BAB 2
LANDASAN TEORI

2.1 Sentiment Analysis

Sentimen analisis adalah proses mengekstraksi, memahami dan mengolah data berupa teks yang tidak terstruktur secara otomatis untuk mendapatkan informasi sentimen yang terdapat pada sebuah kalimat pendapat atau opini[10]. Sentimen analisis juga dapat disebut sebagai *opinion mining* yang merupakan studi dari opini orang lain, appraisal serta emosi yang terdapat dalam atribut yang dimiliki[11]. Dalam *opinion mining* ini dapat ditemukan beberapa tingkatan, yakni :

1. *Document Level Sentiment Analysis*
   Tingkat ini mengklasifikasikan seluruh teks sebagai satu kesatuan, apakah teks tersebut menyatakan sentimen positif, negatif, atau netral terhadap topik tertentu.

2. *Sentence and Phrase-Level Sentiment Analysis*
   Tingkat ini menganalisis sentimen untuk setiap kalimat dalam sebuah dokumen.

3. *Entity and Aspect-level opinions*
   Tingkat ini merupakan analisis yang paling terperinci, di mana sentimen diklasifikasikan untuk setiap aspek atau fitur tertentu dari suatu objek.

2.2 Lexicon Based Approach

Lexicon-based approach merupakan metode ilmiah yang sering digunakan dalam penelitian analisis sentimen. Cara kerja metode ini adalah dengan menggunakan sebuah kamus kata atau corpus yang dilengkapi dengan bobot pada setiap katanya sebagai sumber bahasa. Hasil analisis dengan metode ini berupa klasifikasi sentimen positif, negatif, dan netral.

Pada penelitian ini, proses perhitungan sentimen mengacu pada InSet Lexicon yang dikembangkan untuk bahasa Indonesia. InSet Lexicon memuat kata-kata bermuatan positif dan negatif yang masing-masing diberikan bobot numerik dalam rentang −5 hingga 5, di mana nilai negatif menunjukkan sentimen negatif dan nilai positif menunjukkan sentimen positif.

Setiap kalimat komentar dianalisis dengan mencocokkan kata-kata yang terkandung di dalamnya dengan entri pada InSet Lexicon. Selanjutnya, dilakukan perhitungan *sentiment score* dengan menjumlahkan seluruh bobot kata yang terdeteksi pada kalimat tersebut. Nilai *sentiment score* ini kemudian digunakan sebagai dasar penentuan label sentimen diskrit.

Berdasarkan nilai *sentiment score* yang diperoleh, klasifikasi sentimen ditentukan dengan kriteria sebagai berikut:

$$\begin{aligned} &\text{Jika } sentiment\_score > 0 \rightarrow \text{Sentimen Positif} \\ &\text{Jika } sentiment\_score < 0 \rightarrow \text{Sentimen Negatif} \\ &\text{Jika } sentiment\_score = 0 \rightarrow \text{Sentimen Netral} \end{aligned} \tag{2.1}$$

Pendekatan ini didasarkan pada asumsi bahwa nilai *sentiment score* merepresentasikan kecenderungan polaritas suatu kalimat. Kalimat dengan skor positif menunjukkan dominasi kata bermuatan positif, sedangkan skor negatif menunjukkan dominasi kata bermuatan negatif. Apabila nilai skor sama dengan nol, maka kalimat dianggap tidak memiliki kecenderungan sentimen tertentu dan diklasifikasikan sebagai sentimen netral [12].

2.2.1 InSet Lexicon

InSet Lexicon (Indonesia Sentiment Lexicon) merupakan salah satu sumber daya leksikon yang sering digunakan dan terbukti cukup efektif untuk tugas-tugas analisis sentimen[13]. *InSet Lexicon* terdiri dari ribuan kata positif dan kata negatif berbahasa Indonesia yang telah memiliki bobot nilai atau *polarity score* pada setiap katanya dengan kisaran bobot. *Polarity score* ini lah digunakan untuk mengklasifikasikan jenis sentimen.

Tabel 2.1. Contoh Kata dan score pada InSet Lexicon

Kata	Score Polaritas
bahagia	+4
senang	+3
luar biasa	+5
bagus	+2
suka	+2
marah	-4
buruk	-3
kecewa	-4
benci	-5
bodoh	-3

2.3 Natural Language Processing (NLP)

Natural Language Processing (NLP) merupakan cabang ilmu dari ilmu komputer untuk menjembantani antara bahasa manusia dengan bahasa komputer[14]. Tujuan utama dari NLP adalah agar komputer dapat memahami, memproses, dan menghasilkan bahasa alami seperti yang digunakan oleh manusia dalam komunikasi sehari-hari[14]. Melalui NLP, komputer dapat melakukan berbagai tugas yang melibatkan teks atau ucapan, seperti penerjemahan otomatis, analisis sentimen, ekstraksi informasi, pengenalan suara, dan sistem dialog[15].

Dalam konteks penelitian ini, NLP digunakan sebagai landasan utama untuk melakukan analisis sentimen terhadap teks yang diambil dari media sosial. Melalui penerapan teknik-teknik NLP, sistem dapat mengenali makna dari setiap kalimat, memahami konteks penggunaan kata, dan mengelompokkan sentimen yang terkandung di dalamnya. Dengan demikian, NLP menjadi fondasi penting dalam proses pengolahan teks yang mendukung akurasi model dalam melakukan klasifikasi sentimen.

2.4 Deep Learning

Deep Learning adalah cabang dari *machine learning* yang berfokus pada pembuatan dan pendidikan jaringan saraf tiruan, menggunakan banyak lapisan neuron buatan untuk secara otomatis mengekstraksi fitur-fitur penting dari data mentah, menjadikannya lebih unggul dalam banyak tugas dibandingkan model pembelajaran mesin pada umumnya[16].

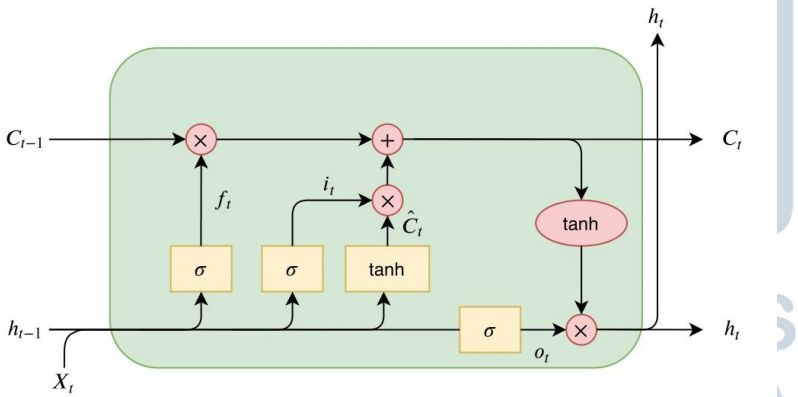
Deep learning telah berkembang menjadi berbagai teknik yang beragam

dengan tujuan-tujuan spesifik yang saling terkait. penelitian [16] mengidentifikasi lima teknik utama deep learning yang masing-masing dirancang untuk mengatasi tantangan tertentu: *Recurrent Neural Networks* (RNN) untuk memproses data sekuensial, *Long Short-Term Memory* (LSTM) untuk menangani dependensi jangka panjang, *Convolutional Neural Networks* (CNN) yang sangat efektif untuk pemrosesan gambar, *Generative Adversarial Networks* (GAN) yang menggunakan sistem dua jaringan berkompetisi untuk menghasilkan data baru, dan *Autoencoders* yang belajar merepresentasikan data dalam bentuk lebih kompak.

Keragaman teknik-teknik ini mencerminkan tujuan utama perkembangan deep learning yang dijelaskan oleh penelitian [16], yaitu memungkinkan komputer belajar dari dataset yang sangat besar dan menyelesaikan tugas-tugas kompleks yang sebelumnya sulit diatasi. Penelitian [17]menambahkan bahwa setiap teknik ini dikembangkan dengan tujuan meningkatkan aplikasi dalam bidang-bidang spesifik seperti deteksi objek, pengenalan visual, pengenalan suara, visi untuk kendaraan otonom, asisten virtual, genomik, dan penemuan obat.

2.5 Long Short Term Memories (LSTM)

Long Short-Term Memory (LSTM) merupakan salah satu varian dari jaringan saraf rekuren (RNN) yang dirancang untuk menangani dependensi jangka panjang dalam data sekuensial dengan tetap menjaga stabilitas pelatihan model [18].

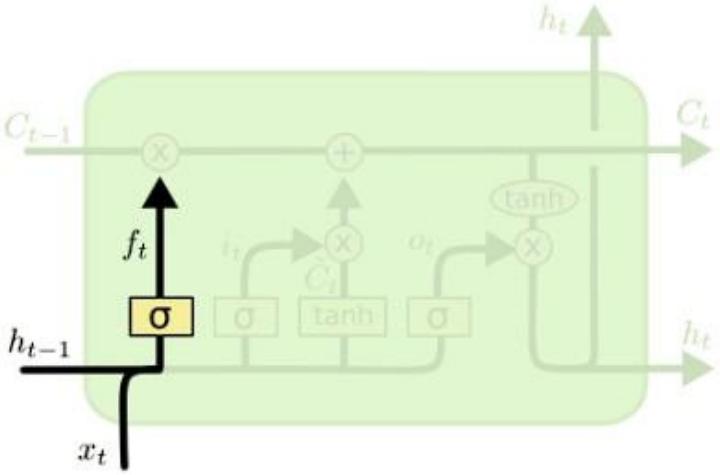


Gambar 2.1. Arsitektur LSTM

Long Short-Term Memory (LSTM) merupakan pengembangan dari Recurrent Neural Network (RNN) yang dirancang untuk mengatasi permasalahan ketergantungan jangka panjang (*long-term dependency*) pada data berurutan melalui penambahan *memory cell*. Arsitektur LSTM terdiri atas *cell state* dan beberapa

mekanisme gerbang, yaitu *input gate*, *forget gate*, dan *output gate*, yang berfungsi untuk mengatur aliran informasi dengan menentukan informasi yang disimpan, diperbarui, atau dihapus, sehingga memungkinkan model mempertahankan informasi relevan dalam urutan data yang panjang[19].

1. *Forget Gate*



Gambar 2.2. Arsitektur Forget Gate pada LSTM

Gambar 2.2 merupakan *Forget gate* pada LSTM berfungsi untuk menentukan informasi mana dari *cell state* sebelumnya yang perlu dipertahankan dan mana yang harus dilupakan. Gerbang ini menerima dua masukan, yaitu keluaran tersembunyi (*hidden state*) dari waktu sebelumnya (h_{t-1}) dan masukan saat ini (x_t), yang kemudian diproses melalui fungsi aktivasi sigmoid. Hasilnya berupa nilai antara 0 dan 1, di mana nilai 0 berarti informasi sepenuhnya dilupakan, sedangkan nilai 1 berarti informasi sepenuhnya dipertahankan. Nilai ini digunakan untuk mengalikan *cell state* sebelumnya (C_{t-1}), sehingga hanya informasi penting yang diteruskan ke tahap berikutnya[20].

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad (2.2)$$

Keterangan:

f_t : Forget Gate.

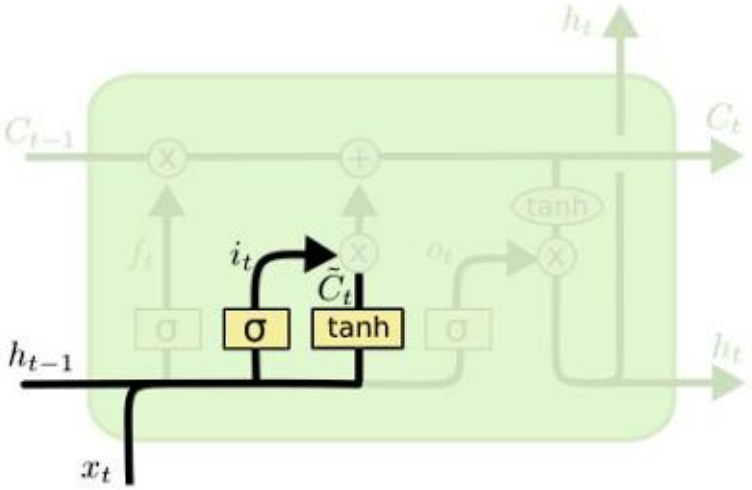
σ : fungsi aktivasi sigmoid

W_f : Nilai Weight untuk forget gate.

$[h_{t-1}, x_t]$: penggabungan antara keluaran tersembunyi sebelumnya (h_{t-1}) dan input saat ini (x_t).

b_f : nilai bias pada forget gate.

2. *Input Gate*



Gambar 2.3. Arsitektur Input Gate pada LSTM

Gambar 2.3 merupakan *input gate* yang berfungsi untuk mengontrol seberapa banyak informasi baru dari input saat ini yang akan disimpan ke dalam cell state. Gerbang ini bekerja dengan mengevaluasi masukan saat ini (x_t) dan keluaran tersembunyi sebelumnya (h_{t-1}) melalui fungsi aktivasi sigmoid untuk menentukan seberapa besar kontribusi input baru terhadap memori. Selain itu, nilai kandidat memori baru ($\tilde{C}_t$) dihasilkan menggunakan fungsi aktivasi tanh untuk menstabilkan nilai dalam rentang -1 hingga 1. Kedua komponen ini digabungkan untuk memperbarui *cell state*, sehingga LSTM dapat menambahkan informasi baru yang relevan tanpa menghapus konteks yang penting dari langkah sebelumnya [20].

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

$$\tilde{C}_t = \tanh(W_C[h_{t-1}, x_t] + b_C) \quad (2.4)$$

Keterangan:

(a) i_t : nilai keluaran dari input gate pada waktu ke-*t*.

(b) $\tilde{C}_t$: kandidat *cell state* baru.

(c) σ : fungsi aktivasi sigmoid.

(d) tanh : fungsi aktivasi tanh untuk normalisasi nilai.

(e) W_i : matriks bobot input gate.

(f) W_C : matriks bobot kandidat memori.

(g) h_{t-1} : *hidden state* waktu sebelumnya.

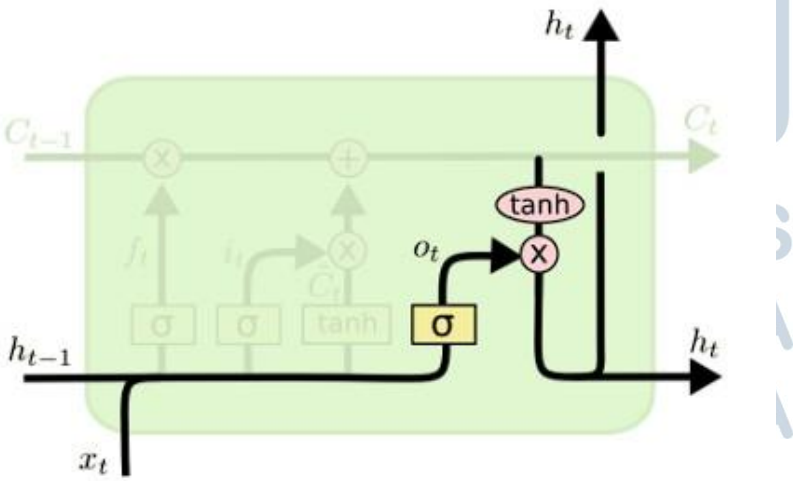
(h) x_t : input pada waktu ke-*t*.

(i) $[h_{t-1}, x_t]$: operasi concatenation.

(j) b_i : bias untuk input gate.

(k) b_C : bias untuk kandidat memori.

3. *Output Gate*



Gambar 2.4. Arsitektur Output Gate pada LSTM

Gambar 2.4 merupakan *Output gate* yang bertanggung jawab untuk menentukan bagian mana dari*cell state*yang akan dikeluarkan sebagai *hidden state* (h_t) pada waktu sekarang. Setelah *cell state* diperbarui, gerbang keluaran menggunakan fungsi sigmoid untuk menghasilkan nilai pengendali (o_t), kemudian nilai ini dikalikan dengan hasil fungsi tanh dari *cell state* untuk menghasilkan *hidden state* akhir. Mekanisme ini memastikan bahwa hanya informasi relevan yang diteruskan ke langkah berikutnya, sekaligus menjaga stabilitas memori jangka panjang dalam jaringan [20].

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{2.5}$$

$$h_t = o_t \times \tanh(C_t) \tag{2.6}$$

Keterangan:

(a) o_t : nilai keluaran output gate pada waktu ke-*t*.

(b) h_t : *hidden state* pada waktu ke-*t*.

(c) C_t : *cell state* pada waktu ke-*t*.

(d) σ : fungsi aktivasi sigmoid.

(e) tanh : fungsi aktivasi tanh.

(f) W_o : matriks bobot output gate.

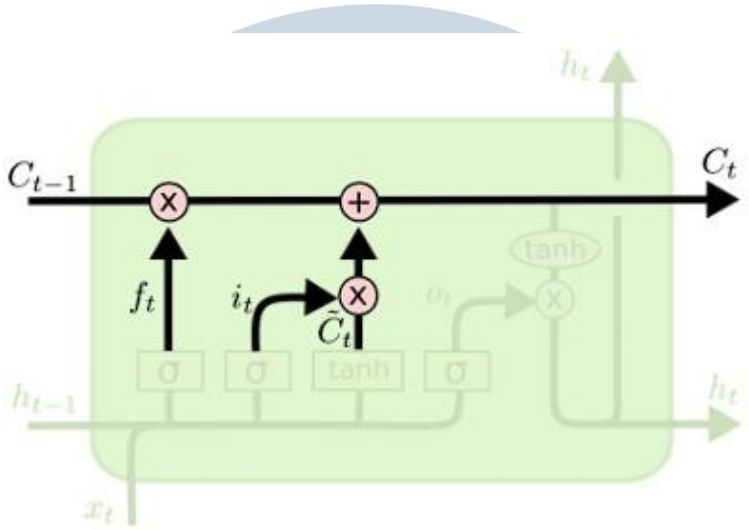
(g) h_{t-1} : *hidden state* waktu sebelumnya.

(h) x_t : input pada waktu ke-*t*.

(i) $[h_{t-1}, x_t]$: operasi concatenation.

(j) b_o : bias untuk output gate.

4. *Cell State*



Gambar 2.5. Arsitektur Cell State pada LSTM

Cell state merupakan komponen dalam arsitektur LSTM yang berfungsi sebagai memori jangka panjang untuk menyimpan informasi penting sepanjang urutan waktu. Berbeda dengan *hidden state* yang merepresentasikan keluaran pada setiap waktu, *cell state* dirancang agar aliran informasinya dapat bergerak melalui jaringan dengan perubahan minimal. Pada setiap langkah waktu, nilai *cell state* diperbarui berdasarkan kombinasi dari informasi yang dipertahankan oleh *forget gate* dan informasi baru yang ditambahkan melalui *input gate*. Dengan mekanisme ini, LSTM mampu mempertahankan informasi yang relevan dalam jangka panjang sekaligus menghindari masalah *vanishing gradient*[21][22].

$$C_t = f_t \times C_{t-1} + i_t \times \tilde{C}_t \quad (2.7)$$

Keterangan Simbol:

(a) C_t : nilai *cell state* pada waktu ke-*t*.

(b) C_{t-1} : nilai *cell state* pada waktu sebelumnya.

(c) f_t : keluaran dari *forget gate* yang menentukan informasi apa yang dipertahankan.

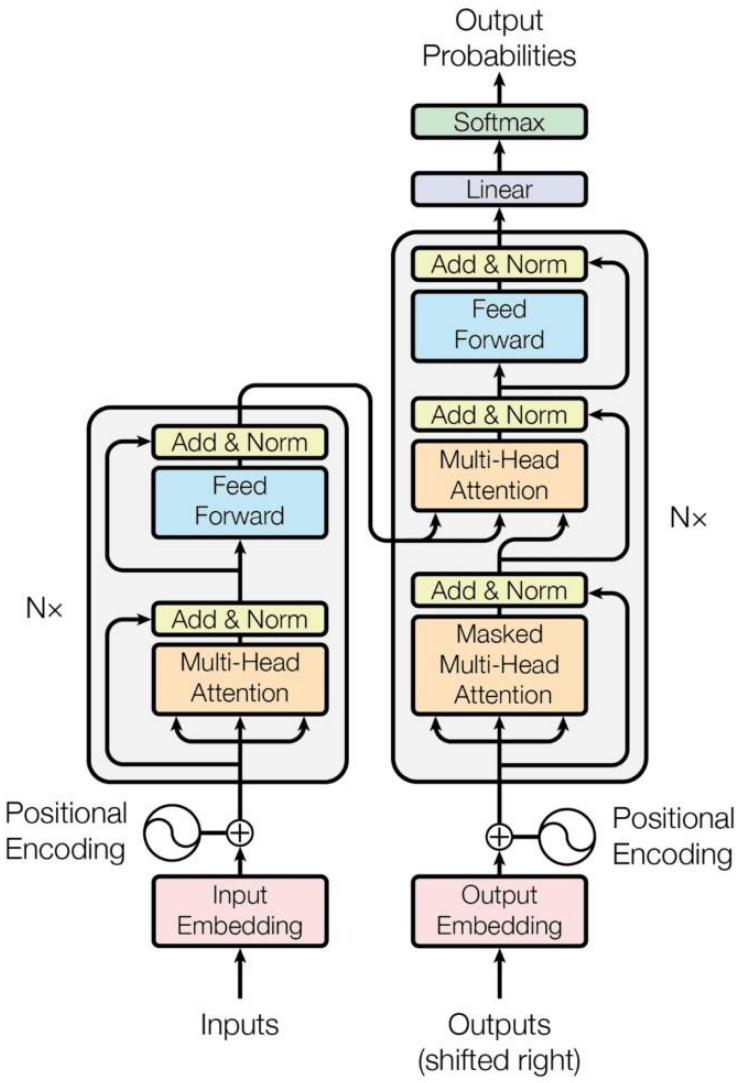
(d) i_t : keluaran dari *input gate* yang mengatur seberapa banyak informasi baru ditambahkan.

(e) $\tilde{C}_t$: kandidat *cell state* yang dihasilkan dari transformasi non-linear (tanh).

(f) $\times$: operasi perkalian elemen-per-elemen (*element-wise multiplication*).

2.6 Transformers

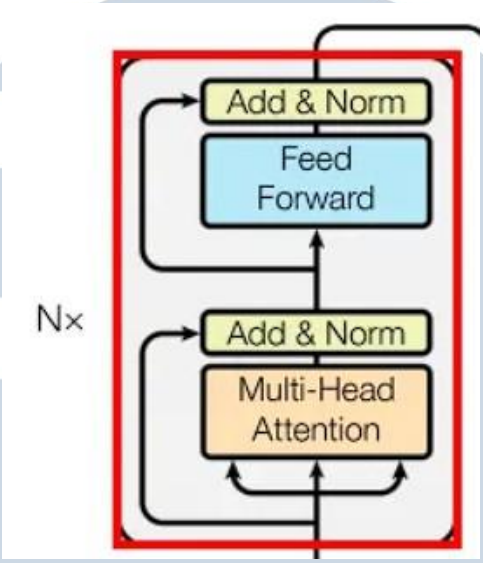
Vaswani et al. memperkenalkan *Transformer* sebagai arsitektur model baru yang sepenuhnya bergantung pada mekanisme perhatian atau *attention mechanism*. Penelitian tersebut juga menjelaskan bahwa pendekatan ini memungkinkan pemrosesan urutan secara paralel, Pernyataan tersebut berarti bahwa setiap token dalam suatu urutan dapat secara langsung mempertimbangkan informasi dari seluruh token lainnya pada lapisan sebelumnya[23]. Mekanisme ini memungkinkan pemodelan hubungan antar kata dilakukan secara paralel tanpa ketergantungan urutan waktu, sehingga lebih efisien dalam menangkap dependensi jarak jauh.



Gambar 2.6. Arsitektur model Transformers

Gambar 2.6 merupakan arsitektur *Transformer* terdiri dari dua komponen utama, yaitu *encoder* dan *decoder*. Bagian *encoder* bertanggung jawab menghasilkan representasi konteks dari masukan teks, sedangkan *decoder* memanfaatkan representasi tersebut untuk menghasilkan keluaran, seperti pada tugas penerjemahan mesin. Masing-masing bagian terdiri dari beberapa lapisan identik yang memiliki struktur internal sama[24].

2.6.1 Encoder

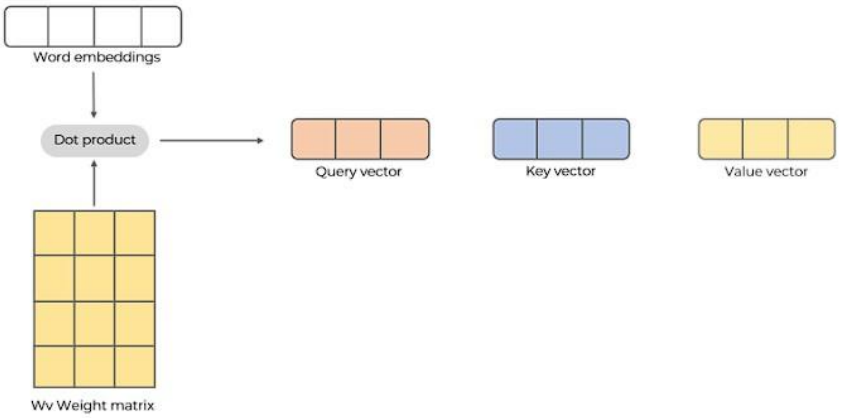


Gambar 2.7. Arsitektur Encoder

Gambar 2.7 adalah arsitektur dari Encoder dalam *Transformer* memiliki beberapa lapisan dengan setiap lapisan terdiri atas 2 komponen utama, yaitu *Multi-Head Self-Attention* dan *Feed Forward Network (FFN)*. Mekanisme *self-attention* memungkinkan model menilai seberapa penting setiap token terhadap token lainnya dalam satu urutan. Dengan membagi perhatian ke dalam beberapa *attention heads*, model mampu menangkap berbagai hubungan semantik secara lebih kaya. Setelah itu, hasil perhatian diproses oleh FFN untuk melakukan transformasi non-linear terhadap representasi token. Kedua sub-lapisan ini dilengkapi dengan *residual connection* dan *layer normalization* agar proses pelatihan lebih stabil dan menghindari degradasi informasi[25][26][27].

A Self-Attention dan Multi-Head Attention

Self-attention memungkinkan setiap token dalam urutan untuk memperhatikan token lain, termasuk dirinya sendiri, sehingga model dapat memahami konteks global dalam satu langkah komputasi. Mekanisme ini menghitung hubungan antara token melalui operasi dot-product antara matriks Query, Key, dan Value[27].



Gambar 2.8. Mekanisme dari Attention

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.8}$$

Keterangan:

1. Q : vektor *query*.
2. K : vektor *key*.
3. V : vektor *value*.
4. d_k : dimensi dari key.
5. softmax : fungsi aktivasi untuk normalisasi skor *attention*.

Self-attention menghitung skor relevansi antara setiap pasangan token melalui operasi dot-product antara Q dan K, kemudian menormalisasikannya menggunakan softmax. Hasilnya digunakan untuk memberi bobot pada nilai V, sehingga setiap token mendapatkan representasi baru berdasarkan konteks seluruh urutan[27].

Multi-head attention memperluas mekanisme *self-attention* dengan membaginya menjadi beberapa *attention head*. Setiap *head* belajar hubungan semantik yang berbeda, misalnya hubungan sintaksis, jarak jauh, atau hubungan frasa. Dengan menggabungkan berbagai perspektif ini, *Transformer* mampu menghasilkan representasi konteks yang lebih kaya dan mendalam[27].

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{2.9}$$

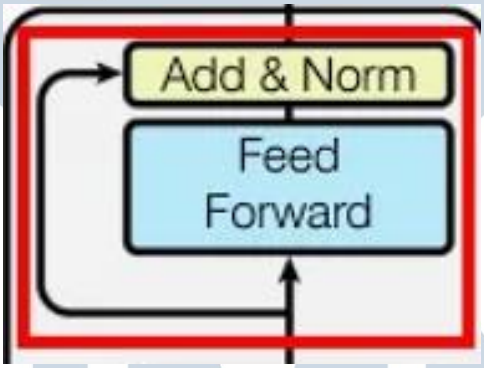
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \quad (2.10)$$

Keterangan:

1. *head* : jumlah attention head.
2. W_i^Q, W_i^K, W_i^V : matriks bobot untuk query, key, dan value pada head ke-*i*.
3. W^O : matriks bobot keluaran setelah penggabungan head.
4. Concat : operasi penggabungan antar head.

B Feed Forward Network (FFN)

Selanjutnya merupakan lapisan *Feed Forward Network*.



Gambar 2.9. Feed Forward Network

Feed Forward Network (FFN) merupakan komponen pada arsitektur Encoder. FFN berfungsi melakukan transformasi non-linear terhadap representasi setiap token secara independen. Berbeda dari mekanisme *attention* yang memodelkan hubungan antar token, FFN bekerja pada setiap token secara terpisah (*token-wise*), sehingga berperan memperkaya representasi semantik dengan pola-pola non-linear yang tidak dapat ditangkap oleh self-attention saja. Struktur FFN terdiri dari dua lapisan linear yang dipisahkan oleh fungsi aktivasi ReLU, dan dilengkapi dengan *residual connection* serta *layer normalization* untuk menjaga stabilitas pelatihan[25][26][27].

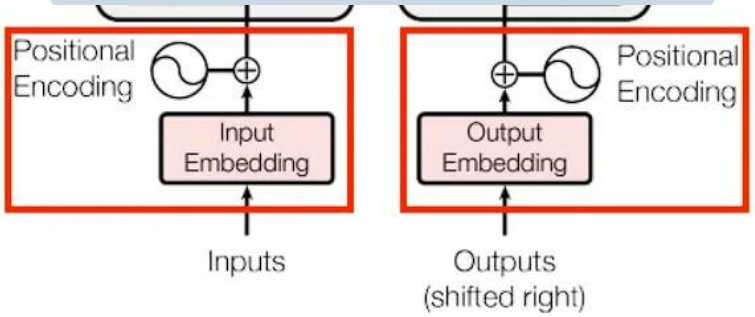
$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \quad (2.11)$$

Keterangan:

1. x : vektor representasi token yang menjadi masukan.
2. W_1 : matriks bobot pada lapisan linear pertama.
3. b_1 : bias pada lapisan pertama.
4. $\max(0, \cdot)$: fungsi aktivasi ReLU.
5. W_2 : matriks bobot pada lapisan linear kedua.
6. b_2 : bias pada lapisan kedua.

2.6.2 Input Embedding dan Output Embedding

Pada model berbasis Transformer, proses *embedding* memegang peranan penting dalam merepresentasikan data teks ke dalam bentuk numerik.



Gambar 2.10. Arsitektur Embedding

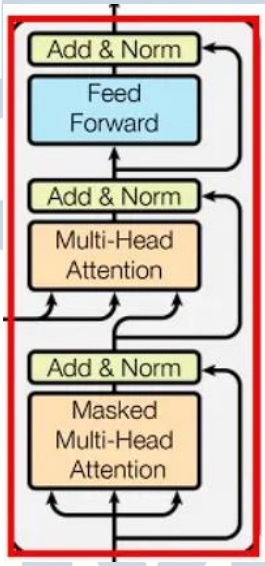
Gambar 2.10 menunjukkan dua komponen awal yang berada pada sisi *encoder* dan*decoder*, yaitu *Input Embedding* serta *Output Embedding*, yang masing-masing ditambahkan dengan *Positional Encoding*. Pada bagian *encoder*, *Input Embedding* berfungsi mengubah setiap token pada kalimat masukan menjadi vektor berdimensi tetap yang dapat diproses oleh model. Karena *Transformer* tidak memiliki mekanisme urutan secara alami, *embedding* ini kemudian dijumlahkan dengan *Positional Encoding* agar model mengetahui posisi setiap token dalam kalimat [25].

Pada bagian decoder, prosesnya serupa namun menggunakan *Output Embedding*, yaitu *embedding* dari token keluaran (target sequence). Token ini digeser ke kanan (*shifted right*) sehingga model hanya dapat melihat token-token yang telah dihasilkan sebelumnya. Setelah diubah menjadi vektor *embedding*, nilai tersebut juga dijumlahkan dengan *Positional Encoding* untuk mempertahankan informasi urutan sebelum memasuki lapisan-lapisan decoder.

Perbedaan utama antara keduanya adalah bahwa *Input Embedding* digunakan untuk memproses kalimat masukan pada encoder, sedangkan *Output Embedding* digunakan untuk memproses kalimat keluaran pada *decoder* yang dioperasikan secara autoregresif. Selain itu, *Output Embedding* selalu mengandung mekanisme *shifted right*, sementara *Input Embedding* tidak memerlukannya. Meskipun demikian, kedua komponen tetap menggunakan prinsip penjumlahan *embedding* dengan positional encoding untuk memberikan informasi posisi token dalam urutan.

2.6.3 Decoder

Decoder pada arsitektur Transformer, yang berperan dalam menghasilkan keluaran berdasarkan representasi konteks yang diperoleh dari *encoder*



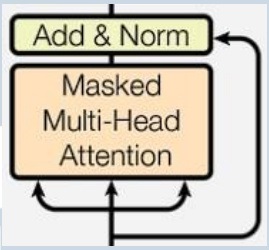
Gambar 2.11. Arsitektur Decoder

Gambar 2.11 menggambarkan struktur Decoder pada Transformer yang tersusun atas beberapa lapisan berurutan. Setiap lapisan memiliki tiga blok utama: *Masked Multi-Head Self-Attention, Multi-Head Cross-Attention, dan Feed Forward Network (FFN)*. Pada blok *masked self-attention*, *decoder* hanya melihat token-token yang sudah dihasilkan sebelumnya, sehingga proses prediksi berlangsung secara bertahap dan tidak mengakses informasi token masa depan. Setelah itu, blok *cross-attention* mengintegrasikan keluaran dari *encoder* dengan token *decoder*, memungkinkan model memahami konteks penuh dari input. Hasil pemrosesan tersebut kemudian diteruskan ke FFN yang bertugas memperkaya representasi token melalui transformasi non-linear. Seluruh blok ini dilengkapi residual

connection dan layer normalization agar aliran informasi tetap stabil dan proses pelatihan berjalan lebih efektif[25][26][27].

A Masked Multi-Head Self-Attention

Masked Multi-Head Self-Attention digunakan pada *decoder* Transformer untuk menjaga ketepatan proses prediksi token.



Gambar 2.12. Arsitektur Masked Multi-Head Attention pada Decoder

Gambar 2.12 menunjukan *Masked Multi-Head Self-Attention* yang merupakan mekanisme *attention* pada bagian *decoder* Transformer yang dirancang untuk memastikan bahwa model hanya memanfaatkan token-token sebelumnya saat memprediksi token berikutnya. Proses ini membuat model bekerja secara *autoregressive*, yaitu menghasilkan keluaran satu per satu tanpa mengetahui token di masa depan. Masking dilakukan dengan memblokir skor perhatian menuju token yang berada di posisi kanan (masa depan) dengan memberikan nilai $-\infty$ sebelum fungsi *softmax*. Dengan demikian, skor perhatian untuk token masa depan menjadi nol. Selain itu, penggunaan beberapa *attention heads* memungkinkan model menangkap beragam pola dependensi secara paralel, sehingga menghasilkan representasi token yang lebih kaya dan kontekstual.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}} + M\right)V \tag{2.12}$$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(h_1, h_2, \ldots, h_H)W^O \tag{2.13}$$

$$h_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{2.14}$$

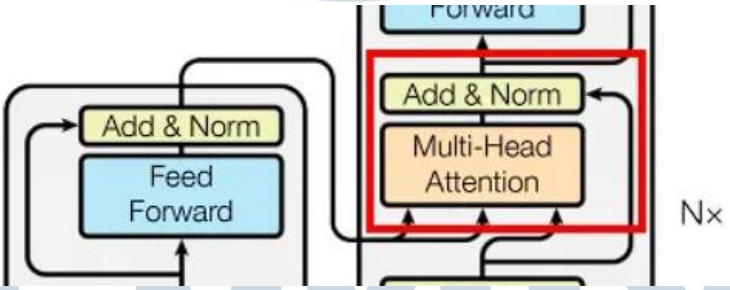
Keterangan:

1. Q : Vektor *query* dari token pada *decoder*.

2. K : Vektor *key* dari token pada *decoder*.
3. V : Vektor *value* dari token pada *decoder*.
4. W_i^Q, W_i^K, W_i^V : Matriks bobot untuk *query*, *key*, dan *value* pada head ke-*i*.
5. W^O : Matriks bobot untuk menggabungkan seluruh *attention heads*.
6. d_k : Dimensi dari vektor *key* untuk normalisasi.
7. M : Matriks *mask* yang berisi 0 untuk token yang boleh diakses dan $-\infty$ untuk token masa depan.
8. h_i : Keluaran *attention* pada head ke-*i*.
9. Concat$(\cdot)$: Operasi penggabungan vektor dari seluruh head.

B Cross-Attention pada Decoder

textitCross-Attention pada *decoder* Transformer yang berfungsi mengintegrasikan informasi dari *encoder* ke dalam proses prediksi keluaran.



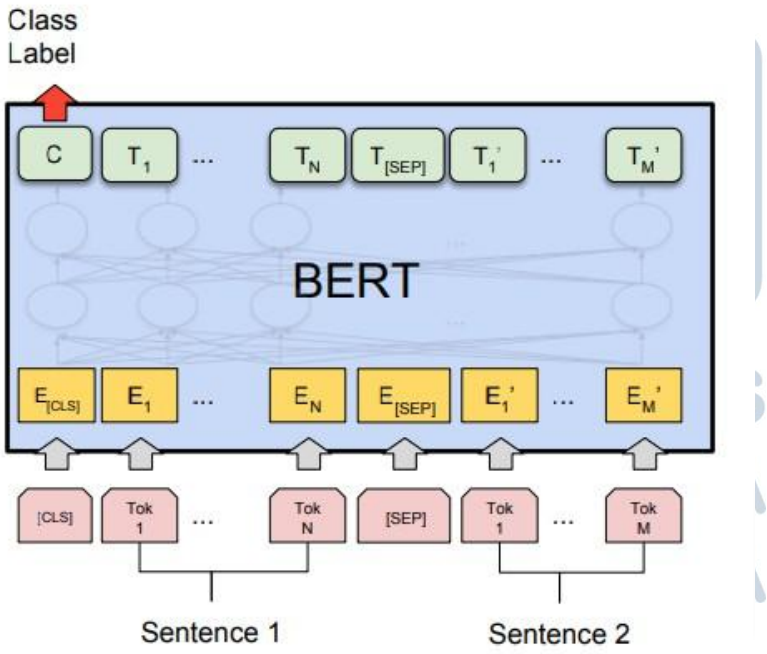
Gambar 2.13. Arsitektur Cross-Attention pada Decoder

Gambar 2.13 adalah gambar *Cross-attention* merupakan mekanisme pada bagian *decoder* Transformer yang memungkinkan model memanfaatkan informasi yang dihasilkan oleh *encoder*. Pada proses ini, matriks *query* (Q) diperoleh dari keluaran *decoder*, sedangkan matriks *key* (K) dan *value* (V) berasal dari representasi yang dihasilkan oleh encoder. Dengan demikian, *decoder* dapat mengakses konteks penuh dari kalimat masukan, sehingga prediksi token berikutnya tidak hanya bergantung pada token sebelumnya dalam keluaran, tetapi juga pada keseluruhan struktur dan makna dari input. Mekanisme ini memainkan peran penting dalam menjaga hubungan antara input dan output, terutama pada tugas seperti penerjemahan dan pemahaman konteks antarkalimat[25].

Cara kerjanya dimulai dengan menghitung skor perhatian antara *query* dari decoder dan *key* dari encoder melalui operasi dot-product. Skor ini kemudian dinormalisasi menggunakan fungsi *softmax* untuk menghasilkan bobot perhatian yang menunjukkan tingkat relevansi setiap token input terhadap token yang sedang diproses pada decoder. Bobot tersebut selanjutnya digunakan untuk melakukan kombinasi linier terhadap *value* encoder, sehingga menghasilkan representasi kontekstual baru yang kaya informasi. Hasil representasi ini kemudian diteruskan ke tahap selanjutnya dalam decoder untuk menghasilkan prediksi token yang lebih akurat dan sesuai dengan konteks input.

2.7 BERT

BERT (*Bidirectional Encoder Representations from Transformers*) merupakan model representasi bahasa yang diperkenalkan oleh Google AI pada tahun 2018[28]. Model ini dirancang untuk membuat representasi kontekstual dari input teks dengan menggunakan *encoder*[29]. Dengan pendekatan bidirectional ini, BERT mampu menangkap makna kata berdasarkan konteks dari kiri maupun kanan, menghasilkan representasi semantik yang lebih kaya dibandingkan model searah seperti RNN[30].



Gambar 2.14. Arsitektur BERT

Gambar 2.14 menggambarkan arsitektur umum dari model BERT yang menggunakan dua kalimat masukan, yaitu *Sentence 1* dan *Sentence 2*. Setiap token pada kalimat masukan diubah menjadi representasi vektor melalui tiga komponen utama, yaitu *Token Embedding*, *Segment Embedding*, dan *Positional Embedding*. Representasi awal dari setiap token digabungkan menjadi satu urutan input yang diawali dengan token khusus [CLS] dan diakhiri dengan token [SEP] sebagai pemisah antar kalimat.

Token [CLS] berfungsi sebagai representasi keseluruhan dari pasangan kalimat dan digunakan untuk tugas klasifikasi, di mana hasil akhir dari token ini akan diteruskan ke lapisan keluaran untuk menghasilkan *class label*. Token [SEP] digunakan untuk menandai batas antara dua kalimat agar model dapat membedakan konteks antar kalimat.

Lapisan di tengah merupakan tumpukan beberapa *Transformer encoder* yang bekerja dengan mekanisme *self-attention*. Mekanisme ini memungkinkan setiap token memperhatikan token lain di seluruh urutan teks, baik dari kiri ke kanan maupun sebaliknya, sehingga model dapat memahami konteks dua arah secara menyeluruh.

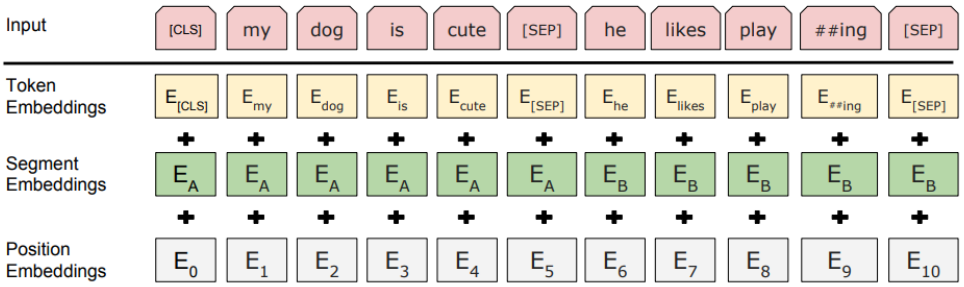
Hasil akhir dari proses ini adalah representasi kontekstual untuk setiap token, dengan keluaran pada token [CLS] mewakili pemahaman global dari seluruh kalimat. Representasi tersebut kemudian digunakan untuk berbagai tugas *Natural Language Processing (NLP)*, seperti *sentiment analysis*, *text classification*, dan *question answering*.

BERT dilatih menggunakan dua tahap utama, yaitu *pre-training* dan *fine-tuning*. Pada tahap *pre-training*, model mempelajari pemahaman bahasa umum melalui korpus teks besar tanpa label, sedangkan pada tahap *fine-tuning*, model disesuaikan dengan dataset spesifik untuk tugas tertentu seperti analisis sentimen atau klasifikasi teks[31][32].

2.7.1 Tahapan Proses pada BERT

Proses kerja BERT secara umum terdiri atas beberapa tahapan berikut:

1. *Input Embedding*



Gambar 2.15. Proses Embedding pada Model BERT

Gambar 2.15 menunjukkan tahapan pembentukan representasi input pada model BERT melalui proses *embedding*. Setiap token yang masuk ke model akan diubah menjadi representasi vektor numerik dengan menjumlahkan tiga jenis embedding, yaitu *Token Embedding*, *Segment Embedding*, dan *Position Embedding*[30].

(a) *Token Embedding*

Setiap kata dalam teks, termasuk token khusus seperti [CLS] dan [SEP], diubah menjadi vektor yang merepresentasikan makna kata tersebut. Pada gambar di atas, kata "my", "dog", dan "is" diubah menjadi E_{my}, E_{dog}, dan E_{is}. Tokenisasi dilakukan menggunakan algoritma WordPiece, yang memungkinkan pemecahan kata menjadi sub-kata seperti "ing" agar model dapat menangani kosakata yang tidak dikenal dengan lebih baik.

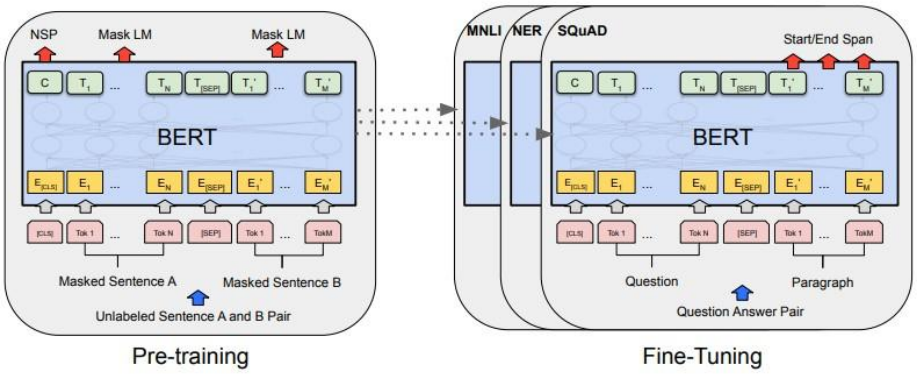
(b) *Segment Embedding*

BERT mampu menerima dua kalimat secara bersamaan sebagai pasangan input. Oleh karena itu, setiap token diberi *segment embedding* yang menunjukkan asal kalimatnya. Kalimat pertama diberi tanda E_A dan kalimat kedua diberi tanda E_B. Komponen ini penting untuk tugas seperti *Next Sentence Prediction (NSP)*, di mana model perlu memahami hubungan antara dua kalimat.

(c) *Position Embedding*

Karena arsitektur *Transformer* tidak memiliki informasi urutan secara alami, maka setiap token juga diberi *position embedding* (E_0, E_1, E_2, . . .) untuk menandai posisi relatifnya dalam kalimat. Hal ini memungkinkan BERT mempertahankan urutan kata sehingga konteks sintaktik tetap terjaga.

2. *Pre-training and Fine-tuning*



Gambar 2.16. Tahapan Pre-training dan Fine-tuning pada Model BERT

Gambar 2.16 memperlihatkan dua tahap utama dalam proses pelatihan model BERT, yaitu tahap *Pre-training* dan *Fine-tuning*. Kedua tahap ini merupakan inti dari cara kerja BERT untuk membangun pemahaman bahasa yang mendalam dan menyesuaikannya dengan berbagai tugas pemrosesan bahasa alami (NLP) [28][30].

(a) *Tahap Pre-training*

Pada bagian kiri gambar, ditunjukkan proses *pre-training* yang melibatkan dua kalimat masukan, yaitu *Sentence A* dan *Sentence B*. Setiap token pada kedua kalimat ini dikonversi menjadi representasi vektor melalui proses *embedding*, yang terdiri atas *Token Embedding*, *Segment Embedding*, dan *Position Embedding*. Token [CLS] diletakkan di awal sebagai representasi keseluruhan teks, sedangkan token [SEP] berfungsi sebagai pemisah antar kalimat.

Pada tahap ini, BERT dilatih menggunakan dua mekanisme utama:

i. *Masked Language Modeling (MLM)* — sebagian token dalam kalimat diganti dengan token khusus [MASK], kemudian model dilatih untuk memprediksi kata yang hilang berdasarkan konteks sekitarnya.

ii. *Next Sentence Prediction (NSP)* — model dilatih untuk menentukan apakah kalimat kedua merupakan lanjutan dari kalimat pertama atau tidak.

Mekanisme MLM ditunjukkan pada panah merah bertuliskan "Mask LM" dalam gambar, sedangkan NSP diwakili oleh panah di sisi kiri bertanda "NSP". Kedua proses ini melatih BERT untuk memahami konteks kata secara dua arah dan hubungan antarkalimat secara global. Hasil dari tahap ini adalah model yang memiliki pemahaman bahasa umum dari data tanpa label (*unsupervised learning*).

(b) *Tahap Fine-tuning*

Bagian kanan gambar menunjukkan tahap *fine-tuning*, di mana model BERT yang telah dilatih secara umum disesuaikan dengan tugas-tugas spesifik menggunakan data berlabel (*supervised learning*). Struktur arsitektur BERT tetap sama, namun ditambahkan lapisan keluaran sesuai jenis tugas.

Sebagai contoh, tugas *MNLI* dan *NER* (*Named Entity Recognition*) menggunakan token [CLS] sebagai masukan ke lapisan klasifikasi akhir untuk menentukan label kategori, sedangkan tugas seperti *SQuAD* (*Question Answering*) menggunakan dua panah merah "Start/End Span" untuk menandai posisi awal dan akhir jawaban dalam teks.

Panah putus-putus di tengah gambar menunjukkan bahwa hasil pelatihan dari tahap *pre-training* digunakan kembali sebagai dasar untuk *fine-tuning*. Dengan pendekatan dua tahap ini, BERT tidak perlu dilatih dari awal setiap kali menghadapi tugas baru, melainkan cukup menyesuaikan parameter terakhir dengan data spesifik.

2.7.2 Aplikasi BERT pada Bahasa Indonesia

Model turunan dari BERT untuk bahasa Indonesia dikenal sebagai *IndoBERT*, yang dikembangkan oleh IndoNLU dan HuggingFace[33]. Model ini dilatih menggunakan lebih dari satu miliar kata berbahasa Indonesia dan terbukti memberikan peningkatan signifikan dalam tugas-tugas seperti analisis sentimen dan klasifikasi teks dibandingkan metode tradisional seperti SVM atau Na¨ıve Bayes.

2.8 indoBERT

IndoBERT merupakan model *Bidirectional Encoder Representations from Transformers* yang telah diadaptasi secara khusus untuk bahasa Indonesia. Model ini dikembangkan oleh tim IndoNLU dan HuggingFace dengan melakukan pra-pelatihan (*pre-training*) menggunakan korpus teks bahasa Indonesia yang sangat besar, terdiri dari lebih dari satu miliar kata yang mencakup berbagai domain seperti berita, media sosial, dan Wikipedia [33].

Keberadaan IndoBERT menjadi tonggak penting dalam pemrosesan bahasa alami (NLP) untuk bahasa Indonesia karena sebelumnya sebagian besar model berfokus pada bahasa Inggris. Dengan arsitektur Transformer yang sama seperti BERT, IndoBERT mampu memahami konteks kata dalam kalimat secara dua arah, sehingga dapat menghasilkan representasi semantik yang lebih akurat dalam bahasa Indonesia[34].

Penelitian yang dilakukan oleh Apriliani et al. membahas penerapan analisis sentimen pada ulasan produk hijab di platform e-commerce Tokopedia dengan memanfaatkan algoritma *Support Vector Machine* (SVM) dan representasi fitur berbasis *IndoBERT embedding*. Dalam penelitian tersebut, IndoBERT digunakan untuk mengekstraksi representasi vektor kata yang bersifat kontekstual dari teks ulasan, sehingga informasi semantik dapat dipertahankan dengan lebih baik sebelum dilakukan proses klasifikasi. Pendekatan ini menunjukkan bahwa penggunaan model bahasa pralatih seperti IndoBERT mampu meningkatkan kualitas representasi teks pada tugas analisis sentimen[35].

```
function Indobert_Large_Algorithm(input_text, labels, max_epochs):
    # Preprocessing
    input_text = Remove_stopwords(input_text)
    input_text = Handle_slangwords(input_text)
    input_text = Remove_numbers_and_symbols(input_text)
    input_text = Stemming(input_text)

    tokens = Tokenize(input_text)
    input_ids = Convert_tokens_to_ids(tokens)
    attention_mask = Create_attention_mask(input_ids)

    # Load Pretraned Indobert Large Model
    model = Load_pretrained_Indobert_Large_model()

    # Prepare data for training
    train_data = Combine_input_labels(input_ids, attention_mask,
```

```
    labels)

    # Split data into train and validation sets
    train_set, val_set = Split_train_validation(train_data)

    # Training loop
    best_loss = infinity
    best_model = None
    epochs_without_improvement = 0

    for epoch in range(max_epochs):
        # Train the model
        train_loss = Train_model(model, train_set)

        # Evaluate on validation set
        val_loss = Evaluate_model(model, val_set)

        # Check for early stopping
        if val_loss < best_loss:
            best_loss = val loss
            best_model = model
            epochs_without_improvement = 0
        else:
            epochs_without_improvement += 1

        if epochs_without_improvement == early_stopping_patience:
            break

    # Postprocessing
    prediction = Extract_prediction(best_model(input_ids,
    attention_mask))

    return prediction
```

Kode 2.1: Pseudocode Indobert

Kode 2.1 merupakan pseudocode IndoBERT, menggambarkan alur umum proses fine-tuning model IndoBERT Large untuk tugas klasifikasi teks. Proses dimulai dengan tahap preprocessing, yaitu membersihkan teks dari stopwords, slangwords, angka, simbol, serta melakukan stemming untuk menyederhanakan bentuk kata. Setelah teks bersih, data diubah menjadi token menggunakan tokenizer IndoBERT, lalu dikonversi menjadi input IDs serta attention mask yang diperlukan sebagai masukan model. Model IndoBERT Large dimuat dan digabungkan dengan

label untuk membentuk data pelatihan. Dataset kemudian dibagi menjadi training set dan validation set untuk mengevaluasi performa model selama pelatihan.

Selanjutnya, Proses pelatihan dilakukan menggunakan training loop dengan mekanisme early stopping, yaitu menghentikan pelatihan jika nilai validation loss tidak membaik dalam beberapa epoch berturut-turut guna mencegah overfitting. Setelah pelatihan selesai, model terbaik digunakan untuk menghasilkan prediksi berdasarkan input IDs dan attention mask.

2.9 Hyperparameter Tuning

Hyperparameter tuning merupakan proses optimasi parameter model pembelajaran mesin yang nilainya ditentukan sebelum proses pelatihan dimulai dan tidak dipelajari secara langsung dari data. Hyperparameter memiliki peran penting dalam menentukan kinerja model, karena konfigurasi yang tidak optimal dapat menyebabkan model mengalami *underfitting* atau *overfitting* [36].

Dalam konteks *deep learning*, beberapa hyperparameter yang umum dioptimalkan meliputi jumlah unit pada lapisan tersembunyi, nilai *learning rate*, ukuran *batch*, serta tingkat *dropout*. Proses hyperparameter tuning bertujuan untuk menemukan kombinasi nilai hyperparameter yang mampu menghasilkan performa model terbaik pada data validasi [37].

Salah satu metode hyperparameter tuning yang efisien adalah *Hyperband*. Metode ini mengombinasikan strategi *random search* dengan teknik *early stopping* untuk mengalokasikan sumber daya pelatihan secara adaptif. Hyperband mampu menghentikan konfigurasi hyperparameter yang kurang menjanjikan lebih awal, sehingga proses pencarian menjadi lebih cepat dan efisien dibandingkan metode pencarian konvensional [38].

Pada penelitian ini, proses hyperparameter tuning diimplementasikan menggunakan *Keras Tuner* dengan algoritma Hyperband untuk mengoptimalkan model *hybrid IndoBERT–LSTM*. Hyperparameter yang dioptimalkan meliputi jumlah unit LSTM, jumlah neuron pada lapisan *dense*, nilai *dropout*, serta *learning rate*. Konfigurasi terbaik dipilih berdasarkan nilai akurasi validasi tertinggi.

2.10 Evaluasi

Pada bagian ini dilakukan proses evaluasi terhadap model yang dibangun untuk mengetahui sejauh mana performa model[39]. Evaluasi dilakukan

menggunakan metrik standar pada tugas klasifikasi teks, yaitu *accuracy*, *precision*, *recall*, dan *F1-score*. Untuk melihat persebaran prediksi model terhadap masing-masing kelas sentimen, digunakan *confusion matrix*. *Confusion matrix* memberikan informasi tentang seberapa sering model melakukan klasifikasi benar dan salah pada tiap kelas, sehingga dapat membantu mengidentifikasi kelas mana yang sering membingungkan model.

2.10.1 Metrik Evaluasi

Evaluasi model dilakukan menggunakan empat metrik utama yang umum digunakan dalam tugas klasifikasi sentimen [40]:

1. *Accuracy*
   Accuracy merupakan perhitungan untuk menggambarkan seberapa akurat model klasifikasi yang telah dibuat.
2. *Precision*
   Precision merupakan perhitungan untuk mengukur ketepatan model dalam memprediksi kelas tertentu.
3. *Recall*
   Recall merupakan perhitungan untuk mengukur kemampuan model dalam mendeteksi seluruh instance dari kelas tertentu.
4. *F1-score*
   F1-score adalah hasil rata rata antara precision dan recall.

Metrik-metrik tersebut dapat diformulasikan sebagai berikut:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.15}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.16}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.17}$$

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.18}$$

dengan:

1. *TP*: *True Positive*
2. *TN*: *True Negative*
3. *FP*: *False Positive*
4. *FN*: *False Negative*