BAB 2
LANDASAN TEORI

2.1 Kecerdasan Buatan

Kecerdasan Buatan (AI) merupakan cabang ilmu yang menggunakan mesin untuk menyelesaikan masalah kompleks dengan cara yang mirip manusia. Ini melibatkan simulasi karakteristik berpikir manusia dalam algoritma komputer, memungkinkan AI untuk bertindak secara fleksibel dan efisien sesuai kebutuhan. AI terkait dengan berbagai bidang, termasuk Ilmu Komputer, Matematika, Psikologi, dan Filosofi, memanfaatkan pengetahuan lintas disiplin untuk kemajuan pembuatan kecerdasan buatan.

Kecerdasan buatan bisa diartikan juga sebagai cabang dari ilmu komputer yang bertujuan membuat komputer mampu menjalankan tugas-tugas yang biasanya dilakukan oleh manusia. Awalnya, komputer hanya digunakan untuk melakukan perhitungan. Namun, sejalan dengan kemajuan teknologi, fungsi komputer telah berkembang jauh lebih luas. Sekarang, komputer tidak hanya terbatas pada tugas-tugas perhitungan tetapi juga diharapkan untuk melakukan berbagai pekerjaan yang sebelumnya hanya bisa dilakukan oleh manusia.

Menurut Prof. Elder Dadios[6] kecerdasan buatan adalah bagian dari ilmu komputer yang mempelajari bagaimana membuat mesin (komputer) dapat melakukan pekerjaan seperti manusia bahkan bisa lebih baik dibandingkan manusia. Agar mesin (komputer) bisa cerdas maka harus diberi bekal pengetahuan dan mempunyai kemampuan untuk menalar. Ada dua bagian utama yang dibutuhkan aplikasi kecerdasan buatan yaitu :

1. Basis Pengetahuan (*Knowledge Base*) adalah berisi fakta-fakta, teori, pemikiran dan hubungan satu dengan lainnya.
2. Motor Inferensi (*Inference Engine*) adalah kemampuan menarik kesimpulan berdasarkan pengetahuan.

Kecerdasan buatan mempunyai kelebihan dibanding dengan kecerdasan alami antara lain:

1. Lebih bersifat permanen. Sebab kecerdasan buatan tidak berubah selama sistem komputer dan program tidak mengubahnya.

2. Lebih mudah diduplikasi dan disebarkan.
3. Lebih murah, karena menyediakan layanan komputer yang lebih mudah dan murah dibandingkan mendatangkan seseorang untuk mengerjakan sejumlah pekerjaan dalam jangka waktu yang lama.
4. Bersifat konsisten dan teliti.
5. Dapat mengerjakan task lebih cepat.

2.2 Deep Learning

Deep Learning adalah cabang dari machine learning yang menggunakan jaringan saraf tiruan yang sangat dalam, atau *Deep Neural Networks* (DNN), untuk mengatasi berbagai permasalahan dalam domain machine learning. Metode ini berusaha meniru proses berpikir manusia dengan membangun representasi data melalui lapisan pembelajaran bertingkat. Dalam *Deep Learning*, *Artificial Neural Networks* (ANN) dikonfigurasi dalam struktur berlapis-lapis, yang sering kali mencakup puluhan hingga ratusan lapisan. Setiap lapisan pada dasarnya bekerja untuk mengolah dan mengubah informasi sebelum meneruskannya ke lapisan berikutnya, memungkinkan model untuk mempelajari fitur data secara progresif dan lebih mendalam.

Salah satu keunggulan utama dari *Deep Learning* adalah kemampuannya untuk secara otomatis dan secara efektif belajar representasi data yang kompleks, yang sering kali tidak terjangkau oleh metode machine learning tradisional. Beberapa algoritma *Deep Learning* yang populer termasuk *Convolutional Neural Networks* (CNN) yang sangat efektif dalam tugas pengolahan gambar, *Long Short-Term Memory* (LSTM) dan *Recurrent Neural Networks* (RNN) yang menangani data sekuensial seperti teks dan deret waktu, serta *Self Organizing Maps* (SOM) yang membantu dalam visualisasi data berdimensi tinggi. Dengan mendalami struktur dan fungsi dari berbagai lapisan ini, *Deep Learning* memfasilitasi penciptaan model yang tidak hanya meniru, tetapi dalam beberapa kasus, dapat melebihi kinerja manusia dalam tugas-tugas spesifik [7].

2.3 Chatbot

Chatbot adalah perangkat lunak komputer yang diciptakan untuk meniru percakapan intelektual berbasis teks dan audio dengan satu atau lebih manusia. Chatbot juga dapat diartikan sebagai sistem komputer yang memungkinkan manusia berinteraksi dengan komputer menggunakan bahasa alami manusia [8].

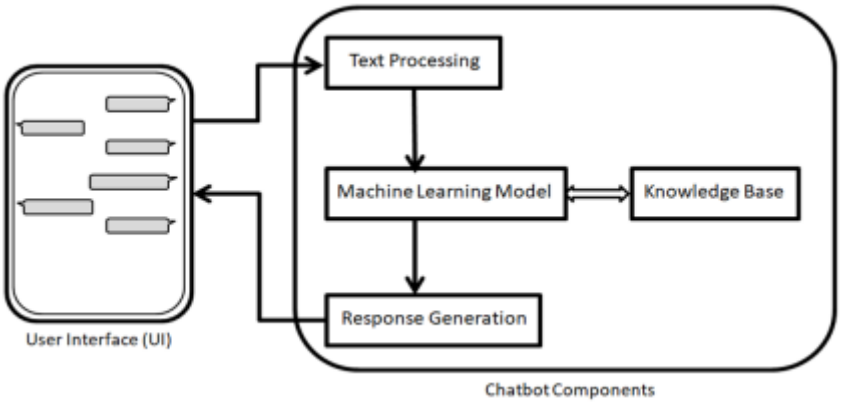
Kata "chatbot" berasal dari dua kata, yaitu "chat" yang dapat dianggap sebagai aktivitas komunikasi media tertulis di dunia komputer, dan "bot", yang merupakan program komputer dengan beberapa titik data yang, ketika diberi input, akan menghasilkan output sebagai respons [3].

Chatbot, sebagai bagian dari evolusi interaksi manusia-komputer, telah diimplementasikan secara luas dalam berbagai sektor, termasuk pendidikan. Dalam konteks akademik, chatbot digunakan untuk memfasilitasi akses informasi yang lebih efisien bagi siswa dan staf. Penelitian oleh Adamopoulou dan Moussiades (2020) menunjukkan bahwa chatbot dapat meningkatkan keterlibatan siswa dan memberikan bantuan administratif secara real-time yang mengurangi beban kerja staf akademik [9]. Ada dua jenis utama chatbot: yaitu chatbot berbasis rule-based dan chatbot contextual.

Chatbot berbasis aturan (rule-based), yang juga dikenal sebagai chatbot skrip, adalah jenis chatbot paling sederhana yang beroperasi seperti pohon keputusan dengan pernyataan if/then (jika pengguna mengatakan/memilih x, maka bot merespons dengan y). Chatbot dasar dalam kategori ini sangat bergantung pada menu atau tombol daripada membiarkan pengguna mengetik permintaan mereka. Pengguna memilih opsi yang disediakan, dan bot merespons dengan tindakan atau jawaban yang telah ditentukan berdasarkan pilihan tersebut. Cara kerjanya mirip dengan sistem IVR telepon, di mana pengguna memilih nomor untuk menandakan tindakan yang ingin mereka lakukan, dan kemudian diarahkan ke informasi atau orang yang tepat.

Sebaliknya, chatbot kontekstual jauh lebih canggih dibandingkan dengan chatbot berbasis aturan. Chatbot ini menggunakan Pemrosesan Bahasa Alami (NLP) untuk secara akurat memahami ucapan atau teks pengguna. Dinamakan kontekstual karena bot ini dapat memahami konteks percakapan serta maksud di baliknya, bukan hanya memindai teks untuk menemukan kata kunci tertentu. Ini

mirip dengan cara kerja asisten virtual seperti Siri atau Alexa. Misalnya, jika pengguna bertanya “Jam berapa sekarang di Bandung?” maka bot dapat memberikan waktu yang tepat. Jika kemudian pengguna bertanya, "Bagaimana dengan Jakarta?" bot dapat menyimpulkan bahwa pengguna masih bertanya tentang waktu dan merespons dengan benar, meskipun kata "waktu" tidak disebutkan dalam pertanyaan kedua. [2]. Berikut contoh ilustrasi dari alur keja chatbot kontekstual pada gambar 2.1.[10]



Gambar 2. 1 Alur Kerja Chatbot [10]

Gambar 2.1 diatas merupakan gambar yang menunjukkan alur kerja chatbot di mana pengguna akan diberikan ui yang kemudian mengirimkan input ke text processing untuk diproses. Selanjutnya inputan tadi akan diproses oleh model dari machine learning dan dilakukan juga pencarian terhadap knowledge base dari modeal tersebut. Jika proses tersebut sudah dilakukan tahap selanjutnya informasi yang tadi diproses dan dicari dalam model dan knowledge base akan dikembalikan lagi kepada pengguna dengan bentuk respons jawaban

2.4 Text Pre-processing

Text pre-processing adalah langkah penting dalam pengolahan data teks sebelum digunakan dalam model pembelajaran mesin. Proses ini bertujuan untuk membersihkan dan menyesuaikan data teks agar sesuai dengan kebutuhan analisis dan model yang akan dibangun. Beberapa teknik dari *text pre-processing* yang umum digunakan meliputi:

2.4.1. Cleaning

Cleaning ini mencakup penghapusan tanda baca dan karakter khusus dari teks, karena elemen-elemen ini sering kali tidak memiliki kontribusi

signifikan. Dengan menghapus tanda baca dan karakter khusus, teks menjadi lebih bersih dan seragam, memudahkan model dalam mengidentifikasi pola dan memahami isi teks dengan lebih baik.

Sebagai contoh, kalimat "Hello, world! How's everything?" akan diubah menjadi "Hello world Hows everything", sehingga mengurangi gangguan dari simbol-simbol yang tidak relevan dan meningkatkan efisiensi analisis selanjutnya.

2.4.2. Case Folding

Proses mengubah semua karakter huruf pada sebuah kalimat menjadi huruf kecil atau huruf besar disebut *Case Folding*. *Case Folding* yang dilakukan pada penelitian ini yaitu mengubah seluruh dataset menjadi huruf kecil. Huruf kapital biasanya terdapat pada setiap awal kalimat seperti "Udara di tempat ini sangat sejuk", menggunakan *Case Folding* kalimat tersebut berubah menjadi "udara di tempat ini sangat sejuk". Tujuan utama *Case Folding* adalah agar kata "udara" tidak lagi mempunyai dua bentuk yaitu, "Udara", dan "udara", namun hanya memiliki satu bentuk huruf kecil saja.

2.4.3. Tokenization

Tokenization ialah tahap proses memangkas kalimat menjadi kata demi kata, proses ini melakukan pemangkasan berdasarkan spasi yang ada pada kalimat. Sebagai contoh, kalimat "Angklung berasal dari Jawa Barat" maka teks tersebut akan dipecah menjadi ['Angklung', 'berasal', 'dari', 'Jawa', 'Barat']. Sehingga proses ini memungkinkan analisis dan pemrosesan teks secara lebih efektif dan efisien. Proses ini membantu dalam berbagai tugas *Natural Language Processing* (NLP) seperti pengindeksan dokumen, pemrosesan teks, analisis sentimen, dan penerjemahan mesin. okenization adalah

2.4.4. Stopword removal

Stopword removal merupakan tahap pengambilan kata-kata penting dan membuang kata-kata yang dianggap tidak penting. Cara untuk membuang kata yang tidak penting disebut *stopword removal*. *Stopword removal* bertujuan untuk menghilangkan kata-kata yang sering muncul

namun tidak memiliki kontribusi dalam proses analisis data. *Stopword removal* berusaha memperkecil dimensi data dan mempercepat waktu komputasi Contoh kata yang tidak penting di bahasa Indonesia seperti kata “dan”,”yang”, ”di”, ”ke”

2.4.5. Stemming

Stemming merupakan suatu proses yang terdapat dalam sistem yang mentransformasikan kata-kata yang terdapat dalam suatu dokumen ke kata-kata akarnya (*root word*) dengan menggunakan aturan-aturan tertentu. Sebagai contoh, kata bersama, kebersamaan, menyamai, akan distem ke root wordnya yaitu “sama”. Untuk kamus stemming yang digunakan dalam penelitian ini ialah Algoritma Stemming Sastrawi. Algoritma *stemming* Sastrawi merupakan pengembangan dari algoritma Nazief dan Adriani Sastrawi sendiri merupakan sebuah *stemmer library* yang tersedia untuk bahasa pemrograman Python, Java, C, Go, PHP, dan Ruby. Proses *stemming* menggunakan algoritma ini sangat bergantung pada kamus kata dasar. Kata dasar pada Sastrawi berasal dari kateglo.com dengan beberapa modifikasi. Langkah-langkah dalam algoritma ini adalah sebagai berikut [11]:

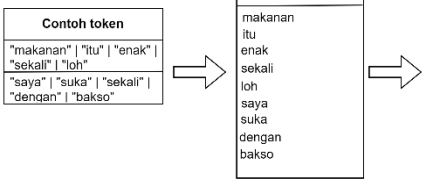
1. Melakukan pemeriksaan apakah kata yang akan di*stemming* ada dalam kamus kata dasar atau tidak. Jika ada, maka proses *stemming* berhenti pada langkah ini.
2. Jika kata tidak ada dalam kamus, artinya kata tersebut merupakan kata berimbuhan. Kemudian menghilangkan kata akhiran “-lah”, “-kah”, “-ku” “-mu”, “-nya”, “tah” atau “-pun”.
3. Menghilangkan kata imbuhan akhiran “-i”, “-kan”, “-an”, kemudian hapus kata imbuhan awalan “be-”, “di-”, “ke-”, “me-”, “pe-”, “-se”, dan “te-”.
4. Jika kata dasar yang dihasilkan dari langkah sebelumnya tidak terdapat di kamus, maka kata tersebut dicek apakah termasuk pada tabel keambiguan atau tidak.

Jika seluruh proses langkah 1-4 gagal dilakukan, maka algoritma akan mengembalikan kata aslinya

2.5 *One hot encoding*

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

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



One hot encoding									
corpus	makanan	itu	enak	sekali	loh	saya	suka	dengan	bakso
makanan	1	0	0	0	0	0	0	0	0
itu	0	1	0	0	0	0	0	0	0
enak	0	0	1	0	0	0	0	0	0
sekali	0	0	0	1	0	0	0	0	0
loh	0	0	0	0	1	0	0	0	0
saya	0	0	0	0	0	1	0	0	0
suka	0	0	0	0	0	0	1	0	0
dengan	0	0	0	0	0	0	0	1	0
bakso	0	0	0	0	0	0	0	0	1

Gambar 2. 2 Proses *one hot encoding*

Pada *proses one-hot encoding* ini, jika terdapat token yang menempati posisi elemen dalam *vocabulary*, maka akan bernilai 1, dan sisanya akan bernilai 0 berdasarkan panjang dari kosa kata (*vocabulary*). Dalam contoh diatas terdapat dua token “makanan”, maka token “makanan” bernilai 1 jika menempati posisi elemen token yang terdapat dalam vocabulary dan seterusnya untuk masing-masing token.

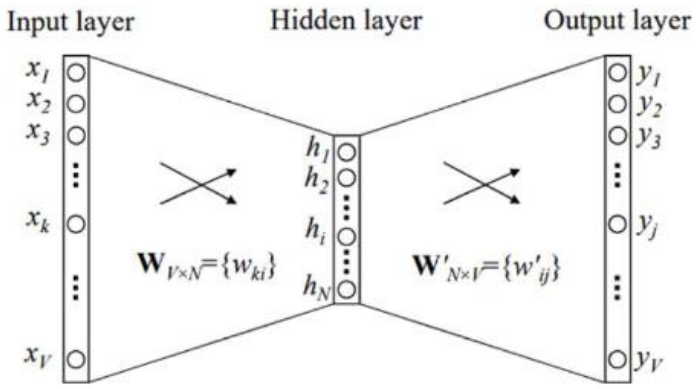
2.6 Word Embedding

Word Embedding adalah representasi kata yang dihasilkan dari teks atau dokumen berdasarkan konteks kalimat. Setiap kata direpresentasikan sebagai vektor yang berisi angka-angka riil. Salah satu tujuan utama dari penggunaan vektor kata ini adalah untuk mengukur kemiripan antar kata berdasarkan nilai vektor masing-masing

Menurut Mikolov [12]ada dua arsitektur model utama untuk menentukan nilai vektor suatu kata, yaitu *continuous bag-of-words* (CBoW) dan skip-gram (SG) . Dengan nilai vektor ini, kita bisa menghitung kemiripan antar kata atau pasangan kata. Kedua model tersebut dapat dibangun dengan menggunakan jaringan saraf tiruan feedforward yang sederhana dan mempertimbangkan konteks kata. Perbedaannya adalah, CBoW dioptimalkan untuk memprediksi kata berdasarkan kata-kata di sekitarnya atau konteksnya, sedangkan *skip-gram* lebih dioptimalkan untuk memprediksi konteks dari kata yang diberikan dan memprediksi kata-kata di sekitarnya. Untuk *word embbeding* pada penelitian ini penulis menggunakan metode Word2Vec CBOW

Word2Vec mengonversi kata menjadi vektor yang dapat menangkap makna semantik dari kata tersebut. Model ini adalah contoh dari pembelajaran tanpa

pengawasan (*unsupervised learning*) yang menggunakan jaringan saraf dengan lapisan tersembunyi (*hidden layer*) dan lapisan sepenuhnya terhubung (*fully connected layer*). Ukuran matriks bobot pada setiap lapisan ditentukan oleh jumlah kata dalam korpus dikalikan dengan jumlah *neuron* tersembunyi pada lapisan tersembunyi. Matriks bobot pada lapisan tersembunyi dari model yang telah dilatih berfungsi sebagai alat untuk mengonversi kata menjadi vektor. Matriks bobot ini berperan seperti tabel pencarian, di mana setiap baris mewakili kata dan setiap kolom mewakili vektor dari kata tersebut. Word2Vec mengandalkan informasi lokal dari bahasa, di mana makna yang dipelajari dari sebuah kata dipengaruhi oleh kata-kata di sekitarnya. Model ini mampu mempelajari pola linguistik sebagai hubungan linear antara vektor kata. [13]. Berikut arsitektur CBOW nya dapat dilihat pada gambar 2.3 [14],



Gambar 2. 3 Arsitektur CBOW dengan satu kata konteks [14]

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

Dimulai dari perhitungan hidden layer (h), namun sebelumnya menentukan matriks bobot W dengan dimensi V × N. Setiap baris dari matriks bobot W merepresentasikan vektor berukuran N-dimensi dari kata terkait di *input layer*. Jadi baris i dari matriks bobot W adalah vwT. Perhitungannya perkalian matriks antara

matriks bobot W *transpose* dengan *input layer* yang berupa *one-hot encoding*. Nilai *hidden layer* didapatkan dengan persamaan (2.6.1)

$$h = w^T x := v_{w_i}^T \quad \text{.....(2.6.1)}$$

Keterangan : w^T adalah matriks bobot W *transpose*

x adalah *input layer*

v_{w_I} adalah representasi vector dari input kata (wI)

Lalu setelah hidden layer dihitung, hitung *output* u, namun matriks bobotnya berbeda dari yang sebelumnya, yaitu bobot matriksnya adalah $W' = \{w'_{ij}\}$ ukurannya adalah N × V. Perhitungannya perkalian matriks antara kolom j bobot matriks W' dengan *hidden layer* pada persamaan (2.6.2)

$$u_j = {v'_{wj}}^T h \quad \text{..... (2.6.2)}$$

Keterangan : v' adalah kolom j dari matriks bobot W *transpose*

h adalah *input layer*

Lalu menghitung output dari unit j di output layer dengan fungsi *softmax* untuk mendapatkan distribusi posterior kata-kata. Catatan bahwa vw adalah representasi baris dari matriks bobot W', ${v'}_w$ representasi kolom dari matriks bobot W' namun analisis berikutnya akan berubah. Nilai output *softmax* didapatkan dengan persamaan (2.8.3).

$$p(w_j|w_I) = y_i = \frac{\exp(u_j)}{\sum_{j'=1}^{V} \exp(u_{j'})} \quad \text{.....(2.6.3)}$$

Keterangan : u_j output u unit ke-j

U_j output dari unit j di *output layer*

V: jumlah *Vocabulary*

Setelah didapatkan hasil keluarannya, update perhitungan *hidden* dan bobot output. Dengan menerapkan *backpropagation* dan melakukan turunan. Tujuan dari pelatihan adalah untuk maksimalkan persamaan (2.6.3) probabilitas bersyarat mengamati output kata sebenarnya dari j* mengingat *input* kata konteks *wI*

berkaitan dengan bobot.Namun sebelum itu perlu menghitung *loss function* atau *error* yang tujuannya untuk meminimumkan nilai E dengan persamaan (2.6.4).

$$max\ p(w_o|w_I) = u_{j^*} - log \sum_{j'=1}^{v} \exp(u_{j'}) \coloneqq -E \quad \text{.....(2.6.4)}$$

Keterangan : wI adalah *input* konteks kata

wO adalah *output* kata sebenarnya atau target kata

$uj*$ adalah *output* u kata sebenarnya

uj' adalah *output* u kata dalam *vocabulary* selain target kata

E adalah loss function

Lalu menghitung turunan bobot untuk memperbarui bobot pada *hidden* dan *output layer*. Turunan parsial E yang berkaitan dengan input unit ke-j dari u_j dapat dihitung dengan persamaan (2.6.5).

$$\frac{\partial E}{\partial u_j} = y_j - t_j \coloneqq e_j \quad \text{.....(2.6.5)}$$

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

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

e_j adalah prediksi *error* di *output layer*

Selanjutnya menghitung turunan parsial E pada W'_{ij} untuk memperoleh gradien yang terdapat pada bobot *hidden* dan *output*. Perhitungannya dengan persamaan (2.6.6).

$$\frac{\partial E}{\partial w'_{ij}} = \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial w'_{ij}} = e_j \otimes h_i \quad \text{.....(2.6.6)}$$

Keterangan : ej adalah prediksi error

hi adalah unit ke-i pada hidden layer

$\otimes$ adalah outer product atau tensor product

Menggunakan *stochastic gradient descent* untuk memperoleh bobot baru (W'). Sebagai catatan bahwa harus menulusuri setiap kemungkinan kata yang terdapat pada vocabulary. Periksalah output dari probabilitas y_j dan bandingkan dengan output yang diharapkan (t_j) 0 atau 1, jika y_j>t_j maka akan melakukan pengurangan proporsi pada vektor hidden layer (h) dari v'_{wj}, (sehingga membuat v'_{wj}, lebih jauh dari v_{wI},), jika y_j<$t_j$yang mana benar hanya jika t_j=1($w_j = w_o$), maka akan menambahkan beberapa h ke v'_{wo}, sehingga membuat v'_{wo}mendekat

ke v_{wI}. Dan jika nilai $y_j$terlalu dekat ke t_j, maka berdasarkan perhitungan baru, perubahan bobot baru akan sangat sedikit atau kecil. Perhitungan memperbarui bobot W' dilakukan dengan persamaan (2.6.7).

$$w_{ij}'^{(baru)} = w_{ij}'^{(lama)} - \eta . e_j . h_i \quad \text{.....(2.6.7)}$$

Keterangan : η adalah *learning rate*

w'_{ij}adalah bobot W' dengan i baris dan j kolom

$h_i$adalah *hidden layer*

e_j adalah prediksi *error*

Kemudian melakukan pembaruan perhitungan bobot antara *input* dan *hidden layer* (W). Mengambil nilai turunan parsial dari E pada output dari hidden layer. Perhitungan turunan E terhadap *hidden layer* dapat dilakukan dengan persamaan (2.6.8).

$$\frac{\partial E}{\partial h_i} = \sum_{j=1}^{v} \frac{\partial E}{\partial uj} . \frac{\partial u_j}{\partial h_i} = \sum_{j=1}^{v} e_j . w'_{ij} = EH_i \quad \text{.....(2.6.8)}$$

Keterangan : h_i adalah *hidden layer*

w'_{ij} adalah matriks bobot W'

h_i adalah prediksi *error* dari kata ke-j pada *output layer*

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

Perhitungan dari turunan E terhadap bobot W sama dengan perhitungan produk *tensor* antara x dan EH dilakukan dengan persamaan (2.6.9)

$$\frac{\partial E}{\partial W} = x \otimes EH = xEH^T \quad \text{.....(2.6.9)}$$

Keterangan : x adalah input vektor bukan nol

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

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

Lalu perhitungan untuk mendapatkan bobot baru v_{wI}, dikarenakan hanya satu nilai x yang bukan nol dan nilai $\frac{\partial E}{\partial W}$ ini sama dengan nilai EH^T jadi menghitung bobot baru W dengan persamaan (2.6.10)

$$v_{wI}^{(baru)} = v_{wI}^{(lama)} - \eta EH^T \quad \text{.....(2.6.10)}$$

Keterangan : v_{wI} adalah baris bobot W, input vektor hanya merupakan kata konteks dan hanya baris dari W yang turunannya bernilai bukan nol

η adalah *learning rate*

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

2.7 K-Fold Cross Validation

Dalam melatih suatu program atau model *learning*, dibutuhkan metoda untuk memvalidasi hasil evaluasi pembelajaran yang dilakukan oleh model. Karena, ketika model menunjukkan performa baik saat dilatih, belum tentu model tersebut bekerja secara optimal pada saat dihadapkan di dunia nyata. Pada kasus tersebut, cross validation datang sebagai suatu metoda statistik yang untuk memvalidasi hasil evaluasi performa suatu model/program *Machine Learning*.

Cross-validation adalah prosedur resampling yang digunakan untuk mengevaluasi model pembelajaran mesin pada sampel data yang terbatas. Prosedur ini memiliki parameter tunggal yang disebut k yang mengacu pada jumlah kelompok yang sampel data yang diberikan akan dibagi menjadi. Dengan demikian, prosedur ini sering disebut *k-fold cross-validation*. Ketika nilai tertentu untuk k dipilih, itu dapat digunakan sebagai pengganti k dalam referensi ke model, seperti k = 5 menjadi 5-*fold cross-validation.*

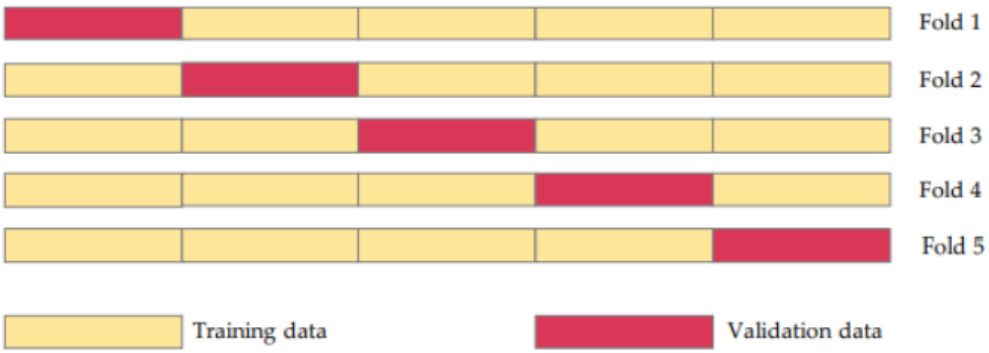
Validasi silang terutama digunakan dalam *machine learning* untuk memperkirakan keterampilan model *machine learning* pada data tak terlihat. Yaitu, menggunakan sampel terbatas untuk memperkirakan bagaimana model diharapkan tampil secara umum ketika digunakan untuk membuat prediksi pada data yang tidak digunakan selama pelatihan model.

Cara kerja *K-Fold Cross Validation* adalah sebagai berikut ;

1. Seluruh data dibagi menjadi K bagian.

2. Fold ke -1 adalah ketika bagian ke-1 menjadi data uji (*testing data*) dan sisanya menjadi data latih (*training data*). Selanjutnya, hitung akurasi berdasarkan porsi data tersebut.
3. Fold ke-2 adalah ketika bagian ke-2 menjadi data uji (*testing data*) dan sisanya menjadi data latih (*training data*). Selanjutnya, hitung akurasi berdasarkan porsi data tersebut.
4. Demikian seterusnya hingga mencapai fold ke-K.
5. Hitung rata-rata akurasi dari N buah akurasi di atas. Rata-rata akurasi ini menjadi akurasi final.

Adapun besaran k seringkali sebesar 5 dan 10, angka tersebut dipilih karena dapat membagi jumlah jumlah data sama rata dan dianggap cukup untuk memberikan estimasi yang secara statistik dianggap akurat. Adapun kondisi ekstrim dimana k = n (jumlah data) sering juga disebut sebagai *Leave-One-Out Cross-Validation* (LOOCV).



Gambar 2. 4 Ilustrasi Dataset pada pelatiham model dengan K-Fold Cros Validation

Dalam penelitian ini juga, metrik yang digunakan untuk mengukur performansi model adalah matriks *loss* dan matriks akurasi. *Loss* (kerugian) adalah nilai fungsi kerugian yang dihitung selama evaluasi. Fungsi *loss* yang digunakan adalah *sparse_categorical_crossentropy*, yang cocok untuk masalah klasifikasi dengan label yang disandikan sebagai angka bulat / integer. Sedangkan untuk metrics *accuracy* (akurasi) adalah proporsi prediksi yang benar dibandingkan dengan total prediksi yang dilakukan oleh model. Akurasi dihitung sebagai metrik performa utama untuk mengevaluasi seberapa baik model dalam mengklasifikasikan data uji.

2.8 Padding

Setiap jaringan saraf membutuhkan input dengan ukuran dan bentuk yang seragam. Namun, dalam pemrosesan teks sebagai input untuk model seperti LSTM, panjang kalimat tidak selalu sama. Secara alami, sebuah kalimat bisa panjang atau pendek. Oleh karena itu, kita perlu menyesuaikan panjang teks agar seragam, yang disebut dengan *padding*.

Padding ini diperlukan karena setiap kalimat dalam teks memiliki jumlah kata yang berbeda. Dalam penelitian ini, panjang maksimal dari data akan dijadikan acuan. Jika ada data yang tidak memiliki panjang yang sama dengan acuan, maka akan ditambahkan angka 0 agar sesuai. Proses ini bertujuan agar saat masuk ke pelatihan model, data dapat diproses secara konsisten. Berikut contoh penggunaan padding pada digambarkan pada tabel 2.1. dibawah ini.

Tabel 2. 1 Contoh penggunaan Padding

Data Asli	**Diubah kedalam Sequence**	**Padding**
[‘halo’, ‘perkenalkan’, 'nama', 'saya', 'ilham', 'rafa', 'nurbakti']	[2,3, 4, 5, 7, 8, 1]	[2,3, 4, 5, 7, 8 ,1]
[‘aku’, ‘tinggal’, ‘di’, ‘bandung’]	[2, 5, 9, 10]	[0, 0, 0, 2, 5, 9, 10]

Dalam contoh tersebut, data pertama memiliki panjang urutan 7 dan akan dijadikan acuan karena merupakan data terpanjang. Data lain yang memiliki panjang kalimat kurang dari 7 akan ditambahkan nol di awal untuk menyamakan panjangnya dengan data acuan.

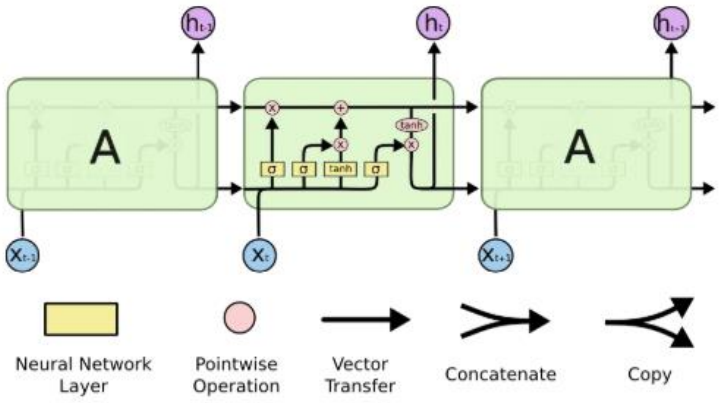
2.9 Label Encoding

Label Encoding adalah proses mengubah data kategori seperti karakter atau teks menjadi data *numerik* sesuai dengan label data yang digunakan. *Encoding* ini mengubah data teks pada kolom tag data menjadi data *numerik* pada tahap ini menggunakan nilai integer untuk nantinya bisa dipakai dalam *Loss Function* dengan metode *Sparse Categorical Cross Entropy*.

2.10 *Long Short-Term Memory (LSTM)*

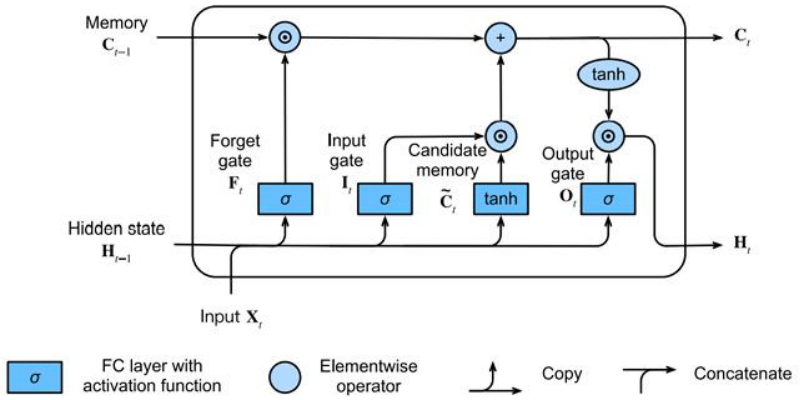
Long Short-Term Memory (LSTM) merupakan salah satu jenis arsitektur jaringan saraf tiruan yang merupakan pengembangan dari *Recurrent Neural Network* (RNN) sebelumnya. LSTM telah dirancang untuk mengatasi kendala yang ditemui dalam RNN salah satunya masalah depedensi jangka panjang. Dalam satu neuron, jaringan pada LSTM mempunyai empat lapisan yang saling berinteraksi, yaitu dua lapisan tanh dan tiga lapisan sigmoid. Fungsi aktivasi sigmoid adalah fungsi yang mengembalikan nilai dalam rentang nol dan satu. Sedangkan fungsi aktivasi tanh mengembalikan nilai dalam rentang negatif satu hingga satu. [15]. Melalui penambahan ini, LSTM dapat mempertahankan informasi kontekstual untuk periode yang lebih lama.

Proses komputasi dalam LSTM terdiri dari empat komponen *gate*, yakni *forget gate, input gate, cell state candidate, dan output gate*. Misalkan x adalah *input sequence* dan h menjadi *hidden state* yang dihasilkan dari LSTM *layer*, maka hubungan antara x dan h yaitu berdasarkan persamaan seperti gambar berikut [16].



Gambar 2. 5 Arsitektur LSTM [16]

Pada umumnya dalam Neural Network terdapat bobot yang digunakan sebagai parameter dalam pemrosesannya, begitu juga jenis arsitektur Recurrent Neural Network. Namun dalam arsitektur Recurrent Neural Network memiliki bobot berulang atau recurrent weight (U) di setiap jaringannya, hal tersebut dikarenakan dalam pemrosesannya dilakukan secara berulang. Berikut gambar arsitektur lstm secara jelasnya [17], bisa dilihat pada gambar berikut



Gambar 2. 6 Arsitektur LSTM secara detail [17]

Forget gate menentukan informasi mana yang dilupakan dalam *cell state* terakhir, inputnya adalah h_{t-1}, dan x_t, nilai output antara (0,1) . Jika *forget gate* = 1 informasi akan disimpan, jika *forget gate* = 0 informasi akan dilupakan . Hasil *forget gate* dihitung dengan persamaan ke (2.10.1).

$$f_t = \sigma\left(w_f x_t + u_f h_{t-1} + b_f\right) \quad (2.10.1)$$

Input gate menentukan informasi harus diperbarui dalam *cell state* pada waktu saat ini . Candidate *cell state* nantinya akan dimasukkan ke dalam persamaan *Cell state*, nilai candidate *cell state* diaktivasi dengan fungsi *hyperbolic tangent* antara -1 dan 1 . *Cell state* tempat menyimpan informasi yang diberikan dari satu *time step* ke *time step* selanjutnya.Perhitungan *input gate* dengan persamaan (2.10.2), perhitungan *Candidate cell state* menggunakan persamaan (2.10.3), dan perhitungan *cell state* menggunakan persamaan (2.10.4).

$$i_t = \sigma(w_i x_t + u_i h_{t-1} + b_i) \quad (2.10.2)$$

$$\tilde{C}_t = tanh(w_c x_t + u_c h_{t-1} + b_c) \quad (2.10.3)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{C}_t \quad (2.10.4)$$

Output gate menentukan nilai output unit LSTM dihitung dengan persamaan (2.10.5), dan *hidden state* dihitung dari perkalian elemen wise antara output gate dan *cell state* nilai *hidden state* intervalnya -1 dan 1 dengan persamaan (2.10.6)

$$O_t = \sigma(w_o x_t + u_o h_{t-1} + b_o) \quad (2.10.5)$$

$$h_t = o_t \odot \tanh(c_t) \quad (2.10.6)$$

Tabel 2. 2 Keterangan notasi dan definisi persamaan LSTM

Notasi	Definisi
t	*Time step t = 1,2, ...*
$\boldsymbol{x_t}$	Vektor input pada saat *time step* ke t
$\boldsymbol{h_{t-1}}$	*Hidden state* pada *time step* ke t-1
$\boldsymbol{h_t}$	*Hidden state time step* ke t
$\boldsymbol{c_{t-1}}$	*Cell memory time step* ke t-1
$\boldsymbol{c_t}$	*Cell memory time step* ke t
$\boldsymbol{\widetilde{C}_t}$	*Candidate* State pada *time step* ke t
$\boldsymbol{W_c}$	Bobot pada *Candidate* layer untuk input kata
$\boldsymbol{W_i}$	Bobot pada input layer untuk input kata
$\boldsymbol{W_o}$	Bobot pada output layer untuk input kata
$\boldsymbol{W_f}$	Bobot pada forget layer untuk input kata
$\boldsymbol{U_c}$	Bobot pada *Candidate* layer untuk *hidden state time step* sebelum
$\boldsymbol{U_i}$	Bobot pada input layer untuk *hidden state time step* sebelum
$\boldsymbol{U_o}$	Bobot pada output layer untuk *hidden state time step* sebelum
$\boldsymbol{U_f}$	Bobot pada forget layer untuk *hidden state time step* sebelum
*	*Hadamard product*

2.11 Pelatihan *Long Short-Term Memory* (LSTM)

Dalam pemrosesannya model LSTM memiliki proses yang diadaptasi dari *Recurrent Neural Network*. Sama seperti model ANN atau RNN, model LSTM memiliki tiga tahapan dalam pemrosesannya, yaitu *feedforward propagation*, *backpropagation*, dan memperbarui nilai parameter. Dalam model LSTM, proses *backpropagation* dilakukan dengan proses *backpropagation through time* (BPTT), hal tersebut dikarenakan model LSTM melakukan proses perulangan dalam node tersembunyi atau hidden node.

Backpropagation sebagai teknik untuk menghitung gradien dari parameter neural network, metode ini melewati jaringan dengan cara berjalan mundur, dari lapisan *output* menuju lapisan input berdasarkan dengan aturan rantai (*chain rule*) yang ada pada kalkulus. *Backpropagation* menyimpan turunan parsial yang dibutuhkan ketika menghitung gradien terhadap beberapa parameter. Dalam

pelatihan *deep learning, forward propagation* dan *backpropagation* saling berhubungan satu sama lain. *Backpropagation Through Time* (BPTT) adalah sebuah teknik untuk memperbarui parameter yang terdapat pada algoritma jenis RNN [17]. Tujuan dari BPTT ini adalah menghitung dan menyimpan nilai gradien terhadap parameter suatu model dengan menggunakan teknik *chain rule* yang terdapat pada kalkulus [18].

Berikut Langkah pelatihan menggunakan model LSTM [19]

1. Inisialisasi parameter yang akan digunakan, yaitu itu inisialisasi bobot (W) dan bobot berulang (U) dengan bilangan acak, serta inisialisasi bias b.

a. Fase I : Feedforward Propagation

2. Menghitung nilai *forget gate* dengan persamaan 2.10.1 dan menghitung nilai fungsi aktivasi *sigmoid* dengan persamaan (2.12.1).
3. Menghitung nilai *input gate* dengan persamaan 2.10.2 dan menghitung nilai fungsi aktivasi *sigmoid* dengan persamaan (2.12.1).
4. Menghitung nilai *candidate cell state* dengan persamaan 2.10.3 dan menghitung nilai fungsi aktivasi *tanh* dengan persamaan (2.12.3).
5. Menghitung nilai *cell state* dengan persamaan 2.10.4.
6. Menghitung nilai *output gate* dengan persamaan 2.10.5 dan menghitung nilai fungsi aktivasi *sigmoid* dengan persamaan (2.12.1).
7. Menghitung nilai keluaran akhir atau *hidden state* dengan persamaan 2.10.6.
8. Menghitung nilai *Loss* atau nilai error dengan persamaan 2.14.1.

b. Fase II : Backpropagation

1. Menghitung turunan parsial dengan persamaan 2.11.1 sebagai berikut :

$$dy_t = -(Y_{rt} - y_t) \quad \text{.....(2.11.1)}$$

Keterangan : Y_{rt}= output label sebenarnya

y_t = output softmax dari *hidden state* LSTM

2. Menghitung turunan bobot dari W_y pada prediksi output LSTM dengan persamaan 2.11.2 berikut ini

$$W_y\, dW_y = \sum_y h_t \otimes dy_t \quad \text{.....(2.11.2)}$$

Keterangan : h_t = *hidden state* token ke -t

dy_t= turunan parsial y_t untuk *output layer*

3. Menghitung bobot batu untuk W_y dengan menggunakan persamaan 2.11.3 sebagai berikut

$$W_{y_{(baru)}} = W_{y_{(lama)}} - \eta \times \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}} \quad (2.11.3)$$

4. Menghitung nilai turunan dari *Hidden State* dengan menggunakan persamaan 2.11.4 sebagai berikut

$$\delta_{ht} = W_y dy_t \quad (2.11.4)$$

5. Menghitung turunan parsial *error* pada *output gate* dengan menggunakan persamaan 2.11.5 berikut

$$\delta_{o_t} = \tanh(c_t) \odot \delta_{h_t} \odot o_t \odot (1 - o_t) \quad (2.11.5)$$

Keterangan : c_t adalah *cell state*

δ_{h_t} adalah turunan parsial hidden state step saat ini

o_t adalah *output gate*

6. Menghitung turunan parsial *error* pada *cell state* dengan menggunakan persamaan 2.11.6 berikut

$$dc_t = (1 - \tanh(c_t)^2) o_t \delta_{o_t} + f_{t+1} \odot \delta_h \quad (2.11.6)$$

Keteranan : c_t adalah *cell state*

δ_{h_t} adalah turunan parsial *hidden state*

δ_{o_t} adalah turunan parsial *output gate*

o_t adalah *output gate*

f_{t+1} adalah *forget gate* pada *time step* 1+t

7. Menghitung turunan Candidate cell state dengan mengunakan persamaan 2.11.7 berikut

$$\delta_c = \mathrm{i_t} \odot dc_t \odot (1 - \tilde{C}_t^{\,2}) \quad (2.11.7)$$

Keterangan : $\mathrm{i_t}$ adalah *input gate*

dc_t adalah turunan parsial pada *cell state*

$\tilde{c}_t$ adalah *candidate cell state*

8. Menghitung turunan parsial *error* terhadap *input gate* dengan dilakukan persamaan 2.11.8 sebagai berikut

$$\delta_i = \tilde{C}_t \odot dc_t \odot i_t \odot (1 - i_t) \quad (2.11.8)$$

Keterangan : $\tilde{c}_t$ adalah *candidate cell state*

dc_t adalah turunan parsial pada cell state

$\mathrm{i_t}$ adalah *input gate*

9. Menghitung turunan parsial error terhadap *forget gate* dengan persamaan 2.11.9 sebagai berikut

$$\delta_f = \mathrm{c_t} \odot dc_t \odot f_t \odot (1 - f_t) \quad \text{.....(2.11.9)}$$

Keterangan : c_t adalah *cell state*

dc_t adalah turunan parsial pada cell state

f_t adalah *forget gate*

10. Menghitung turunan dari bobot W dari setiap gates atau gerbang LSTM dengan persamaan (2.11.11), (2.11.12), (2.11.13), dan (2.11.14) sebagai berikut ini.

$$\frac{\partial E}{\partial W_{f_t}} = \sum_t \delta_{f_t} \otimes x_t \quad \text{.....(2.11.11)}$$

$$\frac{\partial E}{\partial W_{i_t}} = \sum_t \delta_{i_t} \otimes x_t \quad \text{.....(2.11.12)}$$

$$\frac{\partial E}{\partial W_{\tilde{C}_t}} = \sum_t \delta_{\tilde{C}_t} \otimes x_t \quad \text{.....(2.11.13)}$$

$$\frac{\partial E}{\partial W_{o_t}} = \sum_t \delta_{o_t} \otimes x_t \quad \text{.....(2.11.14)}$$

Keterangan : δ_{f_t} adalah turunan parsial terhadap *forget gate*

δ_{i_t} adalah turunan parsial terhadap *input gate*

$\delta_{\tilde{C}_t}$ adalah turunan parsial terhadap *candidate cell state*

δ_{o_t} adalah turunan parsial terhadap *output gate*

x_t adalah *vector* kata inputan pada *time step* saat ini

11. Menghitung turunan dari bobot U dari setiap gates atau gerbang LSTM dengan persamaan (2.11.15), (2.11.16), (2.11.17), dan (2.11.18) sebagai berikut ini.

$$\frac{\partial E}{\partial U_f} = \sum_t \delta_{f_t} \otimes h_{t-1} \quad \text{.....(2.11.15)}$$

$$\frac{\partial E}{\partial U_i} = \sum_t \delta_{i_t} \otimes h_{t-1} \quad \text{.....(2.11.16)}$$

$$\frac{\partial E}{\partial U_{\tilde{C}}} = \sum_t \delta_{\tilde{C}_t} \otimes h_{t-1} \quad \text{.....(2.11.17)}$$

$$\frac{\partial E}{\partial U_o} = \sum_t \delta_{o_t} \otimes h_{t-1} \quad (2.11.18)$$

Keterangan : δ_{f_t} adalah turunan parsial terhadap *forget gate*

δ_{i_t} adalah turunan parsial terhadap *input gate*

$\delta_{\tilde{C}_t}$ adalah turunan parsial terhadap *candidate cell state*

δ_{o_t} adalah turunan parsial terhadap *output gate*

h_{t-1} adalah *hidden state* pada *time step* t - 1

12. Menghitung turunan dari bias pada masing masing *gate* atau gerbang LSTM dengan menggunakan persamaan (2.11.19), (2.11.20), (2.11.21), dan (2.11.22) sebagai berikut.

$$\frac{\partial E}{\partial b_f} = \sum_t \delta_{f_t} \quad (2.11.19)$$

$$\frac{\partial E}{\partial b_i} = \sum_t \delta_{i_t} \quad (2.11.20)$$

$$\frac{\partial E}{\partial b_{\tilde{C}}} = \sum_t \delta_{\tilde{C}_t} \quad (2.11.21)$$

$$\frac{\partial E}{\partial b_O} = \sum_t \delta_{o_t} \quad (2.11.22)$$

Keterangan : δ_{f_t} adalah turunan parsial terhadap *forget gate*

δ_{i_t} adalah turunan parsial terhadap *input gate*

$\delta_{\tilde{C}_t}$ adalah turunan parsial terhadap *candidate cell state*

δ_{o_t} adalah turunan parsial terhadap *output gate*

c. Fase III : Pembaruan Bobot

13. Menghitung nilai perubahan bobot W dengan persamaan 2.19.

2.12 Fungsi Aktivasi

Fungsi aktivasi adalah fungsi yang digunakan dalam jaringan saraf untuk menghitung jumlah input dan bias yang terbobot. Hal ini memanipulasi data yang disajikan melalui beberapa pemrosesan gradien, biasanya *Gradient Descent* dan kemudian menghasilkan output untuk jaringan saraf, yang berisi parameter dalam data. Fungsi aktivasi dapat berupa linier atau non-linier tergantung pada fungsi yang merepresentasikannya, dan digunakan untuk mengontrol output dari jaringan saraf luar.

2.12.1. *Sigmoid Function*

Sigmoid merupakan fungsi aktivasi non linier yang digunakan dalam jaringan saraf. Fungsi Sigmoid mengubah range nilai input x menjadi nilai antara 0 dan 1. Fungsi Sigmoid ditunjukkan oleh persamaan 2.12.1. $S(x)$ akan menghasilkan sebuah kurva dalam rentang 0-1 pada sumbu y. Jika x adalah bilangan 17 positif sangat besar, maka nilai $e\ {-x}$ memiliki nilai mendekati 0 yang menghasilkan output mendekati 1. Sedangkan jika x adalah bilangan negatif sangat besar, maka nilai $e\ {-x}$ memiliki nilai yang besar dan menghasilkan output mendekati 0.

$$f(x) = \frac{1}{(1+e^{-x})} \quad \text{.....(2.12.1)}$$

Keterangan :

$fsigmoid(x)$ = fungsi sigmoid dari x

e = epsilon

2.12.2. *Rectified Linear Unit (ReLU)*

Fungsi ReLU merupakan salah fungsi aktivasi yang telah terbukti menjadi fungsi yang sering digunakan dalam *Deep Learning*. Hal ini dikarenakan fungsi ReLU menawarkan kinerja dan generalisasi yang lebih baik dalam *Deep Learning* dibandingkan dengan fungsi aktivasi sigmoid dan tanh. Aktivasi ReLU akan memberikan keluaran 0 ketika $x < 0$ dan merepresentasikan fungsi linier ketika $x \geq 0$. Fungsi aktivasi ReLU melakukan operasi ambang batas untuk setiap 18 elemen input di mana nilai kurang dari nol diatur ke nol sehingga ReLU dinotasikan sebagai persamaan (2.12.2) sebagai berikut:

$$f(x) = \max(0, x) = \begin{cases} x_i, if\ x_i \geq 0 \\ 0, if\ x_i < 0 \end{cases} \quad \text{.....(2.12.2)}$$

Range dari fungsi aktivasi ReLU adalah [0, ∞], semua input bernilai negatif pada fungsi aktivasi ReLU akan menjadi nilai 0. Fungsi ini memperbaiki nilai input yang kurang dari nol sehingga diubah menjadi nol dan menghilangkan masalah gradien (*vanishing gradient*) yang diamati pada jenis aktivasi sebelumnya.

2.12.3. *Hyperbolic Tangent Function (Tanh)*

Fungsi aktivasi tanh merupakan salah satu fungsi aktivasi yang digunakan untuk kasus multi klasifikasi. Kelemahan fungsi aktivasi tanh adalah tidak

menyelesaikan vanishing gradient yang umum terjadi pada sigmoid. Fungsi aktivasi tanh ditunjukkan oleh persamaan 2.12.3. Jika x merupakan bilangan negatif sangat besar, maka nilai tanh akan mendekati -1, ketika nilai x merupakan bilangan positif sangat besar, maka nilai tanh mendekati 1.

$$f(x) = \left(\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}\right) \quad \text{..... (2.12.3)}$$

Keterangan:

$ftanh(x)$ = fungsi tanh dari x

e = epsilon

2.12.4. *Softmax Function*

Fungi aktivasi *Softmax* adalah jenis dari fungsi aktivasi yang digunakan dalam komputasi saraf. Ini digunakan untuk menghitung distribusi probabilitas dan vektor bilangan real. Fungsi *Softmax* biasanya digunakan sebagai output dari model klasifikasi untuk merepresentasikan distribusi kemungkinan terhadap sejumlah kelas .Fungsi *Softmax* menghasilkan output yang kisaran nilai antara 0 dan 1, dengan jumlah probabilitas sama dengan 1. Fungsi *softmax* dihitung menggunakan persamaan (2.12.4) berikut:

$$f(xi) = \frac{\exp(x_i)}{\sum_j \exp(x_i)} \quad \text{.....(2.12.4)}$$

Keterangan :

x = nilai input lapisan sebelumnya

i,j = indeks unit dan lapisan

Fungsi *softmax* digunakan dalam model multi kelas di mana fungsi ini mengembalikan probabilitas setiap kelas. Fungsi *softmax* sebagian besar muncul di hampir semua lapisan output dari arsitektur *Deep Learning*. Perbedaan utama antara fungsi aktivasi Sigmoid dan *Softmax* adalah Sigmoid digunakan dalam klasifikasi biner sedangkan *Softmax* digunakan untuk tugas klasifikasi multivariant atau multi label

2.13 Adam Optimizers

Optimizers memiliki beberapa jenis untuk model *Deep Learning*, termasuk SGD, Adam, PMSProp, dan lainnya. Penelitian ini menggunakan optimizer *Adam* untuk melatih data. Optimizer *Adam* efektif dalam mengatasi masalah *sparse*

gradient. *Adam* merupakan pengembangan dari *gradient descent stokastik* yang kini banyak digunakan dalam aplikasi *Deep Learning* seperti pemrosesan bahasa alami. *Adam* mengubah bobot melalui *gradient descent*, di mana *learning rate* diubah secara dinamis untuk menghindari jebakan pada minimum lokal. Dibandingkan dengan berbagai jenis optimasi *gradient descent*, *Adam* menunjukkan performa terbaik. *Adam* mudah diimplementasikan, membutuhkan memori relatif kecil, dan efisien dalam komputasi. Algoritma *Adam* menghitung rata-rata bergerak eksponensial dari gradien dan gradien kuadrat [20].

Algoritma optimasi ini memiliki kelebihan dimana sangat efisien secara komputasi dan dalam prosesnya membutuhkan sedikit memori. Berikut adalah proses langkah – langkah algoritma Adam sebagai berikut

1. Menambah t pada setiap iterasi

   $t = t + 1$ .....(2.13.1)

2. Menghitung nilai gradien

   $g_t = \frac{\delta L}{\delta W}$.....(2.13.2)

3. Memperbarui bias pada momen pertama

   $m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$.....(2.13.3)

4. Memperbarui bias pada momen kedua

   $v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot (g_t)^2$.....(2.13.4)

5. Memperbarui koreksi bias pada momen pertama

   $\widehat{m}_t = \frac{m_t}{1-\beta_1^t}$.....(2.13.5)

6. Memperbarui koreksi bias pada momen kedua

   $\hat{v}_t = \frac{v_t}{1-\beta_2^t}$.....(2.13.6)

7. Memperbarui parameter

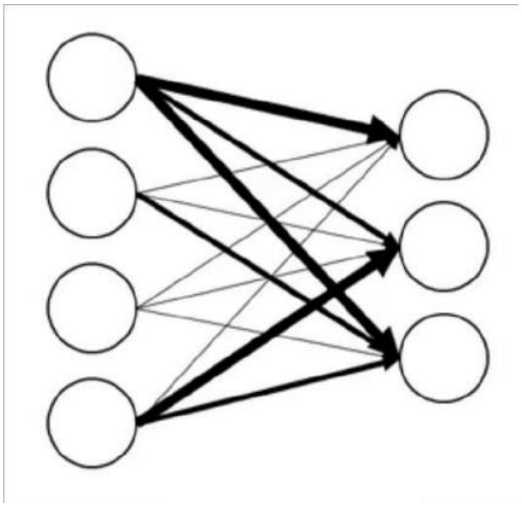
   $\theta_{baru} = \theta_{lama} - \eta \cdot \frac{\widehat{m}_t}{\sqrt{\hat{v}_t + \epsilon}}$.....(2.13.7)

Keterangan :

1. t = iterasi
2. δL = *loss function*
3. δW = bobot
4. m = momen pertama
5. v = momen kedua

6. β_1 , β_2= *exponential decay rates*
7. g = gradien
8. θ = parameter yang diperbaiki
9. η = *learning rate*
10. ϵ = epsilon

Peneliti menentukan nilai default untuk inisialisasi parameter *Adam Optimizer* 0,9 untuk β1, 0,999 untuk β2, dan 10−8 untuk ε. Mereka menunjukkan secara empiris bahwa Adam bekerja dengan baik dalam praktik dan lebih baik dibandingkan dengan algoritma metode pembelajaran adaptif lainnya [21]



Gambar 2. 7 *learning rate* berubah sepanjang training

Pada gambar 2.7 merupakan gambaran *learning rate* yang berubah pada proses training. Karena *learning rate* yang bisa berubah sepanjang proses training ini menyebabkan model akan lebih lebih cepat untuk belajar, karena data yang sering dilatih / diperbarui maka *learning rate* akan semakin kecil yang bisa mencegah terjadinya *osilasi* dan untuk data yang jarang *diupdate* maka *learning ratenya* akan tetap besar sehingga untuk mencapai *konvergen* lebih cepat

2.14 Loss Function

Loss Function merupakan fungsi untuk menghitung kerugian yang terkait dengan semua kemungkinan yang dihasilkan oleh suatu model. Cara kerja *loss function* adalah membandingkan hasil prediksi dari *output layer* dengan target.

Dalam deep learning, loss function ini sangat penting untuk mengukur akurasi model dengan ketentuan nilai tinggi menunjukkan banyak kesalahan prediksi,

sedangkan nilai rendah menunjukkan performa yang baik. Lalu dengan mengukur perubahan nilai loss function, kita dapat menilai efektivitas modifikasi model semakin kecil nilainya, semakin baik performanya.

Terdapat beberapa cara untuk menghitung loss function. Namun, secara umum loss function dapat dikelompokkan menjadi 2, yakni untuk klasifikasi dan regresi.

- **Klasifikasi** - yaitu memprediksi label, dengan mengidentifikasi kategori mana yang dimiliki suatu objek berdasarkan parameter yang berbeda.
- **Regresi** - yaitu memprediksi output yang berkelanjutan, dengan menemukan korelasi antara variabel dependen dan independen.

Pada penelitian ini loss function yang digunakan yaitu *Sparse categorical Cross Entrophy Function* dikarenakan data yang ada dalam penelitian ini berupa klasifikasi yang berisi *multi class label* sehingga fungsi ini yang lebih cocok untuk dipakai. *Categorical Cross Entropy* ini digunakan untuk tugas biner dan multiclass. Jenis *Cross Entropy* ini menghasilkan label berformat integer yang berisi indeks kategori dari kategori yang telah di one hot encoding untuk dipilih mana yang paling cocok. Berikut persamaan dari fungsi *loss* pada rumus (2.14.1) sebagai berikut [22].

$$L(y,\hat{y}) = -\sum_{i=1}^{N} y_i log\hat{y}_1 \quad (2.14.1)$$

Keterangan :

N = banyaknya kelas

y_i = nilai output

$\hat{y}_1$ = nilai prediksi

2.15 Dense Layer

Dalam jaringan saraf, lapisan dense merupakan lapisan yang terhubung secara penuh dengan lapisan sebelumnya, yang berarti bahwa setiap neuron dalam lapisan ini terhubung ke semua neuron di lapisan sebelumnya. Lapisan dense adalah salah satu jenis lapisan yang paling sering digunakan dalam jaringan saraf buatan.

Neuron dalam lapisan dense menerima input dari semua neuron lapisan sebelumnya dan mengolahnya melalui operasi perkalian matriks dengan vektor. Dalam operasi ini, vektor baris yang mewakili output dari lapisan sebelumnya dikalikan dengan vektor kolom dari lapisan dense. Syarat utama untuk perkalian matriks dengan vektor adalah jumlah kolom pada vektor baris harus sama dengan

jumlah kolom pada vektor kolom. Output dari lapisan dense ini kemudian digunakan untuk membuat prediksi berdasarkan informasi yang diterima dari lapisan sebelumnya. Rumus dari output neuron dalam *Dense Layer* bisa dilihat pada persamaan (2.16.1)

$$dense = y = f(w_y h_t + b) \quad \text{.....(2.16.1)}$$

Keterangan : y adalah output neuron

h_t adalah input ke neuron

w_y dalah bobot yang menghubungkan input ke neuron

b adalah bias

f adalah fungsi aktivasi

2.16 User Validation

User validation adalah metode validasi dimana *user* dapat mencoba langsung system yang telah dibangun dan kemudian *user* dapat langsung menilai apakah *output* atau hasil yang diberikan oleh sistem sudah sesuai dengan kebutuhan dari *user*. Selanjutnya akan dihitung akurasi sistem dengan cara membagi jumlah validasi benar dengan total pertanyaan data uji kemudian dikali dengan 100%.

2.17 Akurasi

Pada penelitian ini menggunakan perhitungan akurasi untuk mengetahui tingkat akurasi dari klasifikasi yang dihasilkan oleh model algoritma *long short term memory* (LSTM). Untuk menghitung akurasi dari LSTM itu sendiri dilakukan dengan perhitungan menggunakan rumus (2.18.1) berikut.

$$Akurasi = \frac{Jumlah\ yang\ diklasifikasi\ dengan\ benar}{Jumlah\ sample\ data\ uji} \times 100\% \quad \text{.....(2.18.1)}$$

2.18 Studi Literatur

Penelitian ini memiliki keterkaitan dengan penelitian-penelitian sebelumnya, sehingga terdapat hubungan dalam hal kesamaan dan perbedaan dalam objek yang diteliti. Ringkasan penelitian-penelitian terdahulu dapat dilihat pada tabel 2.3 berikut :

Tabel 2. 3 Tabel Studi Literatur

No	Judul	Metode	Data	Hasil
1.	Implementation of Chatbot for Merdeka Belajar Kampus Merdeka Program Using Long Short-Term Memory [3]	Long Short-Term Memory (LSTM)	Dataset yang digunakan terdiri dari pertanyaan-pertanyaan umum yang diajukan kepada layanan akademik program Merdeka Belajar Kampus Merdeka (MBKM) dan FAQ dari situs web Kampus Merdeka. Pertanyaan ini mencakup definisi, tujuan, persyaratan, jenis program, dan program spesifik MBKM.	Akurasi : 100 % F-1 : 100%
2.	Automated Thai-FAQ Chatbot using RNN-LSTM [23]	RNN - LSTM	2,636 pertanyaan dan jawaban dikategorikan menjadi 80 kelas, kemudian dibagi menjadi tiga set 60% untuk pelatihan, 20% untuk validasi, dan 20% untuk pengujian	akurasi 93.2%
3.	Intelligent Chatbot Adapted	RNN – LSTM	Dataset yang digunakan adalah	akurasi 99%

	from Question and Answer System Using RNN-LSTM Model [24]		Cornell Movie Dialog Corpus, yang berisi sejumlah besar percakapan fiksi yang diambil dari naskah film. Dataset ini mencakup 220,579 percakapan antara 10,292 pasangan karakter film dari 617 film, dengan total 304,713 dialog	
4.	Implementasi Chatbot Pada Pendaftaran Mahasiswa Baru Menggunakan Recurrent Neural Network[1]	RNN - LSTM	Total data berjumlah 300 data lalu dibagi menjadi data latih sebanyak 250 data dan 50 data digunakan sebagai data uji	Akurasi: 99% Presisi: 95% Recall: 92%
5.	Convolutional and Recurrent Neural Networks for Activity Recognition in Smart Environment [5]	CNN & LSTM	Dataset 7 rumah dengan isiannya menjelaskan jumlah penghuni, rumahnya dsb.	Dilakukan beberapa kali perbandingan, dan LSTM memberikan kinerja yang lebih baik daripada CNN dalam kasus umum,

6.	Performance Analysis and Development of QnA Chatbot Model Using LSTM in Answering Questions [25]	LSTM & *Cross Validarion*	4100 pasang pertanyaan dan jawaban yang ada dalam file “data.csv”	Tanpa cross validasi menghasilkan akurasi 91% dan loss 76%. Sedangkan dengan *cross validation* menghasilkan akurasi 96% dan loss 25%.