).

   $\hat{y}_c$ = Probabilitas prediksi *model* bahwa sampel tersebut termasuk dalam kelas c (biasanya hasil dari fungsi *Softmax*).

   log Fungsi logaritma (natural). $sum_{\{c=1\}}^{\{C\}}$ = Simbol penjumlahan untuk seluruh total kelas yang tersedia.

3. *Sparse Categorical Crossentropy*: Memiliki cara perhitungan matematis yang sama dengan *Categorical Crossentropy*, namun dikhususkan untuk struktur data kelas jamak yang mana format labelnya adalah integer berurutan, bukan *array one-hot*.

   Rumus Matematis:

   $$L_{SCCE} = -\frac{1}{N}\sum_{i=1}^{N} \log(\hat{y}_{i,y_i}) \quad (2.37)$$

   Di mana:

   $L_{\{SCCE\}}$ = Nilai total *loss* atau kerugian rata-rata untuk klasifikasi *multi-kelas* dengan format *label sparse*.

   N = Jumlah total sampel data atau observasi dalam satu *batch*.

   y_i = Label sebenarnya (*ground truth*) untuk sampel ke-i, yang direpresentasikan dalam bentuk integer/angka indeks (misal: 0, 1, atau 2), bukan *one-hot encoding*.

   $\hat{y}_{i,y_i}$ = Probabilitas prediksi *model* untuk sampel ke-i pada indeks kelas yang benar (y_i) = Rumus ini secara cerdas hanya mengambil nilai probabilitas dari kelas yang sesuai dengan *label* target.

   log = Fungsi logaritma (natural).

   $sum_{\{i=1\}}^{\{N\}}$ = Simbol penjumlahan dari sampel data pertama hingga ke-N.

© Karya Dilindungi UU Hak Cipta
1. Dilarang menyebarluaskan dokumen tanpa izin.
2. Dilarang melakukan plagiasi.
3. Pelanggaran diberlakukan sanksi sesuai peraturan UU Hak Cipta.

Parameter argumen reduksi (seperti *sum_over_batch_size*, *sum*, atau *none*) di dalam instansiasi kelas *loss function* secara langsung menangani cara *framework* merata-ratakan nilai kesalahan untuk setiap himpunan (*batch size*) sampel selama komputasi. Nilai *loss* ini ditelusuri dengan sangat cermat; penurunan *grafik loss* yang selaras dengan peningkatan *grafik akurasi* (*Accuracy vs Loss Graph*) membuktikan bahwa *model* mempelajari pola generalisasi yang baik dan menghindari *overfitting* [67][68].

2.7.4 Dropout dan SpatialDropout

Dropout merupakan teknik regularisasi pada jaringan saraf tiruan yang bekerja dengan cara menonaktifkan unit atau neuron secara acak selama proses pelatihan dengan probabilitas tertentu p [69]. Tujuannya adalah untuk mencegah *overfitting* dengan memaksa jaringan agar tidak terlalu bergantung pada neuron spesifik tertentu, sehingga mendorong model untuk mempelajari representasi fitur yang lebih kuat dan general [69]. Sementara itu, SpatialDropout adalah varian yang dirancang khusus untuk data dengan spasial, seperti pada *Convolutional Neural Networks* (CNN). Berbeda dengan *Dropout* standar yang mematikan neuron secara mandiri di setiap posisi, *SpatialDropout* mematikan seluruh *feature map* (saluran) secara sekaligus. Hal ini sangat efektif karena pada data citra, piksel yang berdekatan cenderung memiliki korelasi yang sangat kuat; mematikan piksel secara acak (standar *Dropout*) tidak akan memberikan efek regularisasi yang maksimal karena informasi masih bisa "bocor" dari piksel tetangganya.

Selama fase pelatihan, penggunaan *Dropout* akan menyebabkan nilai aktivasi pada lapisan tersebut berkurang karena sebagian neuron dinonaktifkan. Untuk menjaga agar ekspektasi nilai aktivasi tetap konsisten antara fase pelatihan dan fase pengujian (di mana semua neuron aktif), diterapkan mekanisme scaling. Rumus yang digunakan untuk penyesuaian ini adalah:

$$Scale = \frac{1}{1-p} \quad (2.38)$$

© Karya Dilindungi UU Hak Cipta
1. Dilarang menyebarluaskan dokumen tanpa izin.
2. Dilarang melakukan plagiasi.
3. Pelanggaran diberlakukan sanksi sesuai peraturan UU Hak Cipta.

Di mana p adalah probabilitas *dropout* (rasio neuron yang dimatikan). Dengan mengalikan output yang tersisa dengan faktor skala ini selama pelatihan (*inverted dropout*), total jumlah energi atau magnitudo sinyal yang diteruskan ke lapisan berikutnya tetap stabil, sehingga model tidak perlu melakukan penyesuaian ulang saat berpindah ke mode inferensi. Ini adalah perilaku default Keras/TensorFlow *scaling* dilakukan saat *training* agar saat *inference* (prediksi), nilai tidak perlu diubah lagi. Jadi jika nilai p-nya (*dropout*) diganti maka scale-nya otomatis menyesuaikan.

2.8 Evaluasi Model

Evaluasi model merupakan proses yang digunakan untuk mengukur sejauh mana kinerja model *machine learning* dalam menghasilkan prediksi yang tepat terhadap data yang belum pernah dilihat sebelumnya [70][71]. Proses *evaluasi* dilakukan dengan membandingkan hasil prediksi *model* terhadap nilai kelas yang sebenarnya pada data uji [71]. Hasil *evaluasi* ini menjadi acuan dalam menentukan apakah model yang telah dilatih sudah cukup baik untuk digunakan atau masih memerlukan peningkatan lebih lanjut [71].

2.8.1 Confusion Matrix

Confusion Matrix adalah alat evaluasi berbentuk tabel matriks yang digunakan untuk mengukur kinerja model klasifikasi dengan cara membandingkan nilai prediksi yang dihasilkan oleh model dengan nilai sebenarnya atau di sebut *ground truth* dari data [72], [73][73]. Evaluasi menggunakan matriks ini memberikan gambaran yang komprehensif untuk memvisualisasikan jumlah hasil prediksi yang benar dan salah klasifikasi pada masing-masing kelas sentimen [72][73]. Karena penelitian ini menggunakan klasifikasi tiga kelas sentimen (Positif, Netral, dan Negatif), maka matriks yang terbentuk berukuran 3x3 [73].

Tabel 2.2 Ilustrasi *Confusion Matrix* 3 Kelas

Prediksi	**Prediksi Positif**	**Prediksi Netral**	**Prediksi Negatif**
Positif	*True* Positif (Benar)	*False* Netral (Salah)	*False* Negatif (Salah)
Netral	*False* Positif (Salah)	*True* Netral (Benar)	*False* Negatif (Salah)
Negatif	*False* Positif (Salah)	*False* Netral (Salah)	*True* Negatif (Benar)

© Karya Dilindungi UU Hak Cipta
1. Dilarang menyebarluaskan dokumen tanpa izin.
2. Dilarang melakukan plagiasi.
3. Pelanggaran diberlakukan sanksi sesuai peraturan UU Hak Cipta.

Berdasarkan tabel matriks multi-kelas di atas, ekstraksi nilai *True Positive* (TP), *True Negative* (TN), *False Positive* (FP), dan *False Negative* (FN) untuk setiap kelas sentimen dilakukan menggunakan pendekatan *One-vs-Rest* (membandingkan satu kelas spesifik terhadap gabungan kelas lainnya) [73]. Nilai-nilai yang telah diekstrak per kelas tersebut selanjutnya menjadi parameter dasar untuk perhitungan rumus metrik evaluasi.

2.8.2 Accuracy, Precision, Recall, dan F1-Score

Selain divisualisasikan, nilai yang diekstrak dari *Confusion Matrix* dipilih sebagai metode evaluasi utama untuk mengukur keakuratan prediksi model menggunakan metrik *Accuracy, Precision, Recall*, dan *F1-Score* [73]:

1. *Accuracy* (Akurasi): Merupakan proporsi prediksi yang benar (baik kelas positif, negatif, maupun netral) dibandingkan dengan keseluruhan data yang diuji. Metrik ini menunjukkan tingkat ketepatan model secara keseluruhan [73].

   $$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (2.39)$$

2. *Precision* (Presisi): Merupakan rasio data sentimen yang diprediksi benar pada kelas tertentu dibandingkan dengan total keseluruhan data yang diprediksi oleh model ke dalam kelas tersebut. Metrik ini berfokus untuk meminimalkan tingkat *False Positive* [73].

   $$Precision = \frac{TP}{TP+FP} \quad (2.40)$$

3. *Recall* (Sensitivitas): Merupakan rasio data sentimen yang diprediksi benar pada kelas tertentu dibandingkan dengan total keseluruhan data aktual yang memang berada di kelas tersebut. Metrik ini bertujuan untuk meminimalkan tingkat *False Negative* [73].

   $$Recall = \frac{TP}{TP+FN} \quad (2.41)$$

4. *F1-Score*: Merupakan nilai rata-rata harmonik yang menggabungkan *precision* dan *recall*. Metrik ini sangat efektif dan berguna untuk mengevaluasi model pada dataset yang memiliki ketidakseimbangan distribusi jumlah kelas (*imbalanced* data) [66].

   $$F1-Score = 2\,x\,\frac{Precision\ x\ Recall}{Precision+\ Recall} \quad (2.42)$$

© Karya Dilindungi UU Hak Cipta
1. Dilarang menyebarluaskan dokumen tanpa izin.
2. Dilarang melakukan plagiasi.
3. Pelanggaran diberlakukan sanksi sesuai peraturan UU Hak Cipta.