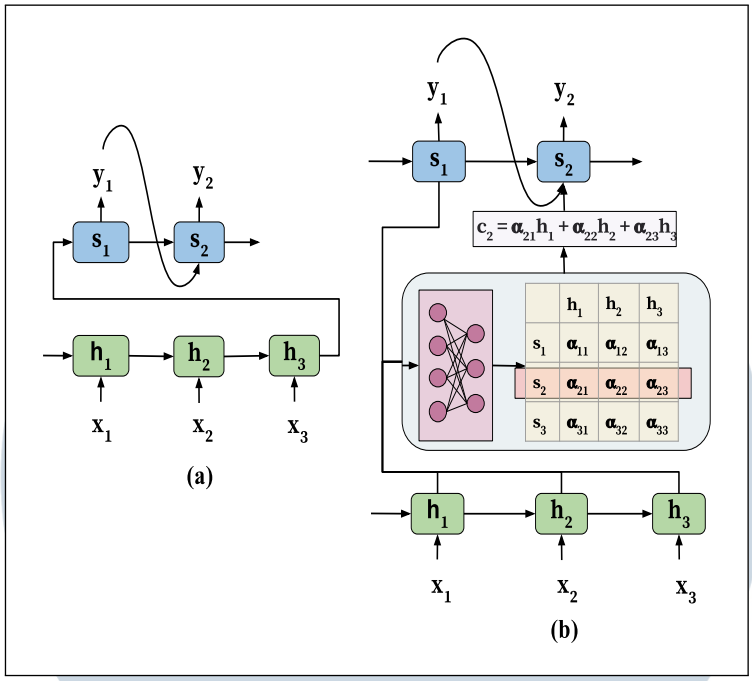
BAB 2
LANDASAN TEORI

2.1 Mekanisme Attention

Mekanisme *attention* adalah mekanisme atau metode yang digunakan dalam *neural network*. Nama dari mekanisme ini terinspirasi dari cara manusia melihat dan memberi perhatian. Mekanisme ini dapat lebih mencakup bagian *input* yang lebih relevan dengan memberi *weight* terhadap *encoding* dari *input*. Ide dari mekanisme telah ada dari tahun 1980, dengan berbagai metode diusulkan untuk *task computer vision* yang berusaha meniru cara manusia melihat [26].

Penggunaan mekanisme *attention* dalam NLP diusulkan oleh Bahdanau et al. [27] untuk kegunaan *Neural-machine translation* (NMT). Model yang digunakan dalam NMT umumnya terdiri atas *encoder* dan *decoder* [28], yang disebut juga dengan model *sequence-to-sequence* (seq2seq) [29]. Bahdanau et al. [27] menggunakan RNN untuk *encoder* dan *decoder*.

Encoder berfungsi untuk mengubah seluruh *input* menjadi *fixed-length vector* untuk diproses oleh *decoder*. *Decoder* menghasilkan *output* berdasarkan state dan konteks sebelumnya. *Encoder-decoder* memiliki permasalahan hilangnya informasi karena *input* yang panjang dipadatkan dengan panjang tetap, dan *decoder* tidak ada mekanisme untuk fokus ke bagian *input* yang relevan ketika menghasilkan *output*. Mekanisme *attention* berfungsi untuk memperbaiki permasalahan tersebut dengan memberikan *decoder* akses ke seluruh *input* melalui konteks yang mencakup seluruh *input* $\{h_1, h_2, \ldots, h_T\}$ dan konteks diberikan *weight* α sehingga *input* yang relevan lebih diprioritaskan saat *decode* [30]. Gambar 2.1 menunjukkan perbedaan model *encoder-decoder* tradisional pada 2.1 (a) dan dengan *attention* pada 2.1 (b).



Gambar 2.1. Arsitektur *Encoder-Decoder* (a) tradisional (b) dengan *attention*
sumber: [30]

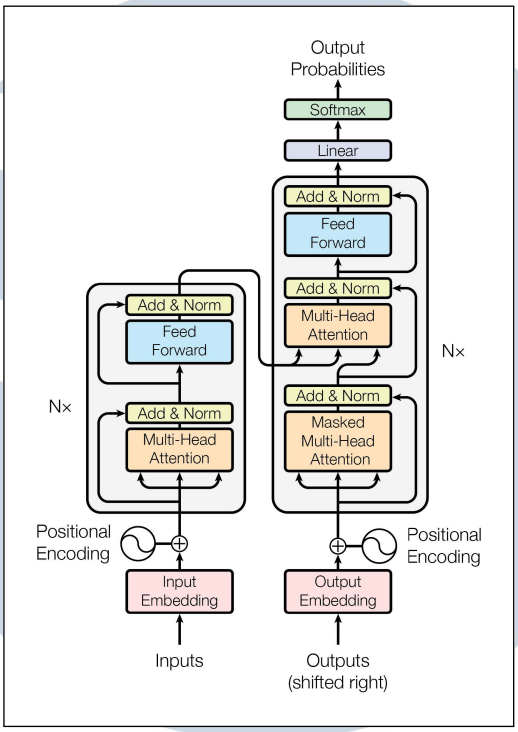
2.2 Transformer

Transformer adalah arsitektur model yang hanya terdiri dari mekanisme *attention*. Sebelumnya, model NN umumnya menggunakan *recurrence* (RNN) atau *convolution* (CNN) yang dapat dihubungkan dengan mekanisme *attention*. Namun kedua *neural-network* tersebut cukup kompleks dan memerlukan sumber daya pemrosesan yang cukup besar. Vaswani et al. [31] mengusulkan arsitektur *Transformer* yang lebih sederhana karena hanya mengandalkan mekanisme *attention* merepresentasikan *input* dan *output*.

2.2.1 Encoder-Decoder

Transformer terdiri atas *encoder* dan *decoder*. *Encoder* terdiri dari enam layer. Setiap layer memiliki dua sub-layer, yaitu *multi-head self-attention mechanism* dan *position-wise fully-connected feed-forward network* sederhana. Hasil dari setiap sub-layer dilakukan normalisasi layer [32] untuk mengurangi waktu *training* dan lebih efisien. Vaswani et al. [31] menetapkan ukuran model berdimensi d_{model} = 512. *Decoder* juga terdiri dari enam layer, dengan setiap layer memiliki sub-layer yang sama dengan pada *encoder*, tetapi ditambah satu sub-layer *multi-head*

attention terhadap *output*. Bentuk arsitektur model *Transformer* dapat dilihat pada Gambar 2.2.



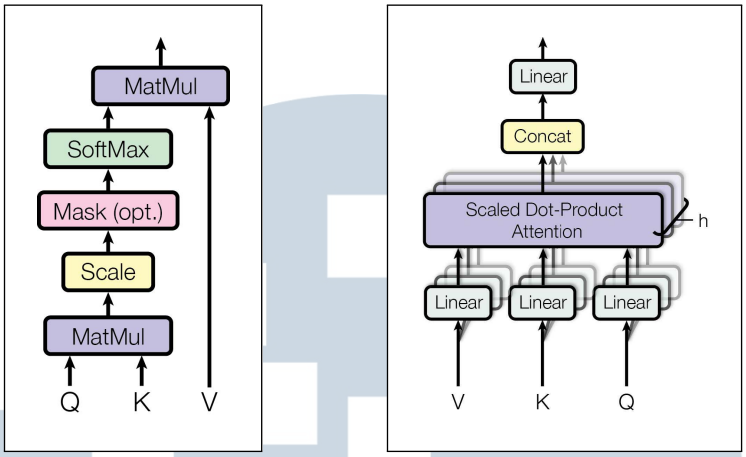
Gambar 2.2. Arsitektur model *Transformer*

sumber: [31]

2.2.2 Attention

Vaswani et al. [31] menggunakan fungsi *self-attention* yang dinamakan "*Scaled Dot-Product Attention*", seperti pada Gambar 2.3(a). *input* untuk fungsi *attention* terdiri dari *query* (Q), *key* (K), dan *value* (V). Fungsi *scaled dot-product attention* melakukan *dot product* pada nilai Q dan K, di-*scaling*, lalu dihitung nilai SoftMax. Kemudian nilai SoftMax dikalikan dengan nilai V, menghasilkan *hidden state* dari proses *attention* ini. Dalam implementasi ketiga variabel tersebut berupa matriks untuk menghitung beberapa *query* secara bersamaan. Fungsi *scaled dot-product attention* dapat dinotasikan dalam persamaan berikut [31]:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.1}$$



(a) *Scaled-Dot-Product Attention* (b) *Multi-Head Attention*

Gambar 2.3. *Attention* dalam *Transformer*
sumber: [31]

Untuk meningkatkan performa, perhitungan fungsi *attention* dapat dilakukan secara parallel dengan membagi *input* sebanyak h kali, menghasilkan h "*heads*" dengan dimensi $d_k = d_{model}/h$, seperti terlihat pada Gambar 2.3(b). Selain meningkatkan performa, setiap *head* juga dapat belajar untuk merepresentasikan *task* berbeda, seperti struktur semantik kalimat. Vaswani et al. [31] menggunakan $h = 8$ *attention head* secara parallel, sehingga setiap *head* mendapatkan informasi matriks berdimensi $d_k = 64$. Fungsi *mutli-head attention* hanya melakukan *concat* terhadap *output* setiap *attention head*, terlihat pada Gambar 2.3(b) dan dapat ditulis dalam persamaan berikut [31]:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \tag{2.2}$$

$$\text{dimana: } \text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{2.3}$$

2.3 BERT (Bidirectional Encoder Representations from Transformers)

Language model yang sudah di-*pre-train* telah ditemukan efektif dalam berbagai *task* NLP [12, 33] (dikutip [14]). *Pre-trained Language model* seperti GPT [12] menggunakan arsitektur *left-to-right*, sehingga token hanya dapat mereferensi token sebelumnya. Hal ini membatasi model untuk dilakukan *fine-tuning* untuk *task* spesifik, khususnya pada *task* yang memerlukan konteks dari dua arah, seperti *question answering*. Oleh karena itu, Devlin et al. [14] mengusulkan arsitektur model baru yang bersifat *bidirectional*. Untuk membuat *pre-trained model* dengan tujuan *fine-tuning* (*fine tuning approach*), BERT menggunakan hanya

Transformer encoder. BERT menghasilkan dua ukuran model, yaitu base dan large, dengan konfigurasi model berbeda.

Devlin et al. [14] berpendapat bahwa model *bidirectional* seharusnya lebih baik dibanding model yang hanya *left-to-right* atau *right-to-left*. Namun, *conditional language model* hanya dapat berbentuk *left-to-right* atau *right-to-left*, karena *bidirectional* akan menyebabkan setiap kata dapat melihat informasi yang seharusnya diprediksi ("*see itself*" [14]), sehingga mengurangi efektivitas model. Untuk mencegah hal tersebut, Devlin et al. [14] menggunakan *masking* saat pre-training. *Masking* dilakukan secara acak pada 15% data. Proses tersebut diberi nama *Masked LM* (MLM) [14]. Selain itu, BERT juga di-*pre-train* dengan tujuan *Next Sentence Prediction* (NSP) karena beberapa *downstream task* memerlukan model untuk memahami hubungan antara dua kalimat.

2.4 DeBERTa

Berbagai penelitian yang berdasarkan BERT berupaya untuk memperbaiki atau meningkatkan BERT [15, 16, 18]. Salah satu perkembangan terbarunya adalah DeBERTa (***D****ecoding-****e****nhanced* ***BERT*** *with disentangled* ***a****ttention*). DeBERTa diusulkan oleh He et al. [34] dari Microsoft. He et al. [34] mengusulkan dua metode baru untuk meningkatkan BERT, yaitu *disentangled attention mechanism* dan *enhanced mask encoder*.

Dengan mekanisme *disentangled attention*, DeBERTa menggunakan dua vektor terpisah untuk *embedding input* konten dan posisi, berbeda dengan BERT yang menjumlahkan vektor konten dan posisi sebagai satu *embedding input*. *Attention weight* dikalkulasi menggunakan *disentangled matrices*. Hal ini didasarkan pada observasi bahwa sepasang kata juga bergantung pada posisinya. Dengan adanya vektor tersendiri untuk posisi *input*, vektor ini juga dimanfaatkan dalam *decoder*. DeBERTa mengintegrasikan vektor posisi dalam *decoder*, tepat sebelum fungsi SoftMax [34].

Untuk *pre-training*, DeBERTa melakukan dengan konfigurasi yang sama dengan BERT [34], tetapi menggunakan *tokenizer* BPE dari GPT2 [17].

2.4.1 DeBERTa-V2

DeBERTa-V2 tidak terdapat publikasi tertulis. Perubahan dalam DeBERTa tertulis pada repo GitHub DeBERTa [35], sebagai berikut:

1. *Vocabulary* atau *tokenizer* yang digunakan diganti menjadi SentencePiece [36].
2. Menambahkan layer *convolution* untuk mempelajari *local dependency*.
3. *Projection matrix* pada *attention* digunakan yang sama untuk posisi dan konten.
4. Menggunakan *bucket* untuk *encode* posisi seperti pada T5.
5. Merilis model 900M dan 1.5B.

2.4.2 DeBERTa-V3

DeBERTa-V3 meningkatkan efisiensi dengan menggunakan metode ELECTRA [37] (***E**fficiently **L**earning an **E**ncoder that **C**lassifies **T**oken **R**eplacements **A**ccurately*) untuk *pre-training* [38]. ELECTRA merupakan metode *pre-training* yang menggantikan metode *Masked LM* (MLM) pada BERT, dengan metode yang dinamakan *replaced token detection* (RTD).

ELECTRA menggunakan dua model yaitu *generator* dan *discriminant*, serupa dalam model GAN [39], tetapi tidak *adversarial* karena kesulitan untuk implementasi GAN pada teks [37]. *Generator* pada ELECTRA merupakan MLM, seperti BERT, yang memprediksi token *output* untuk setiap token "`[MASK]`". *Discriminator* memprediksi apakah data tersebut berasal dari data asli (*input* awal) atau merupakan hasil *output* dari *generator*. Keduanya *Generator* dan *Discriminator* merupakan *Transformer encoder*. Untuk *pre-training*, dilakukan optimasi untuk meminimalisasi nilai *loss function*, gabungan dari *loss generator* dan *loss discriminant*. Pada *fine-tuning*, generator sudah tidak digunakan dan diskriminator digunakan sebagai model BERT.

2.5 Metrik Evaluasi

Metrik evaluasi yang digunakan dalam penelitian ini meliputi *accuracy*, *F1-score*, dan *BLEU*.

2.5.1 Accuracy

Accuracy adalah metrik evaluasi yang mengukur proporsi prediksi benar dari prediksi yang dilakukan. Nilai *accuracy* umumnya tidak digunakan untuk

text classification task, karena mayoritas data teks (email dan tweet) cenderung mempunyai kelas yang tidak seimbang [40]. Namun untuk tugas perbaikan kalimat, nilai akurasi cukup dapat menggambarkan kemampuan model dalam memperbaiki kalimat.

Secara matematis, *accuracy* didefinisikan sebagai:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.4}$$

di mana:

- TP (*True Positive*) adalah jumlah kalimat yang mengandung kesalahan dan dikoreksi dengan benar oleh model,
- TN (*True Negative*) adalah jumlah kalimat yang tidak mengandung kesalahan dan tidak diubah oleh model,
- FP (*False Positive*) adalah jumlah kalimat yang tidak mengandung kesalahan tetapi diubah oleh model, dan
- FN (*False Negative*) adalah jumlah kalimat yang mengandung kesalahan tetapi tidak dikoreksi dengan benar oleh model.

2.5.2 F1-score

F1-score adalah metrik evaluasi yang menggabungkan *precision* dan *recall* menjadi satu nilai tunggal menggunakan rata-rata harmonik.

Precision dan *recall* didefinisikan sebagai berikut:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.5}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.6}$$

dan *F1-score* sebagai berikut:

$$F_1\text{-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.7}$$

2.5.3 BLEU

BLEU (*Bilingual Evaluation Understudy*) adalah metrik yang digunakan untuk menilai kesamaan keluaran model terhadap referensi pada tingkat token. Meskipun evaluasi dalam penelitian ini dilakukan pada level kalimat (*sentence-level*), *BLEU* disertakan untuk memberikan informasi tambahan mengenai kedekatan kata demi kata pada hasil koreksi model terhadap kalimat.

Secara formal, skor BLEU dihitung berdasarkan *n-gram* yang cocok antara keluaran model dan referensi. Untuk BLEU-*n*, proporsi *n-gram* yang benar dihitung, dan dikombinasikan dengan *brevity penalty* untuk menghindari skor tinggi pada keluaran yang terlalu pendek. Skor BLEU didefinisikan sebagai berikut:

$$\text{BP} = \begin{cases} 1, & \text{jika } c > r \\ \exp\left(1 - \frac{r}{c}\right), & \text{jika } c \leq r \end{cases} \tag{2.8}$$

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right) \tag{2.9}$$

di mana:

- p_n adalah proporsi *n-gram* yang cocok antara keluaran model dan referensi,
- w_n adalah bobot untuk setiap *n*-gram (biasanya sama untuk semua *n*), dan

Penggunaan *BLEU* ini memberikan informasi tambahan pada level token untuk kemiripan antara *output* model dengan referensi, melengkapi evaluasi *accuracy* dan *F1-score* yang dilakukan pada level kalimat.