KLASIFIKASI SENTIMEN PADA ULASAN APLIKASI INDRIVE MENGGUNAKAN METODE *LONG SHORT TERM MEMORY*

SKRIPSI

Oleh :
RAHMAT FAUZAN
NIM. 210605110125

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM
MALANG
2025

KLASIFIKASI SENTIMEN PADA ULASAN APLIKASI INDRIVE MENGGUNAKAN METODE *LONG SHORT TERM MEMORY*

SKRIPSI

Diajukan kepada:
Universitas Islam Negeri Maulana Malik Ibrahim Malang
Untuk memenuhi Salah Satu Persyaratan dalam
Memperoleh Gelar Sarjana Komputer (S.Kom)

Oleh :
RAHMAT FAUZAN
NIM. 210605110125

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM MALANG

HALAMAN PERSETUJUAN

KLASIFIKASI SENTIMEN PADA ULASAN APLIKASI INDRIVE MENGGUNAKAN METODE *LONG SHORT TERM MEMORY*

SKRIPSI

Oleh :
RAHMAT FAUZAN
NIM. 210605110125

Telah Diperiksa dan Disetujui untuk Diuji:
Tanggal: 11 Juni 2025

Pembimbing I,	Pembimbing II,
Prof. Dr. Suhartono, M.Kom	Dr. Irwan Budi Santoso, M.Kom
NIP. 19680519 200312 1 001	NIP. 19770103 201101 1 004

Mengetahui,
Ketua Program Studi Teknik Informatika
Fakultas Sains dan Teknologi
Universitas Islam Negeri Maulana Malik Ibrahim Malang

Dr. Ir. Fachrul Kurniawan, M.MT., IPU
NIP. 19771020 200912 1 001

HALAMAN PENGESAHAN

KLASIFIKASI SENTIMEN PADA ULASAN APLIKASI INDRIVE MENGGUNAKAN METODE *LONG SHORT TERM MEMORY*

SKRIPSI

Oleh :
RAHMAT FAUZAN
NIM. 210605110125

Telah Dipertahankan di Depan Dewan Penguji Skripsi
dan Dinyatakan Diterima Sebagai Salah Satu Persyaratan
Untuk Memperoleh Gelar Sarjana Komputer (S.Kom)
Tanggal: 20 Juni 2025

Susunan Dewan Penguji

Ketua Penguji	: Dr. Totok Chamidy, M.Kom NIP. 19691222 200604 1 001	( )
Anggota Penguji I	: Okta Qomaruddin Aziz, M.Kom NIP. 19911019 201903 1 013	( )
Anggota Penguji II	: Prof. Dr. Suhartono, M.Kom NIP. 19680519 200312 1 001	( )
Anggota Penguji III	: Dr. Irwan Budi Santoso, M.Kom NIP. 19770103 201101 1 004	( )

Mengetahui dan Mengesahkan,
Ketua Program Studi Teknik Informatika
Fakultas Sains dan Teknologi
Universitas Islam Negeri Maulana Malik Ibrahim Malang

Dr. Ir. Fachrul Kurniawan, M.MT., IPU
NIP. 19771020 200912 1 001

PERNYATAAN KEASLIAN TULISAN

Saya yang bertanda tangan di bawah ini:

Nama	: Rahmat Fauzan
NIM	: 210605110125
Fakultas / Program Studi	: Sains dan Teknologi / Teknik Informatika
Judul Skripsi	: Klasifikasi Sentimen Pada Ulasan Aplikasi Indrive Menggunakan Metode *Long Short Term Memory*

Menyatakan dengan sebenarnya bahwa Skripsi yang saya tulis ini benar-benar merupakan hasil karya saya sendiri, bukan merupakan pengambil alihan data, tulisan, atau pikiran orang lain yang saya akui sebagai hasil tulisan atau pikiran saya sendiri, kecuali dengan mencantumkan sumber cuplikan pada daftar pustaka.

Apabila dikemudian hari terbukti atau dapat dibuktikan skripsi ini merupakan hasil jiplakan, maka saya bersedia menerima sanksi atas perbuatan tersebut.

Malang, 20 Juni 2025
Yang membuat pernyataan,

Rahmat Fauzan
NIM.210605110125

MOTTO

... Tidak ada perjalanan yang sia-sia, setiap langkah adalah bagian dari proses…

HALAMAN PERSEMBAHAN

Segala puji hanya milik Allah SWT atas segala rahmat dan karunia-Nya, sehingga penulis dapat menyelesaikan skripsi ini. Skripsi ini penulis persembahkan untuk orang tua tercinta, keluarga, dosen, sahabat, serta semua pihak yang telah memberikan dukungan, motivasi, saran, dan doa, yang membuat penelitian ini dapat diselesaikan.

KATA PENGANTAR

Puji dan syukur kepada Allah Subhanahu wa Ta'ala memberikan segala rahmat, hidayah, dan kesehatan-Nya sehingga penulis dapat menyelesaikan skripsi ini dengan baik. Tanpa izin dan pertolongan-Nya, segala usaha dan ikhtiar tidak akan berarti. Semoga segala yang telah diberikan-Nya senantiasa membawa berkah dan manfaat bagi penulis serta orang-orang di sekitar.

Penulis menyampaikan terima kasih kepada semua pihak yang telah mendukung, baik secara moral maupun material. Oleh karenaitu penulis mengucapkan terima kasih yang sebesar-besarnya kepada:

1. Prof. Dr. H. M. Zainuddin, M.A., Rektor Universitas Islam Negeri Maulana Malik Ibrahim Malang.
2. Prof. Dr. Sri Hariani, M.Si., Dekan Fakultas Sains dan Teknologi Universitas Islam Negeri Maulana Malik Ibrahim Malang.
3. Dr. Ir. Fachrul Kurniawan, M.MT., IPU., Ketua Program Studi Teknik Informatika Universitas Islam Negeri Maulana Malik Ibrahim Malang.
4. Prof. Dr. Suhartono, M.Kom., Pembimbing I, yang dengan penuh dedikasi meluangkan waktu untuk memberikan bimbingan dan motivasi selama proses penyusunan skripsi ini.
5. Dr. Irwan Budi Santoso, M.Kom., Pembimbing II, yang dengan tulus memberikan arahan dan dengan teliti meluruskan kesalahan dalam skripsi ini.
6. Dr. Totok Chamidy, M.Kom., Ketua Penguji yang telah memberikan banyak saran dan arahan sejak seminar proposal hingga sidang skripsi.

7. Okta Qomaruddin Aziz, M.Kom., Anggota Penguji 1 yang telah memberikan saran dan masukan berharga selama proses, mulai dari seminar proposal hingga sidang skripsi.
8. Kepada seluruh staf dan dosen Program Studi Teknik Informatika UIN Malang yang telah memberikan bimbingan, pengetahuan, dan dukungan yang sangat berarti bagi penulis
9. Kepada kedua orang tua, Bapak Syafnijar dan Ibu Srimeilefda, yang selalu memberikan dukungan, cinta, dan doa yang tak henti, yang menjadi sumber semangat penulis.
10. Kepada teman-teman kelas E di jurusan Teknik Informatika, UIN Malang, atas kebersamaan, motivasi, dan dukungan dari awal hingga akhir perkuliahan

Malang, Juni 2025

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

ABSTRAK

Fauzan, Rahmat. 2025. **Klasifikasi Sentimen Pada Ulasan Aplikasi Indrive Menggunakan Metode *Long Short Term Memory***. Skripsi. Program Studi Teknik Informatika Fakultas Sains dan Teknologi Universitas Islam Negeri Maulana Malik Ibrahim Malang. Pembimbing: (I) Prof. Dr. Suhartono, M.Kom (II) Dr. Irwan Budi Santoso, M.Kom.

.

Kata Kunci: Indrive, Klasifikasi, *Long Short Term Memory,* Sentimen

Perkembangan teknologi digital telah mendorong transformasi signifikan dalam sektor transportasi, salah satunya melalui hadirnya layanan transportasi online seperti Indrive. Seiring dengan bertambahnya jumlah pengguna, ulasan aplikasi Indrive di Google Play Store juga semakin meningkat, sehingga menjadi sumber informasi yang sangat berharga untuk mengevaluasi tingkat kepuasan pengguna. Berdasarkan hal tersebut, penelitian ini bertujuan untuk mengklasifikasikan sentimen pengguna terhadap aplikasi Indrive dengan menggunakan metode *Long Short-Term Memory* (LSTM). Dalam penelitian ini, sentimen dikategorikan ke dalam tiga kelas utama, yaitu positif, negatif, dan netral. Untuk memperoleh hasil yang optimal, eksperimen dilakukan dengan mengkombinasikan beberapa *hyperparameter*, meliputi jumlah *neuron* LSTM, *dropout rate*, dan *learning rate*. Hasil terbaik dicapai pada konfigurasi 128 *neuron*, *dropout* sebesar 0,2, dan *learning rate* 0,001, dengan performa model yang mencapai akurasi 89,3%, *precision* 80,7%, *recall* 80,4%, dan *F1 score* 80,5%. Pemilihan *hyperparameter* yang tepat menjadi salah satu aspek penting dalam mengoptimalkan kinerja sebuah model. Penambahan jumlah *neuron* hingga 128 dapat meningkatkan performa model karena memungkinkan model untuk memproses dan menyimpan lebih banyak informasi dari konteks kalimat. Sementara itu, penggunaan *dropout rate* yang kecil, seperti 0,2, memberikan hasil yang lebih optimal. Selain itu, *learning rate* sebesar 0,001 menghasilkan konvergensi yang stabil dan memaksimalkan kinerja model.

ABSTRACT

Fauzan, Rahmat. 2025. **Sentiment Classification on Indrive Application Reviews Using the Long Short Term Memory Method.** Thesis. Informatics Engineering Study Program, Faculty of Science and Technology, Maulana Malik Ibrahim State Islamic University, Malang. Promoter: (I) Prof. Dr. Suhartono, M.Kom (II) Dr. Irwan Budi Santoso, M.Kom.

Keywords: Classification, Indrive, Long Short-Term Memory, Sentiment

The development of digital technology has driven significant transformation in the transportation sector, one of which is through the presence of online transportation services such as Indrive. Along with the increasing number of users, Indrive application reviews on the Google Play Store are also increasing, making it a very valuable source of information for evaluating user satisfaction levels. Based on this, this study aims to classify user sentiment towards the Indrive application using the Long Short-Term Memory (LSTM) method. In this study, sentiments are categorized into three main classes, namely positive, negative, and neutral. To obtain optimal results, experiments were conducted by combining several hyperparameters, including the number of LSTM neurons, dropout rate, and learning rate. The best results were achieved in a configuration of 128 neurons, a dropout of 0.2, and a learning rate of 0.001, with model performance reaching 89.3% accuracy, 80.7% precision, 80.4% recall, and 80.5% *F1 score*. Choosing the right hyperparameters is one of the important aspects in optimizing the performance of a model. Increasing the number of neurons to 128 can improve model performance because it allows the model to process and store more information from the sentence context. Meanwhile, the use of a small dropout rate, such as 0.2, provides more optimal results. In addition, a learning rate of 0.001 results in stable convergence and maximizes model performance.

مستخلص البحث

فوزان، رحمت. 2025. **تصنيف المشاعر في تقييمات تطبيق إندرايف باستخدام طريقة الذاكرة قصيرة وطويلة المدى**. رسالة ماجستير. برنامج دراسات تكنولوجيا المعلومات، كلية العلوم والتكنولوجيا، جامعة إسلامية نيجيريا مولانا مالك إبراهيم مالانغ. المشرف (I) :البروفيسور د. سوهارتونو، م.كوم (II) الدكتور إروان بودي سانتوسو، م.كوم.

الكلمات الرئيسية: الذاكرة طويلة وقصيرة الأمد، التصنيف، المشاعر، إندرايف

لقد أدى تطور التكنولوجيا الرقمية إلى حدوث تحول كبير في قطاع النقل، ويتمثل أحد هذه التحولات في وجود خدمات النقل عبر الإنترنت مثل تطبيق .Indrive وإلى جانب تزايد عدد المستخدمين، تتزايد أيضاً مراجعات تطبيق Indrive على متجر Google Play، مما يجعله مصدراً قيماً للغاية للمعلومات لتقييم مستويات رضا المستخدمين. وبناءً على ذلك، تهدف هذه الدراسة إلى تصنيف مشاعر المستخدمين تجاه تطبيق إندرايف باستخدام طريقة الذاكرة طويلة المدى القصيرة الأجل .(LSTM) في هذه الدراسة، يتم تصنيف المشاعر إلى ثلاث فئات رئيسية، وهي الإيجابية والسلبية والمحايدة. وللحصول على أفضل النتائج، أُجريت التجارب من خلال الجمع بين العديد من المعلمات الفائقة، بما في ذلك عدد الخلايا العصبية في الذاكرة طويلة المدى ومعدل التسرب ومعدل التعلم. تم تحقيق أفضل النتائج في تكوين مكون من 128 خلية عصبية، وتسرب 0.2، ومعدل تعلم 0.001، حيث بلغ أداء النموذج دقة 89.3% ودقة 80.7% ودقة 80.7% واسترجاع 80.4% ودرجة .%80.5 F1 يعد اختيار المعلمات الفائقة الصحيحة أحد الجوانب المهمة في تحسين أداء النموذج. يمكن أن تؤدي زيادة عدد الخلايا العصبية إلى 128 خلية عصبية إلى تحسين أداء النموذج لأنه يسمح للنموذج بمعالجة وتخزين المزيد من المعلومات من سياق الجملة. وفي الوقت نفسه، يوفر استخدام معدل تسرب صغير، مثل 0.2، نتائج مثالية أكثر. بالإضافة إلى ذلك، يؤدي معدل التعلم 0.001 إلى تقارب مستقر ويزيد من أداء النموذج إلى أقصى حد.

BAB I

PENDAHULUAN

1.1 Latar Belakang

Perkembangan teknologi saat ini telah membawa transformasi besar dalam berbagai sektor, termasuk pada sektor transportasi. Salah satu perkembangan yang signifikan adalah munculnya layanan transportasi daring yang mengintegrasikan teknologi dengan transportasi konvensional. Dengan aplikasi daring, pengguna dapat memesan kendaraan secara praktis yang memberikan kemudahan dan kenyamanan yang sangat dibutuhkan oleh masyarakat (Aziah & Adawia, 2018). Inovasi ini telah mengubah kebiasaan masyarakat dalam mengakses layanan transportasi, menjadikannya lebih fleksibel dan dapat diakses kapan saja.

Salah satu perusahaan penyedia layanan transportasi online adalah Indrive, yang aplikasinya diluncurkan pada tahun 2012 di Yakutsk, Rusia. Berbeda dengan platform lainnya, Indrive mengusung konsep unik yang berbentuk negosiasi tarif antara pengguna dan pengemudi, yang memberikan tingkat transparansi dan fleksibilitas lebih tinggi (Laras, 2023). Hal ini menjadikan Indrive berbeda dari platform transportasi daring lainnya dalam menghadapi persaingan yang semakin ketat.

Hingga saat ini, Indrive telah berkembang di lebih dari 300 kota di berbagai negara, termasuk Indonesia yang mulai dilayani pada tahun 2019 (Laras, 2023). Kehadiran Indrive di Indonesia memberikan alternatif bagi masyarakat yang

mencari layanan dengan tarif fleksibel yang dapat dinegosiasikan, berbeda dengan pendekatan tarif otomatis yang diterapkan oleh platform lain.

Seiring dengan berkembang Indrive, jumlah ulasan pada aplikasi Indrive pada Google Play Store juga semakin meningkat. Ulasan-ulasan ini menjadi sumber informasi yang penting bagi pengembang untuk menilai tingkat kepuasan pengguna, dan juga untuk menilai kinerja aplikasi serta meningkatkan mutu layanan. Meskipun rating aplikasi sering dijadikan indikator utama kepuasan pengguna, terdapat ketidaksesuaian antara rating dan isi ulasan. Beberapa pengguna memberikan rating rendah dengan komentar positif, sementara yang lain memberikan rating tinggi namun diikuti dengan keluhan (Raffi et al., 2023). Ketidaksesuaian antara rating dan ulasan menunjukkan bahwa rating saja tidak cukup untuk menggambarkan kepuasan pengguna. Karena itu, dibutuhkan metode yang dapat memahami setiap komentar yang ditulis oleh pengguna.

Salah satu pendekatan yang dapat mengatasi permasalahan ini adalah analisis sentimen, merupakan bagian dari bidang *Natural Language Processing* (NLP). Analisis sentimen adalah proses mengklasifikasikan teks berdasarkan sentimen positif, negatif, atau netral yang terkandung dalam opini pengguna (Mutmatinah, 2024). Melalui analisis sentimen, pengembang dapat pemahaman yang lebih mendalam tentang opini publik, mendapatkan wawasan yang berguna untuk mengevaluasi layanan, dan pada akhirnya meningkatkan kualitas aplikasi. Tanggapan negatif dari pengguna dapat digunakan sebagai bahan evaluasi, tanggapan positif sebagai acuan untuk mempertahankan keunggulan layanan, dan

tanggapan netral dapat dimanfaatkan untuk menggali ide-ide baru dalam pengembangan produk (Lisanthoni et al., 2024).

Meningkatkan layanan dengan melakukan evaluasi berdasarkan opini dan pengalaman pengguna juga sejalan dengan ajaran Islam, sebagaimana disebutkan dalam Al-Qur'an surah Al-Hasyr ayat 18:

يٰٓاَيُّهَا الَّذِيْنَ اٰمَنُوا اتَّقُوا اللّٰهَ وَلْتَنْظُرْ نَفْسٌ مَّا قَدَّمَتْ لِغَدٍۚ وَاتَّقُوا اللّٰهَۗ اِنَّ اللّٰهَ خَبِيْرٌۢ بِمَا تَعْمَلُوْنَ ١٨

"Wahai orang-orang yang beriman, bertakwalah kepada Allah dan hendaklah setiap orang memperhatikan apa yang telah diperbuatnya untuk hari esok (akhirat). Bertakwalah kepada Allah. Sesungguhnya Allah Maha Teliti terhadap apa yang kamu kerjakan." (QS. Al- Hasyr: 18)

Menurut penjelasan dalam tafsir Al-Misbah, ayat ini mengandung anjuran bagi setiap individu untuk senantiasa melakukan evaluasi terhadap apa yang telah diperbuatnya. Dalam hal ini diibaratkan seperti tukang yang sedang meninjau kembali hasil pekerjaanya, jika sudah baik, ia menyempurnakannya dan jika masih kurang, ia memperbaikinya. Sikap ini menggambarkan pentingnya introspeksi agar hasil akhirnya sebaik mungkin (Ramadhan & Hidayat, 2024).

Penelitian mengenai analisis sentimen aplikasi Indrive telah dilakukan oleh (Mola et al., 2024) melalui pendekatan *machine learning*. Penelitian ini menenjukkan metode *Support Vector Machine* (SVM) dan *Logistic Regression* memberikan akurasi sebesar 89%. Sementara itu, *Random Forest* hanya mendapatkan akurasi 88%, namun lebih unggul dalam menangani kelas minoritas karena memiliki keseimbangan nilai evaluasi untuk *Recall* dan *Precision*. Penerapan penyimbangan data menggunkan SMOTE tidak memberikan pengaruh yang signifikan terhadap performa SVM dan *Logistic Regression*, namun pada

model *Naïve Bayes*, teknik ini mampu meningkatkan nilai *recall*, khususnya pada kelas netral.

Penelitian sebelumnya masih terbatas pada penggunaan metode *machine learning* konvensional, seperti *Support Vector Machine* (SVM) dan *Naïve Bayes*, yang kurang efektif dalam mengolah data sekuensial, sehingga tidak dapat memahami hubungan antar kata. Salah satu metode yang dapat mengatasi keterbatasan ini adalah *Long Short-Term Memory* (LSTM), yang memang dirancang untuk mengelola data sekuensial dan dapat menangkap konteks dalam teks yang panjang (Basri & Utami, 2025).

LSTM (*Long Short-Term Memory*) adalah pengembangan dari *Recurrent Neural Network* (RNN) yang dilengkapi dengan *cell state*, yang memungkinkan model untuk menyimpan dan mengatur informasi penting dari urutan data yang panjang. Kemampuannya dalam mempertahankan informasi jangka panjang menjadikan LSTM pilihan yang tepat untuk mengelola data dengan rentang waktu panjang, seperti teks (Pradana et al., 2023).

Dengan kemampuan LSTM dalam mengelola data sekuensial dan menyimpan informasi jangka panjang, seperti yang telah dijelaskan pada pembahasan sebelumnya, LSTM menjadi pilihan dalam penelitian ini untuk mengklasifikasikan sentimen pada ulasan pada aplikasi Indrive. Melalui penerapan metode ini, diharapkan dapat memperoleh gambaran mengenai kinerja LSTM dalam mengklasifikasikan sentimen pada ulasan aplikasi Indrive.

1.2 Rumusan Masalah

Bagaimana kinerja metode *Long Short-Term Memory* (LSTM) dalam mengklasifikasikan sentimen pengguna terhadap aplikasi Indrive.

1.3 Batasan Masalah

Batasan masalah yang di tetapkan dalam penelitian ini adalah :

1. Dataset yang digunakan bersumber dari artikel "*Perbandingan Metode Machine Learning* dalam *Analisis Sentimen Komentar Pengguna Aplikasi Indriver* pada *Dataset Tidak Seimbang*".
2. Evaluasi model menggunakan *confusion matrix*, yang mencakup akurasi, *precission*, *recall*, dan *F1 score*.

1.4 Tujuan Penelitian

Mengetahui kinerja metode *Long Short-Term Memory* (LSTM) dalam mengklasifikasikan sentimen pengguna terhadap aplikasi Indrive.

1.5 Manfaat Penelitian

1. Membantu pengembang aplikasi Indrive dalam memahami opini atau sentimen pengguna, sehingga dapat digunakan sebagai bahan evaluasi untuk meningkatkan kualitas layanan aplikasi.
2. Menjadi referensi dalam pengembangan dan penerapan metode *Long Short-Term Mem*ory, dalam klasifikasi sentimen pada aplikasi Indrive

BAB II

STUDI PUSTAKA

2.1 Penelitian Terkait

Penelitian yang dilakukan oleh (Shyahrin et al., 2023) membahas penerapan LSTM dan *Word2Vec* untuk analisis sentimen pada ulasan pengguna aplikasi Ferizy. Ferizy merupakan inovasi digital dari PT ASDP yang dirancang untuk mempermudah pemesanan tiket kapal ferry secara daring. Penelitian ini bertujuan untuk menganalisis sentimen pengguna dengan tujuan untuk mengevaluasi tingkat kepuasan, mengidentifikasi kritik dan saran, serta mengetahui kelemahan yang ada pada aplikasi. Metode yang digunakan adalah LSTM untuk klasifikasi sentimen, sedangkan fitur teks diekstraksi menggunakan *Word2Vec* dari library *Gensim* dengan variasi *Continuous Bag of Words* (CBOW) dan *skip-gram*. Dataset terdiri dari 5.000 ulasan, yang dikelompokkan menjadi sentimen positif dan negatif. Hasil penelitian menunjukkan akurasi skip-gram sebesar 88,20%, lebih tinggi dibandingkan CBOW yang akurasinya hanya 74,20%.

Penelitian selanjutnya yang dilakukan oleh (Muhammad et al., 2021). Penelitian ini mengenai analisis sentimen ulasan hotel berbahasa Indonesia dengan mengombinasikan *Long Short-Term Memory* (LSTM) dan *Word2Vec*. Dalam upaya untuk mencapai akurasi terbaik, penelitian ini melakukan beberapa percobaan dengan total 144 kombinasi parameter. Untuk *Word2Vec*, percobaan mencakup variasi arsitektur model, yaitu *Skip-gram* dan *Continuous Bag of Words* (CBOW), metode evaluasi (*Hierarchical Softmax* dan *Negative Sampling*), serta

dimensi vektor dengan nilai 100, 200, dan 300. Pada model LSTM, peneliti menguji variasi parameter *dropout* (0,2, 0,5, dan 0,7), teknik *pooling* (*average* dan *max*), serta nilai *learning rate* (0,0001 dan 0,001). Dari serangkaian percobaan tersebut, kombinasi parameter yang menunjukkan hasil terbaik adalah arsitektur *Skip-gram* pada *Word2Vec* dengan metode evaluasi *Hierarchical Softmax* dan dimensi vektor 300, serta LSTM dengan *dropout* sebesar 0,2 dan *learning rate* 0,001. Kombinasi ini menghasilkan akurasi klasifikasi sebesar 85,96%.

Penelitian selanjutnya yang relevan adalah penelitian oleh (Rahman et al., 2021), yang menggunakan metode *Long Short-Term Memory* (LSTM) untuk analisis sentimen tweet terkait kebijakan karantina COVID-19 di Indonesia. Penelitian ini menggunakan 1364 tweet berbahasa Indonesia yang dikumpulkan secara mandiri, dengan menambahkan *Word Embedding* sebagai representasi kata. Hasil penelitian menunjukkan bahwa LSTM menghasilkan akurasi sebesar 81%, dengan *precision* dan *recall* masing-masing sebesar 80%. Kinerja metode LSTM dalam penelitian ini lebih unggul dibandingkan dengan dua metode lainnya, yaitu *Naïve Bayes* dan *Recurrent Neural Network* (RNN), dengan peningkatan akurasi sebesar 7% dan 10% lebih baik dibandingkan kedua metode tersebut. LSTM terbukti cukup baik dalam menangani data teks berbahasa Indonesia. Penelitian ini juga mengonfigurasi model LSTM dengan parameter yang optimal, yaitu batch size sebesar 30, ukuran *hidden layer* (*hidden size*) 100, dan *learning rate* 0,0001.

Penelitian oleh (Pradana et al., 2023) menguji konfigurasi optimal model LSTM untuk analisis sentimen terhadap pemindahan ibu kota Indonesia dengan mengatur jumlah *hidden unit*, metode optimasi, dan proporsi pembagian data.

Dalam uji coba *hidden unit*, jumlah yang lebih kecil, seperti 16, mampu mencapai akurasi mendekati sempurna, sementara *hidden unit* yang lebih besar tetap memberikan akurasi tinggi, mencapai 96%. Optimisasi model menggunakan berbagai metode menunjukkan bahwa *Adam optimizer* paling optimal, menghasilkan akurasi mendekati 100%. Algoritma *Adam* menghitung dan memperbarui nilai adaptif dari *learning rate* untuk setiap parameter dalam model berdasarkan perhitungan dari moment pertama dan kedua dari gradien parameter. Dengan demikian, algoritma ini dapat menyesuaikan *learning rate* secara *adaptif* untuk masing-masing parameter, memungkinkan model untuk mengatasi masalah seperti *vanishing gradient* maupun *exploding gradient*.

Penelitian dilakukan oleh (Miftahusalam et al., 2022) yang berjudul "*Perbandingan Algoritma Random Forest, Naïve Bayes, dan Support Vector Machine Pada Analisis* Sentimen *Twitter Mengenai Opini Masyarakat Terhadap Penghapusan Tenaga Honorer*" berhasil meningkatkan akurasi model dengan menyeimbangkan dataset menggunakan teknik *Random Oversampling*, yaitu menggandakan data minoritas secara acak hingga setara dengan data mayoritas. Hasil penelitian menunjukkan bahwa akurasi metode *Random Forest* meningkat dari 62,43% menjadi 66,67%, SVM dari 58,96% menjadi 65,33%, dan *Naïve Bayes* dari 61,85% menjadi 64,00% setelah menerapkan teknik tersebut.

Penelitian oleh (Mola et al., 2024) berjudul "*Perbandingan Metode Machine Learning dalam Analisis Sentimen Komentar Pengguna Aplikasi Indriver pada Dataset Tidak Seimbang*" membandingkan metode *machine learning* dalam analisis sentimen komentar pengguna Indriver. Data diperoleh dari Google Play Store,

dengan 12.147 komentar, yang setelah *preprocessing* (*cleaning, case folding, normalisasi, tokenizing, stemming, convert negation,* dan *stopword removal*) tersisa 12.135 komentar. Sentimen diklasifikasikan sebagai positif, negatif, dan netral, dengan TF-IDF sebagai teknik pembobotan fitur. Model yang diuji meliputi SVM, *Logistic Regression*, *Random Forest*, dan *Naïve Bayes*. Hasil evaluasi tanpa SMOTE menunjukkan SVM dan *Logistic Regression* memiliki akurasi tertinggi (89%), tetapi kurang efektif dalam menangani sentimen netral. *Random Forest* lebih seimbang, dengan *F1 score* 55%, sementara Naïve Bayes memiliki *F1 score* terendah (37%). Penggunaan SMOTE tidak memberikan pengaruh signifikan terhadap performa model, namun pada model *Naïve Bayes*, teknik ini berhasil meningkatkan nilai *recall*, khususnya pada kelas netral.

Berdasarkan penelitian-penelitian di atas, penulis menyusun Tabel 2.1 untuk merangkum metodologi, parameter model, dan hasil yang diperoleh.Tabel ini bertujuan untuk memberikan gambaran mengenai penelitian yang relevan dan sebagai dasar dalam merancang pendekatan penelitian ini.

Tabel 2.1 Penelitian Terkait

No	Nama Peneliti	Judul Penelitian	Metode	Hasil Penelitian
1	Shyahrin et al. (2023)	Penerapan Metode Long Short-Term Memory dan Word2Vec dalam Analisis Sentimen Ulasan pada Aplikasi Ferizy	*Long-Short Term Memory*	LSTM dengan *Word2Vec* (*skip-gram*) menghasilkan akurasi 88,2%, lebih baik dibandingkan CBOW yang hanya 74,20%
2	(Muhammad et al., 2021).	Sentyment Analysis Using Word2vec And Long Short-Term Memory (LSTM) For Indonesian Hotel Reviews	*Long-Short Term Memory*	Arsitektur *Skip-gram* pada *Word2Vec*, dimensi vektor 300, serta LSTM dengan *dropout* 0,2 dan *learning rate* 0,001 menghasilkan akurasi 85,96%
3	(Rahman et al., 2021)	Analisis Sentimen Tweet COVID-19 menggunakan *Word Embedding* dan	*LSTM, Naïve Bayes,* dan *RNN*	Metode LSTM menghasilkan akurasi 81%, lebih unggul dibandingkan metode Naïve Bayes dan RNN

No	Nama Peneliti	Judul Penelitian	Metode	Hasil Penelitian
		Metode *Long Short-Term Memory* (LSTM)		
4	(Pradana et al., 2023)	Analisis Sentimen Pemindahan Ibu Kota Indonesia pada Media Sosial Twitter menggunakan Metode LSTM dan Word2Vec	*LSTM* dan *Word2Vec*	Hidden units kecil yaitu 16, dapat mencapai akurasi tinggi hingga 96%, sementara *Adam optimizer* memberikan hasil terbaik dengan akurasi hampir 100%.
5	(Miftahusalam et al., 2022)	Perbandingan Algoritma Random Forest, Naïve Bayes, dan Support Vector Machine Pada Analisis Sentimen Twitter Mengenai Opini Masyarakat Terhadap Penghapusan Tenaga Honorer	*Random Forest, Naïve Bayes,* dan *Support Vector Machine.*	Penelitian ini mengatasi data tidak seimbang dengan teknik *random oversampling.* Dengan Teknik dapat meningkatkan akurasi pengujian higga 2 hingga 5 persen.
6	(Mola et al., 2024)	Perbandingan Metode Machine Learning dalam Analisis Sentimen Komentar Pengguna Aplikasi Indriver pada Dataset Tidak Seimbang	*SVM, Logistic Regression, Random Forest, Naïve Bayes*	SVM *dan Logistic Regression* memiliki akurasi tertinggi (89%) tetapi kurang efektif pada sentimen netral. *Random Forest* lebih seimbang dengan *F1 score* 55%, sedangkan *Naïve Bayes* memiliki *F1 score* terendah (37%).

2.2 Analisis Sentimen

Analisis Sentimen adalah bagian dari *Natural Language Processing* (NLP) yang bertujuan untuk mengklasifikasikan sentimen teks menjadi kategori positif, negatif, atau netral. Proses ini melibatkan identifikasi emosi dan ekspresi yang terkandung dalam teks. Seiring berkembangnya media sosial pada saat ini, seseorang dapat secara terbuka mengekspresikan pendapat dan perspektif mereka, sehingga penting bagi suatu organisasi untuk memahami sentimen yang mendasari opini-opini tersebut agar dapat membuat keputusan yang tepat. Analisis sentimen menjadi alat yang sangat berguna dalam memahami pandangan masyarakat di berbagai bidang, seperti bisnis, politik, dan lainnya (Tan et al., 2023).

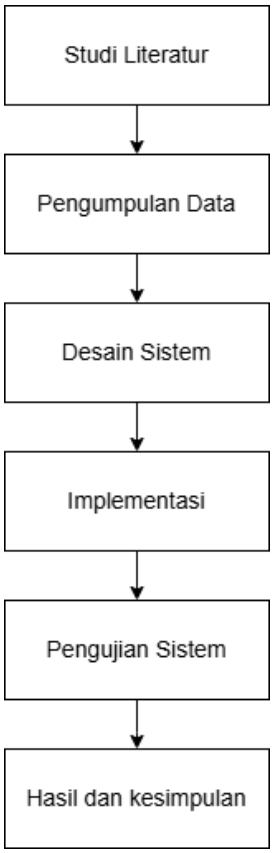
Analisis sentimen terdiri dari beberapa tahapan, yaitu *preprocessing*, *feature extraction*, dan klasifikasi. Dalam tahap *preprocessing*, terdapat beberapa langkah yang perlu dilakukan, antara lain *case folding*, pembersihan teks (*clean text*), normalisasi, penghapusan *stop words*, *stemming*, dan tokenisasi. Setelah tahap ini, data teks diubah menjadi representasi numerik melalui teknik *feature extraction*, seperti *Term Frequency–Inverse Document Frequency* (TF-IDF), GloVe, *fastText*, dan *word2vec*. Representasi ini memungkinkan model untuk memahami makna kata dalam konteks tertentu. Selanjutnya, teks yang telah dikonversi menjadi fitur ini diklasifikasikan ke dalam kategori sentimen menggunakan metode *machine learning*, seperti *logistic regression*, *naive Bayes*, dan *support vector machines*, atau model *deep learning* seperti *long short-term memory* (LSTM) dan *recurrent neural networks* (RNN) (Tan et al., 2023).

BAB III

DESAIN DAN IMPLEMENTASI

3.1 Desain Penelitian

Desain penelitian digunakan sebagai panduan dalam melaksanakan setiap langkah penelitian secara sistematis. Desain penelitian ini divisualisasikan dalam bentuk diagram blok untuk mempermudah pemahaman, seperti yang ditunjukkan pada Gambar 3.1.



Gambar 3.1 Diagram Desain Penelitian

Sesuai dengan Gambar 3.1, penelitian ini dilakukan secara bertahap. Tahap pertama dimulai dengan pengumpulan studi literatur sebagai dasar teori yang

mendukung penelitian. Selanjutnya dilakukan proses pengumpulan data, yang kemudian diikuti dengan perancangan sistem secara menyeluruh. Setelah desain sistem selesai, dilakukan tahap implementasi berdasarkan rancangan yang telah dibuat. Hasil dari implementasi tersebut kemudian diuji dan dianalisis untuk memperoleh kesimpulan dari penelitian ini.

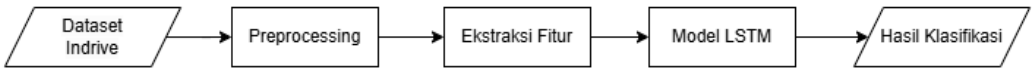
3.2 Pengumpulan Data

Data yang digunakan merupakan data sekunder yang didapatkan dari artikel berjudul "*Perbandingan Metode Machine Learning dalam Analisis Sentimen Komentar Pengguna Aplikasi InDriver pada Dataset Tidak Seimbang*", yang ditulis oleh Sebastianus Adi Santoso Mola, Yufridon Charisma Luttu, dan Dessy Nelci Rumlaklak. Artikel ini dipublikasikan dalam Jurnal Sistem Informasi Bisnis Vol. 3 tahun 2024. Dataset dalam penelitian ini berasal dari komentar pengguna aplikasi Indrive di Google Play Store, yang dikumpulkan selama periode 4 Oktober 2022 hingga 24 Januari 2023. Total komentar yang terkumpul berjumlah 12.141, mencakup berbagai ulasan yang mewakili pengalaman pengguna terhadap aplikasi Indrive.

Setiap komentar dilabeli oleh dosen ahli bahasa Indonesia dari Universitas Nusa Cendana untuk memastikan objektivitas dan akurasi dalam analisis data sentimen. Pelabelan dilakukan untuk mengkategorikan sentimen dalam komentar menjadi positif, negatif, atau netral. Hasil pelabelan dataset dapat diakses pada tautan https://bit.ly/440zMS6. Jumlah data negatif berjumlah 7314, kemudian data netral 1120, dan data positif berjumlah 3707.

3.3 Desain Sistem

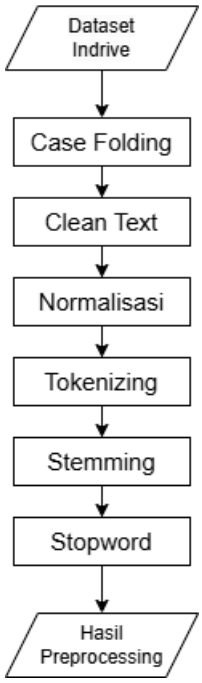
Desain sistem bertujuan untuk menggambarkan alur kerja dari sistem yang akan dikembangkan dalam penelitian ini. Diagram pada Gambar 3.2 memperlihatkan alur proses pengembangan sistem, sebagai berikut:



Gambar 3. 2 Diagram Blok Desain Sistem

3.4 *Preprocessing*

Pra-pemrosesan atau *preprocessing* adalah tahapan yang bertujuan untuk memperbaiki kualitas data supaya dapat diproses dengan lebih efisien dalam proses klasifikasi sentimen (Mutmatinah, 2024). Proses *Preprocessing* terdiri dari beberapa tahapan, yaitu *case folding*, *data cleaning*, normalisasi, *tokenizing*, *stopword removal*, dan *stemming*.



Gambar 3. 3 Diagram Blok *Preprocessing*

3.4.1 *Case Folding*

Case folding adalah mengubah huruf kapital pada teks menjadi huruf kecil. Langkah ini bertujuan untuk membuat teks menjadi lebih seragam. (Rahman et al., 2021). Untuk contoh case folding bisa dilihat dalam Tabel 3.1.

Tabel 3.1 Contoh Hasil *Case Folding*

Sebelum Case Folding	Setelah Case Folding
aplikasi makin aneh	aplikasi makin aneh
Terima Kasih	terima kasih
supir nya baik dan ramah...	supir nya baik dan ramah...

3.4.2 *Data Cleaning*

Data cleaning merupakan tahap membersihkan data dari elemen-elemen yang tidak relevan. Dalam tahap ini, proses pembersihan meliputi penghapusan emotikon, simbol, URL, *mention*, *hashtag*, angka dalam kata, tanda baca khusus, spasi berlebih, baris baru, dan tag HTML (Azrul et al., 2024). Untuk contohnya dapat dilihat pada Tabel 3.2.

Tabel 3.2 Contoh Hasil *Data Cleaning*

Sebelum Data Cleaning	Setelah Data Cleaning
aplikasi makin aneh	aplikasi makin aneh
terima kasih	terima kasih
supir nya baik dan ramah...	supir nya baik dan ramah

3.4.3 Normalisasi

Normalisasi adalah proses yang bertujuan untuk mengubah kata-kata tidak baku menjadi bentuk baku. Proses ini digunakan untuk menangani berbagai bentuk variasi bahasa, seperti singkatan, bahasa gaul (*slang*), serta kesalahan ejaan yang umum ditemukan dalam kalimat (Ivan et al., 2019). Dalam penelitian ini, proses normalisasi menggunakan kamus kata baku yang diperoleh dari GitHub dengan

nama akun *ramaprakoso*. Contoh hasil dari proses normalisasi dapat dilihat pada Tabel 3.3.

Tabel 3.3 Contoh Hasil Normalisasi

Sebelum Normalisasi	Setelah Normalisasi
aplikasi makin aneh	aplikasi makin aneh
terima kasih	terima kasih
supir nya baik dan ramah	supir nya baik dan ramah

3.4.4 *Tokenization*

Tokenization merupakan tahap mengubah setiap teks pada data ulasan menjadi rangkaian kata atau token (Shyahrin et al., 2023). Contoh hasil *tokenization* dapat dilihat dari Tabel 3.4.

Tabel 3.4 Contoh Hasil *Tokenizazion*

Sebelum Tokenizazion	**Sebelum** Tokenizazion
aplikasi makin aneh	[‘aplikasi’, 'makin', 'aneh']
terima kasih	['terima', 'kasih']
supir nya baik dan ramah	['supir', 'nya', 'baik', 'dan', 'ramah']

3.4.5 *Stemming*

Stemming adalah proses mengubah kata berimbuhan atau kata turunan menjadi kata dasar. Langkah ini bertujuan untuk menyederhanakan bentuk kata. (Shyahrin et al., 2023). Misalnya, kata "berjalan," "berjalanlah," dan "berjalannya" semuanya akan diubah menjadi kata dasar "jalan". Dalam penelitian ini, proses *stemming* akan menggunkan *Library Sastrawi*. Contoh penerapan stemming dapat dilihat pada Tabel 3.5.

Tabel 3.5 Contoh Hasil *Stemming*

Sebelum Stemming	Sebelum Stemming
[‘aplikasi’, 'makin', 'aneh']	[‘aplikasi’, 'makin', 'aneh']
['terima', 'kasih']	['terima', 'kasih']
['supir', 'nya', 'baik', 'dan', 'ramah']	['supir', 'nya', 'baik', 'dan', 'ramah']

3.4.6 *Stopword Removal*

Stopword removal adalah proses penghapusan kata hubung yang tidak memiliki makna yang signifikan. Kata-kata ini, seperti "dan," "atau," "di," dan "ke," sering kali hanya berfungsi sebagai penghubung atau pelengkap dalam kalimat tetapi tidak memengaruhi konteks atau makna utama teks (Azrul et al., 2024). Dalam penelitian ini, proses *stopword removal* dilakukan dengan menggunakan daftar *stopword* dari NLTK. Berikut contoh proses *stopword removal* yang dapat dilihat pada Tabel 3.6.

Tabel 3.6 Contoh *Stopword Removal*

Sebelum Stopword Removal	Sebelum Stopword Removal
[‘aplikasi’, 'makin', 'aneh']	[‘aplikasi’, 'makin', 'aneh']
['terima', 'kasih']	['terima', 'kasih']
['supir', 'nya', 'baik', 'dan', 'ramah']	['supir', 'baik', 'ramah']

3.4.7 *One-hot encoding*

One-hot encoding adalah mengubah label kelas menjadi format numerik berupa vektor biner, di mana setiap kelas unik direpresentasikan oleh vektor yang terdiri dari angka nol dan satu (Lisanthoni et al., 2024). Proses ini dilakukan karena algoritma *machine learning* tidak dapat mengelola data dalam bentuk teks secara langsung. Oleh karena itu, data teks perlu dikonversi terlebih dahulu ke dalam bentuk numerik agar dapat diproses oleh metode seperti *Long Short-Term Memory* (LSTM) (Sari et al., 2020).

Dalam penelitian ini, *one-hot encoding* digunakan untuk merepresentasikan setiap kelas sebagai berikut:

- Negatif (0) : [1,0,0]
- Netral (1) : [0,1,0]

– Positif: (2) : [0,0,1]

Tabel 3.7 Contoh Hasil *One-Hot Encoding*

Data Preprocessing	Kelas	One-Hot Encoding
[‘aplikasi’, 'makin', 'aneh']	Negatif	[1,0,0]
['terima', 'kasih']	Netral	[0,1,0]
['supir', 'baik', 'ramah']	Positif	[0,0,1]

3.5 Ekstraksi Fitur

Ekstraksi fitur adalah proses mengubah teks menjadi representasi numerik agar dapat diolah oleh model. Proses ekstraksi fitur terdiri dari beberapa tahapan, yaitu pembuatan indeks dan penambahan padding, kemudian diubah ke dalam bentuk representasi vektor kata (*word embeddings*) yang diperoleh dari model *Word2Vec* yang perlu di latih terlebih dahulu (*pre-trained*). Berikut tahapan dari ekstraksi fitur:

3.5.1 Indeks kata

Indeks kata merupakan proses mengonversi teks ke dalam bentuk numerik agar dapat diolah oleh algoritma. Hal ini dilakukan karena *machine learning* hanya dapat mengolah data dalam format angka. Proses ini diawali dengan pembuatan daftar kata unik (*vocab*), dengan setiap kata unik diberi angka sebagai representasinya (Pane & Ramdan, 2022). Selanjutnya, setiap baris data diubah menjadi angka berdasarkan *vocab* tersebut. Hasil *vocab* bisa dilihat dalam Tabel 3.8, dan contoh hasil dari proses indeks bisa dilihat dalam Tabel 3.9.

Tabel 3.8 *Vocab*

Kata Unik	Indeks
<OOV>	1
tidak	2
driver	3
aplikasi	4
.....	.....
amien	7139

Kata Unik	Indeks
unistal	7140
moding	7141

Tabel 3.9 Contoh hasil indeks

Data Preprocessing	Indeks
[‘aplikasi’, 'makin', 'aneh']	[4,44,162]
['terima', 'kasih']	[40,23]
['supir', 'baik', 'ramah']	[833,7,124]

3.5.2 *Padding*

Padding bertujuan untuk memastikan semua input memiliki panjang yang sama, sehingga dapat diproses dengan baik oleh model LSTM. Kalimat yang panjangnya kurang dari nilai maksimum akan diberi *padding* dengan menambahkan nilai 0 (nol) di awal kalimat. Dengan demikian, *padding* memungkinkan model untuk menangani data teks yang panjang bervariasi secara konsisten. (Pane & Ramdan, 2022). Untuk hasil *padding* bisa dilihat pada Tabel 3.10

Tabel 3.10 Contoh hasil indeks dan pading

Hasil Preprocessing	Indeks
[‘aplikasi’, 'makin', 'aneh']	[[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,4,22,162]
['terima', 'kasih']	[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,40,23]
['supir', 'baik', 'ramah']	[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,833,7,124]

3.5.3 *Word2Vec*

Word2Vec diperkenalkan pertama kali oleh Tomas Mikolov dan timnya pada tahun 2013, dengan tujuan memetakan kata-kata ke dalam ruang vektor, sehingga kata-kata dengan makna yang mirip atau sering muncul bersama memiliki vektor yang berdekatan (Pradana et al., 2023). *Word2Vec* memiliki dua arsitektur utama, yaitu *skip-gram* dan *continuous bag of words* (CBOW). Dalam penelitian ini, peneliti menggunakan arsitektur *Skip-gram*, yang bertujuan untuk memprediksi

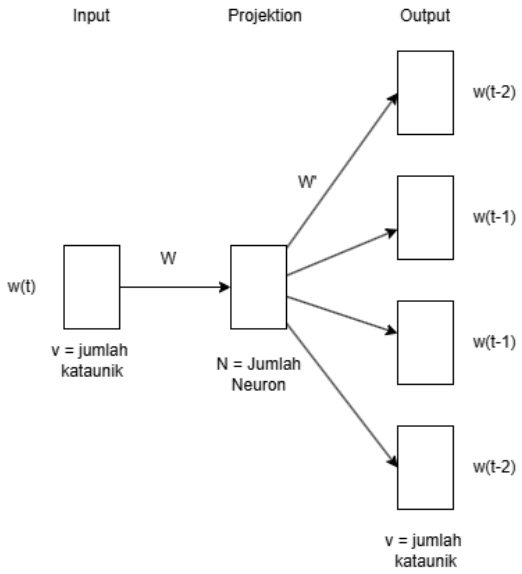
konteks kata berdasarkan satu kata target. Dengan kata lain, model ini mempelajari probabilitas kemunculan kata-kata konteks yang mungkin berada di sekitar kata target tertentu (Niasita et al., 2019).

Dalam penelitian ini, peneliti akan menggunakan pustaka *Gensim* untuk membangun model *Word2Vec*. Pustaka ini dipilih karena mampu menyederhanakan proses pelatihan model serta memberikan fleksibilitas dalam menyesuaikan berbagai parameter penting, seperti panjang vektor (*vector size*) dan ukuran jendela konteks (*window size*). Pada penelitian ini, panjang vektor yang digunakan ditetapkan sebesar 300, dengan ukuran jendela konteks sebesar 5.

Terdapat beberapa tahapan sebelum melakukan pelatihan model *Word2Vec* . Langkah pertama dalam proses ini adalah melakukan *one-hot encoding* terhadap setiap kata dalam kosakata. *One-hot encoding* merepresentasikan setiap kata dalam bentuk vektor biner berdimensi sama dengan ukuran kosakata (*vocabulary size*), di mana hanya satu elemen dalam vektor yang bernilai 1 (menunjukkan indeks kata tersebut), sementara elemen lainnya bernilai 0 (Aufa & Qoiriah, 2023).

Setelah proses *one-hot encoding* diterapkan, langkah selanjutnya adalah menentukan pasangan antara kata target dan kata konteks, yang ditentukan berdasarkan *window size* (jendela konteks). Kata konteks merupakan kata-kata yang berada di sekitar kata target, yaitu kata-kata sebelum dan sesudahnya dalam satu baris data. Dalam proses ini, setiap kata dalam baris data akan secara bergantian berperan sebagai kata target. Sementara itu, kata-kata yang berada dalam jangkauan *window* akan dianggap sebagai kata konteks. Pasangan-pasangan ini kemudian digunakan untuk memprediksi kata konteks berdasarkan kata target.

Arsitektur *skip-gram* terdiri dari tiga lapisan utama, yaitu *input layer*, *projection layer* (atau *hidden layer*), dan *output layer*, sebagaimana ditunjukkan pada Gambar 3.3 (Aufa & Qoiriah, 2023). Pada *input layer*, model menerima kata target yang telah direpresentasikan dalam bentuk one-hot encoding. Sementara itu, *output layer* bertugas untuk memprediksi kata-kata dalam konteks sekitar dari kata target tersebut.



Gambar 3. 4 Arsitektur *Skip-Gram*

Pelatihan model *Word2Vec* diawali dengan inisialisasi bobot secara acak pada *hidden layer* (W) dan *output layer* (W'). Bobot W menghubungkan *input layer* ke *hidden layer*, sedangkan bobot W' menghubungkan *hidden layer* ke *output layer*. Ukuran masing-masing bobot adalah W = v × n dan W' = n × v, di mana v merupakan jumlah kosakata, dan n adalah dimensi vektor (*vector size*).

Setelah bobot diinisialisasi, dilakukan proses *forward pass* untuk menghitung aliran data dari *input layer* ke *hidden layer*, lalu ke *output layer*. Proses ini bertujuan untuk menghasilkan prediksi terhadap kata-kata konteks berdasarkan kata target.

Adapun perhitungan dilakukan melalui beberapa tahap, dimulai dengan menghitung nilai pada *hidden layer* sesuai dengan persamaan 3.1 berikut:

$$h = W^T . x \tag{3.1}$$

Keterangan:
h = vektor keluaran pada hidden layer.
W^T = matriks bobot *hidden layer* yang ditranspos.
x = vektor input, yang direpresentasikan dalam bentuk *one-hot encoding*.

Selanjutnya, dilakukan perhitungan keluaran dari *hidden layer* menuju *output layer* dengan menggunakan persamaan 3.2 berikut:

$$U = W'^T \cdot h \tag{3.2}$$

Keterangan:
U = *output* layer
W'^T = matriks bobot *output layer* yang ditranspos.

Keluaran dari *output layer* kemudian diubah menjadi probabilitas dengan menggunakan fungsi aktivasi *Softmax*. Persamaan *softmax* adalah sebagai berikut:

$$\hat{y}_j = \frac{\exp(u_j)}{\sum_{j'=1}^{V} \exp(u_{j'})} \tag{3.3}$$

Keterangan:
$\hat{y}_j$ = Probabilitas baris j
$u_{j'}$ = *output* baris *j'*
V = jumlah kata unik

Setelah mendapatkan nilai probabilitas, langkah selanjutnya adalah menghitung selisih atau *error* antara probabilitas yang diprediksi oleh model dan nilai sebenarnya. Penghitungan ini dilakukan dengan menggunakan fungsi *cross-entropy loss*. Rumus perhitungan *error* dengan menggunakan *cross-entropy loss* adalah sebagai berikut:

$$e = -\sum_{j=1}^{v} y_j \log(\hat{y}_j) \quad (3.4)$$

Keterangan:
e = nilai *error*
y_j = Nilai sebenarnya dibaris j
$\hat{y}_j$ = Probabilitas baris j

Proses berikutnya adalah *backpropagation*, yaitu proses untuk memperbarui bobot-bobot model berdasarkan nilai *error* yang telah dihitung sebelumnya. Tahap pertama dalam *backpropagation* ini adalah menghitung gradien dari *loss* terhadap keluaran pada *output layer*.

$$\frac{\partial l}{\partial U} = y - \hat{y} \quad (3.5)$$

Keterangan:
$\frac{\partial l}{\partial U}$ = Gradien *output layer*
Y = Nilai *output* sebenarnya
$\hat{y}$ = Nilai probabilitas

Perubahan bobot antara *hidden layer* dan *output layer* dihitung menggunakan gradien dengan persamaan 3.6 berikut:

$$\frac{\partial l}{\partial W'} = h.\partial U^T \quad (3.6)$$

Keterangan:
$\frac{\partial l}{\partial W'}$ = Gradien bobot *output layer*
h = Nilai *hidden layer*
∂U^T = Nilai gradien *output layer* di *trapose*

Sedangkan perubahan bobot antara *input layer* dan *hidden layer* dihitung dengan:

$$\frac{\partial l}{\partial W} = x.(W'.\partial U)^T \quad (3.7)$$

Keterangan:
$\frac{\partial l}{\partial W}$ = Gradien bobot *output layer*

h = Nilai *hidden layer*
x = Nilai input
∂U^T = Nilai gradien *output layer*

Setelah gradien dihitung, bobot diperbarui dengan rumus:

$$W = W_{lama} - (\eta \cdot \frac{\partial l}{\partial W}) \quad (3.8)$$

$$W' = W'_{lama} - (\eta \cdot \frac{\partial l}{\partial W'}) \quad (3.9)$$

Keterangan:
η = learning rate
W = bobot antara input layer dan hidden layer.
W' = bobot antara hidden layer dan output layer.

Proses ini dilakukan dalam beberapa iterasi (*epoch*) hingga model mencapai konvergensi atau performa yang diinginkan. Dalam setiap iterasi, bobot yang ada (W) diubah berdasarkan nilai gradien yang dihitung untuk mengurangi kesalahan antara prediksi dan nilai yang sebenarnya. Nilai vektor nantinya didapatkan dari representasi bobot (W). Contoh hasil vektor yang diperoleh dapat dilihat pada Tabel 3.11.

Tabel 3.11 Contoh Output *Word2Vec*

Output Word2Vec
[[0.00000000e+00, 0.00000000e+00, 0.00000000e+00, . . . , 0.00000000e+00, 0.00000000e+00, 0.00000000e+00], [-6.97642616e-01, 1.87695541e-01, -5.58217462e-01, . . . , -5.96168169e-01, -7.00651705e-01, 8.19292757e-01], . . . , [7.01845204e-03, 2.63108760e-02, 4.40757518e-04, . . . , -1.40538616e-02, 1.32307298e-02, -3.67004704e-03], [9.90398135e-03, 2.78079417e-02, -2.46612029e-03, . . . , -1.15900813e-02, 1.27192587e-02, -2.83975503e-03]]

3.6 *Balancing Data*

Kelas yang tidak seimbang (*imbalance*) pada dataset dapat memengaruhi performa model secara signifikan. Ketidakseimbangan ini menyebabkan model

cenderung mengklasifikasikan data ke dalam kelas mayoritas, sehingga mengabaikan kelas minoritas. Jika jarak antara kelas minoritas dan mayoritas terlalu jauh, maka kelas mayoritas akan mendominasi dan menutupi informasi dari kelas minoritas. Akibatnya, keputusan yang dihasilkan oleh model menjadi bias (Sofyan & Prasetyo, 2021).

Untuk mengatasi ketidakseimbangan kelas, salah satu teknik yang dapat digunakan adalah *random oversampling*, yaitu dengan menambah jumlah sampel pada kelas minoritas dengan cara menggandakan data secara acak hingga jumlah datanya seimbang dengan kelas mayoritas (Nurhidayat & Dewi, 2023). Teknik ini hanya diterapkan pada data *training*, sedangkan data testing tetap dibiarkan dalam kondisi aslinya agar evaluasi model mencerminkan performa pada data nyata secara objektif.

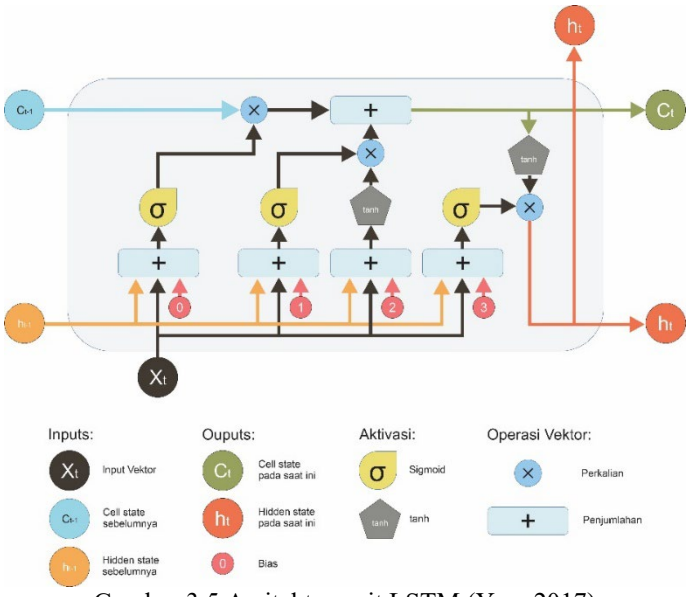
3.7 *Long Short Term Memory*

Long Short-Term Memory (LSTM) diperkenalkan oleh Sepp Hochreiter dan Jürgen Schmidhuber pada tahun 1997 sebagai solusi terhadap masalah yang dihadapi oleh *Recurrent Neural Network* (RNN) dalam mengelola data berurutan. RNN kovensional mengalami kesulitan dalam menangani ketergantungan jangka panjang pada data sekuensial, terutama karena masalah *vanishing gradient*, di mana gradien yang dihitung pada lapisan-lapisan terdalam menjadi sangat kecil, sehingga pembaruan bobot menjadi terbatas dan memperlambat proses konvergensi model (Pradana et al., 2023)

LSTM mengatasi keterbatasan tersebut dengan menambahkan fitur *cell state*. *Cell state* berfungsi sebagai jalur memori utama dalam jaringan, memungkinkan

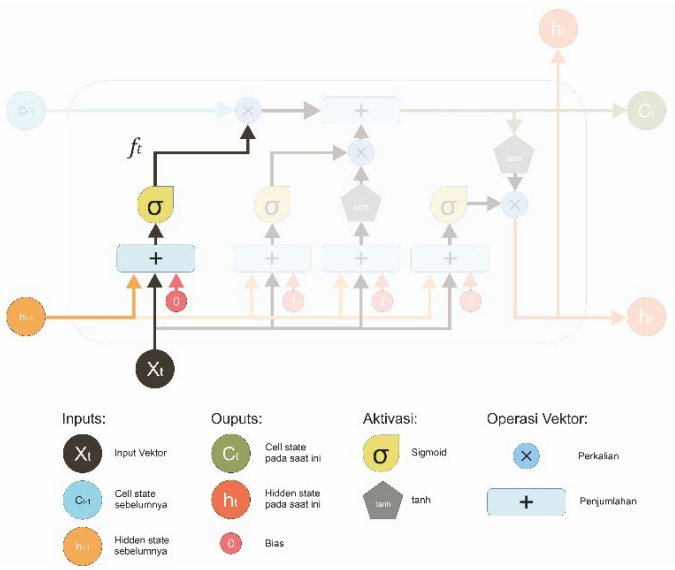
model untuk menyimpan informasi jangka panjang. Dengan adanya *cell state*, LSTM dapat mempertahankan informasi yang relevan lebih lama, bahkan dalam urutan data yang sangat panjang, sehingga efektif mengatasi masalah *vanishing gradient* yang sering dihadapi oleh RNN (Pradana et al., 2023).

LSTM (Long *Short-Term Memory*) dalam pengelolaan data berbentuk teks bekerja dengan memproses setiap kata secara berurutan dalam satu urutan data. Setiap kata diproses secara bertahap melalui unit-unit LSTM (Widayat, 2021). Dalam prosesnya unit LSTM menghasilkan *cell state* sebagai memori jangka panjang dan *hidden state* sebagai memori jangka pendek (Alhamdani et al., 2021). Kedua memori ini memungkinkan informasi mengalir dan dipertahankan dari satu unit ke unit lainnya. Aliran informasi dalam *cell state* dan *hidden state* diperbarui oleh gerbang (*gates*), yang bertanggung jawab untuk menentukan informasi yang perlu dilupakan, disimpan, atau diteruskan sebagai output (Pradana et al., 2023). Bentuk lebih rinci mengenai komponen-komponen dalam satu unit LSTM bisa dilihat pada Gambar 3.5.



Gambar 3.5 Arsitektur unit LSTM (Yan, 2017)

Dalam Gambar 3.5, dapat diketahui bahwa LSTM memiliki arsitektur yang kompleks. Arsitektur ini terdiri dari beberapa gerbang (*gates*) yang berfungsi untuk mengatur aliran informasi dalam jaringan. Gerbang tersebut meliputi *forget gate*, *input gate*, dan *output gate*. Setiap gerbang memiliki peran masing-masing dalam mengelola input yang diterima dan menentukan informasi mana yang akan dipertahankan atau dilupakan, serta bagaimana informasi tersebut diteruskan ke unit selanjutnya. Berikut adalah penjelasan mengenai bagaimana LSTM dalam mengelola informasi dalam unit LSTM:



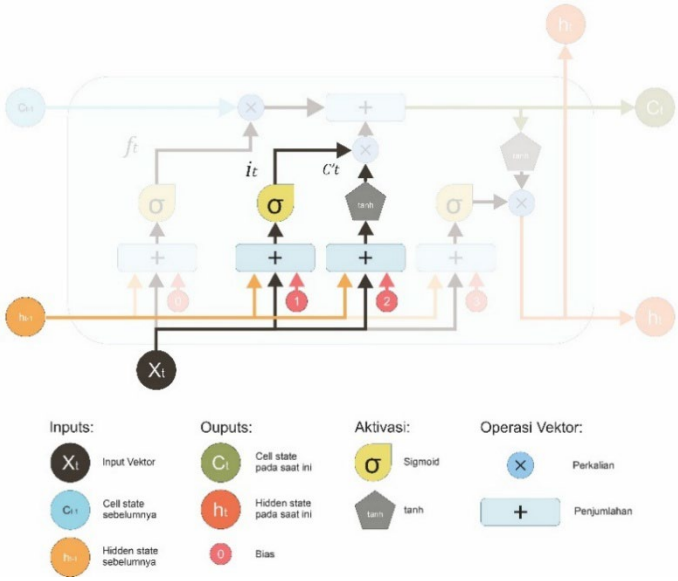
Gambar 3.6 *Forget gate* pada LSTM (Yan, 2017)

Tahap pertama dalam proses LSTM adalah menentukan informasi yang perlu dilupakan dari *cell state* sebelumnya, yang diatur oleh *forget gate* (Pradana et al., 2023). Gambar 3.6 menunjukkan proses penjumlahan antara *hidden state* sebelumnya, input saat ini, dan bias, setelah masing-masing dikalikan dengan bobotnya. *Hidden state* sebelumnya (h_{t-1}) dikalikan dengan bobot U_f , sedangkan input (x_t) dikalikan dengan bobot W_f.

Hasil penjumlahan tersebut kemudian diproses menggunakan fungsi aktivasi *sigmoid*, yang menghasilkan nilai antara 0 dan 1. Nilai ini menunjukkan seberapa besar informasi yang harus dipertahankan atau dilupakan dalam *cell state*. Nilai mendekati 0 berarti informasi dilupakan, sementara nilai mendekati 1 berarti informasi dipertahankan (Pradana et al., 2023). Persamaan untuk *forget gate* dapat dilihat pada persamaan 3.10:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{3.10}$$

Keterangan:
f_t = *forget gate* time step t
x_t = input pada *time step* t
W_f = bobot *forget gate*
U_f = bobot *hidden state* di *forget gate*
h_{t-1} = *hidden state* sebelumnya
σ = fungsi aktivasi sigmoid
b_f = bias *forget gate*



Gambar 3. 7 *Input gate* pada LSTM (Yan, 2017)

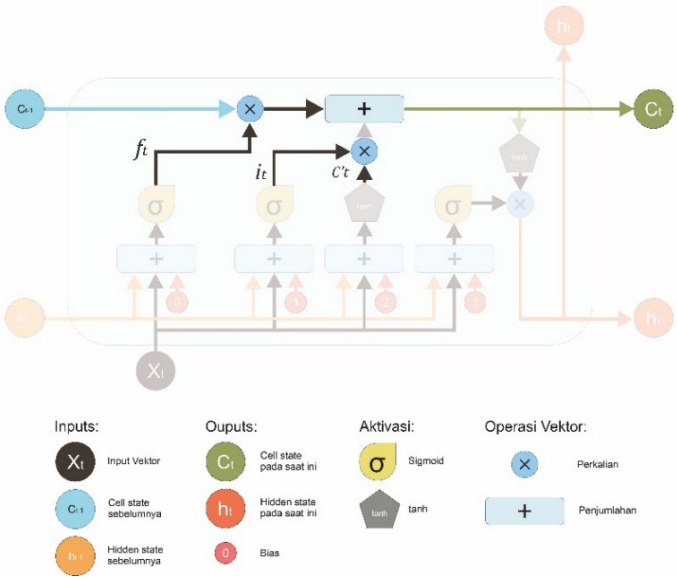
Langkah selanjutnya dalam proses LSTM adalah menentukan informasi yang akan disimpan ke dalam *cell state*. Proses ini dapat dilihat pada Gambar 3.7, yang menggambarkan dua jalur yang terlibat. Jalur pertama adalah menentukan *input*

gate dengan menggunakan fungsi *sigmoid*, yang berfungsi untuk menentukan seberapa banyak informasi yang akan disimpan. Jalur kedua adalah menentukan *candidate state* dengan fungsi *tanh* mengubah nilai *cell state* ke rentang antara -1 dan 1, yang berfungsi untuk menentukan informasi baru yang akan ditambahkan ke dalam *cell state* (Maharani et al., 2024). Persamaan matematisnya adalah:

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \quad (3.11)$$

$$\tilde{C}'_t = tanh(W_c x_t + U_c h_t - 1 + b_c) \quad (3.12)$$

Keterangan:
i_t = *input gate* time step t
$\tilde{C}'_t$ = *candidate state* time step t
x_t = input pada *time step* t
W_i = bobot *input gate*
W_c = bobot *candidate gate*
U_i = bobot *hidden state* di *input gate*
U_c = bobot *hidden state* di *candidate state*
h_{t-1} = *hidden state* sebelumnya
σ = fungsi aktivasi sigmoid



Gambar 3.8 *Update cell State* (Yan, 2017)

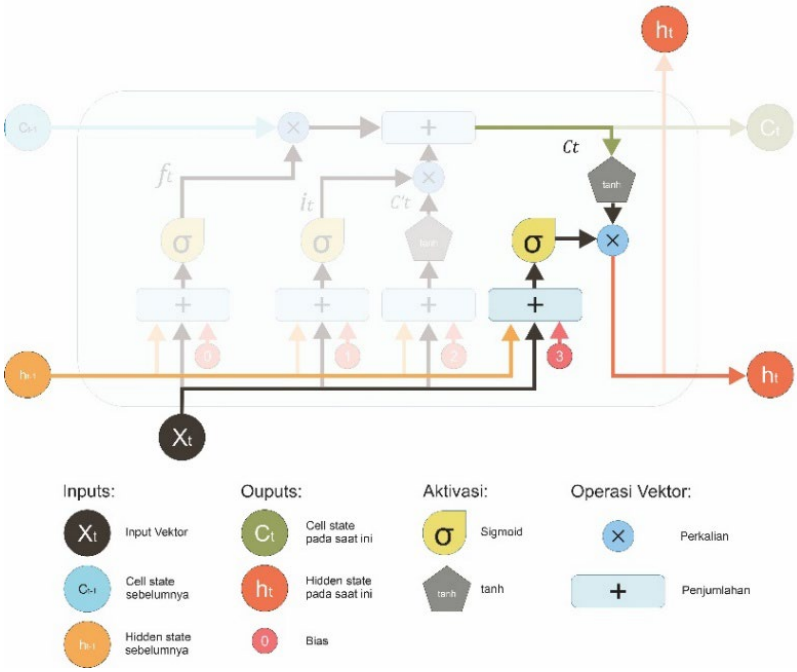
Gambar 3.8 merupakan proses pembaruan dari *cell state*. Informasi yang didapatkan dari *forget gate* dan *input gate* digabungkan untuk memperbarui *cell*

state. Pada proses ini, hasil keluaran dari *forget gate* dikalikan dengan *cell state* sebelumnya, yang menunjukkan informasi lama yang tetap dipertahankan. Sementara itu, hasil keluaran dari *input gate* dikalikan dengan kandidat nilai baru dari *cell state*, yang menunjukkan informasi baru yang akan ditambahkan.

Proses pembaruan *cell state* dapat dirumuskan sebagai berikut:

$$C_t = i_t \circ \tilde{C}'_t + f_t \circ C_{t-1} \tag{3.13}$$

Keterangan:
i_t = *input gate* time step t
$\tilde{C}'_t$ = *candidate state* time step t
f_t = *forget gate* time step t
$\tilde{C}_{t-1}$ = *cell state* sebelumnya



Gambar 3.9 *Output Gate* (Yan, 2017)

Langkah selanjutnya dalam proses *Long Short-Term Memory* (LSTM) adalah *output gate*, yang bertanggung jawab untuk menghasilkan *hidden state* sebagai keluaran. Pada Gambar 3.9, terlihat bagaimana *output gate* bekerja untuk memutuskan informasi mana yang akan diteruskan sebagai output dari unit LSTM.

Output gate ini menggunakan fungsi *sigmoid* untuk menghasilkan nilai kontinu antara 0 dan 1. Nilai ini menunjukkan seberapa banyak informasi dari *cell state* yang akan diteruskan ke *hidden state* pada *time step* saat ini. Semakin mendekati nilai 1, semakin banyak informasi yang diteruskan, sedangkan semakin mendekati nilai 0, semakin sedikit informasi yang diteruskan.

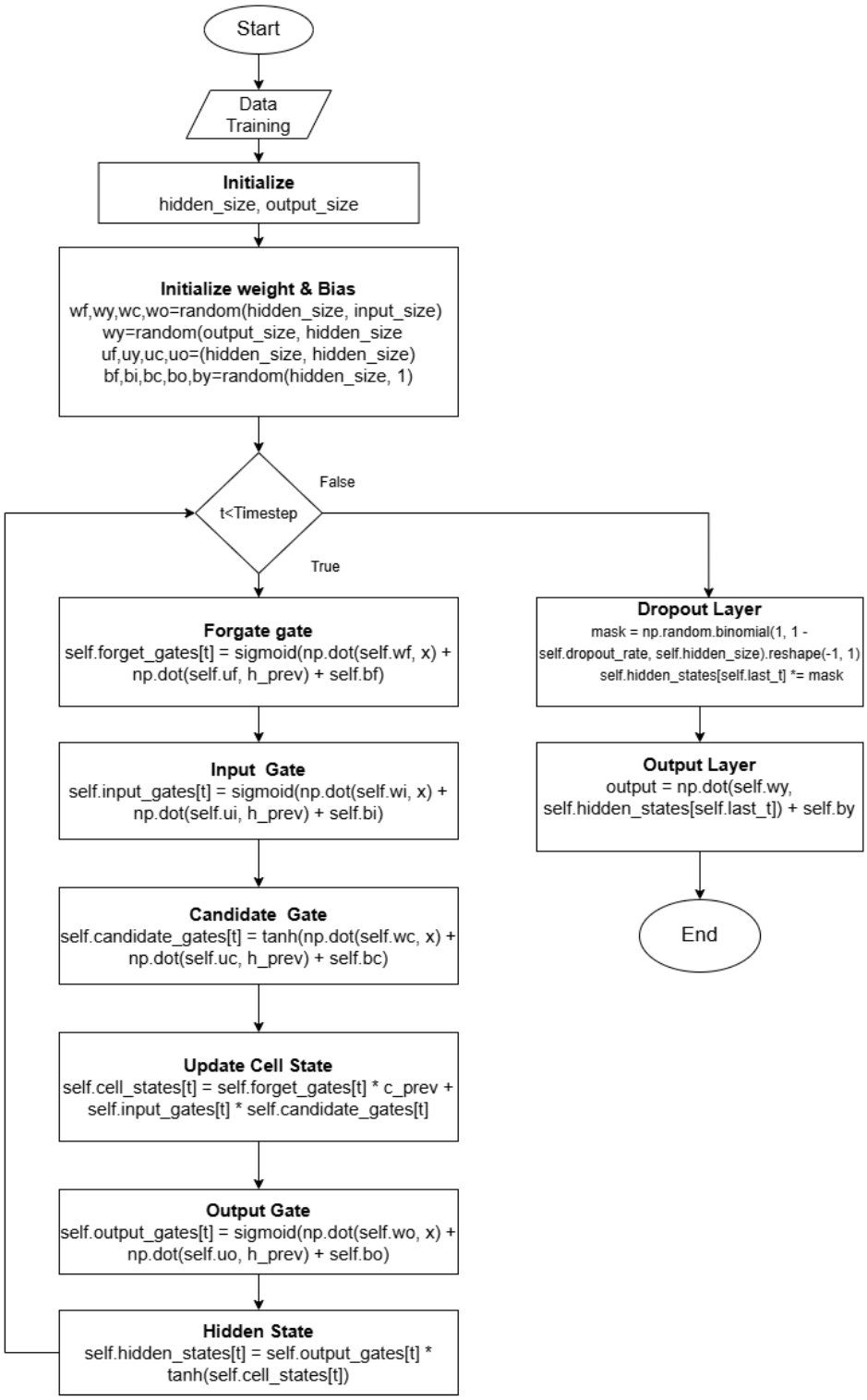
Setelah nilai dari output gate diperoleh, hasil ini kemudian dikalikan dengan fungsi *tanh* dari *cell state* saat ini (C_t) untuk mendapatkan nilai *hidden state*. Fungsi *tanh* mengubah nilai *cell state* menjadi rentang antara -1 dan 1,Persamaan sebagai berikut.

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{3.14}$$

$$h_t = o_t \circ \tanh(C_t) \tag{3.15}$$

Keterangan:
o_t = *output gate*
h_t = *hidden state*
x_t = input pada *time step t*
W_o = bobot *candidate gate*
U_o = bobot *hidden state* di *output gate*
b = bias
σ = fungsi aktifasi sigmoid

Setelah mengetahui rumus *feedforward* pada unit LSTM, untuk alur yang lebih jelas dapat dilihat melalui *flowchart* pada Gambar 3.10. *flowchart* ini menggambarkan alur kerja *feedforward* secara bertahap, mulai dari inisialisasi parameter hingga pengolahan input pada setiap langkah waktu. Melalui visualisasi tersebut, dilihat bagaimana masing-masing komponen seperti *forget gate*, *input gate*, *candidate gate*, dan *output gate* diolah secara sistematis untuk menghasilkan output yang merepresentasikan informasi dalam data sekuensial.



Gambar 3.10 *Flowchart* Proses *Feedforward*

Dari Gambar 3.10, proses dimulai dengan memasukkan dataset, kemudian dilanjutkan dengan inisialisasi ukuran *hidden size* dan *output size* yang digunakan untuk mengatur ukuran matriks bobot. Selanjutnya, dilakukan inisialisasi bobot secara acak, dengan ukuran matriks untuk bobot input (W = *hidden size* × *output size*) dan untuk bobot *hidden state* (U = *hidden size* × *hidden size*) serta bias (b = *hidden size* × 1). Setelah itu, data dimasukkan ke dalam LSTM *layer*, yang kemudian diproses secara berulang sesuai dengan jumlah unit LSTM. Di dalam LSTM *layer*, pertama-tama dihitung *forget gate* dengan rumus (3.10), *input gate* (3.11), dan *candidate gate* (3.12). Setelah itu, *cell state* diperbarui dengan menggunakan rumus (3.13). Proses ini berlanjut dengan perhitungan *output gate* (3.14), yang digunakan untuk menentukan seberapa banyak informasi dari *cell state* yang akan dipindahkan ke *hidden state* (3.14). Proses ini akan dilakukan hingga mencapai *timestep* terakhir. Setelah itu, data diteruskan ke *output layer*, di mana *neuron* akan dinonaktifkan secara acak dengan mengubahnya menjadi nilai 0. Selanjutnya, proses dilanjutkan dengan *output layer* untuk menghitung probabilitas.

Setelah menyelesaikan perhitungan pada *time-step* terakhir dan *output layer*, langkah selanjutnya adalah melakukan *backpropagation*. Proses *backpropagation* bertujuan untuk menghitung gradien bobot pada LSTM dengan mengikuti aturan rantai (*chain rule*). *Backpropagation* ini menyimpan turunan parsial yang diperlukan untuk menghitung gradien bobot, yang kemudian digunakan untuk memperbarui bobot yang lama (Qur'atul et al., 2024). Adapun persaman dari *backpropagation* sebagai berikut:

$$\partial o_t = \partial h_t . tanh(C_t) . O_t . (1 - O_t) \quad (3.16)$$

$$\partial \mathrm{C}_t = \partial h_t . O_t . (1 - tanh(C_t)^2) + \partial C_{t+1} . f_{t+1} \quad (3.17)$$

$$\partial \mathrm{C'}_t = \partial C_t . i_t . (1 - C'^2_t) \quad (3.18)$$

$$\partial i_t = \partial C_t . \partial C'_t . i_t . (1 - i_t) \quad (3.19)$$

$$\partial f_t = \partial C_t . C_{t-1} . f_t . (1 - f_t) \quad (3.20)$$

$$\partial h_{t-1} = \partial U^T . \partial gates \quad (3.21)$$

Keterangan:
∂o_t = Turunan *output gate*
$\partial \mathrm{C}_t$ = Turunan *cell state*
$\partial \mathrm{C'}_t$ = Turunan *candidate cell state*
∂i_t = Turunan *input gate*
∂f_t = Turunan *forget gate*
∂h_{t-1} = Turunan *hidden state*

Setelah proses backpropagation selesai, langkah selanjutnya adalah menghitung gradien bobot dan bias untuk memperbarui parameter dalam jaringan. Gradien terhadap bobot input (W), bobot *hidden state* (U), dan bias (b) dihitung menggunakan persamaan berikut:

$$\partial W_{gates} = \sum_{t=1}^{T} \partial gates_t . x_t \quad (3.22)$$

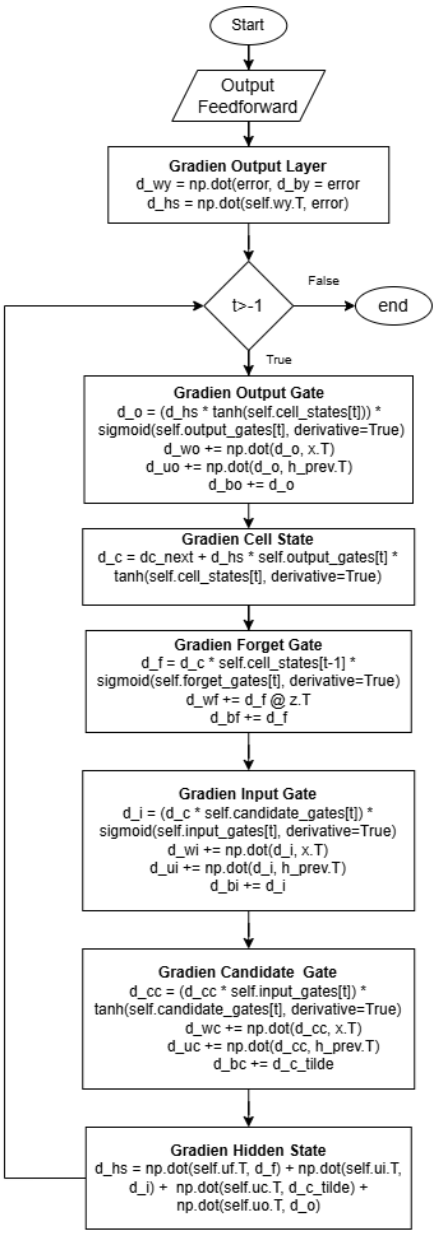
$$\partial U_{gates} = \sum_{t=1}^{T} \partial gates_t . h_{t-1} \quad (3.23)$$

$$\partial b_{gates} = \sum_{t=1}^{T} \partial gates_t \quad (3.24)$$

Keterangan:
∂W_{gates} = Gradien bobot input
∂U_{gates} = Gradien terhadap bobot hidden state
∂b_{gates} = Gradien terhadap bias

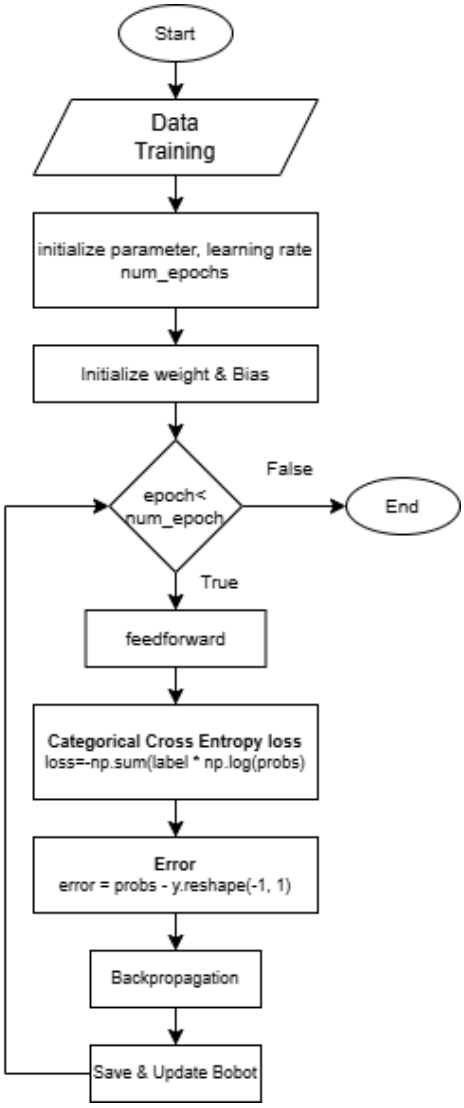
Persamaan 3.22 digunakan untuk menghitung total gradien terhadap bobot input pada setiap gerbang. Gradien ini diperoleh dengan menjumlahkan hasil

perkalian antara turunan dari setiap gerbang terhadap input pada waktu t. Selanjutnya, Persamaan 3.23 menghitung gradien terhadap *hidden state* untuk setiap gerbang, dengan cara menjumlahkan hasil perkalian antara turunan gerbang pada waktu t dan *hidden state* pada waktu sebelumnya (t-1). Terakhir, Persamaan 3.24 digunakan untuk menghitung gradien terhadap bias pada setiap gerbang, yang dihitung dengan menjumlahkan seluruh gradien yang telah dihitung sebelumnya.



Gambar 3.11 *Flowchart* Proses *Backpropagation*

Dari Gambar 3.11, langkah-langkah perhitungan gradien selama proses backpropagation pada jaringan LSTM dimulai dengan menghitung gradien pada *output layer* (d_wy) menggunakan *error* yang diperoleh dari output *feedforward*. Setelah itu, gradien untuk *output gate* (d_o) dihitung menggunakan rumus (3.16), diikuti dengan pembaruan bobot (d_w) menggunakan rumus (3.22) dan pembaruan bobot *hidden state* (d_u) menggunakan rumus (3.23), serta pembaruan bias (d_b) menggunakan rumus (3.24). Selanjutnya, gradien untuk *cell state* (d_c) dihitung dengan rumus (3.17). Proses dilanjutkan dengan menghitung gradien untuk *forget gate* (d_f) menggunakan rumus (3.20), diikuti dengan pembaruan bobot dan bias untuk *forget gate* dengan menggunakan rumus (3.22), (3.23), dan (3.24) untuk pembaruan bobot dan bias *forget gate* (d_wf, d_uf, dan d_bf). Kemudian, gradien untuk *input gate* (d_i) dihitung, dan pembaruan bobot serta bias dilakukan dengan menggunakan rumus yang sama, yaitu (3.22), (3.23), dan (3.24) untuk *input gate* (d_wi, d_ui dan d_bi). Selanjutnya, gradien untuk candidate gate (d_cc) dihitung menggunakan rumus (3.18), diikuti dengan pembaruan bobot dan bias untuk candidate gate (d_wc, d_uc, dan d_bc), yang juga menggunakan rumus (3.22), (3.23), dan (3.24) untuk pembaruan bobot dan bias candidate gate. Terakhir, gradien untuk hidden state (d_hs) dihitung dengan menggabungkan semua gradien yang telah dihitung sebelumnya, seperti gradien dari forget gate, input gate, output gate, dan candidate gate, sesuai dengan persamaan (3.25).

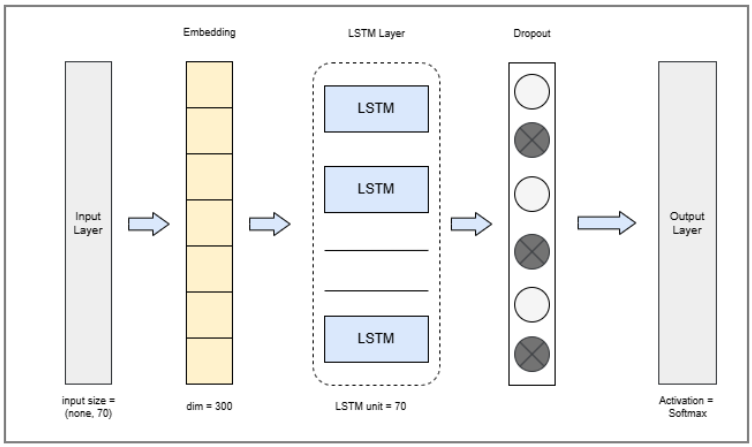


Gambar 3.12 *Flowchart* proses *Training*

Untuk proses *training* secara keseluruhan, bisa dilihat dalam Gambar 3.12 yang menunjukkan tahapan-tahapan utama dalam pelatihan model. Proses dimulai dari inisialisasi parameter, *learning rate*, serta bobot dan bias. Selanjutnya, dilakukan proses feedforward diikuti dengan perhitungan nilai *loss* menggunakan fungsi *Categorical Cross Entropy*. Setelah itu, kesalahan (*error*) dihitung dari selisih antara prediksi dan label sebenarnya. Proses backpropagation kemudian dijalankan untuk menyebarkan *error* ke belakang dan menghitung gradien. Bobot

diperbarui berdasarkan hasil perhitungan tersebut, dan proses ini diulang hingga mencapai jumlah *epoch* yang ditentukan.

3.8 Arsitektur Model



Gambar 3.13 Arsitektur Model

Penelitian ini menggunakan arsitektur model yang terdiri dari lima lapisan utama, yaitu *input layer*, *embedding layer*, LSTM layer, *dropout layer*, dan *output layer*, seperti yang ada dalam Gambar 3.4. Masing-masing lapisan memiliki peran tersendiri dalam memproses data dan menghasilkan klasifikasi sentimen.

Input layer berfungsi untuk menerima data teks yang telah melalui tahap preprocessing, pengindeksan, dan padding dengan panjang 70. Selanjutnya, embedding layer mengonversi kata-kata pada teks menjadi vektor yang berdasarkan indeks yang nilai vektornya didapatkan dari model *Word2Vec*. Kemudian masuk kedalam LSTM layer yang terdiri dari 70 unit LSTM, sesuai dengan panjang padding.

Setelah itu, *dropout layer* diterapkan sebagai teknik regulasi untuk mencegah *overfitting* pada model. Dengan menonaktifkan sejumlah *neuron* secara acak,

dropout memastikan bahwa model tidak hanya mengandalkan pola yang terdapat pada unit-unit tertentu, melainkan dapat belajar untuk mengenali pola yang lebih umum yang berlaku pada data yang belum pernah dilihat sebelumnya (Verianto, 2024).

Terakhir, *output layer* mengonversi hasil dari LSTM menjadi vektor dengan jumlah elemen yang sesuai dengan jumlah kelas sentimen (positif, negatif, atau netral) kemudian dihitung probabilitas setiap kelas menggunakan fungsi aktivasi *softmax*.

3.9 Evaluasi Model

Evaluasi model ini bertujuan untuk menilai kualitas dan kinerja model dalam mengklasifikasikan sentimen secara keseluruhan. Pada penelitian ini, digunakan *confusion matrix* sebagai alat evaluasi. *Confusion matrix* memetakan hasil prediksi model terhadap nilai sebenarnya, yang kemudian digunakan untuk menghitung berbagai metrik evaluasi, yaitu *precision*, *recall*, *F1 score*, dan akurasi.

Recall mengukur seberapa baik model dapat memprediksi kelas positif dari total kelas positif tersebut. Sedangkan *precision* mengukur seberapa baik model untuk memprediksi kelas positif yang dari total yang diprediksi sebagai kelas positif. *F1 score* digunakan untuk memberikan gambaran menyeluruh mengenai keseimbangan antara *recall* dan *precision*, sehingga dapat memastikan kinerja model tidak hanya fokus pada salah satu metrik. Terakhir, akurasi menunjukkan proporsi keseluruhan prediksi yang benar dibandingkan dengan total prediksi yang dilakukan oleh model. Dengan menggunakan metrik-metrik ini, kinerja model dapat dianalisis secara mendalam untuk menentukan apakah model telah mencapai

performa yang memadai atau masih memerlukan perbaikan lebih lanjut (Lisanthoni et al., 2024). Adapun persamaan matematis untuk evaluasi model dijelaskan sebagai berikut (Mola et al., 2024)

$$\text{Akurasi} = \frac{TP + TN + TN_t}{TP + FP + TN + FN + FN_t + TN_t} \quad (3.25)$$

$$Precision = \frac{TP}{TP + FP} \quad (3.26)$$

$$Recall = \frac{TP}{TP + FN + FN_t} \quad (3.27)$$

$$F1\ Score = 2 \times \frac{precision \times recall}{precision + recall} \quad (3.28)$$

3.10 Skenario Pengujian

Dalam penelitian ini, dilakukan serangkaian uji coba untuk menentukan konfigurasi hyperparameter yang paling optimal dalam membangun model. Uji coba difokuskan pada tiga *hyperparameter*, yaitu jumlah *neuron* pada LSTM, tingkat *dropout*, dan *learning rate*. Masing-masing *hyperparameter* memiliki peran yang berbeda dalam memengaruhi kinerja model. Jumlah *Neuron* pada LSTM berfungsi untuk menyimpan dan memproses informasi dari data yang diberikan, karena *neuron* merupakan komponen utama dalam struktur jaringan LSTM. *Dropout* berfungsi sebagai teknik regulasi untuk mencegah *overfitting*, sehingga pemilihan nilai *dropout* yang tepat sangat penting agar model tidak terlalu mengandalkan data latih tertentu (Verianto, 2024). Sementara itu, *learning rate* berperan dalam mengatur besar pembaruan bobot selama pelatihan (Saputra, 2020).

Kombinasi konfigurasi *hyperparameter* yang diuji dalam penelitian ini dapat dilihat pada Tabel 3.12.

Tabel 3. 12 *Hyperparameter*

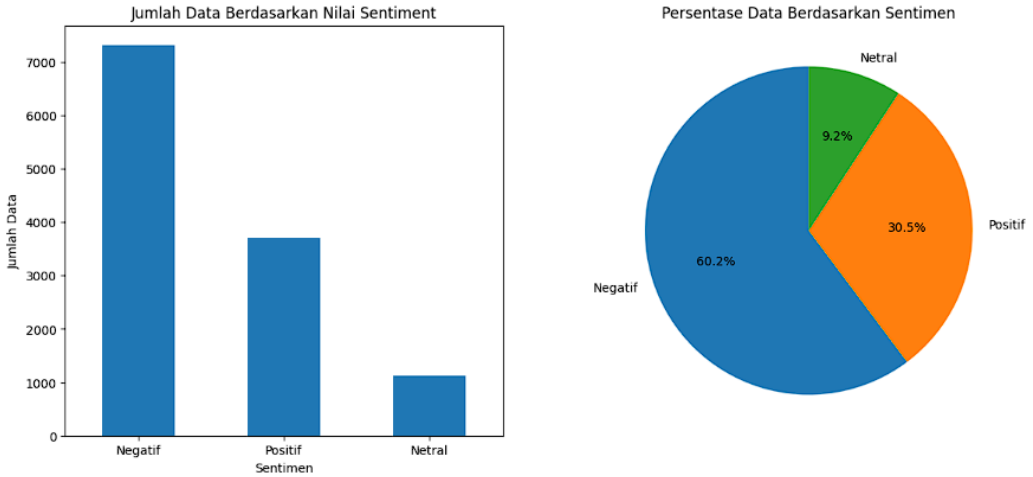
NO	Neuron	Learning Rate	Drop Out
1	32	0.2	0.01
2	32	0.3	0.01
3	32	0.5	0.01
4	32	0.2	0.001
5	32	0.3	0.001
6	32	0.5	0.001
7	32	0.2	0.0001
8	32	0.3	0.0001
9	32	0.5	0.0001
10	64	0.2	0.01
11	64	0.3	0.01
12	64	0.5	0.01
13	64	0.2	0.001
14	64	0.3	0.001
15	64	0.5	0.001
16	64	0.2	0.0001
17	64	0.3	0.0001
18	64	0.5	0.0001
19	128	0.2	0.01
20	128	0.3	0.01
21	128	0.5	0.01
22	128	0.2	0.001
23	128	0.3	0.001
24	128	0.5	0.001
25	128	0.2	0.0001
26	128	0.3	0.0001
27	128	0.5	0.0001

BAB IV

HASIL DAN PEMBAHASAN

4.1 Pengumpulan Data

Dataset pada penelitian ini diperoleh dari penelitian yang telah dilakukan oleh (Mola et al., 2024). Seperti yang telah dijelaskan pada bab sebelumnya, dataset ini dari tiga kategori sentimen, yaitu positif, negatif, dan netral, dengan total sebanyak 12.141 data ulasan. Jumlah data negatif berjumlah 7314, kemudian data netral 1120, dan data positif berjumlah 3707. Visulisasi perbandingan dataset ulasan positif, negatif dan netral dapat dilihat pada Gambar 4.1. Dari Gambar 4.1 dapat diketahui bahwa data negatif sebanyak 60,2% kemudian netral 9,2% dan positif 30.5%.



Gambar 4.1 Grafik Perbandingan Kelas

4.2 *Preprocessing*

Setelah proses pengumpulan data, proses berikutnya adalah *preprocessing*. *Preprocessing* bertujuan untuk membersihkan data dari elemen-elemen yang tidak relevan serta meningkatkan kualitas data agar sesuai dengan kebutuhan pemodelan. *Preprocessing* dilakukan melalui beberapa tahapan, yaitu *case folding*, *clean text*, normalisasi, tokenisasi, *stemming*, dan *stopword removal*. Hasil dari tahapan *preprocessing* ini dapat dilihat pada Tabel 4.1.

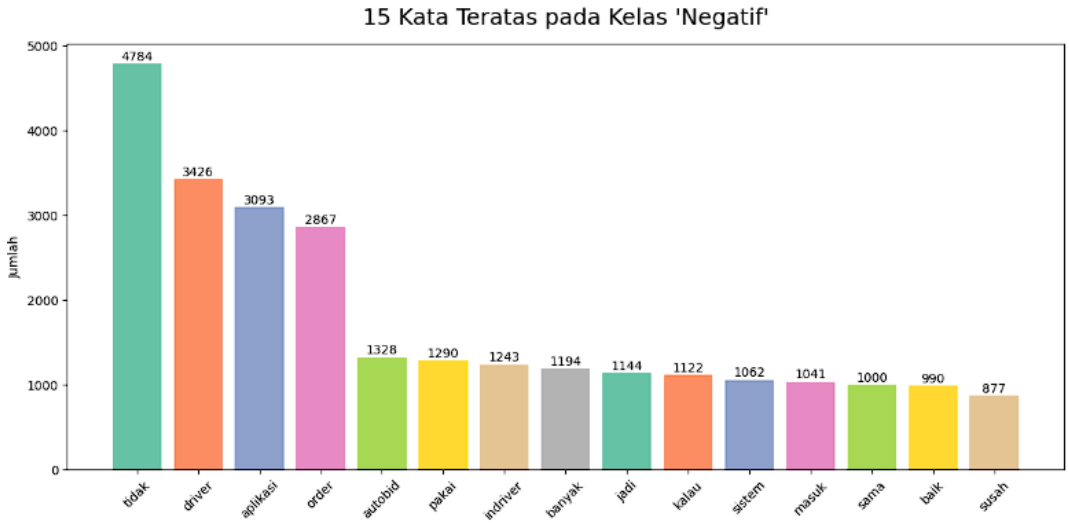
Tabel 4.1 Hasil *Preprocessing*

Sebelum Preprocessing	Setelah Preprocessing
Indriver halo Admin,Yth pihak indriver, ko Argo nya makin lama makin jauh makin murah knp Tega sekali halo admin murah harga Argonya	['indriver', 'halo', 'admin', 'yth', 'pihak', 'indriver', 'ko', 'argo', 'makin', 'lama', 'makin', 'jauh', 'makin', 'murah', 'tega', 'sekali', 'halo', 'admin', 'murah', 'harga', 'argonya']
sangat mudah di pahami dan yang lebih enak nya lagi bisa milih driver berikut ongkos nya	['sangat', 'mudah', 'paham', 'lebih', 'enak', 'milih', 'driver', 'ikut', 'ongkos']
Harusnya di buat kan auto bid supaya tidak terlalu berbahaya menerima orderan sambil berkendara ini malah di hilangkan auto bid nya	['buat', 'kan', 'auto', 'bid', 'tidak', 'terlalu', 'bahaya', 'terima', 'order', 'kendara', 'bahkan', 'hilang', 'auto', 'bid']
aplikasi perlu autobid, km perlu bawa uang pulang kerumah buat keluarga	['aplikasi', 'perlu', 'autobid', 'kamu', 'perlu', 'bawa', 'uang', 'pulang', 'rumah', 'buat', 'keluarga']
Tolong di berikan system yang adil untuk driver, padahal jarak 100-200 meter dari lokasi saya, tapi saya tidak mendapatkan tawaran lelang lelang sama sekali, dan bukan hanya satu atau dua kali, ini sangat sering terjadi	['ikan', 'system', 'adil', 'driver', 'padahal', 'jarak', 'meter', 'lokasi', 'tidak', 'tawar', 'lelang', 'lelang', 'sama', 'sekali', 'bukan', 'satu', 'kali', 'sangat', 'sering', 'jadi']
....	....
Di tunggu autobid nya segera mungkin	['tunggu', 'autobid', 'segera', 'mungkin']
bagus	['bagus']
Sangat membantu Semoga aplikasi nya bisa lebih baik lagi	['sangat', 'bantu', 'moga', 'aplikasi', 'lebih', 'baik']
Tolong dong, auto bit nya di aktifkan lagi dari pada memakai apk tambahan seperti Botering dan netguardian yg mungkin merigukan driver lain	['dong', 'auto', 'bit', 'aktif', 'pakai', 'aplikasi', 'tambah', 'botering', 'netguardian', 'mungkin', 'merigukan', 'driver']
Pihak aplikator tidak ada tindakan soal driver yg memakai aplikasi tambahan	['pihak', 'aplikator', 'tidak', 'tindak', 'soal', 'driver', 'pakai', 'aplikasi', 'tambah']

4.3 Visulisasi Data

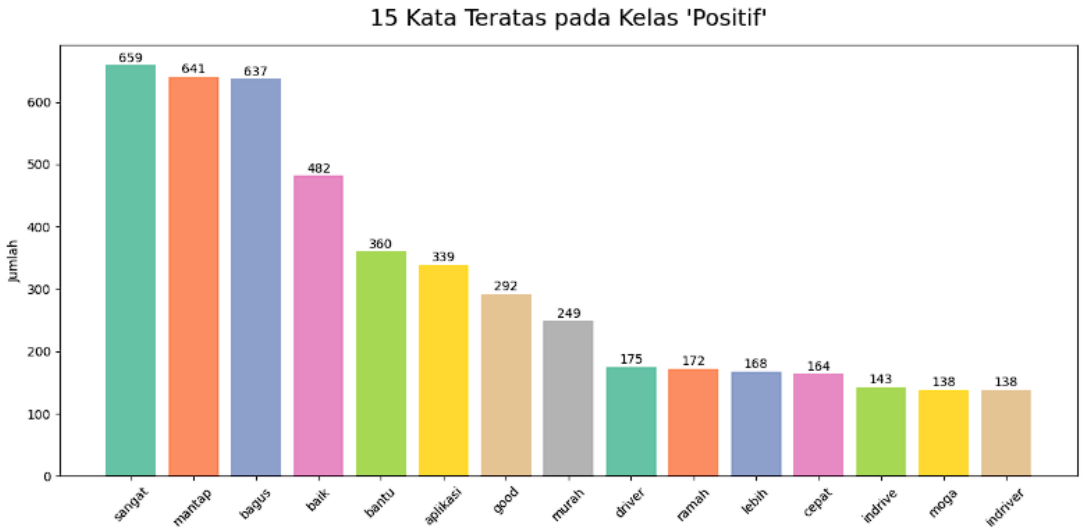
Visualisasi data dalam penelitian ini digunakan untuk menampilkan kata-kata yang paling sering muncul dalam ulasan pengguna berdasarkan masing-masing

kelas sentimen, yaitu positif, negatif, dan netral. Visualisasi ini membantu mengidentifikasi fokus atau topik utama yang sering dibahas oleh pengguna dalam setiap kategori sentimen.



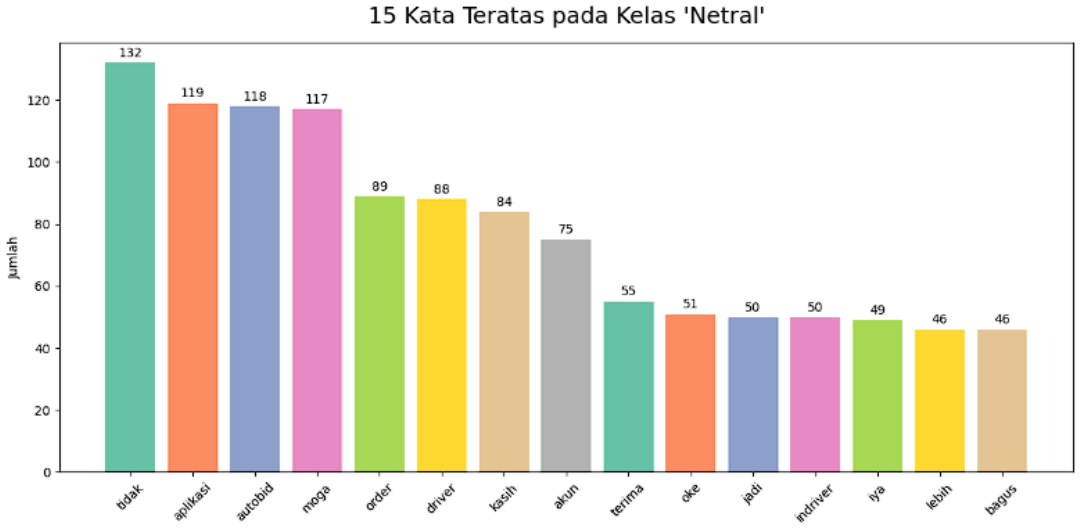
Gambar 4.2 15 Kata Teratas di Kelas Negatif

Gambar 4.2 menampilkan visualisasi kata-kata yang paling sering muncul dalam ulasan dengan sentimen negatif. Kata-kata seperti "tidak", "driver", "aplikasi", dan "order" mendominasi. Hal ini menunjukkan bahwa ulasan dengan sentimen negatif umumnya berkaitan dengan masalah pada layanan driver, ketidakpuasan terhadap sistem aplikasi, dan kendala dalam proses pemesanan.



Gambar 4.3 Kata Teratas di Kelas Posistif

Gambar 4.3 menampilkan visualisasi kata-kata yang paling sering muncul dalam ulasan dengan sentimen positif. Kata-kata seperti "sangat", "mantap", "bagus", dan "baik" sering ditemukan, mencerminkan apresiasi pengguna terhadap kinerja aplikasi, baik dari sisi pelayanan maupun fungsionalitasnya.



Gambar 4.4 Kata Teratas di Kelas Netral

Gambar 4.4 menunjukkan visualisasi kata-kata yang paling sering muncul dalam ulasan dengan sentimen netral. Kata-kata seperti "aplikasi" dan "autobid"

banyak dibahas, mengindikasikan bahwa pengguna fokus pada fitur dan fungsionalitas aplikasi secara umum. Kata "tidak" juga cukup dominan, yang dapat menunjukkan adanya ketidakmampuan atau keterbatasan dalam menggunakan layanan atau fitur tertentu. Selain itu, kemunculan kata "moga" mencerminkan harapan pengguna terhadap perbaikan aplikasi di masa mendatang, sementara kata "oke" menunjukkan bentuk penerimaan yang netral atau tidak terlalu emosional.

4.4 *Splitting Data*

Proses ini melibatkan pembagian dataset menjadi dua bagian, yaitu data pelatihan (*training*) dan data pengujian (*testing*), dengan sebesar 80% untuk pelatihan dan 20% untuk pengujian. Rincian jumlah data berdasarkan label sentimen setelah pembagian dataset bisa dilihat pada Tabel 4.2.

Tabel 4.2 *Splitting Data*

Data	Negatif	Netral	Positif
Data *training*	5853	913	1946
Data *testing*	1461	207	761
Total	7320	1120	3707

4.5 Ekstraksi Fitur

Dalam penelitian ini, ekstraksi fiturnya menggunakan *Word2Vec*, yang bertujuan untuk membentuk representasi vektor kata (*word embeddings*). Sesuai dengan penjelasan pada bab sebelumnya, pendekatan yang digunakan adalah *Skip-Gram*, yang memprediksi konteks di sekitar kata target.

Model *Word2Vec* dilatih menggunakan pustaka *Gensim* dengan panjang vektor sebesar 300, ukuran *window* sebesar 5, dan jumlah kemunculan minimal kata yang dipertimbangkan dalam pelatihan model adalah 1. Parameter-parameter

tersebut serta proses pembuatan model dapat dilihat pada *source code* dalam Tabel 4.3.

Tabel 4.3 *Source code word2Vec*

```
size_embedding = 300
windows = 5
min_count = 1
maxlen = 70

# Ubah data teks menjadi list of list
text_train_splited = text_train.tolist()

# Melatih model Word2Vec dengan algoritma Skip-Gram (sg=1)
w2v_model = gensim.models.Word2Vec(
    sentences=text_train_splited,
    vector_size=size_embedding,
    window=windows,
    sg=1,
    min_count=min_count,
    epochs=100
)
```

Karena model LSTM membutuhkan input dalam bentuk vektor numerik, maka representasi kata dari model *Word2Vec* perlu dikonversi ke dalam bentuk matriks bobot (*weight matrix*). Pembuatan fungsi untuk mengubah *Word2Vec* ke matriks dapat dilihat pada *source code* dalam Tabel 4.4.

Tabel 4.4 *Source Code* Konversi Model *word2Vec* ke *Array*

```
def w2v_to_keras_weights(model, vocab, size_embedding):
    vocab_size = len(vocab) + 1
    weights = np.zeros((vocab_size, size_embedding))

    for word, index in vocab.items():
        if word in model.wv:
            weights[index] = model.wv[word]
        else:
            weights[index] = np.random.normal(size=size_embedding)

    return weights

# Membuat matriks embedding dari model Word2Vec dan word_index
embedding_vectors = w2v_to_keras_weights(w2v_model, word_index, size_embedding)
```

Hasil dari konversi vektor ke matriks dapat dilihat pada tabel 4.5. Matrik *embedding* tersebut merupakan hasil konversi dari model *Word2Vec* ke dalam bentuk matriks dua dimensi. Setiap baris pada matriks mewakili vektor suatu kata yang diatur berdasarkan indeks.

Tabel 4.5 Hasil konversi vektor kata ke dalam bentuk matriks

```
array([
   [ 0.0,  0.0,  0.0, ...,  0.0,  0.0,  0.0 ],
   [-1.77120505,  0.7186979, -0.57442347, ...,  0.5656422, -0.22926401, -0.1098892 ],
   [ 0.25527677,  0.25034735,  0.16453634, ..., -0.04347499,  0.14092743, -0.29456776 ],
   ...,
   [ 0.09317119,  0.0810328,  0.06643042, ..., -0.21568681,  0.1780656,  0.12611993 ],
   [-0.04651726,  0.10947423, -0.14517167, ..., -0.00403241,  0.03469072, -0.16027783 ],
   [ 0.02174663,  0.08741777,  0.12025511, ...,  0.09670959,  0.04682323, -0.11266061 ]
])
```

Setelah model *Word2Vec* dilatih, Data teks dikonversi ke dalam bentuk numerik melalui proses pengindeksan menggunakan *Tokenizer*, kemudian dilakukan proses *padding* agar seluruh data memiliki panjang yang seragam sesuai kebutuhan input model. Untuk *source code* ada di tabel 4.6 dan hasil padding di tabel 4.7.

Tabel 4.6 Source code indeks dan padding

```
# Buat tokenizer dan handle kata tidak dikenal dengan token <OOV>
tokenizer = Tokenizer(oov_token="<OOV>")
tokenizer.fit_on_texts(text_train_splited)

# Ubah kalimat ke dalam urutan angka (token)
text_train_tok = tokenizer.texts_to_sequences(text_train_splited)

# Ambil indeks kata
word_index = tokenizer.word_index
print('Ukuran kosakata:', len(word_index))

# Padding
text_train_tok_pad = pad_sequences(text_train_tok, maxlen=maxlen)
```

Tabel 4.7 Hasil *Padding*

```
[[   0,    0,    0, ...,  225,   17,    8 ],
 [   0,    0,    0, ...,  134,  748,  375 ],
 [   0,    0,    0, ...,   16,  785,  274 ],
 ...,
 [   0,    0,    0, ...,    4,   47,  790 ],
 [   0,    0,    0, ...,    0,    0, 1881 ],
 [   0,    0,    0, ...,    0,    0,  457 ]]
```

Selain teks ulasan, label kelas pada data juga perlu diubah ke dalam bentuk numerik karena model tidak dapat memproses data dalam format teks secara langsung. Oleh karena itu, label sentimen harus dikonversi ke dalam bentuk numerik menggunakan metode *one-hot encoding*. Hasil dari proses ini ditampilkan pada tabel 4.8.

Tabel 4.8 Hasil *One-Hot Encoding*

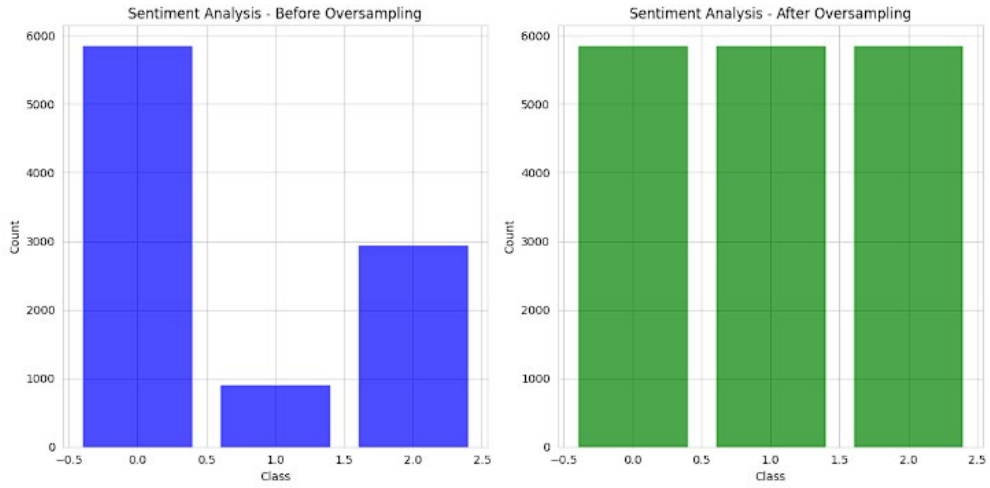
```
[[1., 0., 0.],
 [1., 0., 0.],
 [1., 0., 0.],
       ...,
 [0., 0., 1.],
 [0., 0., 1.],
 [0., 0., 1.]]
```

4.6 *Balancing Data*

Untuk mengatasi masalah ketidakseimbangan data dalam penelitian ini, digunakan metode *random oversampling*. Proses *oversampling* dilakukan dengan memanfaatkan fungsi *RandomOverSampler* dari *library imblearn*. *Source code* penerapan metode berada di dalam Tabel 4.9, sedangkan hasil setelah dilakukan *oversampling* bisa dilihat pada Gambar 4.5. Kelas-kelas minoritas ditingkatkan jumlahnya hingga setara dengan kelas mayoritas, yaitu kelas negatif, yang masing-masing mencapai 5.835 data.

Tabel 4.9 *Source Code Random Oversampling*

```
oversampler = RandomOverSampler(sampling_strategy='auto', random_state=42)
X_train_res, y_train_res = oversampler.fit_resample(text_train_tok_pad, y_train_encoded)
```



Gambar 4.5 Data *training* sebelum dan sesudah *oversampling*

4.7 Pemodelan LSTM

Setelah melalui proses preprocessing dan pembobotan, tahap selanjutnya adalah pembuatan model. Dalam penelitian ini, digunakan library *TensorFlow* untuk membangun arsitektur model. Model terdiri dari lima lapisan utama, yaitu: *input layer*, *embedding layer*, LSTM *layer*, *dropout layer*, dan *output layer*. Parameter yang divariasikan dalam penelitian ini meliputi jumlah *neuron* LSTM, tingkat *dropout* (*dropout rate*), dan laju pembelajaran (*learning rate*), sesuai dengan skenario yang dijelaskan pada Tabel 3.16. Pelatihan model akan menggunakan optimisasi *Adam*, dengan jumlah *epoch* sebanyak 100 dan ukuran *batch* sebesar 32.

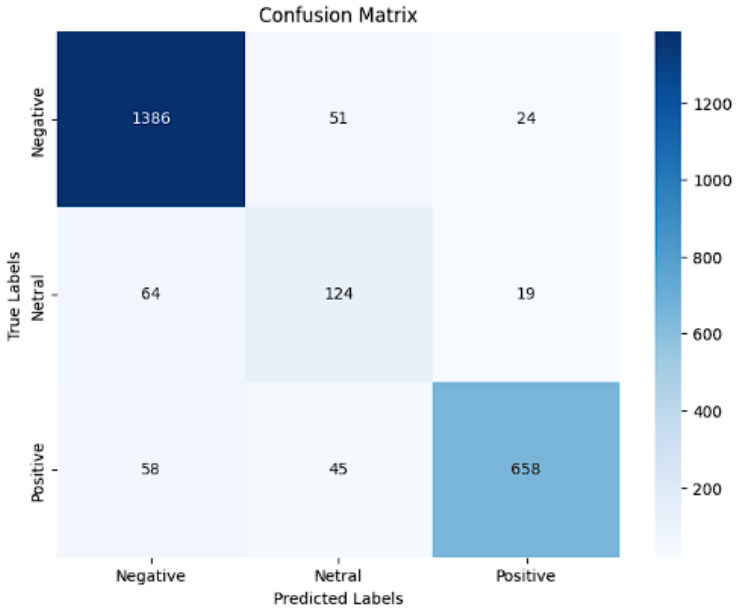
4.8 Hasil Uji Coba

Percobaan penelitian ini dilakukan dengan menguji model menggunakan kombinasi parameter, yaitu jumlah *neuron* LSTM, *dropout rate*, dan *learning rate*, yang nilainya telah ditentukan pada bab sebelumnya. Berikut ini adalah hasil pengujian dari 27 model dengan variasi parameter yang digunakan, yang dapat dilihat pada tabel 4.10 berikut ini.

Tabel 4. 10 Hasil evaluasi skenario pengujian

Model Name	Neuron LSTM	Dropout	Learning Rate	Accuracy	Precision	Recall	*F1 score*
Model 1	32	0.2	0.01	0.879	0.787	0.8	0.792
Model 2	32	0.3	0.01	0.875	0.784	0.792	0.787
Model 3	32	0.5	0.01	0.872	0.772	0.793	0.78
Model 4	32	0.2	0.001	0.89	0.816	0.784	0.798
Model 5	32	0.3	0.001	0.887	0.8	0.808	0.803
Model 6	32	0.5	0.001	0.89	0.809	0.802	0.804
Model 7	32	0.2	0.0001	0.863	0.763	0.806	0.775
Model 8	32	0.3	0.0001	0.846	0.742	0.791	0.76
Model 9	32	0.5	0.0001	0.867	0.771	0.811	0.781
Model 10	64	0.2	0.01	0.877	0.794	0.778	0.785
Model 11	64	0.3	0.01	0.879	0.785	0.797	0.79
Model 12	64	0.5	0.01	0.867	0.775	0.789	0.781
Model 13	64	0.2	0.001	0.888	0.801	0.801	0.801
Model 14	64	0.3	0.001	0.888	0.816	0.781	0.796
Model 15	64	0.5	0.001	0.886	0.804	0.793	0.798
Model 16	64	0.2	0.0001	0.869	0.768	0.801	0.779
Model 17	64	0.3	0.0001	0.879	0.79	0.797	0.791
Model 18	64	0.5	0.0001	0.868	0.769	0.812	0.782
Model 19	128	0.2	0.01	0.876	0.78	0.786	0.783
Model 20	128	0.3	0.01	0.872	0.784	0.77	0.775
Model 21	128	0.5	0.01	0.868	0.771	0.795	0.779
Model 22	128	0.2	0.001	0.893	0.807	0.804	0.805
Model 23	128	0.3	0.001	0.885	0.801	0.791	0.795
Model 24	128	0.5	0.001	0.891	0.825	0.789	0.805
Model 25	128	0.2	0.0001	0.883	0.797	0.8	0.797
Model 26	128	0.3	0.0001	0.878	0.784	0.81	0.791
Model 27	128	0.5	0.001	0.882	0.796	0.797	0.795
Tanpa Oversampling	128	0.3	0.001	0.887	0.807	0.78	0.793

Berdasarkan hasil uji coba diatas, performa terbaik diperoleh pada Model 22 dengan kombinasi parameter *neuron* LSTM = 128, *dropout* = 0.2, dan *learning rate* = 0.001. Model ini menghasilkan akurasi sebesar 89,3%, *precision* 80.7%, *recall* 80,4 dan *F1 score* 80.5%.



	precision	recall	f1-score	support
Negative	0.919	0.949	0.934	1461
Netral	0.564	0.599	0.581	207
Positive	0.939	0.865	0.900	761
accuracy			0.893	2429
macro avg	0.807	0.804	0.805	2429
weighted avg	0.895	0.893	0.893	2429

Gambar 4.6 *Confusion Matrix*

4.9 Pembahasan

Berdasarkan hasil evaluasi pada tabel 4.11, performa terbaik diperoleh model dengan kombinasi parameter *neuron* LSTM sebesar 128, *dropout rate* sebesar 0,2, dan *learning rate* sebesar 0,001 (Model 22). Model ini berhasil mencapai akurasi sebesar 89,3%, *precision* 80,7%, *recall* 80,4%, dan *F1 score* sebesar 80,5%. Jika ditinjau lebih lanjut melalui *confusion matrix* pada Gambar 4.6, terlihat bahwa performa model dalam mengklasifikasikan sentimen kategori *netral* masih relatif

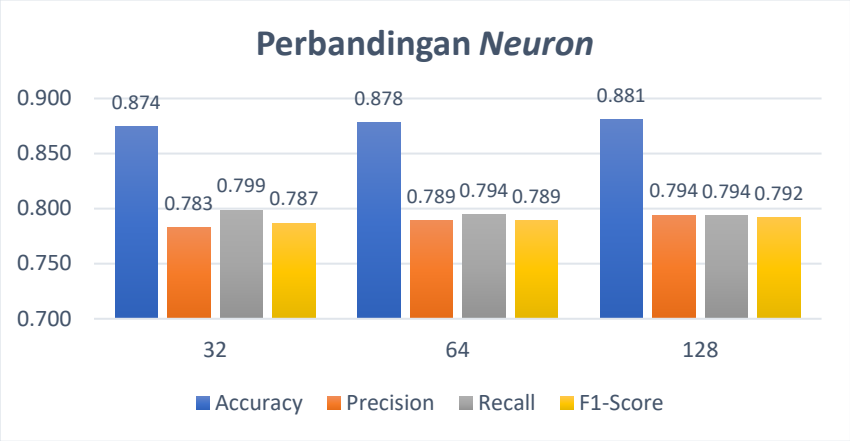
lebih rendah dibanding kategori negatif dan positif. Hal ini terlihat dari nilai *recall* untuk kelas netral yang hanya sebesar 59.9%, dan *precision* sebesar 56.4%, dengan *F1 score* sebesar 58.1%. Akibatnya, data yang seharusnya termasuk dalam kategori netral sering kali diklasifikasikan sebagai sentimen negatif atau positif.

Sebaliknya, performa model dalam mengklasifikasikan kategori negatif dan positif menunjukkan hasil yang lebih tinggi dibandingkan kategori netral. Untuk kelas negatif, model memperoleh *precision* sebesar 91,9%, *recall* sebesar 94,9%, dan *F1 score* sebesar 93,4%, yang. Sementara itu, untuk kelas positif, model juga menunjukkan hasil *precision* sebesar 93,9%, *recall* sebesar 86,5%, dan *F1 score* mencapai 90,0%. Performa kelas netral masih berada di bawah dua kelas lainnya disebabkan oleh karakteristik data netral yang cenderung ambigu, sehingga lebih sulit dikenali oleh model. Selain itu, jumlah data pada kelas netral yang relatif sedikit dibandingkan kelas lainnya turut menjadi faktor yang memengaruhi akurasi prediksi pada kelas tersebut. Meskipun telah dilakukan penyeimbangan data menggunakan *random oversampling*, pengaruhnya terhadap performa model tidak signifikan. Berdasarkan Tabel 4.10, model dengan *oversampling* hanya meningkatkan akurasi sebesar 0,6%. Metode ini hanya menggandakan data yang sudah ada, tanpa menambah informasi baru atau keragaman dalam pola data. Sebagai akibatnya, kemampuan model dalam mengenali pola sentimen netral tetap terbatas, karena tidak ada penambahan representasi yang lebih bervariasi yang dapat membantu model membedakan sentimen netral dengan lebih akurat.

Untuk mengetahui pengaruh masing-masing *hyperparameter* terhadap performa model, dilakukan evaluasi dengan menghitung rata-rata performa model pada setiap variasi nilai *hyperparameter*.

4.9.1 *Neuron*

Analisis terhadap pengaruh jumlah *neuron* terhadap performa model dilakukan dengan membandingkan rata-rata hasil evaluasi model pada setiap konfigurasi jumlah *neuron*, sebagaimana ditunjukkan grafik pada Gambar 4.7.



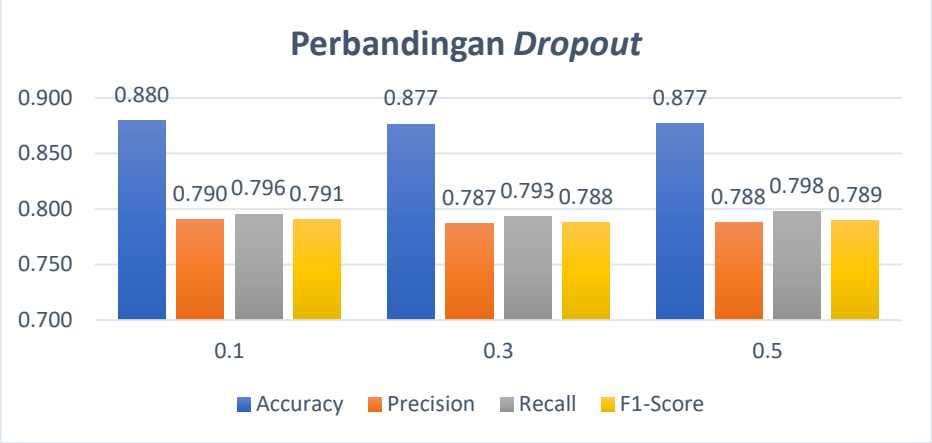
Gambar 4.7 Grafik *Neuron*

Model dengan 128 *neuron* memberikan performa terbaik dengan akurasi 88,1%, *precision* 79,4%, dan *F1 score* 79,4%. Jika dibandingkan dengan model yang menggunakan 64 *neuron*, model dengan 128 *neuron* menghasilkan performa yang sedikit lebih tinggi. Akurasi pada model 64 neuron adalah 87,8%, atau 0,3% lebih rendah dibandingkan model 128 neuron. Kedua model memiliki nilai *recall* yang sama, yaitu 79,4%. Namun, *precision* pada model dengan 64 *neuron* lebih rendah sebesar 0,5%, sehingga menyebabkan penurunan pada *F1 score*.

Model dengan 32 *neuron* menunjukkan performa terendah, dengan akurasi 87,4%, yang turun 0,7% dibandingkan model dengan 128 *neuron*. *Precision* pada model ini adalah 78,3%, turun 1,1% dibandingkan dengan model 128 *neuron*. *Recall* pada model ini lebih tinggi dibandingkan model lainnya, namun karena *precision* lebih rendah menyebakan nilai *F1 score* tidak mengalami peningkatan. *F1 score* pada model 32 *neuron* adalah 78,7%, yang turun 0,7% dibandingkan model 128 *neuron*. Peningkatan jumlah *neuron* cenderung meningkatkan performa model, hal ini disebabkan oleh semakin besar memori yang dihasilkan dan juga membuatnya dapat mempelajari pola-pola yang lebih kompleks.

4.9.2 *Dropout*

Dropout merupakan teknik regulasi yang digunakan untuk mencegah terjadinya *overfitting* pada model. Dengan menerapkan *dropout*, sejumlah *neuron* secara acak dinonaktifkan selama proses pelatihan, sehingga model tidak terlalu bergantung pada neuron tertentu dan dapat belajar secara lebih umum. Hasil pengaruh variasi *dropout* terhadap performa model dapat dilihat pada grafik berikut:

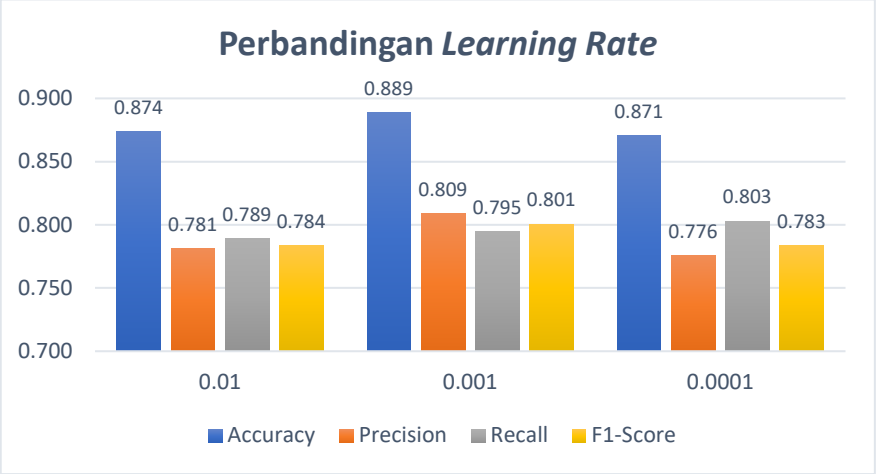


Gambar 4.8 Grafik *Dropout*

Berdasarkan data pada Gambar 4.8, diketahui bahwa *dropout* 0.2 menghasilkan performa terbaik secara keseluruhan, terutama pada metrik akurasi (88,0%) dan *F1 score* (79,1%). Sementara itu, nilai *dropout* yang lebih tinggi, seperti 0.3 dan 0.5, menunjukkan sedikit penurunan performa, yang disebabkan oleh hilangnya terlalu banyak informasi sehingga model tidak dapat mempelajari pola secara optimal. Namun, variasi nilai *dropout* tidak memberikan dampak signifikan terhadap peningkatan kinerja model. Hal ini ditunjukkan oleh kenaikan akurasi yang hanya sebesar 0,3% dibandingkan dengan model yang menggunakan nilai dropout yang lebih tinggi, serta peningkatan *F1 score* yang juga terbatas, yaitu sebesar 0,3%

4.9.3 *Learning Rate*

Learning rate berperan dalam mengatur besar pembaruan bobot selama pelatihan. Untuk mengetahui seberapa besar pengaruhnya terhadap performa model, dilakukan perhitungan rata-rata dari nilai *learning rate* 0.01, 0.001, dan 0.0001. Perhitungan ini bertujuan untuk mengevaluasi dan membandingkan dampak masing-masing nilai *learning rate* terhadap metrik-metrik evaluasi seperti akurasi, *precision*, *recall*, dan *F1 score*. Hasil rata-rata performa model dapat dilihat pada gambar 4.9.



Gambar 4.9 Grafik *Learning Rate*

Berdasarkan Gambar 4.9, *learning rate* sebesar 0.001 menghasilkan performa terbaik dengan akurasi sebesar 88,9%, *precision* 80,9%, *recall* 79,5%, dan *F1 score* 80,1%. Sementara itu, nilai *learning rate* yang terlalu besar (0,01) maupun terlalu kecil (0,0001) menunjukkan performa yang kurang optimal. *Learning rate* 0,01 menghasilkan performa rendah karena selama proses pelatihan, model cenderung tidak stabil dan sering melewati titik konvergensi yang optimal. Sebaliknya, *learning rate* 0,0001 membuat proses pelatihan berlangsung sangat lambat, sehingga kurang efisien dari segi waktu dan menghambat pencapaian hasil yang optimal. Adapun *learning rate* sebesar 0,001 memberikan performa terbaik karena proses pelatihan berlangsung stabil.

Jika dibandingkan *learning rate* 0.001 dengan *learning rate* 0.01, penggunaan *learning rate* 0.001 menghasilkan peningkatan akurasi sebesar 1,5%, *precision* 2,8%, *recall* 0,6%, dan *F1 score* 1,7%. Sedangkan jika dibandingkan dengan *learning rate* 0.0001, penggunaan *learning rate* 0.001 menunjukkan peningkatan sebesar 1,8% untuk akurasi, 3,3% untuk *precision*, dan 1,8% untuk *F1 score*.

Meskipun nilai *recall* pada *learning rate* 0.0001 sedikit lebih tinggi dibandingkan dengan 0.001, rendahnya nilai *precision* pada 0.0001 menyebabkan *F1 score* yang dihasilkan tetap lebih rendah. Hal ini menunjukkan bahwa *learning rate* 0.001 memberikan keseimbangan yang lebih baik antara *precision* dan *recall*, sehingga menghasilkan *F1 score* tertinggi dan kinerja model yang paling optimal.

4.10 Integrasi Islam

Mendengarkan dan menilai pendapat serta pandangan dengan cara yang objektif merupakan sikap yang sejalan dengan prinsip-prinsip dalam ajaran Islam. Hal ini ditekankan dalam Surah Az-Zumar ayat 18, yang berbunyi:

الَّذِينَ يَسْتَمِعُونَ الْقَوْلَ فَيَتَّبِعُونَ أَحْسَنَهُ ۚ أُولَٰئِكَ الَّذِينَ هَدَاهُمُ اللَّهُ ۖ وَأُولَٰئِكَ هُمْ أُولُو الْأَلْبَابِ

"yang mendengarkan perkataan lalu mengikuti apa yang paling baik di antaranya. Mereka itulah orang-orang yang telah diberi Allah petunjuk dan mereka itulah orang-orang yang mempunyai akal." (QS. Az-Zumar: 18)

Menurut interpretasi Imam Fakhruddin Ar-Razi, ayat tersebut merujuk pada individu yang memiliki kemampuan untuk berpikir secara mendalam (*ulul albab*) terhadap tanda-tanda kekuasaan Allah. Mereka memiliki kemampuan untuk mendengarkan nasihat dengan hati terbuka, mengambil hikmah dari pengalaman orang lain, serta menyaring dan menganalisis informasi dengan teliti. Sikap ini mencerminkan pemikiran kritis, selektivitas, dan kebijaksanaan dalam menanggapi beragam pandangan (Nafi' et al., 2023). Prinsip ini sangat relevan dalam konteks klasifikasi sentimen, yang mengharuskan pemahaman terhadap berbagai pendapat dengan cara yang objektif.

Konsep klasifikasi sejalan dengan ajaran yang terkandung dalam Surah Al-Waqi'ah ayat 7–10, yang menyatakan:

وَّكُنْتُمْ اَزْوَاجًا ثَلٰثَةً ۗ ﴿٧﴾ فَاَصْحٰبُ الْمَيْمَنَةِ ۙ مَآ اَصْحٰبُ الْمَيْمَنَةِ ۗ ﴿٨﴾ وَاَصْحٰبُ الْمَشْئَمَةِ ۙ مَآ اَصْحٰبُ الْمَشْئَمَةِ ۗ ﴿٩﴾ وَالسّٰبِقُوْنَ السّٰبِقُوْنَ ۙ ﴿١٠﴾

"Kamu menjadi tiga golongan, yaitu golongan kanan, alangkah mulianya golongan kanan itu. dan golongan kiri, alangkah sengsaranya golongan kiri itu. Selain itu, (golongan ketiga adalah) orang-orang yang paling dahulu (beriman). Merekalah yang paling dahulu (masuk surga)" (QS. Al-Waqi'ah:7-10)

Ayat ini menggambarkan pembagian atau pengelompokan manusia berdasarkan kondisi dan pencapaian mereka. Dalam konteks analisis sentimen, ayat ini dapat dijadikan acuan untuk mengkategorikan berbagai opini yang muncul dalam data. Sentimen positif mencerminkan ekspresi yang menunjukkan dukungan, kepuasan, atau penghargaan terhadap sesuatu. Sebaliknya, sentimen negatif menggambarkan perasaan ketidakpuasan, penolakan, atau kritik. Sementara itu, sentimen netral mencakup pernyataan yang bersifat informatif, deskriptif, atau tidak mengarah pada penilaian tertentu.

Dengan menganalisis sentimen pengguna, pengembang aplikasi dapat menilai kelemahan dan kelebihan dari layanan yang diberikan. Proses ini sejalan dengan prinsip muhasabah dalam Islam, yaitu kebiasaan untuk melakukan introspeksi secara berkelanjutan demi perbaikan. Hal ini tercermin dalam Surah Al-Hasyr ayat 18, di mana Allah SWT berfirman:

يٰٓاَيُّهَا الَّذِيْنَ اٰمَنُوا اتَّقُوا اللّٰهَ وَلْتَنْظُرْ نَفْسٌ مَّا قَدَّمَتْ لِغَدٍۚ وَاتَّقُوا اللّٰهَ ۗاِنَّ اللّٰهَ خَبِيْرٌۢ بِمَا تَعْمَلُوْنَ ﴿١٨﴾

"Wahai orang-orang yang beriman, bertakwalah kepada Allah dan hendaklah setiap orang memperhatikan apa yang telah diperbuatnya untuk hari esok (akhirat). Bertakwalah kepada Allah. Sesungguhnya Allah Maha Teliti terhadap apa yang kamu kerjakan." (QS. Al- Hasyr: 18)

Dalam Tafsir Al-Misbah, dijelaskan bahwa ayat ini mengandung dorongan bagi setiap individu untuk terus-menerus melakukan evaluasi terhadap tindakan

yang telah dilakukannya. Seperti seorang pengrajin yang memeriksa kembali hasil kerjanya, jika sudah baik, ia akan menyempurnakannya, dan jika masih kurang, ia akan memperbaikinya. Pendekatan ini menekankan pentingnya evaluasi yang berkelanjutan untuk mencapai hasil yang lebih optimal (Ramadhan & Hidayat, 2024). Pendekatan serupa sangat relevan dalam pengembangan aplikasi, terutama dalam merespons umpan balik pengguna. Ulasan positif dapat menjadi landasan untuk mempertahankan kualitas, sementara kritik dapat digunakan sebagai bahan perbaikan untuk meningkatkan mutu layanan secara berkelanjutan.

Semangat muhasabah mendorong individu untuk tidak merasa puas secara cepat, tetapi untuk terus mengevaluasi dan memperbaiki hasil kerjanya. Proses evaluasi ini menciptakan dorongan untuk meningkatkan kualitas kerja dan menghasilkan output yang lebih baik. Dalam konteks pengembangan aplikasi, semangat tersebut tercermin dalam kesiapan untuk menerima kritik dengan sikap terbuka dan menjadikannya sebagai landasan untuk perbaikan layanan agar lebih optimal. Prinsip ini selaras dengan sabda Rasulullah SAW yang diriwayatkan oleh Aisyah RA:

عَنْ عَائِشَةَ، أَنَّ رَسُولَ اللَّهِ صَلَّى اللهُ عَلَيْهِ وَسَلَّمَ قَالَ: إن اللَّهَ عَزَّ وَجَلَّ يُحِبُّ إِذَا عَمِلَ أَحَدُكُمْ عَمَلًا أَنْ يُتْقِنَهُ

“Allah 'azza wa jalla menyukai jika salah seorang di antara kalian melakukan suatu amal secara itqan” (HR. Al-Baihaqi).

Menurut Mat dan Ghani, konsep *itqan* menggambarkan bahwa ketekunan dalam melaksanakan suatu pekerjaan akan menghasilkan produk berkualitas tinggi, yang bebas dari praktik penipuan atau pengelapan. Oleh karena itu, Islam tidak hanya menekankan aspek kehalalan dalam proses konsumsi dan produksi, tetapi

juga mendorong terciptanya kualitas yang unggul. Kualitas tersebut sering disebut dalam Al-Qur'an dengan istilah *thayyiban*, yang berarti "baik" dan biasanya digandengkan dengan kata halalan (Fathrul Quddus, 2021).

Penelitian ini diharapkan dapat menjadi kontribusi yang bermanfaat untuk membantu sesama, khususnya bagi pengembang aplikasi Indrive, dalam upaya perbaikan dan peningkatan kualitas layanan. Langkah ini mencerminkan implementasi nyata dari prinsip tolong-menolong yang diajarkan dalam Islam, sebagaimana tercantum dalam Surah Al-Mā'idah ayat 2:

وَتَعَاوَنُوْا عَلَى الْبِرِّ وَالتَّقْوٰىۖ وَلَا تَعَاوَنُوْا عَلَى الْاِثْمِ وَالْعُدْوَانِۖ وَاتَّقُوا اللّٰهَۗ اِنَّ اللّٰهَ شَدِيْدُ الْعِقَابِ ۝٢

"Tolong-menolonglah kamu dalam (mengerjakan) kebajikan dan takwa, dan jangan tolong-menolong dalam berbuat dosa dan permusuhan. Bertakwalah kepada Allah, sesungguhnya Allah sangat berat siksaan-Nya." (QS. Al-Ma'idah: 2)

Dalam tafsir Al-Maraghi, dijelaskan bahwa ajaran untuk saling tolong-menolong dalam kebaikan dan ketakwaan merupakan salah satu prinsip sosial utama yang terdapat dalam Al-Qur'an. Allah SWT mewajibkan umat manusia untuk saling membantu dalam hal-hal yang bermanfaat bagi masyarakat, baik dalam konteks individu maupun kelompok, serta dalam urusan agama maupun kehidupan duniawi(Puspitasari, 2022).

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan penelitian yang telah dilakukan, metode *Long Short-Term Memory* (LSTM) menunjukkan kinerja terbaik dalam mengklasifikasikan sentimen ulasan aplikasi Indrive dengan konfigurasi *hyperparameter* yang meliputi jumlah *neuron* sebanyak 128, *dropout rate* sebesar 0,2, dan *learning rate* sebesar 0,001. Model ini berhasil memperoleh akurasi sebesar 89,3%, *precision* 80,7%, *recall* 80,4%, dan *F1 score* 80,5%. Analisis terhadap *hyperparameter* masing-masing menunjukkan bahwa peningkatan jumlah *neuron* hingga 128 menghasilkan peningkatan kinerja dibandingkan dengan jumlah neuron yang lebih rendah. *Dropout rate* yang lebih kecil, yaitu 0,2, memberikan hasil yang sedikit lebih baik dibandingkan dengan nilai *dropout rate* yang lebih tinggi. Selain itu, variasi pada *learning rate* dapat memengaruhi kinerja model, dengan nilai 0,001 memberikan hasil optimal dibandingkan dengan nilai yang lebih rendah (seperti 0,0001) atau lebih tinggi (seperti 0,01).

5.2 Saran

Berdasarkan temuan yang diperoleh dari penelitian ini, terdapat beberapa aspek yang masih dapat diperbaiki untuk mencapai hasil yang lebih optimal pada penelitian yang akan datang. Beberapa saran berikut dapat dijadikan pertimbangan untuk pengembangan penelitian selanjutnya:

1. Meningkatkan jumlah data serta memastikan distribusi yang seimbang antar kategori sentimen, terutama untuk kategori netral yang menunjukkan performa precision dan recall yang masih rendah.
2. Meningkatkan jumlah LSTM layer untuk memungkinkan model mempelajari pola yang lebih kompleks dari data.
3. Mengeksplorasi penerapan metode alternatif, seperti *Bidirectional* LSTM (Bi-LSTM) atau *Gated Recurrent Unit* (GRU), untuk membandingkan performanya dengan model LSTM tradisional.

DAFTAR PUSTAKA

Alhamdani, F. D. S., Marthasari, G. I., & Aditya, C. S. K. (2021). Prediksi Harga Emas menggunakan Metode Time Series Long Short—Term Memory Neural Network. *Jurnal Repositor*, *3*(4), 375–386.

Aufa, M. J., & Qoiriah, A. (2023). Analisis Sentimen Pengguna Platform Belajar Online Coursera menggunakan Random Forest dengan Metode Ekstraksi Fitur Word2vec. *Journal of Informatics and Computer Science (JINACS)*, 244–255. https://doi.org/10.26740/jinacs.v4n02.p244-255

Aziah, A., & Adawia, P. R. (2018). Analisis Perkembangan Industri Transportasi Online di Era Inovasi Disruptif (Studi Kasus PT Gojek Indonesia). *Jurnal Humaniora*, *18*(2). https://doi.org/10.31294/jc.v18i2.4117

Azrul, A., Irma Purnamasari, A., & Ali, I. (2024). Analisis Sentimen Pengguna Twitter Terhadap Perkembangan Artificial Intelligence dengan Penerapan Algoritma Long Short-Term Memory (LSTM). *JATI (Jurnal Mahasiswa Teknik Informatika)*, *8*(1), 413–421. https://doi.org/10.36040/jati.v8i1.8416

Basri, N. F., & Utami, E. (2025). Application of Word2Vec and LSTM Models in Sentiment Analysis of Mobile Legends User Reviews. *Sistemasi: Jurnal Sistem Informasi*, *14*(2), 856. https://doi.org/10.32520/stmsi.v14i2.5074

Fathrul Quddus, M. (2021). Kritik Konsumerisme dalam Etika Konsumsi Islam. *Malia: Jurnal Ekonomi Islam*, *13*(1), 43–60. https://doi.org/10.35891/ml.v13i1.2771

Ivan, Sari, Y. A., & Adikara, P. P. (2019). Klasifikasi Hate Speech Berbahasa Indonesia di Twitter Menggunakan Naive Bayes dan Seleksi Fitur Information Gain dengan Normalisasi Kata. *Jurnal Pengembangan Teknologi Informasi Dan Ilmu Komputer*, *3(5)*, 4914–4922.

Laras, A. (2023, March 28). *Mengenal inDrive, Aplikasi Ojol Asal Rusia yang Bisa Tawar Harga*. Bisnis.com. https://entrepreneur.bisnis.com/read/20230328/52/1641483/mengenal-indrive-aplikasi-ojol-asal-rusia-yang-bisa-tawar-harga

Lisanthoni, A., Gunawan, E. L., Adhigiadany, C. A., & Prasetya, A. (2024). Penerapan LSTM dalam Analisis Sentimen Berbasis Lexicon untuk Meningkatkan Sistem Pemantauan Citra PLN di Platform Digital. *Prosiding Seminar Nasional Sains Data*, *4(1)*, 581–591. https://doi.org/10.33005/senada.v4i1.287

Maharani, C. A., Warsito, B., & Santoso, R. (2024). Analisis Sentimen Vaksin Covid-19 pada Twitter Menggunakan Recurrent Neural Network (RNN)

dengan Algoritma Long Short-Term Memory (LSTM). *Jurnal Gaussian*, *12*(3), 403–413. https://doi.org/10.14710/j.gauss.12.3.403-413

Miftahusalam, A., Nuraini, A. F., Khoirunisa, A. A., & Pratiwi, H. (2022). Perbandingan Algoritma Random Forest, Naïve Bayes, dan Support Vector Machine pada Analisis Sentimen Twitter Mengenai Opini Masyarakat terhadap Penghapusan Tenaga Honorer. *Seminar Nasional Official Statistics*, *2022*(1), 563–572. https://doi.org/10.34123/semnasoffstat.v2022i1.1410

Mola, S. A. S., Luttu, Y. C., & Rumlaklak, D. N. (2024). Perbandingan Metode Machine Learning dalam Analisis Sentimen Komentar Pengguna Aplikasi InDriver pada Dataset Tidak Seimbang. *Jurnal Sistem Informasi Bisnis*, *14*(3), 247–255. https://doi.org/10.21456/vol14iss3pp247-255

Muhammad, P. F., Kusumaningrum, R., & Wibowo, A. (2021). Sentiment Analysis Using Word2vec and Long Short-Term Memory (LSTM) for Indonesian Hotel Reviews. *Procedia Computer Science*, *179*, 728–735. https://doi.org/10.1016/j.procs.2021.01.061

Mutmatinah, S. (2024). Metode Deep Learning LSTM dalam Analisis Sentimen Aplikasi PeduliLindungi. *Scientific: Journal of Computer Science and Informatics*, *1(1)*, 9–19. https://doi.org/10.34304/scientific.v1i1.231

Nafi', N. A., Mufid, M. A., Zainuddin, A., & Rohtih, W. A. (2023). Konsep Berpikir Kritis Perspektif Imam Fakhruddin Ar-Razi (interpretasi Qs. Ali Imran: 190-191 Dan Qs. Az-Zumar:18). *Triwikrama: Jurnal Multidisiplin Ilmu Sosial*, *01*(02), 23–40. https://doi.org/10.6578/tjmis.v1i2.53

Niasita, A. F., Adikara, P. P., & Adinugroho, S. (2019). Analisis Sentimen Pembangunan Infrastruktur di Indonesia dengan Automated Lexicon Word2Vec dan Naive-Bayes. *Jurnal Pengembangan Teknologi Informasi Dan Ilmu Komputer*, *3*(3), 2673–2679.

Nurhidayat, R., & Dewi, K. E. (2023). Penerapan Algoritma K-Nearest Neighbor dan Fitur Ekstraksi N-Gram dalam Analisis Sentimen Berbasis Aspek. *Komputa : Jurnal Ilmiah Komputer dan Informatika*, *12*(1), 91–100. https://doi.org/10.34010/komputa.v12i1.9458

Pane, S. F., & Ramdan, J. (2022). Pemodelan Machine Learning: Analisis Sentimen Masyarakat terhadap Kebijakan PPKM Menggunakan Data Twitter. *Jurnal Sistem Cerdas*, *5*(1), 12–20. https://doi.org/10.37396/jsc.v5i1.191

Pradana, Y. A., Cholissodin, I., & Kurnianingtyas, D. (2023). Analisis Sentimen Pemindahan Ibu Kota Indonesia pada Media Sosial Twitter menggunakan Metode LSTM dan Word2Vec. *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*, *7*(5), 2389–2397.

Puspitasari, M. (2022). Kerjasama dalam Lembaga Pendidikan Berdasarkan Tafsir Al-Qur'an Surat Al-Maidah Ayat 2. *LEARNING : Jurnal Inovasi Penelitian Pendidikan dan Pembelajaran*, *2*(3), 209–221. https://doi.org/10.51878/learning.v2i3.1521

Qur'atul, N., Pramono, B., & Wibowo, A. H. (2024). Penerapan Metode LSTM pada Sistem Klasifikasi Komentar Publik yang Termasuk Jenis Pelanggaran Undang-Undang ITE. *Animator*, *2*(1).

Raffi, M., Suharso, A., & Maulana, I. (2023). Analisis Sentimen Ulasan Aplikasi Binar Pada Google Play Store Menggunakan Algoritma Naïve Bayes. *INTECOMS: Journal of Information Technology and Computer Science*, *6*(1), 450–462. https://doi.org/10.31539/intecoms.v6i1.6117

Rahman, M. Z., Sari, Y. A., & Yudistira, N. (2021). Analisis Sentimen Tweet COVID-19 menggunakan Word Embedding dan Metode Long Short-Term Memory (LSTM). *Jurnal Pengembangan Teknologi Informasi Dan Ilmu Komputer*, *5*(11), 5120–5127.

Ramadhan, F. S., & Hidayat, A. S. (2024). Tafsir Ayat-Ayat Al-Qur'an Tentang Fungsi Manajemen Pendidikan (Studi Tafsir Maudhu'i dalam Qs. Al-Hasyr: 18, Qs. Ali-Imran: 103, Qs. Al-Kahfi: 2, Dan Qs. Al-Infhithar: 10-12). *INOVATIF: Jurnal Penelitian Pendidikan, Agama, dan Kebudayaan*, *10*(1), 86–107. https://doi.org/10.55148/inovatif.v10i1.788

Saputra, A. C. (2020). Penentuan Parameter Learning Rate Selama Pembelajaran Jaringan Syaraf Tiruan Backpropagation menggunakan Algoritma Genetika. *Jurnal Teknologi Informasi: Jurnal Keilmuan dan Aplikasi Bidang Teknik Informatika*, *14*(2), 202–212. https://doi.org/10.47111/jti.v14i2.1141

Sari, W. K., Rini, D. P., Malik, R. F., & Azhar, I. S. B. (2020). Klasifikasi Teks Multilabel pada Artikel Berita menggunakan Long Short- Term Memory dengan Word2Vec. *Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi)*, *4*(2), 276–285. https://doi.org/10.29207/resti.v4i2.1655

Shyahrin, M. V., Sibaroni, Y., & Puspandari, D. (2023). Penerapan Metode Long Short-Term Memory dan Word2Vec dalam Analisis Sentimen Ulasan pada Aplikasi Ferizy. *Techno.Com*, *22*(4), 833–842. https://doi.org/10.33633/tc.v22i4.9205

Sofyan, S., & Prasetyo, A. (2021). Penerapan Synthetic Minority Oversampling Technique (SMOTE) Terhadap Data Tidak Seimbang pada Tingkat Pendapatan Pekerja Informal Di Provinsi D.I. Yogyakarta Tahun 2019. *Seminar Nasional Official Statistics*, *2021*(1), 868–877. https://doi.org/10.34123/semnasoffstat.v2021i1.1081

Tan, K. L., Lee, C. P., & Lim, K. M. (2023). A Survey of Sentiment Analysis: Approaches, Datasets, and Future Research. *Applied Sciences*, *13*(7), 4550. https://doi.org/10.3390/app13074550

Verianto, E. (2024). Penerapan LSTM dengan Regularisasi untuk Mencegah Overfitting pada Model Prediksi Tingkat Inflasi di Indonesia. *Simkom*, *9*(2), 195–204. https://doi.org/10.51717/simkom.v9i2.460

Widayat, W. (2021). Analisis Sentimen Movie Review menggunakan Word2Vec dan metode LSTM Deep Learning. *Jurnal Media Informatika Budidarma*, *5*(3), 1018. https://doi.org/10.30865/mib.v5i3.3111

Yan, S. (2017, November 15). *Understanding LSTM and its diagrams*. Medium. https://blog.mlreview.com/understanding-lstm-and-its-diagrams-37e2f46f1714

LAMPIRAN

```
import numpy as np
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# --- Helper Functions ---
def sigmoid(x, derivative=False):
    return x * (1 - x) if derivative else 1 / (1 + np.exp(-x))

def tanh(x, derivative=False):
    return 1 - x**2 if derivative else np.tanh(x)

def softmax(x):
    x = x - np.max(x, axis=0, keepdims=True)
    exps = np.exp(x)
    return exps / np.sum(exps, axis=0, keepdims=True)

def initWeights(output_size, input_size):
    return np.random.uniform(-1, 1, (output_size, input_size)) * np.sqrt(6 / (input_size +
output_size))


class LSTM:
    def __init__(self, hidden_size, output_size, num_epochs, learning_rate, embedding_vectors,
batch_size=32, dropout_rate=0.5):
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.num_epochs = num_epochs
        self.learning_rate = learning_rate
        self.embedding_vectors = embedding_vectors
        self.vector_size = embedding_vectors.shape[1]
        self.batch_size = batch_size
        self.dropout_rate = dropout_rate  # Dropout rate

        input_size = self.vector_size

        # Weights - Initialized using Glorot/Xavier
        self.wf = initWeights(hidden_size, input_size)
        self.wi = initWeights(hidden_size, input_size)
        self.wc = initWeights(hidden_size, input_size)
        self.wo = initWeights(hidden_size, input_size)
        self.wy = initWeights(output_size, hidden_size)

        self.uf = initWeights(hidden_size, hidden_size)
        self.ui = initWeights(hidden_size, hidden_size)
        self.uc = initWeights(hidden_size, hidden_size)
        self.uo = initWeights(hidden_size, hidden_size)

        # Biases - Initialized to zeros
        self.bf = np.zeros((hidden_size, 1))
        self.bi = np.zeros((hidden_size, 1))
        self.bc = np.zeros((hidden_size, 1))
        self.bo = np.zeros((hidden_size, 1))
        self.by = np.zeros((output_size, 1))
```

```
    def reset_states(self):
        self.hidden_states = {-1: np.zeros((self.hidden_size, 1))}
        self.cell_states = {-1: np.zeros((self.hidden_size, 1))}
        self.forget_gates = {}
        self.input_gates = {}
        self.candidate_gates = {}
        self.output_gates = {}
        self.inputs_sequence = {}  # Store inputs for backward pass

    def forward(self, inputs, train=True):
        self.reset_states()  # Reset states for each new forward pass
        self.last_t = len(inputs) - 1  # Store the last timestep index
        for t in range(len(inputs)):
            x = inputs[t].reshape(-1, 1)  # Current input vector
            self.inputs_sequence[t] = x  # Store input for backward pass
            h_prev = self.hidden_states[t - 1]  # Previous hidden state
            c_prev = self.cell_states[t - 1]  # Previous cell state

            # Calculate gate activations
            self.forget_gates[t] = sigmoid(np.dot(self.wf, x) + np.dot(self.uf, h_prev) + self.bf)
            self.input_gates[t] = sigmoid(np.dot(self.wi, x) + np.dot(self.ui, h_prev) + self.bi)
            self.candidate_gates[t] = tanh(np.dot(self.wc, x) + np.dot(self.uc, h_prev) + self.bc)
            self.output_gates[t] = sigmoid(np.dot(self.wo, x) + np.dot(self.uo, h_prev) + self.bo)

            # Update cell state and hidden state
            self.cell_states[t] = self.forget_gates[t] * c_prev + \
                                  self.input_gates[t] * self.candidate_gates[t]
            self.hidden_states[t] = self.output_gates[t] * tanh(self.cell_states[t])

            # Apply dropout if in training mode (dropout after hidden state)
        if train:
            mask = np.random.binomial(1, 1 - self.dropout_rate, self.hidden_size).reshape(-1, 1)
            self.hidden_states[self.last_t] *= mask

        # Output layer calculation for the last hidden state
        output = np.dot(self.wy, self.hidden_states[self.last_t]) + self.by
        return output

    def categorical_cross_entropy(self, probs, label):
        eps = 1e-10  # Small epsilon for numerical stability
        probs = np.clip(probs, eps, 1 - eps)  # Clip probabilities to avoid log(0)
        return -np.sum(label * np.log(probs))

    def backward(self, error, sequence_length):
        # Initialize gradients for the current sequence to zeros
        d_wf = np.zeros_like(self.wf)
        d_wi = np.zeros_like(self.wi)
        d_wc = np.zeros_like(self.wc)
        d_wo = np.zeros_like(self.wo)
        d_wy = np.dot(error, self.hidden_states[self.last_t].T)  # Gradient for output weights
        d_by = error  # Gradient for output bias

        d_uf = np.zeros_like(self.uf)
        d_ui = np.zeros_like(self.ui)
        d_uc = np.zeros_like(self.uc)
```

```
d_uo = np.zeros_like(self.uo)

d_bf = np.zeros_like(self.bf)
d_bi = np.zeros_like(self.bi)
d_bc = np.zeros_like(self.bc)
d_bo = np.zeros_like(self.bo)

# Initial gradients for hidden and cell states (from the output layer)
d_hs = np.dot(self.wy.T, error)
dc_next = np.zeros_like(self.cell_states[0])  # Gradient from future cell state

# Backpropagation through time loop (from last timestep to first)
for t in reversed(range(sequence_length)):
    x = self.inputs_sequence[t]  # Input for current timestep
    h_prev = self.hidden_states[t - 1]  # Hidden state from previous timestep
    c_prev = self.cell_states[t - 1]  # Cell state from previous timestep

    # Gradients through output gate (o_t)
    # d_o represents dL/d(output_gate_input)
    d_o = (d_hs * tanh(self.cell_states[t])) * sigmoid(self.output_gates[t], derivative=True)
    d_wo += np.dot(d_o, x.T)
    d_uo += np.dot(d_o, h_prev.T)
    d_bo += d_o

    # Gradients through cell state (c_t)
    # d_c is the total gradient flowing into the current cell state
    d_c = dc_next + d_hs * self.output_gates[t] * tanh(self.cell_states[t], derivative=True)

    # Gradients through input gate (i_t)
    # d_i represents dL/d(input_gate_input)
    d_i = (d_c * self.candidate_gates[t]) * sigmoid(self.input_gates[t], derivative=True)
    d_wi += np.dot(d_i, x.T)
    d_ui += np.dot(d_i, h_prev.T)
    d_bi += d_i

    # Gradients through forget gate (f_t)
    # d_f represents dL/d(forget_gate_input)
    d_f = (d_c * c_prev) * sigmoid(self.forget_gates[t], derivative=True)
    d_wf += np.dot(d_f, x.T)
    d_uf += np.dot(d_f, h_prev.T)
    d_bf += d_f

    # Gradients through candidate gate (c_tilde_t)
    # d_c_tilde represents dL/d(candidate_gate_input)
    d_c_tilde = (d_c * self.input_gates[t]) * tanh(self.candidate_gates[t], derivative=True)
    d_wc += np.dot(d_c_tilde, x.T)
    d_uc += np.dot(d_c_tilde, h_prev.T)
    d_bc += d_c_tilde

    # Gradients flowing to the previous hidden state (h_{t-1}) and cell state (c_{t-1})
    # These are passed to the next iteration (t-1)
    dc_next = d_c * self.forget_gates[t]  # Gradient of c_t wrt c_{t-1}
    d_hs = np.dot(self.uf.T, d_f) + np.dot(self.ui.T, d_i) + \
           np.dot(self.uc.T, d_c_tilde) + np.dot(self.uo.T, d_o)
```

```
        # Return all calculated gradients for the current sequence
        return {
            'd_wf': d_wf, 'd_wi': d_wi, 'd_wc': d_wc, 'd_wo': d_wo, 'd_wy': d_wy,
            'd_uf': d_uf, 'd_ui': d_ui, 'd_uc': d_uc, 'd_uo': d_uo,
            'd_bf': d_bf, 'd_bi': d_bi, 'd_bc': d_bc, 'd_bo': d_bo, 'd_by': d_by
        }

    def embedding(self, idx_seq):
        # Ensure embedding_vectors are normalized externally if needed for better training
stability.
        return self.embedding_vectors[idx_seq]

    def train(self, X_train, y_train):
        for epoch in range(self.num_epochs):
            total_loss_epoch = 0
            correct_predictions_epoch = 0
            total_samples_epoch = 0

            # Shuffle data at the beginning of each epoch for better generalization
            permutation = np.random.permutation(len(X_train))
            X_train_shuffled = X_train[permutation]
            y_train_shuffled = y_train[permutation]

            # Iterate through data in mini-batches
            for i in range(0, len(X_train_shuffled), self.batch_size):
                batch_X = X_train_shuffled[i:i+self.batch_size]
                batch_y = y_train_shuffled[i:i+self.batch_size]

                # Initialize accumulated gradients for the current batch
                batch_grads = {
                    'd_wf': np.zeros_like(self.wf), 'd_wi': np.zeros_like(self.wi),
                    'd_wc': np.zeros_like(self.wc), 'd_wo': np.zeros_like(self.wo),
                    'd_wy': np.zeros_like(self.wy), 'd_uf': np.zeros_like(self.uf),
                    'd_ui': np.zeros_like(self.ui), 'd_uc': np.zeros_like(self.uc),
                    'd_uo': np.zeros_like(self.uo), 'd_bf': np.zeros_like(self.bf),
                    'd_bi': np.zeros_like(self.bi), 'd_bc': np.zeros_like(self.bc),
                    'd_bo': np.zeros_like(self.bo), 'd_by': np.zeros_like(self.by)
                }

                current_batch_size = 0  # Track actual samples in batch (for last batch)

                # Process each sample in the current mini-batch
                for x_seq, y_true in zip(batch_X, batch_y):
                    current_batch_size += 1
                    x_embed = self.embedding(x_seq)  # Get embeddings for the sequence
                    logits = self.forward(x_embed, train=True)  # Forward pass with dropout
                    probs = softmax(logits)  # Apply softmax for probabilities

                    loss = self.categorical_cross_entropy(probs, y_true.reshape(-1, 1))
                    total_loss_epoch += loss  # Accumulate loss for epoch summary

                    # Check prediction for accuracy
                    if np.argmax(probs) == np.argmax(y_true):
                        correct_predictions_epoch += 1
                    total_samples_epoch += 1
```

```
                error = probs - y_true.reshape(-1, 1)  # Calculate output error

                # Get gradients for this single sample
                grads_for_sample = self.backward(error, len(x_seq))

                # Accumulate gradients from this sample into batch_grads
                for key in batch_grads:
                    batch_grads[key] += grads_for_sample[key]

            # Apply mini-batch gradient updates (once per batch)
            if current_batch_size > 0:  # Avoid division by zero for potentially empty last batch
                for key, grad_sum in batch_grads.items():
                    # Average the gradients over the batch
                    avg_grad = grad_sum / current_batch_size

                    # Update corresponding weights/biases using averaged gradients
                    if key == 'd_wf': self.wf -= self.learning_rate * avg_grad
                    elif key == 'd_wi': self.wi -= self.learning_rate * avg_grad
                    elif key == 'd_wc': self.wc -= self.learning_rate * avg_grad
                    elif key == 'd_wo': self.wo -= self.learning_rate * avg_grad
                    elif key == 'd_wy': self.wy -= self.learning_rate * avg_grad
                    elif key == 'd_uf': self.uf -= self.learning_rate * avg_grad
                    elif key == 'd_ui': self.ui -= self.learning_rate * avg_grad
                    elif key == 'd_uc': self.uc -= self.learning_rate * avg_grad
                    elif key == 'd_uo': self.uo -= self.learning_rate * avg_grad
                    elif key == 'd_bf': self.bf -= self.learning_rate * avg_grad
                    elif key == 'd_bi': self.bi -= self.learning_rate * avg_grad
                    elif key == 'd_bc': self.bc -= self.learning_rate * avg_grad
                    elif key == 'd_bo': self.bo -= self.learning_rate * avg_grad
                    elif key == 'd_by': self.by -= self.learning_rate * avg_grad

        # Print epoch summary
        avg_loss_epoch = total_loss_epoch / total_samples_epoch
        acc_epoch = correct_predictions_epoch / total_samples_epoch
        print(f"Epoch {epoch+1}/{self.num_epochs}, Loss: {avg_loss_epoch:.4f}, Accuracy:
{acc_epoch*100:.2f}%")
        print("Correct predictions this epoch: ", correct_predictions_epoch)  # Total correct for
the entire epoch

    def test(self, X_test, y_test):
        correct = 0
        preds = []
        trues = []
        for x, y in zip(X_test, y_test):
            x_embed = self.embedding(x)  # Get embeddings
            logits = self.forward(x_embed, train=False)  # Forward pass without dropout (for
testing)
            probs = softmax(logits)  # Probabilities
            pred = np.argmax(probs)  # Predicted class
            true = np.argmax(y)  # True class
            preds.append(pred)
            trues.append(true)
            if pred == true:
                correct += 1
```

```
    acc = correct / len(X_test)
    class_names = ['Negative', 'Netral', 'Positive']  # Define your class names
    print(f"\nTest Accuracy: {acc * 100:.2f}%")

    # Generate and display Confusion Matrix
    cm = confusion_matrix(trues, preds)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_names,
yticklabels=class_names)
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    plt.title('Confusion Matrix')
    plt.show()

    # Print Classification Report
    print(classification_report(trues, preds, target_names=class_names, digits=3))
```