UNIVERSITAS INDONESIA

PEMERINGKATAN TEKS BAHASA INDONESIA DENGAN BERT

SKRIPSI

CARLES OCTAVIANUS
2006568613

FAKULTAS FAKULTAS MATEMATIKA DAN ILMU PENGATAHUAN ALAM
PROGRAM STUDI MATEMATIKA
DEPOK
DESEMBER 2023

UNIVERSITAS INDONESIA

PEMERINGKATAN TEKS BAHASA INDONESIA DENGAN BERT

SKRIPSI

Diajukan sebagai salah satu syarat untuk memperoleh gelar

Sarjana Sains

CARLES OCTAVIANUS

2006568613

FAKULTAS FAKULTAS MATEMATIKA DAN ILMU PENGATAHUAN ALAM

PROGRAM STUDI MATEMATIKA

DEPOK

DESEMBER 2023

HALAMAN PERNYATAAN ORISINALITAS

Skripsi ini adalah hasil karya saya sendiri,
dan semua sumber baik yang dikutip maupun dirujuk
telah saya nyatakan dengan benar.

Nama	:	**Carles Octavianus**
NPM	:	**2006568613**
Tanda Tangan	:	
Tanggal	:	**2 Desember 2023**

HALAMAN PENGESAHAN

Skripsi ini diajukan oleh	:	
Nama	:	Carles Octavianus
NPM	:	2006568613
Program Studi	:	Matematika
Judul Skripsi	:	Pemeringkatan Teks Bahasa Indonesia Dengan BERT

Telah berhasil dipertahankan di hadapan Dewan Penguji dan diterima sebagai bagian persyaratan yang diperlukan untuk memperoleh gelar Sarjana pada Program Studi Matematika, Fakultas Fakultas Matematika dan Ilmu Pengatahuan Alam, Universitas Indonesia.

DEWAN PENGUJI

Pembimbing 1	:	Sarini Abdullah S.Si., M.Stats., Ph.D.	( )
Penguji 1	:	Penguji Pertama Anda	( )
Penguji 2	:	Penguji Kedua Anda	( )

Ditetapkan di	:	Depok
Tanggal	:	2 Desember 2023

KATA PENGANTAR

Template ini disediakan untuk orang-orang yang berencana menggunakan LaTeX untuk membuat dokumen tugas akhir.

@todo

Silakan ganti pesan ini dengan pendahuluan kata pengantar Anda.

Ucapan Terima Kasih:

1. Pembimbing.
2. Dosen.
3. Instansi.
4. Orang tua.
5. Sahabat.
6. Teman.

Penulis menyadari bahwa laporan Skripsi ini masih jauh dari sempurna. Oleh karena itu, apabila terdapat kesalahan atau kekurangan dalam laporan ini, Penulis memohon agar kritik dan saran bisa disampaikan langsung melalui *e-mail* `emailanda@mail.id`.

Terkait template ini, gambar lisensi di atas diambil dari `http://creativecommons.org/licenses/by-nc-sa/1.0/deed.en_CA`. Jika ingin mengentahui lebih lengkap mengenai *Creative Common License 1.0 Generic*, silahkan buka `http://creativecommons.org/licenses/by-nc-sa/1.0/legalcode`. Seluruh dokumen yang dibuat dengan menggunakan template ini sepenuhnya menjadi hak milik pembuat dokumen dan bebas didistribusikan sesuai dengan keperluan masing-masing. Lisensi hanya berlaku jika ada orang yang membuat template baru dengan menggunakan template ini sebagai dasarnya.

Penyusun template ingin berterima kasih kepada Andreas Febrian, Lia Sadita, Fahrurrozi Rahman, Andre Tampubolon, dan Erik Dominikus atas kontribusinya dalam template yang menjadi pendahulu template ini. Penyusun template juga ingin mengucapkan terima kasih kepada Azhar Kurnia atas kontribusinya dalam template yang menjadi pendahulu template ini.

Semoga template ini dapat membantu orang-orang yang ingin mencoba menggu-

nakan LaTeX. Semoga template ini juga tidak berhenti disini dengan ada kontribusi dari para penggunanya. Jika Anda memiliki perubahan yang dirasa penting untuk disertakan dalam template, silakan lakukan *fork* repositori Git template ini di `https://gitlab.com/ichlaffterlalu/latex-skripsi-ui-2017`, lalu lakukan *merge request* perubahan Anda terhadap *branch* `master`. Kami berharap agar *template* ini dapat terus diperbarui mengikuti perubahan ketentuan dari pihak Rektorat Universitas Indonesia, dan hal itu tidak mungkin terjadi tanpa kontribusi dari teman-teman sekalian.

Depok, 2 Desember 2023

Carles Octavianus

HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI TUGAS AKHIR UNTUK KEPENTINGAN AKADEMIS

Sebagai sivitas akademik Universitas Indonesia, saya yang bertanda tangan di bawah ini:

Nama	:	Carles Octavianus
NPM	:	2006568613
Program Studi	:	Matematika
Jenis Karya	:	Skripsi

demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Universitas Indonesia **Hak Bebas Royalti Noneksklusif (*Non-exclusive Royalty Free Right*)** atas karya ilmiah saya yang berjudul:

Pemeringkatan Teks Bahasa Indonesia Dengan BERT

beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Noneksklusif ini Universitas Indonesia berhak menyimpan, mengalihmedia/formatkan, mengelola dalam bentuk pangkalan data (*database*), merawat, dan memublikasikan tugas akhir saya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyataan ini saya buat dengan sebenarnya.

Dibuat di : Depok
Pada tanggal : 2 Desember 2023
Yang menyatakan

(Carles Octavianus)

ABSTRAK

Nama	:	Carles Octavianus
Program Studi	:	Matematika
Judul	:	Pemeringkatan Teks Bahasa Indonesia Dengan BERT
Pembimbing	:	Sarini Abdullah S.Si., M.Stats., Ph.D.

Isi abstrak.

Kata kunci:
Keyword satu, kata kunci dua

ABSTRACT

Name	:	Carles Octavianus
Study Program	:	Mathematics
Title	:	Text Ranking in Indonesian Using BERT
Counselor	:	Sarini Abdullah S.Si., M.Stats., Ph.D.

Abstract content.

Key words:
Keyword one, keyword two

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR KODE PROGRAM

DAFTAR LAMPIRAN

BAB 1
PENDAHULUAN

@todo
wew

BAB 2
LANDASAN TEORI

2.1 Masalah Pemeringkatan Teks

Permasalahan pemeringkatan teks adalah Permasalahan untuk menentukan urutan dokumen yang paling relevan dengan kueri q yang diberikan. Dalam bahasa yang lebih formal, diberikan kueri q dan himpunan dokumen terbatas $\mathcal{D} = \{d_1, d_2, ..., d_n\}$, keluaran yang diinginkan dari permasalahan ini adalah barisan dokumen $D_k = (d_{i_1}, d_{i_2}, ..., d_{i_k})$ yang merupakan k dokumen yang paling relevan dengan kueri q. Selain itu, biasanya nilai k akan lebih kecil dari banyaknya dokumen yang ada, sehingga permasalahan pemeringkatan sering juga disebut sebagai *top-k retrieval*. Untuk mengukur performa suatu model pemeringkatan, biasanya digunakan metrik evaluasi seperti presisi, *recall*, *reciprocal rank*, dan *normalized discounted cumulative gain* (nDCG) yang akan dijelaskan pada Subbab 2.1.2.

2.1.1 Bentuk Umum *Dataset*

Sebelum menjelaskan metrik evaluasi, akan dijelaskan terlebih dahulu bentuk umum dari *dataset* yang digunakan untuk mengevaluasi sebuah sistem pemeringkatan teks. Bentuk umum dari *dataset* yang digunakan biasanya terdiri dari 3 *file*, yaitu *file* korpus, *file* kueri, dan *file judgements*. *File* korpus adalah kumpulan dokumen/teks yang ingin di-*retreive* oleh sebuah sistem pemeringkatan teks. Biasanya, pada *file* korpus terdapat 3 kolom, yaitu id teks, judul teks, dan isi dari teks tersebut. Tabel 2.1 menunjukkan potongan dari *file* korpus.

Tabel 2.1: Potongan *file* korpus *dataset* MIRACL.

_id	title	text
1342516#1	Colobothea biguttata	Larva kumbang ini biasanya mengebor ke dalam kayu dan dapat menyebabkan kerusakan pada batang kayu hidup atau kayu yang telah ditebang.
1342517#0	Ichthyodes rufipes	Ichthyodes rufipes adalah spesies kumbang tanduk panjang yang berasal dari famili Cerambycidae. Spesies ini juga merupakan bagian dari genus Ichthyodes, ordo Coleoptera, kelas Insecta, filum Arthropoda, dan kingdom Animalia.

file kueri berisi kumpulan kueri yang digunakan untuk mengambil dokumen dari *file* korpus. performa dari sistem pemeringkatan teks akan diukur dengan mengambil k dokumen dari *file* korpus untuk setiap kueri pada *file* kueri. Biasanya, pada *file* kueri terdapat 2 kolom, yaitu id kueri dan isi dari kueri tersebut. Tabel 2.2 menunjukkan potongan dari *file* kueri.

Tabel 2.2: Potongan *file* kueri *dataset* MIRACL.

_id	text
3	Dimana James Hepburn meninggal?
4	Dimana Jamie Richard Vardy lahir?
11	berapakah luas pulau Flores?
17	Siapakah yang menulis Candy Candy?
19	Apakah karya tulis Irma Hardisurya yang pertama?

Selanjutnya *file judgements* berisi pemetaan relevansi antara kueri pada *file* kueri dengan dokumen pada *file* korpus. Biasanya, pada *file* judgements terdapat 3 kolom, yaitu id kueri, id dokumen, dan relevansi antara kueri dan dokumen tersebut (r). Pasangan $(\text{kueri}, \text{dokumen})$ yang relevan akan memiliki nilai $r > 0$ dan nilai r yang makin besar menunjukkan relevansi yang makin tinggi. Selain itu, pasangan $(\text{kueri}, \text{dokumen})$ yang tidak relevan akan memiliki nilai $r = 0$ dan biasanya pasangan $(\text{kueri}, \text{dokumen})$ yang tidak relevan tidak dituliskan pada *file judgements*. tak menutup kemungkinan jika sebuah *dataset* hanya menggunakan nilai relevansi biner ($r \in \{0, 1\}$). Terakhir, Tabel 2.3 menunjukkan potongan dari *file judgements*.

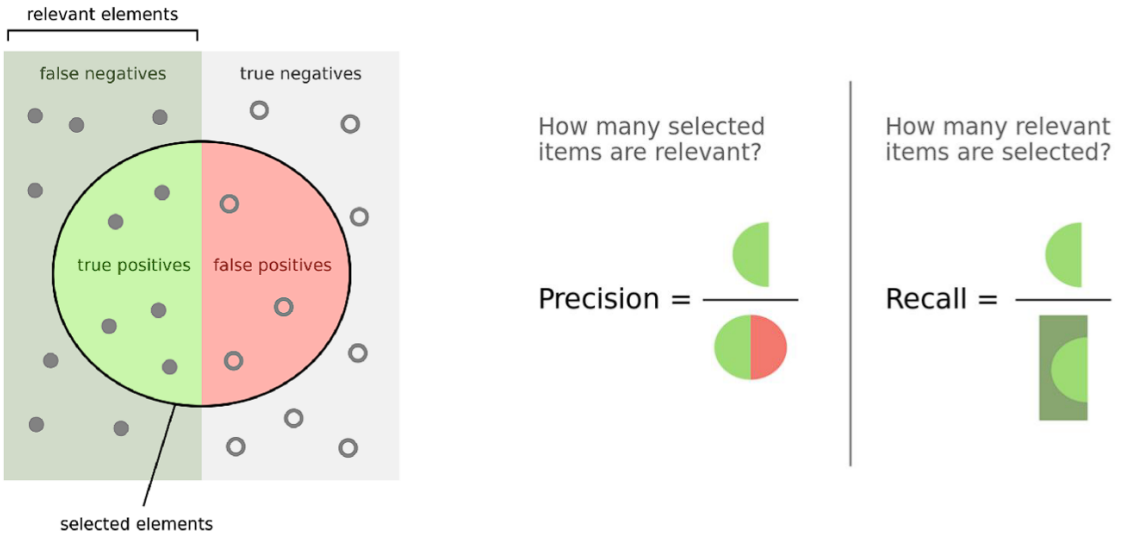
Tabel 2.3: Potongan *file* judgements *dataset* MIRACL.

query-id	corpus-id	score
3	115796#6	1
3	77689#48	1
4	1852373#0	1

2.1.2 Metrik Evaluasi dalam Pemeringkatan Teks

Subbab ini menjelaskan beberapa metrik evaluasi yang sering digunakan untuk mengukur performa dari sistem pemeringkatan teks. Metrik evaluasi yang akan dijelaskan adalah *recall*, presisi, *reciprocal rank*, dan *normalized discounted cumulative gain* (nDCG). Metrik tersebut digunakan untuk mengukur performa dari sistem pemeringkatan teks dengan mengambil k dokumen dari *file* korpus pada satu kueri. Untuk mendapatkan performa dari sistem pemeringkatan teks secara keseluruhan, biasanya metrik evaluasi tersebut akan dihitung untuk setiap kueri pada *file* kueri dan kemudian diambil nilai rata-ratanya.

2.1.2.1 *Recall* dan Presisi



Gambar 2.1: Ilustrasi *recall* dan presisi. Nilai *recall* dihitung sebagai rasio dokumen relevan yang diambil oleh sistem terhadap seluruh dokumen yang relevan dengan kueri q. Sedangkan nilai presisi dihitung sebagai rasio dokumen relevan yang diambil oleh sistem terhadap seluruh dokumen yang diambil oleh sistem.

Presisi dan *recall* adalah metrik yang paling sederhana untuk mengukur kemampuan dari suatu sistem pemeringkatan teks. *Recall* mengukur kemampuan sistem dalam mengemba-

likan semua dokumen yang relevan dengan kueri q dari himpunan dokumen $\mathcal{D}$, sedangkan presisi mengukur kemampuan sistem dalam mengembalikan dokumen yang relevan dengan kueri q dari himpunan dokumen $\mathcal{D}$. Untuk suatu kueri q, kumpulan dokumen $\mathcal{D} = \{d_1, d_2, ..., d_n\}$, dan barisan k dokumen yang diambil oleh sistem, $D_k = (d_{i_1}, d_{i_2}, ..., d_{i_k})$, *recall* dan presisi dapat dihitung dengan Persamaan 2.1 hingga Persamaan 2.4.

$$\mathcal{D} = \{d_1, d_2, \ldots, d_n\} \tag{2.1}$$

$$D_k = (d_{i_1}, d_{i_2}, \ldots, d_{i_k}) \tag{2.2}$$

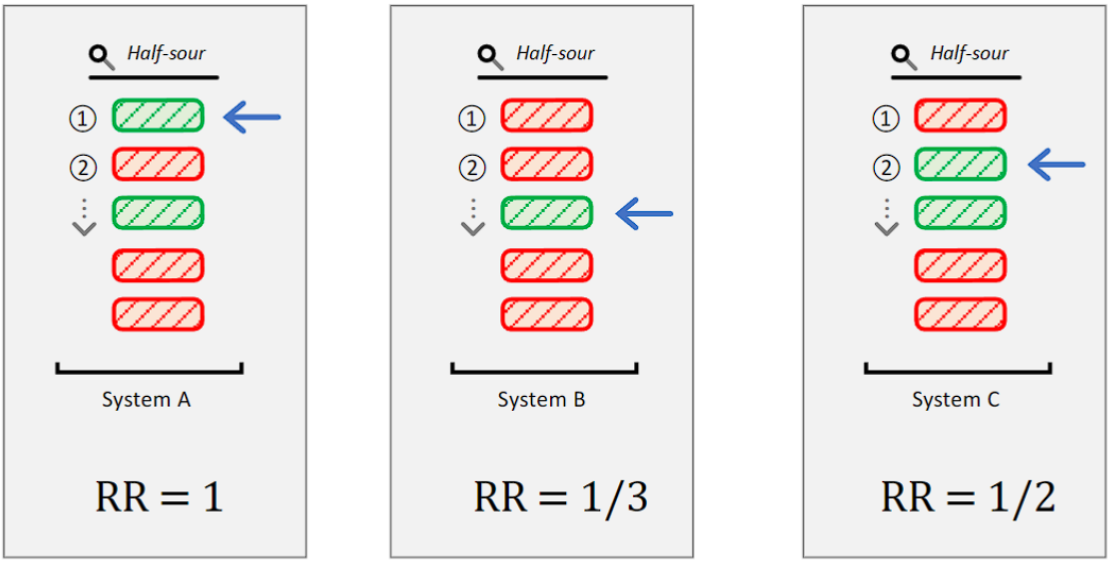
$$\text{recall}(q, D_k)@\text{k} = \frac{\sum_{d \in D_k} \text{rel}(q,d)}{\sum_{d \in \mathcal{D}} \text{rel}(q,d)} \in [0,1] \tag{2.3}$$

$$\text{precision}(q, D_k)@\text{k} = \frac{\sum_{d \in D_k} \text{rel}(q,d)}{|D_k|} \in [0,1] \tag{2.4}$$

$$\text{dengan rel}(q,d) = \begin{cases} 1 & \text{jika } r > 1 \\ 0 & \text{jika } r = 0 \end{cases} \tag{2.5}$$

Sebagai Contoh, Jika terdapat 10 dokumen yang relevan dengan kueri q, dan sistem mengembalikan $k = 100$ dokumen, namun hanya terdapat 5 dokumen yang relevan pada D_k maka *recall* dan presisi dari sistem tersebut adalah 0.5 ($\frac{5}{10}$) dan 0.05 ($\frac{5}{100}$) masing-masing. Baik *recall* maupun presisi memiliki rentang nilai dari 0 hingga 1, dengan nilai 1 menunjukkan performa sistem yang terbaik. perhitungan *recall* biasanya dilakukan untuk k yang cukup besar ($k = 100, 1000$), sedangkan perhitungan presisi dilakukan untuk k yang kecil ($k = 1, 3, 5$) (Hofstätter et al., 2021).

2.1.2.2 *Reciprocal Rank*



Gambar 2.2: Ilustrasi *reciprocal rank*.

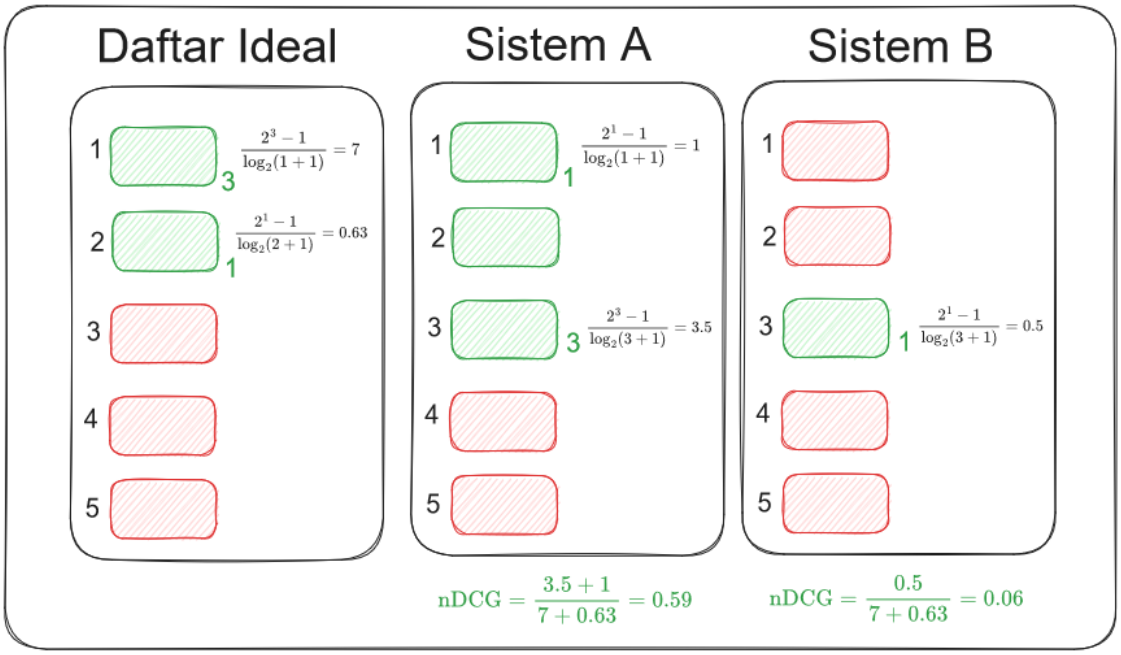
Metrik lainnya yang sering digunakan untuk mengukur performa sistem pemeringkatan adalah *reciprocal rank* (RR). Metrik RR menitikberatkan pada peringkat dari dokumen relevan pertama dengan kueri q. Persamaan 2.6 hingga Persamaan 2.7 menunjukkan cara menghitung RR dari suatu kueri q dan barisan k dokumen yang diambil oleh sistem (Lin et al., 2020; **?**).

$$\mathrm{RR}(q, D_k)@\mathrm{k} = \begin{cases} \frac{1}{\mathrm{FirstRank}(q, D_k)} & \text{jika } \exists d \in D_k \text{ dengan } \mathrm{rel}(q,d) = 1 \\ 0 & \text{jika } \forall d \in D_k,\ \mathrm{rel}(q,d) = 0 \end{cases} \in [0,1] \quad (2.6)$$

$$\mathrm{FirstRank}(q, D_k) = \text{posisi dokumen relevan pertama } d \in D_k \text{ dengan } \mathrm{rel}(q,d) = 1 \quad (2.7)$$

Gambar 2.2 mengilustrasikan metrik RR. Pada gambar tersebut, nilai RR dari sistem A adalah 1 $(\frac{1}{1})$ karena posisi dari dokumen yang relevan pertama adalah 1. Nilai RR dari sistem B dan sistem C masing-masing adalah 0.33 $(\frac{1}{3})$ dan 0.5 $(\frac{1}{2})$ karena posisi dari dokumen yang relevan pertama adalah 3 dan 2. Selain itu, jika tidak terdapat dokumen yang relevan dengan kueri q pada D_k, nilai RR dari sistem tersebut adalah 0.

2.1.2.3 *Normalized Discounted Cumulative Gain* (nDCG)



Gambar 2.3: *Creative Common License 1.0 Generic*.

Normalized Discounted Cumulative Gain (nDCG) adalah metrik yang umumnya digunakan untuk mengukur kualitas dari pencarian situs web. Tidak seperti metrik yang telah disebutkan sebelumnya, nDCG dirancang untuk suatu *judgements r* yang tak biner. Fungsi rel(q,d) pada Persamaan 2.5 berubah menjadi rel(q,d) $= r$ ketika menghitung metrik nDCG. Persamaan 2.8 hingga Persamaan 2.10 menunjukkan cara menghitung nDCG dari suatu kueri q dan barisan k dokumen yang diambil oleh sistem.

$$\text{nDCG}(q,D_k)@\text{k} = \frac{\text{DCG}(q,D_k)@\text{k}}{\text{DCG}(q,D_k^{\text{ideal}})@\text{k}} \in [0,1] \tag{2.8}$$

$$\text{DCG}(q,D_k)@\text{k} = \sum_{d \in D_k} \frac{2^{\text{rel}(q,d)}-1}{\log_2(\text{rank}(d,D_k)+1)} \tag{2.9}$$

$$\text{rank}(d,D_k) = \text{Posisi } d \text{ dalam } D_k \tag{2.10}$$

$$\text{rel}(q,d) = r \tag{2.11}$$

Perhitungan *discounted cumulative gain* (DCG) pada Persamaan 2.9 dapat dijelaskan menjadi dua faktor, yaitu:

1. faktor $2^{\text{rel}(q,d)}-1$ menunjukkan bahwa dokumen yang lebih relevan akan memiliki

nilai yang lebih tinggi dari dokumen yang kurang relevan untuk posisi dokumen yang sama.

2. faktor $\frac{1}{\log_2(\text{rank}(d,D_k)+1)}$ menunjukkan bahwa dokumen yang relevan yang muncul pada peringkat yang lebih tinggi akan memiliki nilai yang lebih tinggi dari dokumen dengan relevansi yang sama, tetapi muncul pada peringkat yang lebih rendah.

nilai dari nDCG pada Persamaan 2.8 adalah nilai DCG pada barisan dokumen D_k yang dinormalisasi oleh nilai DCG pada barisan dokumen ideal D_k^{ideal}. Barisan dokumen ideal D_k^{ideal} adalah barisan dokumen yang diurutkan berdasarkan relevansinya dengan kueri q.

Selain itu, jika pada *dataset* memiliki *judgements* biner, faktor $2^{\text{rel}(q,d)} - 1$ pada Persamaan 2.9 dapat diubah menjadi $\text{rel}(q,d)$. Akibatnya, Persamaan 2.9 akan menjadi Persamaan 2.12.

$$\text{DCG}(q,D_k)@\text{k} = \sum_{d \in D_k} \frac{\text{rel}(q,d)}{\log_2(\text{rank}(d,D_k)+1)}. \tag{2.12}$$

2.2 Pemeringkatan Teks dengan Statistik

Untuk mengambil k dokumen dari kumpulan $\mathcal{D}$ diperlukan suatu fungsi skor $s(q,d,\mathcal{D})$ yang mengukur relevansi antara kueri q dan dokumen d. dengan mencari skor antara q terhadap semua dokumen pada $\mathcal{D}$, Barisan dokumen $D_k = (d_{i_1}, d_{i_2}, \ldots, d_{i_k})$ dapat dipilih sehingga $\text{score}(q,d_{i_1}) \geq \text{score}(q,d_{i_2}) \geq \cdots \geq \text{score}(q,d_{i_k})$ adalah k dokumen dengan skor tertinggi.

Bagian ini menjelaskan dua fungsi skor stastistik sederhana yang menjadi *baseline* ketika membandingkan performa dari model pemeringkatan teks yang lebih kompleks. Subbab 2.2.1 menjelaskan fungsi skor statistik yang berdasarkan pada frekuensi kemunculan kata pada dokumen dan kueri. Selanjutnya, Subbab 2.2.2 membahas fungsi skor statistik yang menjadi standar *de facto* dalam pemeringkatan teks.

2.2.1 *Term Frequency - Inverse Document Frequency* (TF-IDF)

fungsi skor TF-IDF adalah fungsi skor statistik yang menghitung score antara kueri q dan dokumen d dengan menghitung frekuensi kemunculan kata pada dokumen dan kueri. Untuk suatu kueri q, misalkan $T_q = \{t_1, t_2, \ldots, t_{L_1}\}$ adalah himpunan kata yang terdapat pada kueri q. Selain itu, misalkan juga $T_d = \{t_1, t_2, \ldots, t_n\}$ adalah himpunan kata yang terdapat

pada dokumen d. nilai skor antara q dan d diberikan oleh persamaan Persamaan 2.13 sampai Persamaan 2.21.

$$\mathcal{D} = \{d_1, d_2, \ldots, d_n\} \tag{2.13}$$

$$T_q = \{t_1, t_2, \ldots, t_{L_1}\} \tag{2.14}$$

$$T_d = \{t_1, t_2, \ldots, t_{L_2}\} \tag{2.15}$$

$$\text{tf}(t,d) = \frac{\text{Count}(t,d)}{|d|} \tag{2.16}$$

$$\text{Count}(t,d) = \text{jumlah kemunculan } t \text{ dalam } d \tag{2.17}$$

$$\text{df}(t,\mathcal{D}) = \text{jumlah dokumen pada } \mathcal{D} \text{ yang mengandung } t \tag{2.18}$$

$$\text{idf}(t,\mathcal{D}) = \begin{cases} \log_2\left(\frac{|\mathcal{D}|}{\text{df}(t,\mathcal{D})}\right) & \text{jika df}(t,\mathcal{D}) > 0 \\ 0 & \text{jika df}(t,\mathcal{D}) = 0 \end{cases} \tag{2.19}$$

$$\text{TF-IDF}(t,d,\mathcal{D}) = \text{tf}(t,d) \times \text{idf}(t,\mathcal{D}) \tag{2.20}$$

$$\text{RelevantScore}(q,d,\mathcal{D}) = \sum_{t \in T_q \cap T_d} \text{tf-idf}(t,d,\mathcal{D}) \tag{2.21}$$

	doc_1	doc_2	doc_3	doc_4
A	10	10	10	10
B	10	10	10	0
C	10	10	0	0
D	0	0	0	1

⇒

	IDF
A	0.00
B	0.29
C	0.69
D	1.39

⇓

TF

	doc_1	doc_2	doc_3	doc_4
A	0.33	0.33	0.50	0.91
B	0.33	0.33	0.50	0.00
C	0.33	0.33	0.00	0.00
D	0.00	0.00	0.00	0.09

TF-IDF

	doc_1	doc_2	doc_3	doc_4
A	0.00	0.00	0.00	0.00
B	0.10	0.10	0.14	0.00
C	0.23	0.23	0.00	0.00
D	0.00	0.00	0.00	0.13

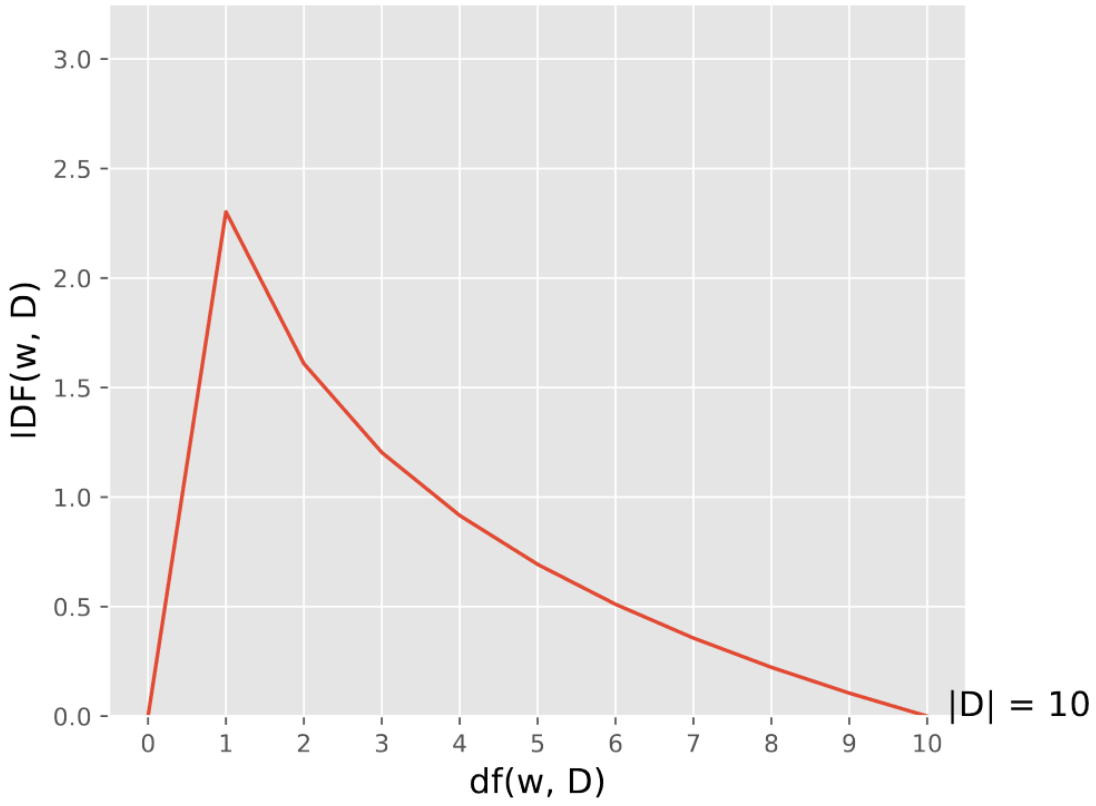
Gambar 2.4: *Creative Common License 1.0 Generic.*

skor untuk pasangan (q,d) dihitung dengan menjumlahkan skor TF-IDF dari setiap kata yang terdapat pada kueri q dan dokumen d. skor TF-IDF dari suatu kata t adalah perkalian antara *term frequency* ($\text{tf}(q,d)$) dan *inverse document frequency* ($\text{idf}(t,\mathcal{D})$). fungsi skor pada Persamaan 2.21 dapat dijelaskan menjadi dua bagian faktor utama

berikut:

1. faktor $\text{tf}(t,d)$ menunjukkan bahwa nilai TF-IDF meningkat seiring dengan bertambahnya frekuensi kemunculan kata t pada dokumen d.

2. Faktor $\text{idf}(t,\mathcal{D})$ menunjukkan bahwa nilai TF-IDF meningkat seiring dengan *rarity* dari kata t pada himpunan dokumen $\mathcal{D}$. Akibatnya, kata yang jarang muncul pada himpunan dokumen $\mathcal{D}$ dan muncul pada suatu dokumen tertentu akan menghasilkan skor yang tinggi. Sementara itu, kata-kata yang sering muncul pada koleksi dokumen $\mathcal{D}$ memiliki nilai *downgraded*.



Gambar 2.5: *Creative Common License 1.0 Generic.*

Kata-kata seperti preposisi atau kata ganti akan menghasilkan skor TF-IDF yang sangat rendah. Hal ini menyiratkan bahwa kata-kata tersebut memiliki sedikit relevansi dalam dokumen dan bisa diabaikan. Di sisi lain, kata-kata yang muncul secara berlebihan dalam satu dokumen tetapi jarang muncul dalam dokumen lainnya akan menghasilkan nilai $\text{tf}(t,d)$ dan $\log\left(\frac{\mathcal{D}}{\text{df}(t,\mathcal{D})}\right)$ yang relatif besar. Dampaknya adalah skor TF-IDF yang dihasilkan juga menjadi signifikan. Gambar 2.4 menunjukkan contoh perhitungan skor

TF-IDF untuk suatu kumpulan dokumen dan Gambar 2.5 menunjukkan grafik dari fungsi idf.

2.2.2 *Best Match 25* (BM25)

BM25 (*Best Match attempt* 25) merupakan pengembangan dari fungsi skor TF-IDF dengan perbedaan utama pada fungsi nilai yang berkaitan dengan frekuensi kata – digunakan $\text{score}_{\text{BM25}}(q,d)$ (Persamaan 2.24) daripada $\text{tf}(q,d)$ (Persamaan 2.20). Pada fungsi $\text{score}_{\text{BM25}(q,d)}$ terdapat 2 parameter yang dapat diatur, yaitu b, dan k_1. Setiap parameter mempunyai efek yang berbeda terhadap nilai $\text{score}_{\text{BM25}(q,d)}$ yang dihasilkan. Sebelum menjelaskan efek dari setiap parameter, Persamaan 2.22 hingga Persamaan 2.26 menunjukkan cara menghitung skor relevansi dari suatu kueri q dan dokumen d.

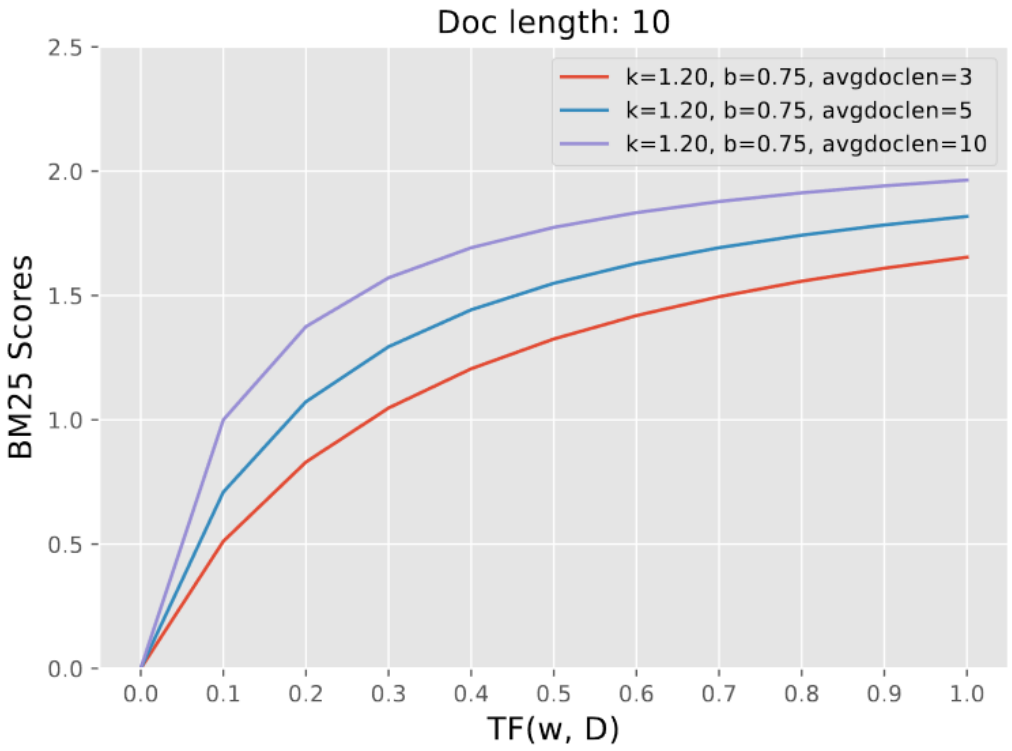
$$\text{idf}_{\text{BM25}}(t,\mathcal{D}) = \log\left(1 + \frac{|\mathcal{D}| - \text{df}(t,\mathcal{D}) + 0.5}{\text{df}(t,\mathcal{D}) + 0.5}\right) \tag{2.22}$$

$$\text{score}_{\text{BM25}}(t,d) = \frac{\text{tf}(t,d) \times (k_1+1)}{\text{tf}(t,d) + k_1 \times (1 - b + b \times \frac{|d|}{\text{avgdl}})} \tag{2.23}$$

$$\text{BM25}(t,d,\mathcal{D}) = \text{idf}_{\text{BM25}}(t,\mathcal{D}) \times \text{score}_{\text{BM25}}(q,d,\mathcal{D}) \tag{2.24}$$

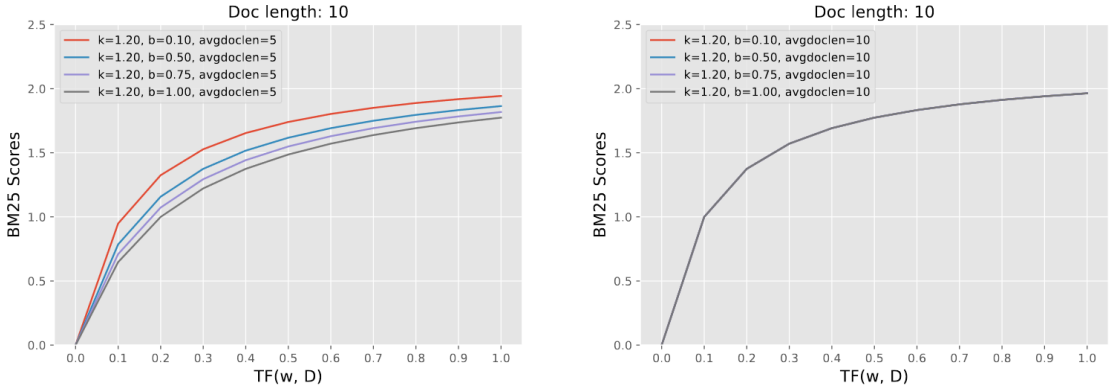
$$\text{avgdl} = \text{rata-rata panjang dokumen pada koleksi } \mathcal{D} \tag{2.25}$$

$$\text{RelevantScore}(q,d,\mathcal{D}) = \sum_{t \in T_q \cap T_d} \text{BM25}(t,d,\mathcal{D}) \tag{2.26}$$



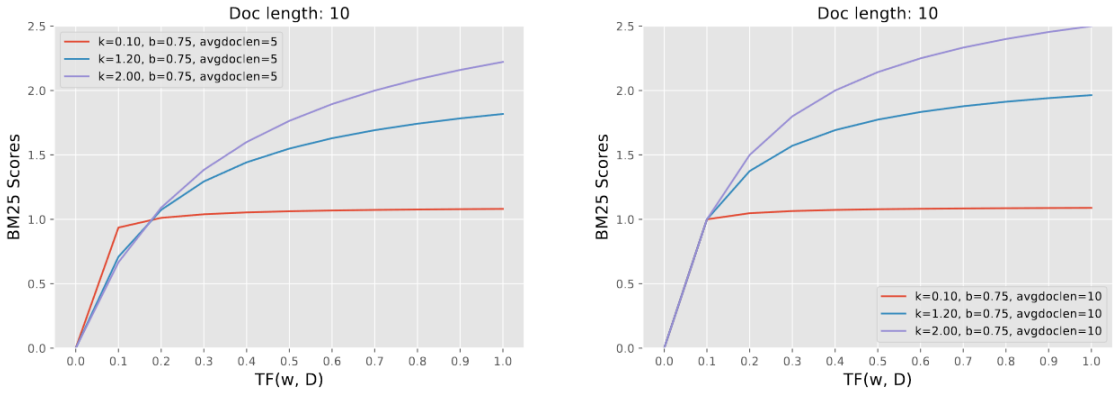
Gambar 2.6: *Creative Common License 1.0 Generic*.

;



Gambar 2.7: *Creative Common License 1.0 Generic*.

;



Gambar 2.8: *Creative Common License 1.0 Generic*.

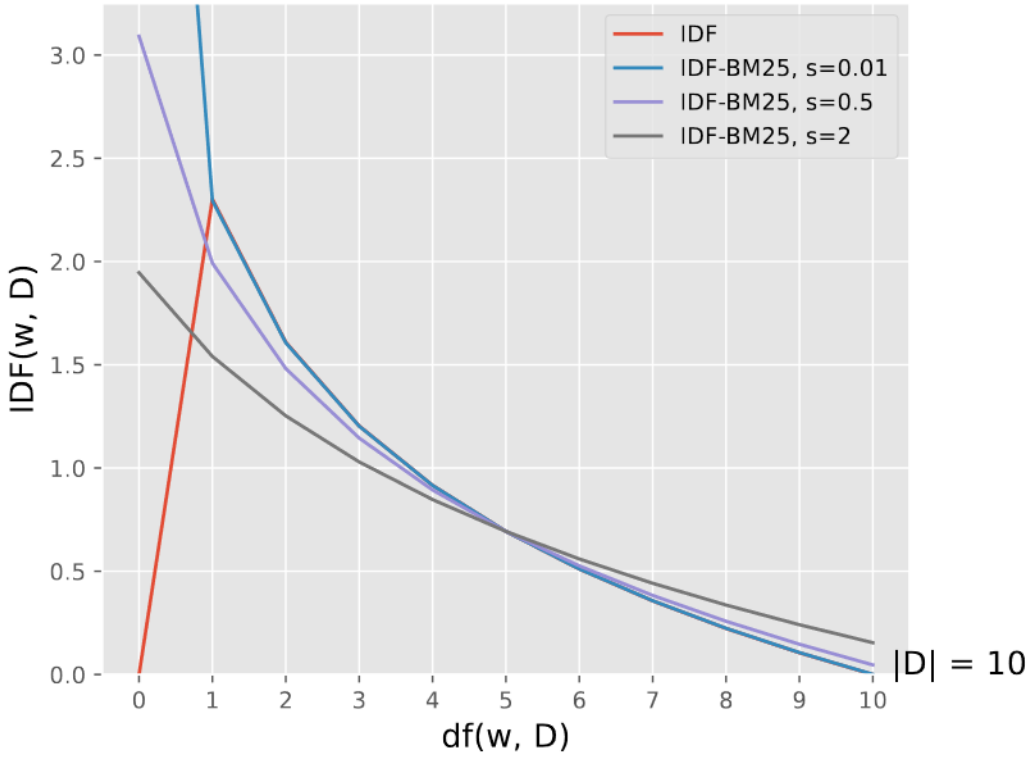
;

Efek dari masing-masing parameter dan faktor pada $\text{score}_{\text{BM25}}(t,d)$ dapat dijelaskan sebagai berikut:

1. faktor $\frac{|\text{d}|}{\text{avgdl}}$ pada $\frac{\text{tf}(t,d)\times(k_1+1)}{\text{tf}(t,d)+k_1\times\left(1-b+b\times\frac{|d|}{\text{avgdl}}\right)}$ men-*penalize* skor pada dokumen yang panjangnya lebih besar dari rata-rata panjang dokumen pada himpunan dokumen $\mathcal{D}$. Gambar 2.6 Menjukkan efek dari perbedaan nilai avgdl terhadap skor yang dihasilkan, makin besar rasio $\frac{|d|}{\text{avgdl}}$ makin kecil skor yang dihasilkan.

2. nilai b menentukan seberapa besar efek dari faktor $\frac{|d|}{\text{avgdl}}$ terhadap skor yang dihasilkan. Gambar 2.7 Menunjukkjan efek dari perbedaan nilai b terhadap skor yang dihasilkan. Untuk $\frac{|d|}{\text{avgdl}} = 1$, Faktor b tidak memiliki pengaruh terhadap skor. Nilai b yang umum dipilih berada pada rentang $[0.5, 0.8]$.

3. nilai k_1 men-*penalize* kemunculan kata t pada dokumen d yang berlebih. Gambar 2.8 Menunjukkan efek dari perbedaan nilai k_1 terhadap skor yang dihasilkan. untuk nilai k_1 yang ekstrim, nilai $\text{score}_{\text{BM25}}(t,d)$ hanya menjadi indikator saja dari kemunculan kata t pada dokumen d. Nilai k_1 yang umum dipilih berada pada rentang $[1.2, 2.0]$.

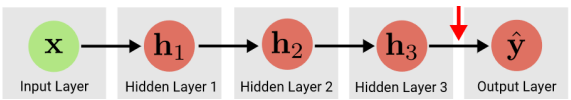


Gambar 2.9: *Creative Common License 1.0 Generic*.

;

Perbedaan *minor* lainnya ada pada fungsi idf. Fungsi idf pada BM25 merupakan versi *smoothing* dari idf dengan tujuan untuk menghindari nilai idf yang bernilai 0 ketika kata t tidak muncul pada himpunan dokumen $\mathcal{D}$ – semata-mata untuk konsistensi dengan asumsi bahwa kata t yang tidak muncul pada himpunan dokumen $\mathcal{D}$ memiliki nilai idf yang paling tinggi. Gambar 2.9 Menunjukkan Perbedaan antara idf_{BM25} dan idf. Perbedaan utamanya terjadi ketika $\text{df}(t, \mathcal{D}) = 0$, nilai dari idf_{BM25} tak nol dan mengikuti pola yang diharapkan. Ketika $\text{df}(t, \mathcal{D}) > 0$, nilai dari idf_{BM25} dan idf hampir serupa.

2.3 *Deep Learning*



Gambar 2.10: *Creative Common License 1.0 Generic*.

Arsitektur *Deep learning* merujuk pada model *machine learning* yang tersusun dari fungsi-fungsi terturunkan (yang biasa disebut sebagai *layer*), dimana komposisi antara

fungsi-fungsi tersebut dapat digambarkan sebagai *directed acyclic graph* (DAG) yang memetakan suatu *input* ke suatu *output*. Biasanya, setiap fungsi dalam Arsitektur *Deep learning* memiliki parameter yang ingin diestimasi atau dicari dengan data.

Gambar 2.10 menunjukkan arsitektur deep learning yang sederhana, yaitu *feed-forward neural network* (FFN). Pada Gambar 2.10, *input* $\mathbf{x}$ akan dipetakan ke *output* $\hat{y}$ melalui serangkaian fungsi f_1, f_2, f_3 yang disebut sebagai *layer*. Setiap *layer* f_i memiliki parameter θ_i yang akan diestimasi dengan data. Selain itu, *Output* dari *layer* f_i akan menjadi *input* dari *layer* f_{i+1}. *Output* dari *layer* f_3 adalah *output* dari model. Model pada Gambar 2.10 dapat ditulis sebagai Persamaan 2.27.

$$\hat{y} = f_{\text{model}}(\mathbf{x};\boldsymbol{\theta}) = f_3(f_2(f_1(\mathbf{x};\boldsymbol{\theta}_1);\boldsymbol{\theta}_2);\boldsymbol{\theta}_3) \tag{2.27}$$

$$\text{dengan } \boldsymbol{\theta} = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_3\} \tag{2.28}$$

2.3.1 *Multilayer Perceptron* (MLP)

Multi-layer perceptron (MLP) adalah *feed-forward neural network* dengan setiap fungsi f_i adalah fungsi linear yang diikuti oleh fungsi aktivasi non-linear ϕ yang diterapkan *element-wise* pada setiap *output*-nya. *hyperparameter* lainnya selain fungsi aktivasi adalah kedalamaan model L, dan dimensi *output* dari setiap *layer* $d_1, d_2, \ldots, d_L$.

Untuk permasalahan regresi dengan *input* $\mathbf{x} \in \mathbb{R}^{d_0}$ dan *output* $\mathbf{y} \in \mathbb{R}^{d_L}$, Persamaan 2.29 hingga Persamaan 2.35 menunjukkan arsitektur MLP untuk permasalahan regresi dengan L *layer* dan fungsi aktivasi ϕ.

$$f_l(\mathbf{x};\mathbf{W}_l,b_l) = \phi(\mathbf{x}\mathbf{W}_l + \mathbf{b}_l) \in \mathbb{R}^{d_l}, \quad l = 1,2,\ldots,L-1 \tag{2.29}$$

$$f_L(\mathbf{x}) = \mathbf{x}\mathbf{W}_L + \mathbf{b}_L \in \mathbb{R}^{d_L} \tag{2.30}$$

$$f_{\text{model}}(\mathbf{x};\boldsymbol{\theta}) = f_L(f_{L-1}(\ldots f_1(\mathbf{x}))\ldots) \tag{2.31}$$

$$\phi(\mathbf{x}) = \text{fungsi aktivitasi non-linear} \tag{2.32}$$

$$\boldsymbol{\theta} = \{\mathbf{W}_1, \mathbf{b}_1, \mathbf{W}_2, \mathbf{b}_2, \ldots, \mathbf{W}_L, \mathbf{b}_L\} \tag{2.33}$$

$$\mathbf{W}_l = \text{matriks bobot} \in \mathbb{R}^{d_{l-1} \times d_l} \tag{2.34}$$

$$\mathbf{b}_l = \text{vektor bias} \in \mathbb{R}^{d_l} \tag{2.35}$$

Untuk Permasalahan Klasifikasi Biner dengan *input* $\mathbf{x} \in \mathbb{R}^{d_0}$ dan *output* $\mathbf{y} \in \{0,1\}$, Persamaan 2.36 hingga Persamaan 2.40 menunjukkan arsitektur MLP untuk permasa-

lahan klasifikasi biner.

$$f_{\text{model}}(\mathbf{x};\boldsymbol{\theta}) = f_L(f_{L-1}(\dots f_1(\mathbf{x}))\dots), \tag{2.36}$$

$$f_L(\mathbf{x}) = \sigma(\mathbf{x}\mathbf{W}_L + \mathbf{b}_L \in \mathbb{R}), \tag{2.37}$$

$$\sigma(x) = \frac{1}{1+e^x} \in (0,1), \tag{2.38}$$

$$\text{decision}(\mathbf{x};\boldsymbol{\theta}) = \begin{cases} 1 & \text{jika } f(\mathbf{x};\boldsymbol{\theta}) \geq \text{threshold} \\ 0 & \text{jika } f(\mathbf{x};\boldsymbol{\theta}) < \text{threshold} \end{cases}, \tag{2.39}$$

$$\text{threshold} \in [0,1]. \tag{2.40}$$

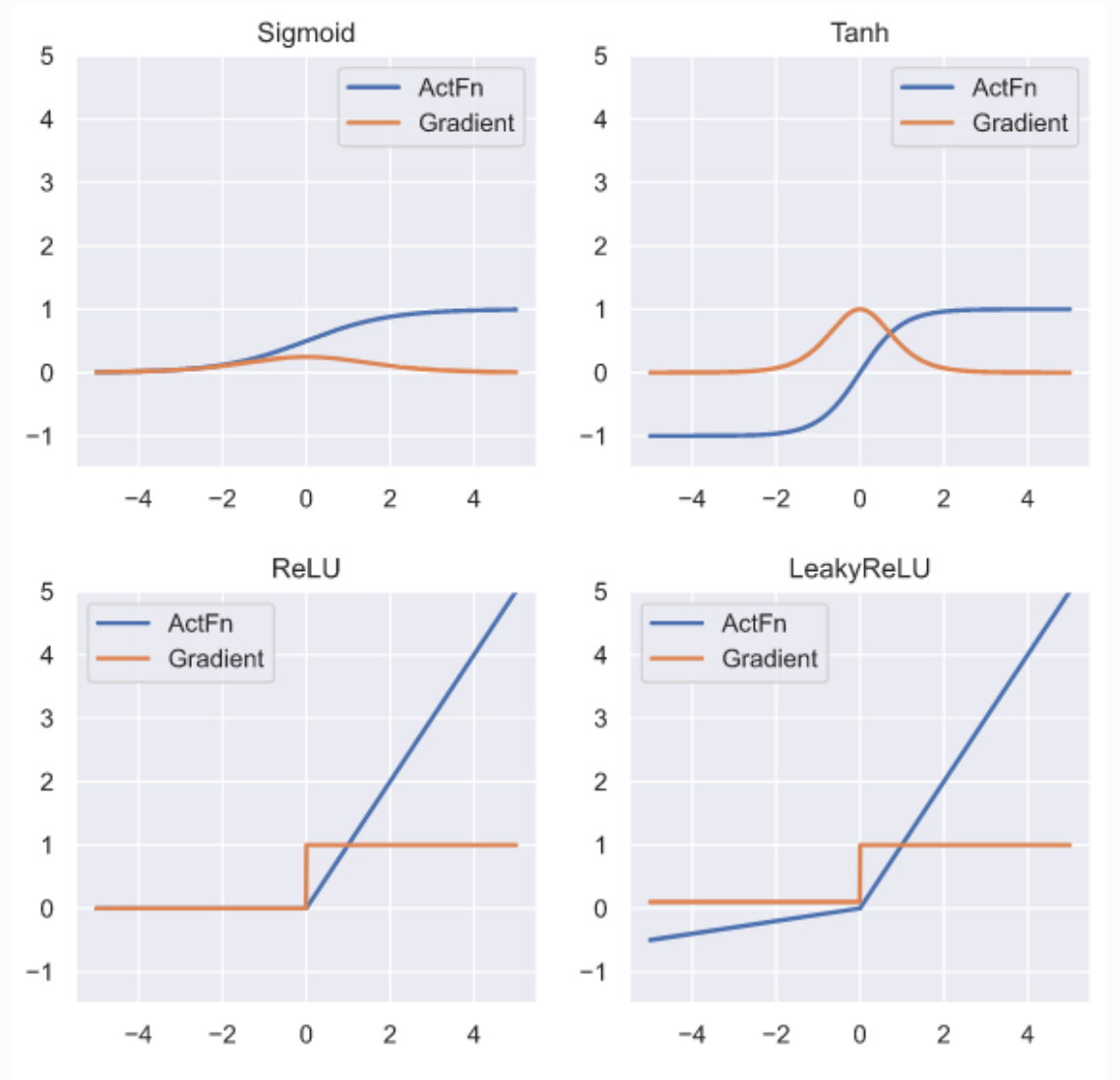
Perbedaan utama antara MLP untuk permasalahan regresi dan klasifikasi adalah fungsi aktivasi pada *output layer*. Pada permasalahan regresi, fungsi aktivasi pada *output layer* adalah fungsi identitas, sedangkan pada permasalahan klasifikasi, fungsi aktivasi pada *output layer* adalah fungsi *sigmoid*. Tujuan pengunaan fungsi *sigmoid* pada permasalahan klasifikasi adalah untuk memastikan bahwa *output* dari model berada pada rentang $[0,1]$, nilai tersebut dapat diinterpretasikan sebagai probabilitas $\mathbf{x}$ termasuk pada kelas positif. Selain itu, *threshold* pada Persamaan 2.40 digunakan untuk menentukan kelas dari $\mathbf{x}$.

2.3.2 Fungsi Aktivasi

Fungsi Aktivasi pada setiap fungsi f_i pada *multilayer perceptron* digunakan untuk menambahkan non-linearitas pada model. Sebab, tanpa adanya fungsi aktivasi non-linear, model *multilayer perceptron* akan menjadi model linear. Selain itu, fungsi aktivasi juga biasanya adalah fungsi yang terturunkan, meskipun tidak perlu terturunkan disetiap titik. Tabel 2.4 menunjukkan beberapa fungsi aktivasi yang sering digunakan pada *multilayer perceptron*. Gambar 2.11 menunjukkan grafik dari fungsi aktivasi pada Tabel 2.4 dan turunannya.

Tabel 2.4: Beberapa fungsi aktivasi yang sering digunakan pada *multilayer perceptron*.

Fungsi Aktivasi	**Persamaan**
Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$
Tanh	$\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
ReLU	$\text{ReLU}(x) = \max(0,x)$
Leaky ReLU	$\text{LeakyReLU}(x) = \max(\alpha x, x), \alpha \in [0,1]$



Gambar 2.11: *Creative Common License 1.0 Generic*.

2.3.3 Fungsi *Loss*

Misalkan $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \ldots, (\mathbf{x}_n, \mathbf{y}_n)\}$ adalah *dataset* yang terdiri dari n pasangan *input* dan *output*. Parameter $\boldsymbol{\theta}$ pada f_{model} diestimasi dengan melakukan *fitting* pada *dataset* $\mathcal{D}$. Untuk melakukan *fitting* pada *dataset* $\mathcal{D}$, diperlukan suatu fungsi *loss* yang mengukur seberapa baik hasil pemetaan f_{model} pada *input* $\mathbf{x}_i$ terhadap *output* $\mathbf{y}_i$. Meskipun sembarang fungsi yang terturunkan dapat digunakan sebagai fungsi *loss*, namun pemilihan fungsi *loss* berdasarkan *maximum likelihood estimation* (MLE) lebih disarankan.

Untuk permasalahan klasifikasi biner, fungsi *loss* yang sering digunakan adalah *binary cross entropy* (BCE) seperti yang ditinjukkan pada Persamaan 2.49. Penurunan fungsi *loss* BCE dengan mengikuti prinsip MLE yang akan dijelaskan pada bagian berikut.

Misalkan $y_i \mid \mathbf{x}$ mengikuti distribusi bernouli dengan parameter $\text{p} = f_{\text{model}}(\mathbf{x}; \boldsymbol{\theta})$ yang saling independen antara satu sama lainnya. Persamaan 2.41 menunjukkan definisi dari $y_i \mid \mathbf{x}$.

$$y_i \mid \mathbf{x} \overset{\text{iid}}{\sim} \text{Bernoulli}(f_{\text{model}}(\mathbf{x}; \boldsymbol{\theta})) \tag{2.41}$$

$$p(y_i \mid \mathbf{x}) = f_{\text{model}}(\mathbf{x}; \boldsymbol{\theta})^{y_i} (1 - f_{\text{model}}(\mathbf{x}; \boldsymbol{\theta}))^{1-y_i} \tag{2.42}$$

Fungsi *likelihood* dari $\boldsymbol{\theta}$ terhadap *dataset* $\mathcal{D}$ dapat ditulis sebagai berikut:

$$\mathcal{L}(\boldsymbol{\theta}) = \prod_{i=1}^{N} p(y_i \mid \mathbf{x}_i; \boldsymbol{\theta}). \tag{2.43}$$

Dengan prinsip MLE, parameter $\boldsymbol{\theta}$ yang dicari adalah parameter $\boldsymbol{\theta}$ yang memaksimalkan fungsi *likelihood* $\mathcal{L}(\boldsymbol{\theta})$,

$$\boldsymbol{\theta}_{\text{MLE}} = \arg\max_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}). \tag{2.44}$$

Untuk mempermudah perhitungan, fungsi *likelihood* diubah menjadi negatif *log-likelihood* $\ell(\boldsymbol{\theta})$, sehingga permasalahan optimasi dapat ditulis seperti Persamaan 2.45

hingga Persamaan 2.47.

$$\ell(\boldsymbol{\theta}) = -\log \mathcal{L}(\boldsymbol{\theta}), \tag{2.45}$$

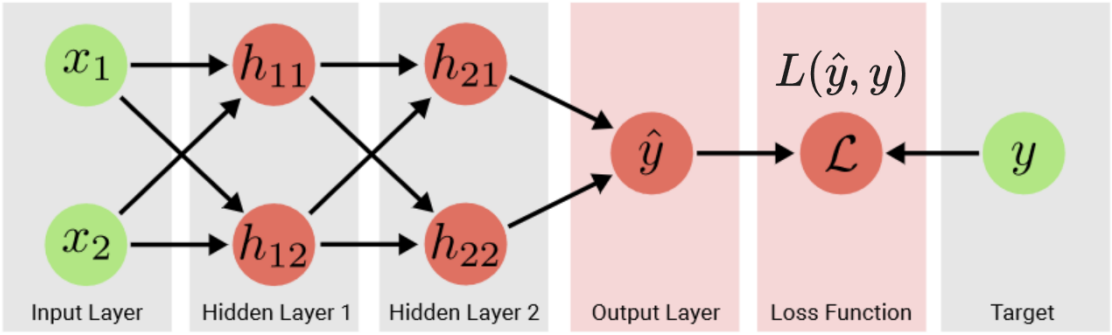
$$= -\sum_{i=1}^{N} \log\left(p(y_i \mid \mathbf{x}_i; \boldsymbol{\theta})\right), \tag{2.46}$$

$$\boldsymbol{\theta}_{\text{MLE}} = \arg\min_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}). \tag{2.47}$$

Dengan mengganti $p(y_i \mid \mathbf{x}_i; \boldsymbol{\theta})$ dengan fungsi distribusi-nya, maka fungsi *loss* yang digunakan untuk permasalahan klasifikasi biner adalah *binary cross entropy* (BCE) seperti pada Persamaan 2.49. Gambar 2.12 mengilustrasikan *directed acyclic graph* (DAG) dari model ketika proses pelatihan dilakukan.

$$\boldsymbol{\theta}_{\text{MLE}} = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} \underbrace{-y_i \log\left(f_{\text{model}}(\mathbf{x}_i; \boldsymbol{\theta})\right) - (1 - y_i) \log\left(1 - f_{\text{model}}(\mathbf{x}_i; \boldsymbol{\theta})\right)}_{\text{Binary Cross Entropy Loss } L(y_i, f_{\text{model}}(\mathbf{x}_i; \boldsymbol{\theta}))}, \tag{2.48}$$

$$L(y_i, f_{\text{model}}(\mathbf{x}_i; \boldsymbol{\theta})) = -y_i \log\left(f_{\text{model}}(\mathbf{x}_i; \boldsymbol{\theta})\right) - (1 - y_i) \log\left(1 - f_{\text{model}}(\mathbf{x}_i; \boldsymbol{\theta})\right). \tag{2.49}$$



Gambar 2.12: *Creative Common License 1.0 Generic*.

Untuk mendapatkan f_{model} dengan performa yang baik, dibutuhkan model dengan nilai $\ell(\boldsymbol{\theta})$ seminimum mungkin. Namun, pencarian $\boldsymbol{\theta}$ sehingga $\ell(\boldsymbol{\theta})$ minumum secara analitik tidak dapat dilakukan karena non-linearitas yang ada pada model, dengan kata lain solusi dari $\nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}) = 0$ tidak dapat dicari secara analitik. Sebagai gantinya, pencarian $\boldsymbol{\theta}$ dilakukan secara numerik dengan menggunakan metode *gradient descent* yang akan dijelaskan pada bagian selanjutnya.

2.3.4 *Backpropagation*

2.3.5 *Optimasi* Parameter

Gradient Descent adalah metode numerik yang digunakan untuk mencari nilai $\boldsymbol{\theta}$ yang meminimalkan fungsi *loss* $\ell(\boldsymbol{\theta})$. Pada metode *gradient descent*, nilai $\boldsymbol{\theta}$ diupdate secara iteratif dengan mengikuti arah negatif dari *gradient* $\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta})$ yang menunjukkan arah dari penurunan fungsi *loss* $\mathcal{L}(\boldsymbol{\theta})$. Persamaan 2.51 menunjukkan algoritma *gradient descent*.

$$\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\} \tag{2.50}$$

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \frac{1}{n} \sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} L(y_i, f_{\text{model}}(\mathbf{x}_i; \boldsymbol{\theta}^{(t)})), \tag{2.51}$$

$$\text{dengan } \eta \in \mathbb{R}^+ \text{ adalah } \textit{learning rate}. \tag{2.52}$$

Perlu diketahui bahwa pada metode *gradient descent* memperbarui parameter dengan mengambil rata-rata *gradient* dari semua data pada *dataset* pelatihan $\mathcal{D}$. Hal ini menciptakan masalah ketika model menggunakan banyak parameter dan jumlah data pada *datasets* latih besar, yaitu komputasi *forward pass* dan *backward pass* menjadi sangat mahal dan diperlukan memori yang besar untuk menyimpan gradient dari semua data pada *dataset* latih. Untuk mengatasi masalah tersebut, digunakan metode *stochastic gradient descent* (SGD) dimana setiap *update* dari parameter $\boldsymbol{\theta}$ dihitung dengan mengambil rata-rata *gradient* dari sebagian data pada *dataset* $\mathcal{B} \subseteq \mathcal{D}$. Persamaan 2.55 menunjukkan algoritma *stochastic gradient descent*.

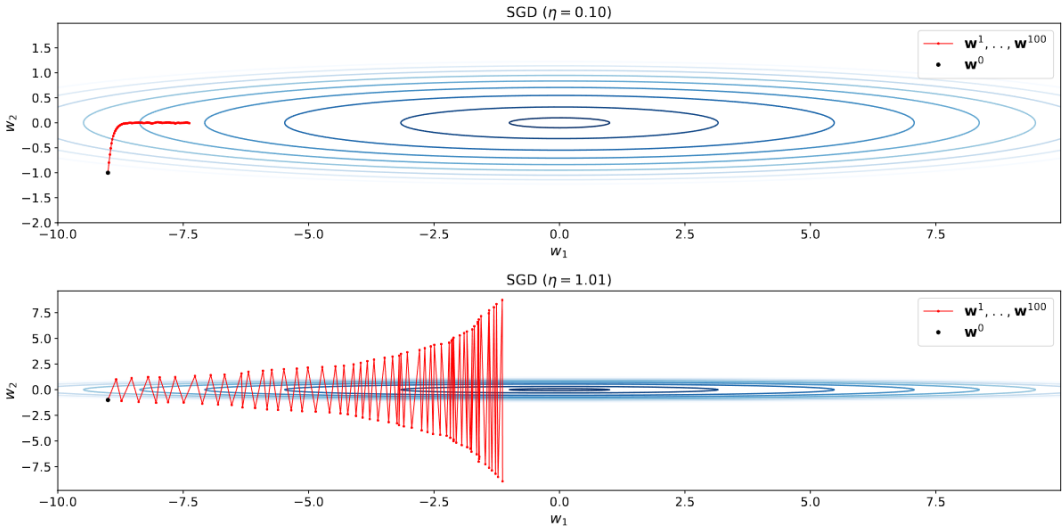
$$\mathcal{B} = \{(\mathbf{x}_{i_1}, y_{i_1}), (\mathbf{x}_{i_2}, y_{i_2}), \ldots, (\mathbf{x}_{i_b}, y_{i_b})\} \subseteq \mathcal{D}, |\ \mathcal{B}\ | \ll |\ \mathcal{D}\ |, \tag{2.53}$$

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}) = \frac{1}{b} \sum_{i=1}^{b} \nabla_{\boldsymbol{\theta}} L(y_i, f_{\text{model}}(\mathbf{x}_i; \boldsymbol{\theta})), \tag{2.54}$$

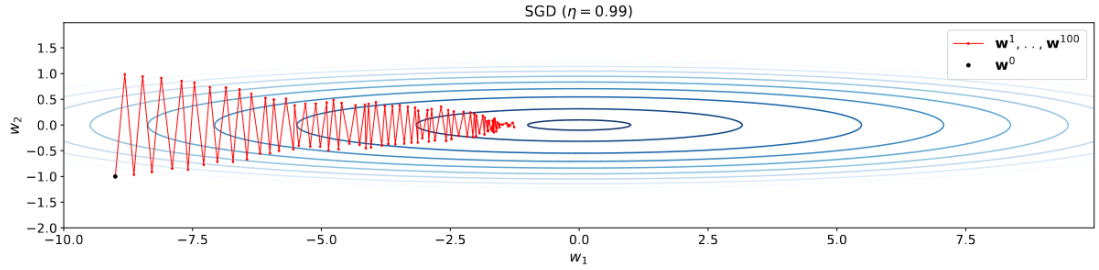
$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}^{(t)}). \tag{2.55}$$

hyperparameter learning rate mengatur laju dari perubahan parameter $\boldsymbol{\theta}$ pada setiap iterasi pembaruan. Dengan demikian, pemilihan *learning rate* berpengaruh terhadap kekonvergenan optimasi yang dilakukan. Jika *learning rate* yang digunakan terlalu kecil, model membutuhkan waktu yang jauh lebih lama untuk mencapai nilai parameter $\boldsymbol{\theta}$ yang optimal. Di lain sisi, pemilihan *learning rate* yang terlalu besar dapat membuat model tidak dapat menemukan nilai parameter $\boldsymbol{\theta}$ yang optimal. Gambar 2.13 mengilustrasikan

proses pembaruan parameter $\boldsymbol{\theta}$ dengan *learning rate* yang terlalu kecil dan terlalu besar, dan Gambar 2.14 mengilustrasikan proses pembaruan parameter $\boldsymbol{\theta}$ dengan *learning rate* yang baik.

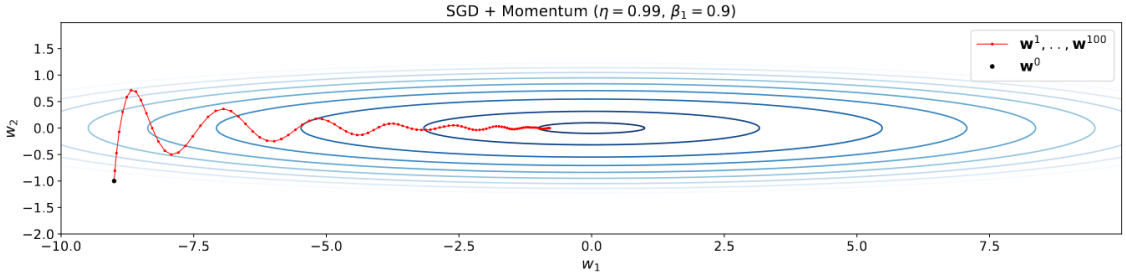


Gambar 2.13: *Creative Common License 1.0 Generic*.



Gambar 2.14: *Creative Common License 1.0 Generic*.

Untuk mempercepat proses pembaruan parameter $\boldsymbol{\theta}$, digunakan metode *stochastic gradient descent* dengan momentum untuk mengurangi osilasi pada proses pembaruan parameter. daripada memperbarui parameter $\boldsymbol{\theta}$ dengan gradien pada iterasi sekarang saja, metode *stochastic gradient descent* dengan momentum memperbarui parameter $\boldsymbol{\theta}$ dengan gradien pada iterasi sekarang dan gradien pada iterasi sebelumnya. gradien yang digunakan untuk melakukan pembaruan parameter $\boldsymbol{\theta}$ *exponential moving average* dari gradien pada iterasi sekarang dan gradien pada iterasi sebelumnya. Persamaan 2.56 menunjukkan algoritma *stochastic gradient descent* dengan momentum dan Gambar 2.15 mengilustrasikan pembaruan parameter $\boldsymbol{\theta}$ dengan momentum.

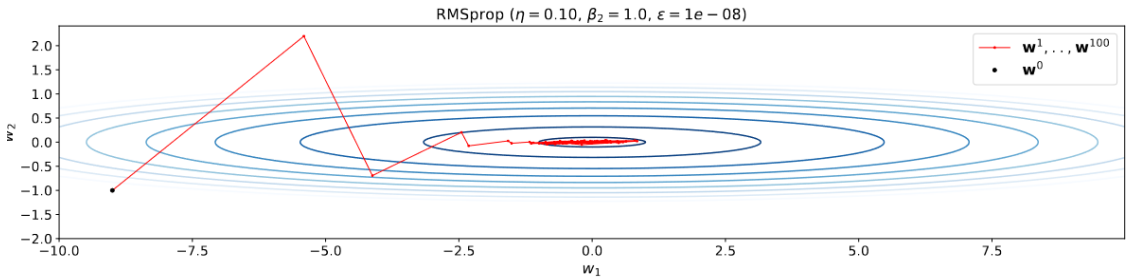


Gambar 2.15: *Creative Common License 1.0 Generic.*

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta m^{(t+1)}, \tag{2.56}$$

$$\boldsymbol{m}^{(t+1)} = \beta_1 \boldsymbol{m}^{(t)} + (1-\beta_1)\nabla_{\boldsymbol{\theta}} L_{\mathcal{B}}(\boldsymbol{\theta}^{(t)}). \tag{2.57}$$

Metode lainnya yang dapat digunakan untuk mempercepat proses pembaruan parameter $\boldsymbol{\theta}$ adalah metode *adaptive learning rate*. Metode *adaptive learning rate* mengubah *learning rate* pada setiap parameter $\boldsymbol{\theta}$ dengan membagi *learning rate* awal dengan *moving average* dari kuadrat gradien – biasanya disebut sebagai *running variance* – pada parameter $\boldsymbol{\theta}$ tersebut. Pembagian antara gradien dan *running variance* tersebut dilakukan secara *element-wise*. Persamaan 2.58 menunjukkan algoritma *stochastic gradient descent* dengan *adaptive learning rate* dan Gambar 2.16 menngilustrasikan pembaruan parameter $\boldsymbol{\theta}$ dengan *adaptive learning rate*.



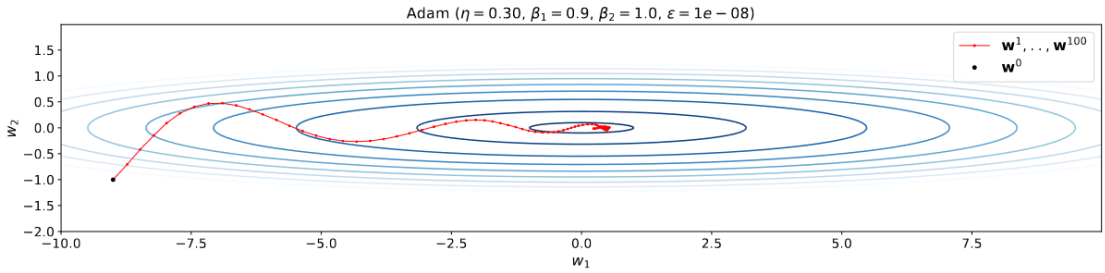
Gambar 2.16: *Creative Common License 1.0 Generic.*

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \frac{\eta \nabla_{\boldsymbol{\theta}} L_{\mathcal{B}}(\boldsymbol{\theta}^{(t)}).}{\sqrt{v^{(t+1)}} + \varepsilon}, \tag{2.58}$$

$$\boldsymbol{v}^{(t+1)} = \beta_2 \boldsymbol{v}^{(t)} + (1-\beta_2)\nabla_{\boldsymbol{\theta}} L_{\mathcal{B}}(\boldsymbol{\theta}^{(t)}) \odot \nabla_{\boldsymbol{\theta}} L_{\mathcal{B}}(\boldsymbol{\theta}^{(t)}). \tag{2.59}$$

faktor ε yang ditambahkan pada Persamaan 2.58 digunakan untuk menghindari pembagian dengan nol pada awal iterasi karena inisialiasi awal vektor $\boldsymbol{v}^{(0)}$ adalah nol.

Terakhir, metode optimasi *Adaptive Moment Estimation* (Adam) menggabungkan metode *stochastic gradient descent* dengan momentum dan *adaptive learning rate*. Persamaan 2.60 menunjukkan algoritma *stochastic gradient descent* dengan Adam dan Gambar 2.17 mengilustrasikan pembaruan parameter $\boldsymbol{\theta}$ dengan Adam. Persamaan 2.60 menunjukkan persamaan dari metode optimasi Adam dan Gambar 2.17 mengilustrasikan pembaruan parameter $\boldsymbol{\theta}$ dengan Adam.



Gambar 2.17: *Creative Common License 1.0 Generic.*

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \frac{\eta \hat{\mathbf{m}}^{(t+1)}}{\sqrt{\hat{\mathbf{v}}^{(t+1)}} + \varepsilon}, \tag{2.60}$$

$$\hat{\mathbf{m}}^{(t+1)} = \frac{\mathbf{m}^{(t+1)}}{1-\beta_1}, \tag{2.61}$$

$$\hat{\mathbf{v}}^{(t+1)} = \frac{\mathbf{v}^{(t+1)}}{1-\beta_2}, \tag{2.62}$$

$$\mathbf{m}^{(t+1)} = \beta_1 \mathbf{m}^{(t)} + (1-\beta_1) \nabla_{\boldsymbol{\theta}} L_{\mathcal{B}}(\boldsymbol{\theta}^{(t)}), \tag{2.63}$$

$$\mathbf{v}^{(t+1)} = \beta_2 \mathbf{v}^{(t)} + (1-\beta_2) \left(\nabla_{\boldsymbol{\theta}} L_{\mathcal{B}}(\boldsymbol{\theta}^{(t)}) \odot \nabla_{\boldsymbol{\theta}} L_{\mathcal{B}}(\boldsymbol{\theta}^{(t)}) \right). \tag{2.64}$$

Alasan dilakukan pembagian dengan $(1-\beta_1)$ dan $(1-\beta_2)$ Persamaan 2.61 dan Persamaan 2.62 adalah untuk menghilangkan bias pada *momentum* dan *running variance* pada awal iterasi.

2.3.6 Inisialisasi Bobot

2.4 Pembelajaran Representasi

2.4.1 Fungsi *Loss* pada Pembelajaran Representasi

BAB 3
BIDIRECTIONAL ENCODER REPRESENTATION FROM TRANSFORMER (BERT) UNTUK PEMERINGKATAN TEKS

3.1 Mekanisme *Attention*

Mekanisme *Attention* dapat ditinjau sebagai *Dictinoary Lookup*, yaitu untuk sebuah vektor kueri $\mathbf{q}$ dan sekumpulan pasangan terurut vektor $\mathcal{KV} = \{(\mathbf{k}_1, \mathbf{v}_2), (\mathbf{k}_2, \mathbf{v}_2), \ldots, (\mathbf{k}_n, \mathbf{v}_n)\}$, mekanisme *attention* akan mengembalikan vektor nilai $\mathbf{v}_i$ yang memiliki vektor kunci $\mathbf{k}_i$ yang cocok dengan vektor kueri $\mathbf{q}$. Persamaan 3.1 hingga Persamaan 3.6 menunjukkan bagaimana mekanisme *attention* dilakukan.

$$\mathcal{KV} = \{(\mathbf{k}_1, \mathbf{v}_2), (\mathbf{k}_2, \mathbf{v}_2), \ldots, (\mathbf{k}_n, \mathbf{v}_n)\}, \tag{3.1}$$

$$\text{tulis kembali } \mathbf{K} = \begin{bmatrix} \mathbf{k}_1 \\ \mathbf{k}_2 \\ \vdots \\ \mathbf{k}_n \end{bmatrix} \in \mathbb{R}^{n \times d_k}, \tag{3.2}$$

$$\mathbf{V} = \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \vdots \\ \mathbf{v}_n \end{bmatrix} \in \mathbb{R}^{n \times d_v}, \tag{3.3}$$

$$\text{Attention}(\mathbf{q}, \mathbf{K}, \mathbf{V}) = \boldsymbol{\alpha}\mathbf{V} \in \mathbb{R}^{d_v}, \tag{3.4}$$

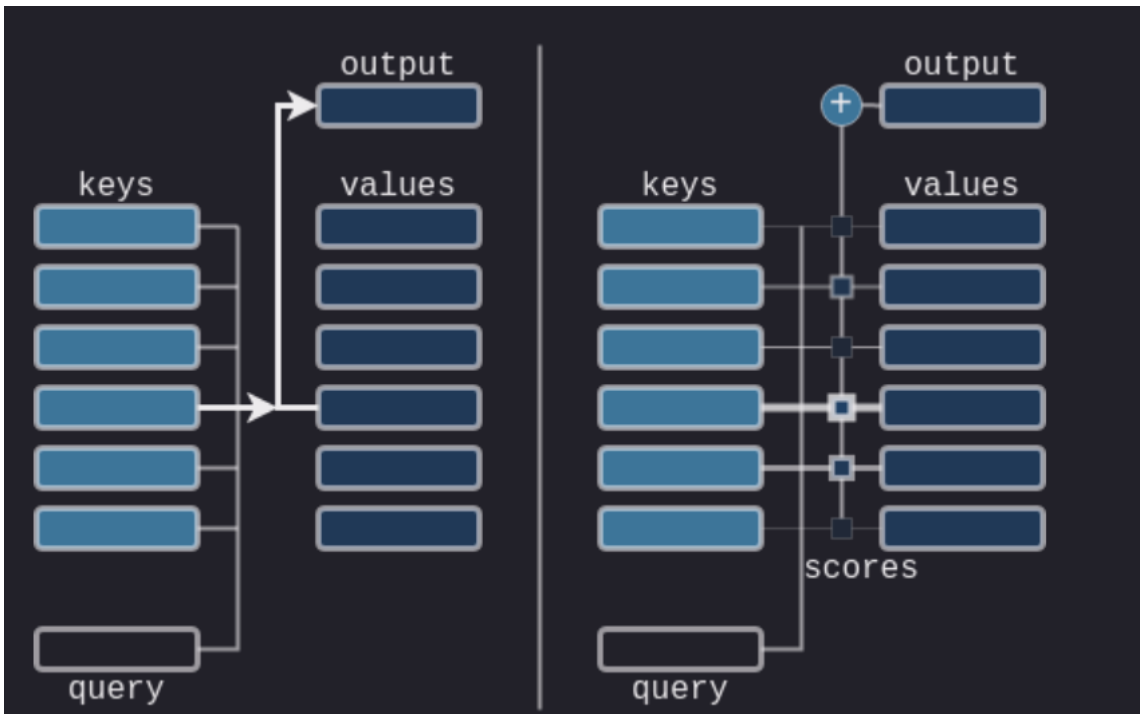
$$\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \ldots, \alpha_n], \tag{3.5}$$

$$\text{dengan } \alpha_i = \begin{cases} 1, & \text{jika } i = \arg\max_j f_{attn}(\mathbf{q}, \mathbf{k}_j) \\ 0, & \text{lainnya} \end{cases}. \tag{3.6}$$

$f_{attn}(\mathbf{q}, \mathbf{k})$ adalah fungsi atensi yang menghitung nilai atensi antara vektor kueri $\mathbf{q}$ dan vektor kunci $\mathbf{k}$. α_i pada persamaan di atas disebut sebagai bobot atensi dan nilai $f_{attn}(\mathbf{q}, \mathbf{k})$ disebut sebagai nilai atensi.

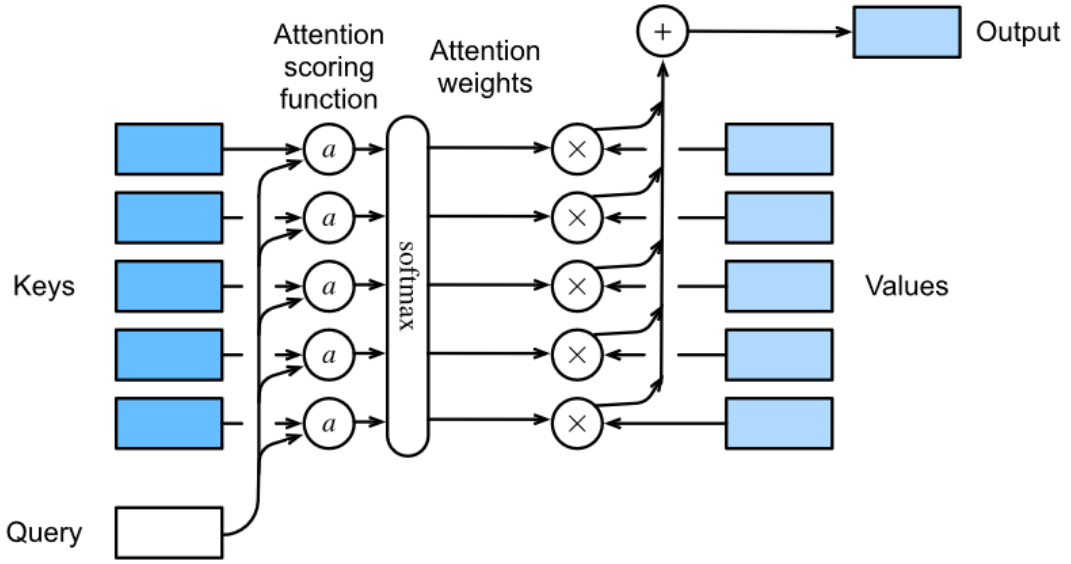
Sebagai contoh, untuk $\mathbf{q} = [1,2]$, $\mathcal{KV} = \{([2,1],[1,0]),([1,2],[0,1])\}$ serta fungsi

$f_{\text{attn}}(\mathbf{q}, \mathbf{k}) = \mathbf{q} \cdot \mathbf{k}$, nilai dari Attention$(\mathbf{q}, \mathbf{K}, \mathbf{V})$ adalah $[0, 1]$, karena nilai maksimal f_{attn} terjadi ketika $\mathbf{k} = [1, 2]$.



Gambar 3.1: Perbedaan mekanisme *hard attention* dan *soft attention* (pi tau, 2023)

Mekansime *attention* pada Persamaan 3.1 hingga Persamaan 3.6 disebut sebagai *hard attention* karena hanya satu vektor nilai $\mathbf{v}_i$ yang dipilih dari sekumpulan vektor nilai $\mathbf{V}$. Berbeda dengan *hard attention* yang tidak terturunkan, *soft attention* mengambil seluruh vektor nilai $\mathbf{V}$ dan menghitung bobot α_i untuk setiap vektor nilai $\mathbf{v}_i$ dengan fungsi *softmax*. Hasil dari *soft attention* adalah rata-rata terbobot dari seluruh vektor nilai $\mathbf{V}$. Persamaan 3.7 hingga Persamaan 3.11 menunjukkan bagaimana mekanisme *soft attention* dilakukan.



Gambar 3.2: Ilustrasi dari mekanisme *soft attention* (A. Zhang et al., 2023)

$$\text{Attention}(\mathbf{q},\mathbf{K},\mathbf{V}) = \alpha\mathbf{V} \in \mathbb{R}^{d_v}, \tag{3.7}$$

$$\text{dengan } \boldsymbol{\alpha} = [\alpha_1, \alpha_2, \ldots, \alpha_n], \tag{3.8}$$

$$\text{dan } \alpha_i(\mathbf{q},\mathbf{k}_i) = \text{Softmax}_i(\boldsymbol{\alpha}) = \frac{\exp(f_{attn}(\mathbf{q},\mathbf{k}_i))}{\sum_{j=1}^{n}\exp(f_{attn}(\mathbf{q},\mathbf{k}_j))}, \tag{3.9}$$

$$\sum_{i=1}^{n}\alpha_i = 1, \tag{3.10}$$

$$0 \leq \alpha_i \leq 1. \tag{3.11}$$

Dengan rata-rata terbobot dari $\mathbf{V}$, transformasi *soft attention* dapat dicari turunannya dengan *backpropagation* yang merupakan syarat *fundamental* yang harus dimiliki oleh sebuah model *deep learning*.

Sebagai contoh, hasil dari $\text{Attention}(\mathbf{q},\mathbf{K},\mathbf{V})$ untuk $\mathbf{q} = [1,2]$, $\mathcal{KV} = \{([2,1],[0,1]),([1,2],[1,0])\}$ serta fungsi $f_{\text{attn}}(\mathbf{q},\mathbf{k}) = \mathbf{q}\cdot\mathbf{k}$ adalah $0.268[0,1] + 0.732[1,0] = [0.732, 0.268]$ dengan $\alpha_1 = \frac{\exp(4)}{\exp(4)+\exp(5)} \approx 0.268$ dan $\alpha_2 = \frac{\exp(5)}{\exp(4)+\exp(5)} \approx 0.732$.

Pada kasus kumpulan kueri $\mathcal{Q} = \{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_m\}$, mekanisme *attention* untuk setiap triplet $(\mathbf{q}_i,\mathbf{K},\mathbf{V})$ dapat dihitung secara bersamaan seperti yang ditunjukkan pada Per-

samaan 3.12 hingga Persamaan 3.15.

$$\text{tulis } \mathbf{Q} = \begin{bmatrix} \mathbf{q}_1 \\ \mathbf{q}_2 \\ \vdots \\ \mathbf{q}_m \end{bmatrix} \in \mathbb{R}^{m \times d_k}, \tag{3.12}$$

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathbf{AV} \in \mathbb{R}^{m \times d_v}, \tag{3.13}$$

$$\mathbf{A} = \begin{bmatrix} \boldsymbol{\alpha}_1 \\ \boldsymbol{\alpha}_2 \\ \vdots \\ \boldsymbol{\alpha}_m \end{bmatrix} = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \dots & \alpha_{1n} \\ \alpha_{21} & \alpha_{22} & \dots & \alpha_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{m1} & \alpha_{m2} & \dots & \alpha_{mn} \end{bmatrix} \in \mathbb{R}^{m \times n}, \tag{3.14}$$

$$\alpha_{ij}(\mathbf{q}_i, \mathbf{k}_j) = \text{Softmax}_j(\alpha_i) = \frac{\exp(f_{attn}(\mathbf{q}_i, \mathbf{k}_j))}{\sum_{k=1}^{n} \exp(f_{attn}(\mathbf{q}_i, \mathbf{k}_k))}. \tag{3.15}$$

α_{ij} adalah bobot yang menunjukkan bobot atensi antara vektor kueri $\mathbf{q}_i$ dengan vektor kunci $\mathbf{k_j}$.

3.1.1 *Attention* Parametrik

Mekanisme *attention* yang dilakukan oleh Vaswani et al. (2017) merupakan mekanisme *attention* parametrik. Pada mekanisme *attention* parametrik, nilai f_{attn} antar vektor kueri $\mathbf{q}$ dan $\mathbf{v}$ dibandingkan pada ruang vektor yang akan dipelajari (*learned embedding space*) daripada ruang vektor aslinya. Sebagai contoh, untuk suatu kueri $\mathbf{q} \in \mathbb{R}^{d_q}$, dan vektor kunci $\mathbf{k} \in \mathbb{R}^{d_k}$, *additive attention* yang diperkenalkan oleh Bahdanau, Cho, dan Bengio (2016) menghitung nilai keserupaan antara $\mathbf{q}$ dan $\mathbf{k}$ seperti pada Persamaan 3.16

$$f_{attn}(\mathbf{q}\mathbf{W}^q, \mathbf{k}\mathbf{W}^k) = (\mathbf{q}\mathbf{W}^q + \mathbf{k}\mathbf{W}^k)\mathbf{W}^{\text{out}} \in \mathbb{R}, \tag{3.16}$$

$$\text{dengan } \mathbf{W}^q \in \mathbb{R}^{d_q \times d_{\text{attn}}}, \mathbf{W}^k \in \mathbb{R}^{d_k \times d_{\text{attn}}}, \mathbf{W}_{\text{out}} \in \mathbb{R}^{d_{\text{attn}} \times 1}. \tag{3.17}$$

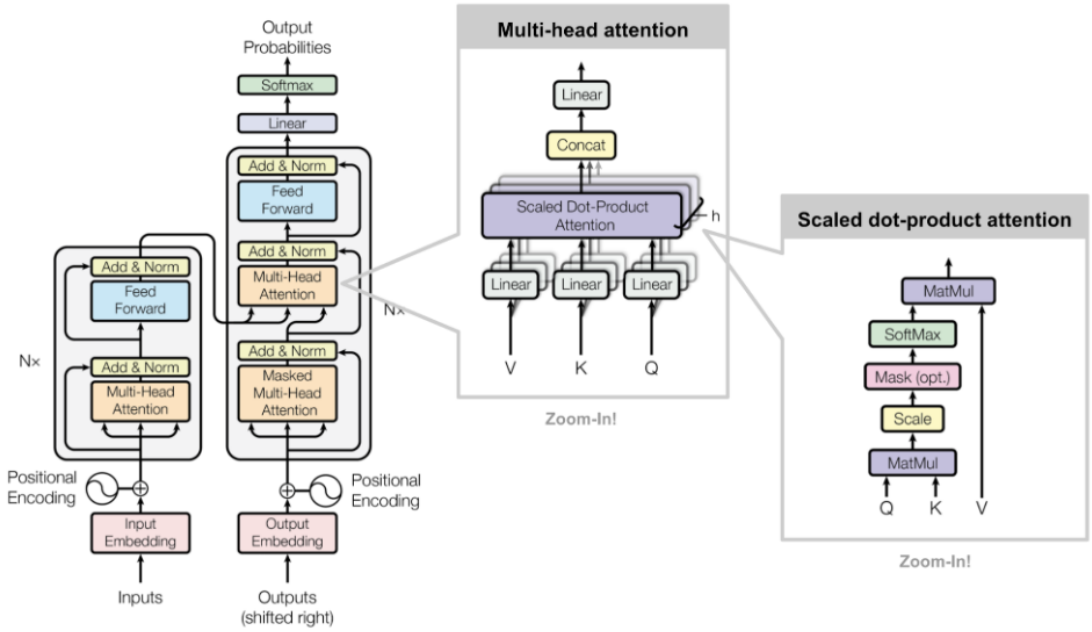
Matriks $\mathbf{W}^q$, $\mathbf{W}^k$, dan $\mathbf{W}^{\text{out}}$ adalah matriks paramter yang akan diestimasi selama proses pelatihan. Contoh *attention* parametrik yang lebih sederhana adalah *dot-product attention*. Fungsi f_{attn} yang digunakan adalah perkalian titik antara $\mathbf{q}$ dan $\mathbf{k}$ di ruang vektor yang dipelajari (*learned embedding space*). Persamaan 3.18 menunjukkan bagaimana

dot-product attention dihitung.

$$f_{attn}(\mathbf{q}\mathbf{W}^q, \mathbf{k}\mathbf{W}^k) = (\mathbf{q}\mathbf{W}^q)(\mathbf{k}\mathbf{W}^k)^\top \tag{3.18}$$

$$\text{dengan } \mathbf{W}^q \in \mathbb{R}^{d_q \times d_{\text{attn}}}, \mathbf{W}^k \in \mathbb{R}^{d_k \times d_{\text{attn}}}. \tag{3.19}$$

3.2 Transformer



Gambar 3.3: Arsitektur *transformer* (Weng, 2018).

Transformers merupakan Arsitektur *deep learning* yang pertama kali diperkenalkan oleh Vaswani et al. (2017). Awalnya Transformers merupakan model *sequance to sequance* yang diperuntukkan untuk permasalahan mesin translasi neural (*neural machine translation*). Namun, sekarang *transformer* juga digunakan untuk permasalahan pemrosesan bahasa alami lainnya. model-model yang berarsitektur *transformer* menjadi model *state-of-the-art* untuk permasalahan pemrosesan bahasa alami lainnya, seperti *question answering*, *sentiment analysis*, dan *named entity recognition*.

Berbeda dengan arsitektur mesin translasi terdahulu, transformer tidak mengunakan *recurrent neural network* (RNN) atau *convolutional neural network* (CNN), melainkan transformer adalah model *feed foward network* yang dapat memproses seluruh *input* pada barisan secara paralel. Untuk menggantikan kemampuan RNN dalam mempelajari ketergantungan antar *input* yang berurutan dan kemampuan CNN dalam mempelajari fitur

lokal, transformer bergantung pada mekanisme *attention*.

Terdapat tiga jenis *attention* yang digunakan dalam model *transformer* (Vaswani et al., 2017):

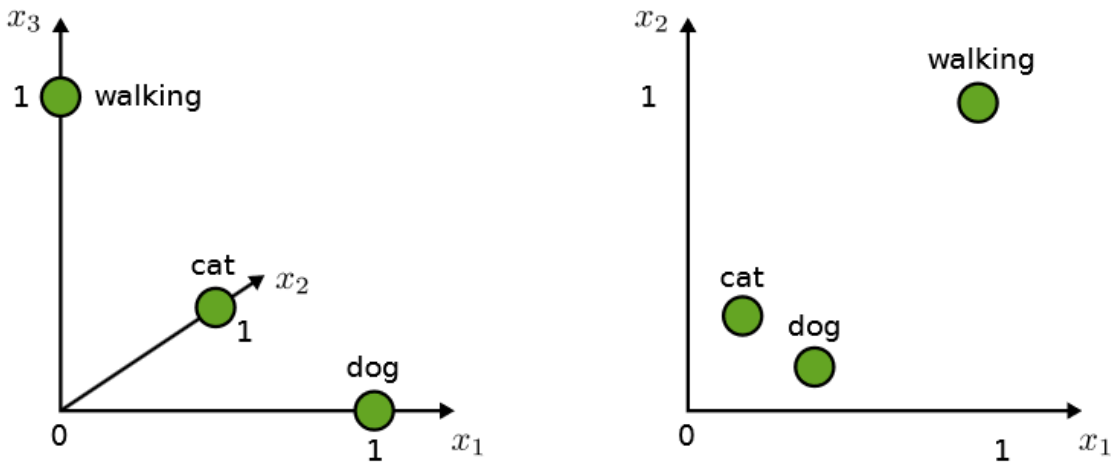
1. *Encoder self-attention*: menggunakan barisan *input* yang berupa barisan token atau kata sebagai masukan untuk menghasilkan barisan representasi kontekstual, berupa vektor, dari *input*. Setiap representasi token tersebut memiliki ketergantungan dengan token lainnya dari barisan *input*.

2. *Decoder self-attention*: menggunakan barisan *target* yang berupa kalimat terjemahan parsial, barisan token, sebagai masukan untuk menghasilkan barisan representasi kontekstual (vektor) dari *target*. Setiap representasi token tersebut memiliki ketergantungan dengan token sebelumnya dalam urutan masukan.

3. *Decoder-encoder attention*: menggunakan barisan representasi kontekstual dari *input*, dan barisan representasi kontekstual dari *target* untuk menghasilkan token berikutnya yang merupakan hasil prediksi dari model. Barisan *target* yang digabung dengan token hasil prediksi tersebut akan menjadi barisan *target* untuk prediksi selanjutnya.

Arsitektur dari *transformer* mengimplementasikan struktur pasangan encoder-decoder. Aristektur dari *transformer* dapat dilihat pada Gambar 3.3. Lapisan *encoder* berfungsi untuk memahami konteks suatu kata dalam teks atau kalimat, sementara lapisan *decoder* digunakan untuk menyelesaikan masalah translasi menuju bahasa berbeda. Pada permasalahan klasifikasi seperti analisis sentimen dan pemeringkatan teks, lapisan *decoder* tidak digunakan. Permasalahan tersebut *output* dari lapisan *encoder* yang digunakan sebagai masukan untuk lapisan *classifier*. Subbab 3.2.1 hingga Subbab 3.2.8 menjelaskan arsitektur model *transformer* dan berbagai mekanisme yang menyusun model *transformer*, khususnya *transformer encoder* yang menjadi *building block* dari *Biderectional Encoder Representations from Transformers* (BERT).

3.2.1 *Token Embedding (Input Embedding)*



Gambar 3.4: Ilustrasi dari representasi token. Gambar kiri menunjukkan representasi token dengan *one-hot encoding*, sedangkan gambar kanan menunjukkan representasi token dengan *token embedding* (Geiger et al., 2022)

Perlu diingat kembali bahwa *input* dari *Attention* (dan tentunya *transformer*) adalah barisan vektor. Jika *Attention* ingin dapat digunakan pada permasalahan bahasa, barisan kata atau subkata (selanjutnya disebut token) harus terlebih dahulu diubah menjadi barisan vektor.

Representasi vektor dari token yang paling sederhana adalah dengan *one-hot encoding*. Andaikan $\mathcal{T} = \{t_1, t_2, \ldots, t_{|\mathcal{T}|}\}$ adalah semua kemungkinan token yang mungkin muncul dalam permasalahan bahasa yang ingin diselesaikan. Untuk sembarang barisan token $t = (t_{i_1}, t_{i_2}, \ldots, t_{i_L})$, representasi vektor dari token t_{i_j} adalah vektor $\mathbf{oh}_{i_j} = [0, \ldots, 0, 1, 0, \ldots, 0] \in \mathbb{R}^{|\mathcal{T}|}$, dengan nilai 1 pada indeks ke j dan nilai 0 pada indeks lainnya. *One-hot encoding* tentunya memiliki kelemahan:

1. Vektor yang dihasilkan adalah *sparse vector*, dan ukuran vektor yang dihasilkan cukup besar, yaitu $|\mathcal{T}|$.

2. Representasi token yang buruk. Operasi vektor yang dilakukan pada *one-hot encoding* tidaklah bermakna. Misalnya, Jarak antar token akan selalu sama pada *one-hot encoding*, yaitu $\sqrt{2}$.

Untuk Mengatasi kekurangan dari representasi *one-hot encoding*, reprentasi yang digunakan adalah vektor padat yang akan dipelajari ketika proses pelatihan. Misalkan $\mathbf{E}_{\mathcal{T}} \in \mathbb{R}^{|\mathcal{T}| \times d_{\text{token}}}$ adalah matriks parameter yang merupakan representasi vektor padat dari

seluruh token ada. Persamaan 3.20 hingga Persamaan 3.22 menunjukkan bagaimana representasi vektor dari barisan suatu token t dihitung.

$$t = (t_{i_1}, t_{i_2}, \ldots, t_{i_L}), \tag{3.20}$$

$$\mathbf{e}_{i_j} = \mathbf{oh}_{i_j}\mathbf{E}_{\mathcal{T}} \in \mathbb{R}^{d_{\text{token}}}, \tag{3.21}$$

$$\text{Embed}(t) = \mathbf{E}_t = \begin{bmatrix} \mathbf{e}_{i_1} \\ \mathbf{e}_{i_2} \\ \vdots \\ \mathbf{e}_{i_L} \end{bmatrix} \in \mathbb{R}^{L \times d_{\text{token}}}. \tag{3.22}$$

Gambar 3.4 mengilustrasikan perbedaan antara *one-hot encoding* dan *token embedding*. Pada representasi token dengan vektor padat vektor yang secara semantik atau sintaksis mirip akan memiliki jarak yang lebih dekat. Selain itu, representasi token dengan vektor padat memiliki dimensi d_{token} yang lebih kecil daripada *one-hot encoding* yang memiliki dimensi $|\mathcal{T}|$.

3.2.2 *Scaled Dot-Product Attention*

Scaled dot-product attention adalah mekanisme *Attention* parametrik yang digunakan dalam *transformers*. *Scaled dot-product attention* menghitung keserupaan antara vektor kueri $\mathbf{q}$ dan vektor kunci $\mathbf{k}$ pada ruang vektor yang dipelajari (*learned embedding space*) dengan fungsi keserupaan $f_{attn}(\mathbf{q}\mathbf{W}^q, \mathbf{k}\mathbf{W}^k)$ adalah perkalian titik antara $\mathbf{q}\mathbf{W}^q$ dan $\mathbf{k}\mathbf{W}^k$ yang kemudian dibagi dengan $\sqrt{d_{attn}}$, seperti pada Persamaan 3.23.

$$f_{attn}(\mathbf{q}\mathbf{W}^q, \mathbf{k}\mathbf{W}^k) = \frac{\mathbf{q}\mathbf{W}^q(\mathbf{k}\mathbf{W}^k)^\top}{\sqrt{d_{attn}}} \in \mathbb{R}, \tag{3.23}$$

$$\text{dengan } \mathbf{W}^q \in \mathbb{R}^{d_q \times d_{\text{attn}}}, \mathbf{W}^k \in \mathbb{R}^{d_k \times d_{\text{attn}}}. \tag{3.24}$$

pembagian dengan $\sqrt{d_{attn}}$ dilakukan untuk menjaga variansi dari nilai atensi $\mathbf{q}\mathbf{W}^q(\mathbf{k}\mathbf{W}^k)^\top$ tetap serupa dengan variansi $\mathbf{q}\mathbf{W}^q$ dan $\mathbf{k}\mathbf{W}^k$. Tanpa pembagian $\sqrt{d_{attn}}$, variansi dari nilai atensi akan memiliki faktor tambahan $\sigma^2 d_{attn}$, seperti yang ditunjukkan

pada Persamaan 3.25 hingga Persamaan 3.26.

$$\mathbf{q}\mathbf{W}^q \sim \mathcal{N}(0,\sigma^2) \text{ dan } \mathbf{k}\mathbf{W}^k \sim \mathcal{N}(0,\sigma^2). \tag{3.25}$$

$$\text{Var}(\mathbf{q}\mathbf{W}^q(\mathbf{k}\mathbf{W}^k)^\top) = \sum_{i=1}^{d_{attn}} \text{Var}\left((\mathbf{q}\mathbf{W}^q)_i((\mathbf{k}\mathbf{W}^k)_i^\top\right) = \sigma^4 d_{attn}. \tag{3.26}$$

Akibatnya, untuk nilai d_{attn} yang cukup besar, akan terdapat satu elemen atensi acak $(\mathbf{q}\mathbf{W}^q(\mathbf{k}\mathbf{W}^k)^\top)_i$ sehinnga $\mid (\mathbf{q}\mathbf{W}^q(\mathbf{k}\mathbf{W}^k)^\top)_i \mid \gg \mid (\mathbf{q}\mathbf{W}^q(\mathbf{k}\mathbf{W}^k)^\top)_j \mid$ untuk sembarang nilai atensi lainnya. Jika faktor d_{attn} tidak dihilangkan, *softmax* dari nilai atensi akan jenuh ke 1 untuk satu elemen acak tersebut dan 0 untuk elemen lainnya – atau sebaliknya. Akibatnya, gradien pada fungsi *softmax* akan mendekati nol sehingga model tidak dapat belajar parameter dengan baik.

Dengan *scaled dot product attention*, tidak ada faktor d_{attn} pada variansi nilai atensi. faktor σ^4 pada Persamaan 3.27 tidak menjadi masalah karena dengan *normalization layer* yang dijelaskan pada Subbab 3.2.7 mengakibatkan $\sigma^2 \approx 1$ sehingga $\sigma^4 \approx \sigma^2 \approx 1$.

$$\text{(scaled dot product attention) Var}\left(\frac{\mathbf{q}\mathbf{W}^q(\mathbf{k}\mathbf{W}^k)^\top}{\sqrt{d_{attn}}}\right) = \frac{\sigma^4 d_{attn}}{d_{attn}} = \sigma^4 \tag{3.27}$$

Terakhir, untuk kumpulan vektor kueri $\mathcal{Q} = \{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_m\}$, dan kumpulan vektor kunci dan nilai $\mathcal{KV} = \{(\mathbf{k}_1, \mathbf{v}_2), (\mathbf{k}_2, \mathbf{v}_2), \ldots, (\mathbf{k}_n, \mathbf{v}_n)\}$, *scaled dot product attention* dapat

dihitung secara bersamaan seperti pada Persamaan 3.28 hingga Persamaan 3.31.

$$\text{Tulis Kembali } \mathbf{Q} = \begin{bmatrix} \mathbf{q}_1 \\ \mathbf{q}_2 \\ \vdots \\ \mathbf{q}_m \end{bmatrix} \in \mathbb{R}^{m \times d_q}, \tag{3.28}$$

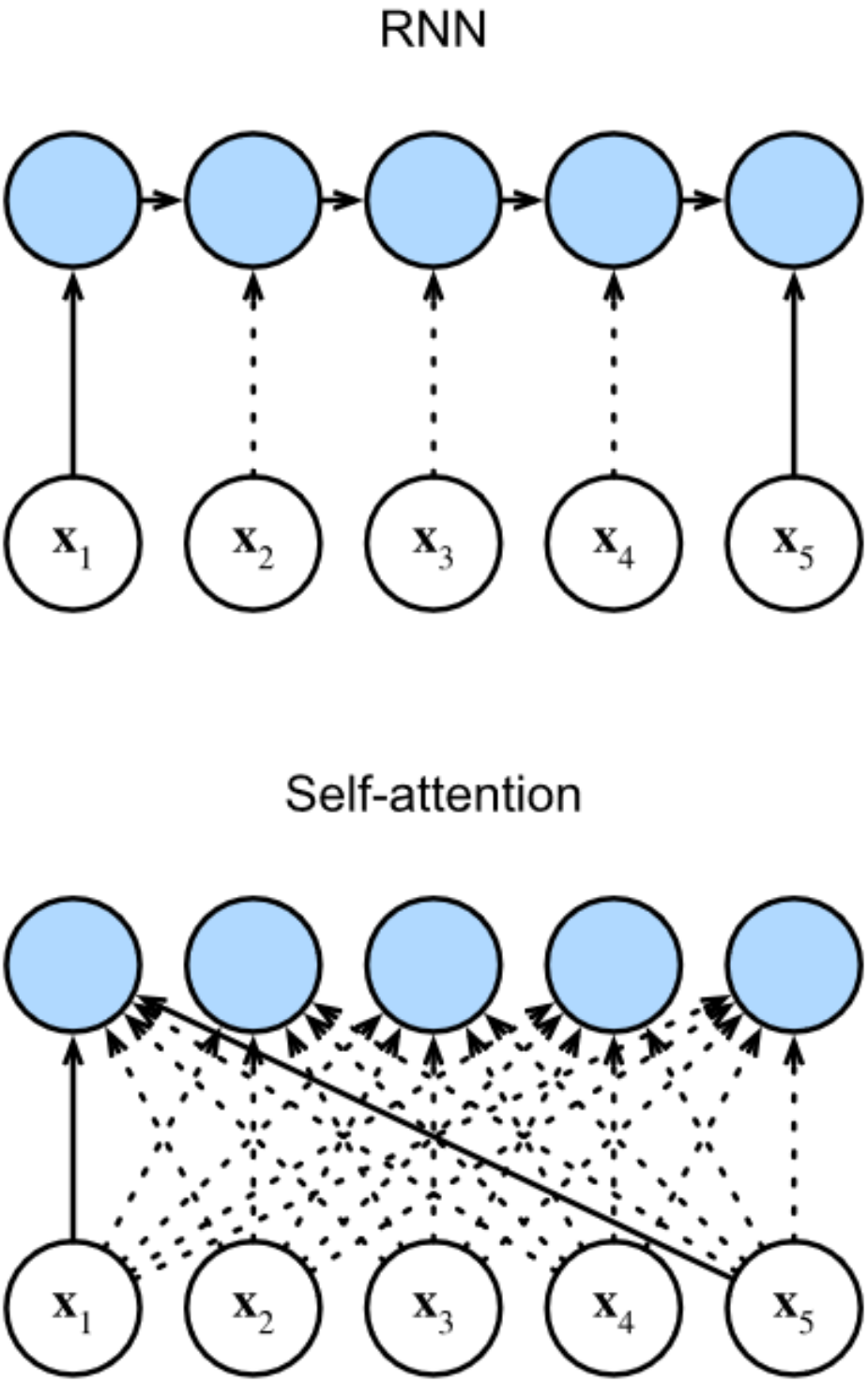
$$\mathbf{K} = \begin{bmatrix} \mathbf{k}_1 \\ \mathbf{k}_2 \\ \vdots \\ \mathbf{k}_n \end{bmatrix} \in \mathbb{R}^{n \times d_k}, \tag{3.29}$$

$$\text{dan } \mathbf{V} = \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \vdots \\ \mathbf{v}_n \end{bmatrix} \in \mathbb{R}^{n \times d_v}, \tag{3.30}$$

$$\text{Attention}(\mathbf{QW}^q, \mathbf{KW}^k, \mathbf{V}) = \text{Softmax}(\frac{\mathbf{QW}^q(\mathbf{KW}^k)^\top}{\sqrt{d_{attn}}})\mathbf{V} \in \mathbb{R}^{m \times d_v}, \tag{3.31}$$

$$\text{dengan } \mathbf{W}^q \in \mathbb{R}^{d_q \times d_{\text{attn}}}, \mathbf{W}^k \in \mathbb{R}^{d_k \times d_{\text{attn}}}. \tag{3.32}$$

3.2.3 *Self-Attention*



Gambar 3.5: Perbandingan RNN dan *self-attention* dalam menghasilkan representasi vektor kontekstual. Pada RNN, representasi vektor kontekstual setiap token bergantung pada perhitungan token sebelumnya. Pada *self-attention*, representasi vektor kontekstual setiap token dihitung secara independen dan paralel.

Self-Attention layer adalah *layer* yang digunakan *transformer* untuk menghasilkan representasi vektor yang kontekstual dari barisan token *input*. Berbeda dengan RNN dalam menghasilkan representasi vektor kontekstual, *self-attention* tidak memerlukan ketergantungan sekuensial, yang berarti representasi vektor kontekstual setiap tokennya dapat dihitung secara independen dan paralel. Gambar 3.5 mengambarkan perbedaan kedua arsitektur dalam menghasilkan representasi vektor kontekstual. Kemampuan Paralelisme dari *self-attention* membuat proses komputasi menjadi lebih cepat pada *hardware* yang mendukung paralelisme.

Perhitungan *self-attention* pada *transformer* yang digunakan adalah *scaled dot product attention* yang telah dijelaskan pada Subbab 3.2.2. Pada *self-attention*, vektor kueri **q**, vektor kunci **k**, dan vektor nilai **v** adalah vektor yang sama, yaitu *embedding* dari token **E** yang dijelaskan pada Subbab 3.2.1. Selain itu, Dimensi dari *learned embedding space* d_{attn} yang digunakan untuk perhitungan nilai atensi adalah d_{token} yaitu dimensi dari *token embedding*. Persamaan 3.33 hingga Persamaan 3.35 menunjukkan bagaimana *self-attention* dihitung.

$$\mathbf{E} = \text{Embedding dari barisan token} \in \mathbb{R}^{L \times d_{\text{token}}}, \tag{3.33}$$

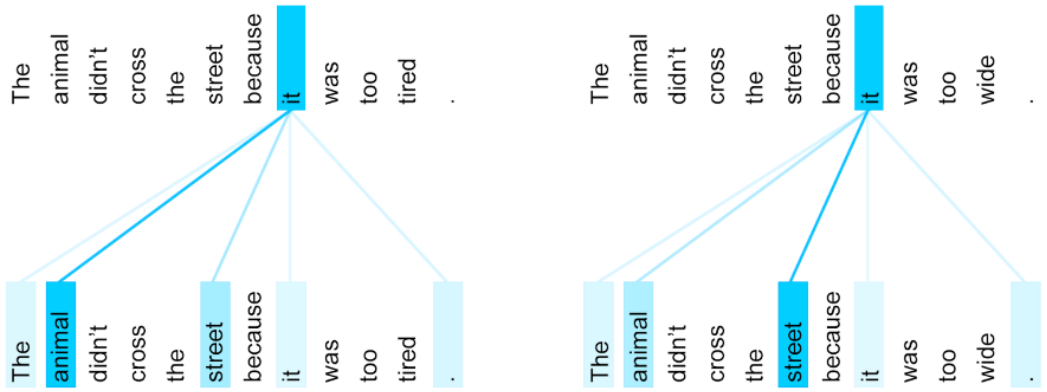
$$\text{Self-Attention}(\mathbf{E}) = \text{Attention}(\mathbf{E}\mathbf{W}^q, \mathbf{E}\mathbf{W}^k, \mathbf{E}\mathbf{W}^v), \tag{3.34}$$

$$= \text{Softmax}(\frac{\mathbf{E}\mathbf{W}^q(\mathbf{E}\mathbf{W}^k)^\top}{\sqrt{d_{attn}}})(\mathbf{E}\mathbf{W}^v) \in \mathbb{R}^{L \times d_{\text{token}}}, \tag{3.35}$$

$$\text{dengan } \mathbf{W}^q, \mathbf{W}^k, \in \mathbb{R}^{d_{\text{token}} \times d_{\text{token}}}, \mathbf{W}^v \in \mathbb{R}^{d_{\text{token}} \times d_{\text{token}}}. \tag{3.36}$$

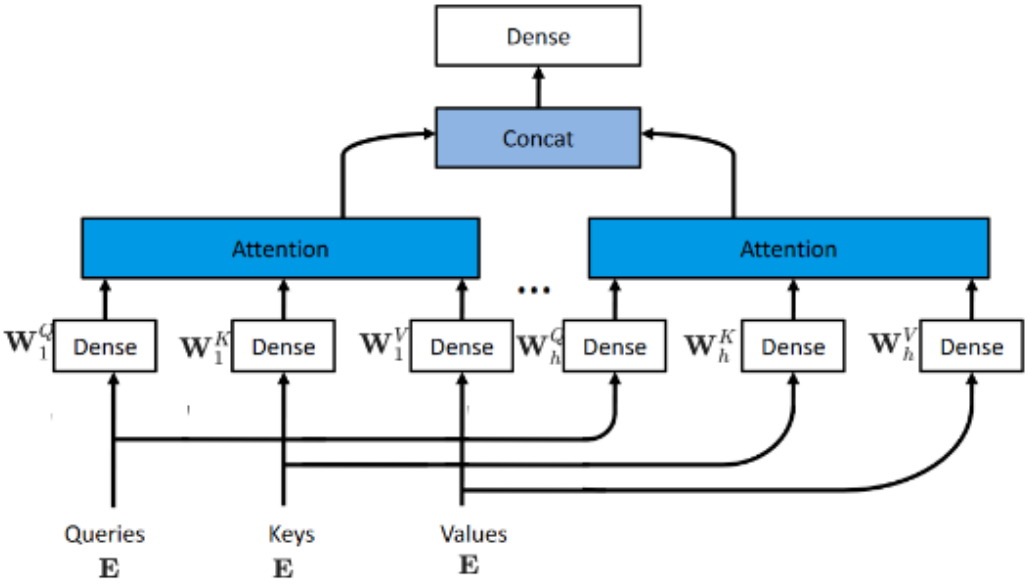
self-attention dapat dikonsepsikan sebagai proses pembentukan representasi token yang kontekstual. untuk setiap tokennya, *self-attention* menghitung keserupaan antara token $\mathbf{e}_{t_i}\mathbf{W}^q$ dengan seluruh token lainnya $\mathbf{E}\mathbf{W}^k$ dengan *scaled dot product attention*. Hasil dari *scaled dot product attention* adalah vektor yang menunjukkan bobot atensi dari token tersebut terhadap token lainnya. Bobot atensi tersebut kemudian digunakan untuk menghitung rata-rata terbobot dari seluruh token lainnya ($\mathbf{E}\mathbf{W}^v$). Hasil dari rata-rata terbobot tersebut adalah representasi vektor kontekstual dari token tersebut. Gambar 3.6 adalah contoh dari *self-attention* yang menghasilkan representasi vektor kontekstual pada token *it*. Pada Gambar 3.6 kiri token *it* memiliki bobot atensi yang tinggi terhadap token dan *animal* sehingga representasi vektor kontekstual dari token *it* akan memiliki nilai yang serupa dengan representasi token *animal*. Di lain sisi, token *it* pada Gambar 3.6

memiliki bobot atensi yang tinggi terhadap token *street*.



Gambar 3.6: Ilustrasi *self-attention* dalam menghasilkan representasi vektor kontekstual dari barisan token. Representasi vektor dari token *it* akan bergantung terhadap barisan token *input*.

3.2.4 *Multi-Head Self-Attention*



Gambar 3.7: Ilustrasi *multi-head self-attention* pada *transformer*. *Multi-head self-attention* menghitung *self-attention* sebanyak *h* kali pada subruang yang berbeda.

Multi-Head Self-Attention adalah arsiktetur pada *transformer* untuk melakukan mekanisme *self-attention* beberapa kali pada subruang (*learned embedded space*) yang berbeda. dengan melakukan hal tersebut, diharapkan bahwa model dapat menangkap relasi atau keserupaan antar token dari sudut pandang yang berbeda.

Secara teknis, *embedding* dari barisan token $\mathbf{E}$ akan dipetakan sebanyak h kali dengan *linear layer* yang kemudian hasil *attention* dari setiap *head* akan digabungkan dan dilakukan transformasi sekali lagi dengan *linear layer*. Persamaan 3.37 hingga Persamaan 3.40 menunjukkan bagaimana *multi-head self-attention* dihitung.

$$\text{MHSA}(\mathbf{E}) = \text{Concat}(\text{head}_i, \ldots, \text{head}_h)\mathbf{W}^O \in \mathbb{R}^{L \times d_{\text{token}}}, \tag{3.37}$$

$$\text{head}_i = \text{Self-Attention}_i(\mathbf{E}) = \text{Softmax}(\frac{\mathbf{E}\mathbf{W}_i^q(\mathbf{E}\mathbf{W}_i^k)^\top}{\sqrt{d_{\text{token}}/h}})\mathbf{E}\mathbf{W}_i^v \in \mathbb{R}^{L \times \frac{d_{\text{token}}}{h}}, \tag{3.38}$$

$$\text{Concat}(\text{head}_1, \ldots, \text{head}_h) = [\text{head}_1 | \ldots | \text{head}_h] \in \mathbb{R}^{L \times d_{\text{token}}}, \tag{3.39}$$

$$\text{dengan } \mathbf{W}_i^q, \mathbf{W}_i^k, \mathbf{W}_i^v, \in \mathbb{R}^{\frac{d_{\text{token}}}{h} \times \frac{d_{\text{token}}}{h}}, \mathbf{W}^O \in \mathbb{R}^{d_{\text{token}} \times d_{\text{token}}}. \tag{3.40}$$

perhatikan bahwa dimensi dari *learned embedding space* menjadi $\frac{d_{\text{token}}}{h}$ untuk setiap *head*-nya. Hal ini dilakukan untuk menjaga dimensi dari *output* terakhir tetap sama dengan dimensi dari *input*, yaitu d_{token}. Selain itu, justifikasi lainnya yang dapat dibuat adalah setiap *head* hanya perlu menggunakan dimensi yang lebih kecil dari d_{token} untuk menangkap ketergantungan antar-token (pi tau, 2023).

3.2.5 *Positional Encoding*

Mekanisme *self-attention* yang dijelaskan sebelumnya tidak memperhatikan informasi mengenai urutan token selama proses komputasinya. Representasi vektor kontekstual dari suatu token akan sama meskipun urutan tokennya berbeda. Lebih tepatnya, mekanisme *self-attention* bersifat *permutation equivariant*, yaitu untuk *token embedding* $\mathbf{E}$ dan matriks permutasi $\mathbf{P}_\pi$, Persamaan 3.41 terpenuhi.

$$\text{Self-Attention}(\mathbf{E}\mathbf{P}_\pi) = \text{Self-Attention}(\mathbf{E})\mathbf{P}_\pi \tag{3.41}$$

Namun, urutan dari token penting dalam pemrosesan bahasa alami. Kalimat `saya makan nasi` dan `nasi makan saya` memiliki makna yang berbeda. Oleh karena itu, informasi mengenai urutan token haruslah diperhatikan dalam pemrosesan bahasa alami.

Vaswani et al. (2017) menambahkan informasi posisi dengan Menjumlahkan *token embedding* $\mathbf{E}$ dengan suatu matriks *positional encoding* $\mathbf{PE}$. setiap entri dari $\mathbf{PE}$ adalah fungsi sinusoidal dari posisi token dan dimensi dari *token embedding* seperti yang ditunjukkan pada Persamaan 3.42. Gambar 3.8 menunjukkan ilustrasi dari *positional encoding*

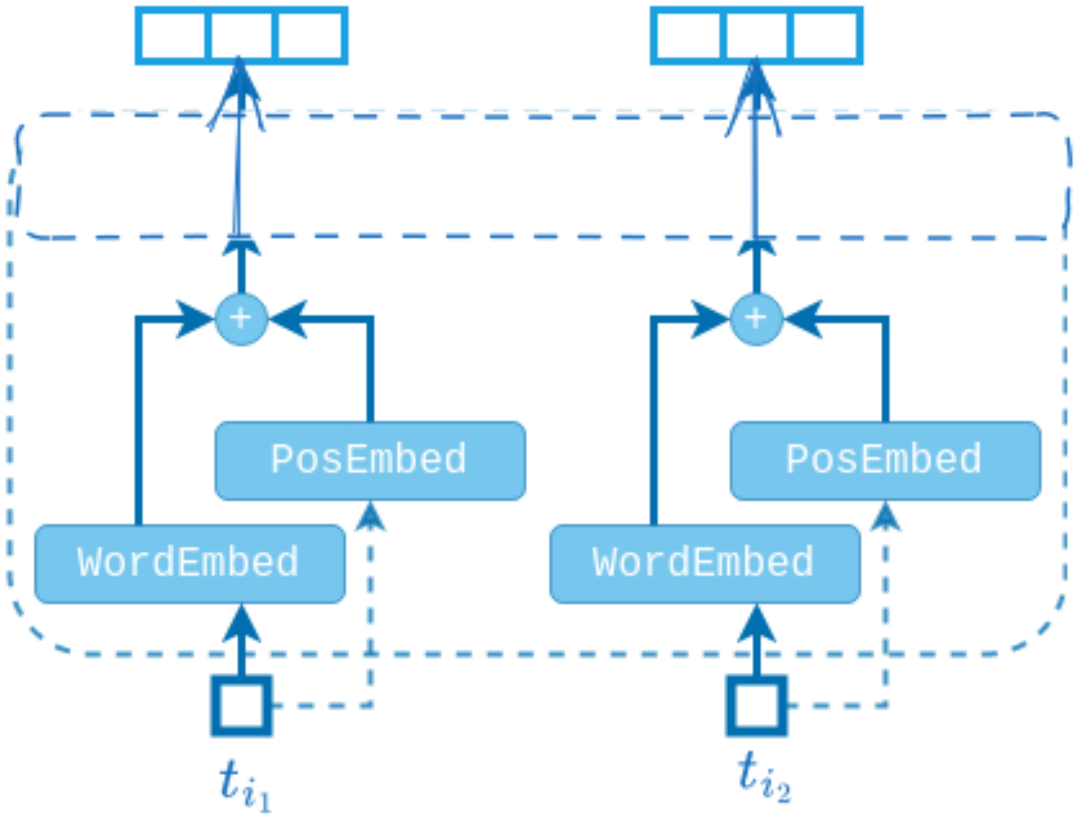
pada *transformer*.

$$\mathrm{PE}_{\mathrm{pos},i} = \begin{cases} \sin\left(\frac{pos}{10000^{i/d_{\mathrm{token}}}}\right) & \text{jika } i \bmod 2 = 0, \\ \cos\left(\frac{pos}{10000^{(i-1)/d_{\mathrm{token}}}}\right) & \text{lainnya.} \end{cases} \tag{3.42}$$

berbeda dengan Vaswani et al. (2017), Devlin, Chang, Lee, dan Toutanova (2018) menggunakan matriks parameter $\mathbf{W}^{pe} \in \mathbb{R}^{L_{\max} \times d_{\mathrm{token}}}$ untuk menghitung matriks *positional encoding*$\mathbf{PE} \in \mathbb{R}^{L \times d_{\mathrm{token}}}$ seperti yang ditunjukkan pada Persamaan 3.43 hingga Persamaan 3.44.Kekurangan dari pendekatan ini adalah model tidak dapat melakukan inferensi pada barisan token yang lebih panjang dari $L_{\max}$. Gambar 3.8 mengilustrasikan *positional encoding* pada *transformer*.

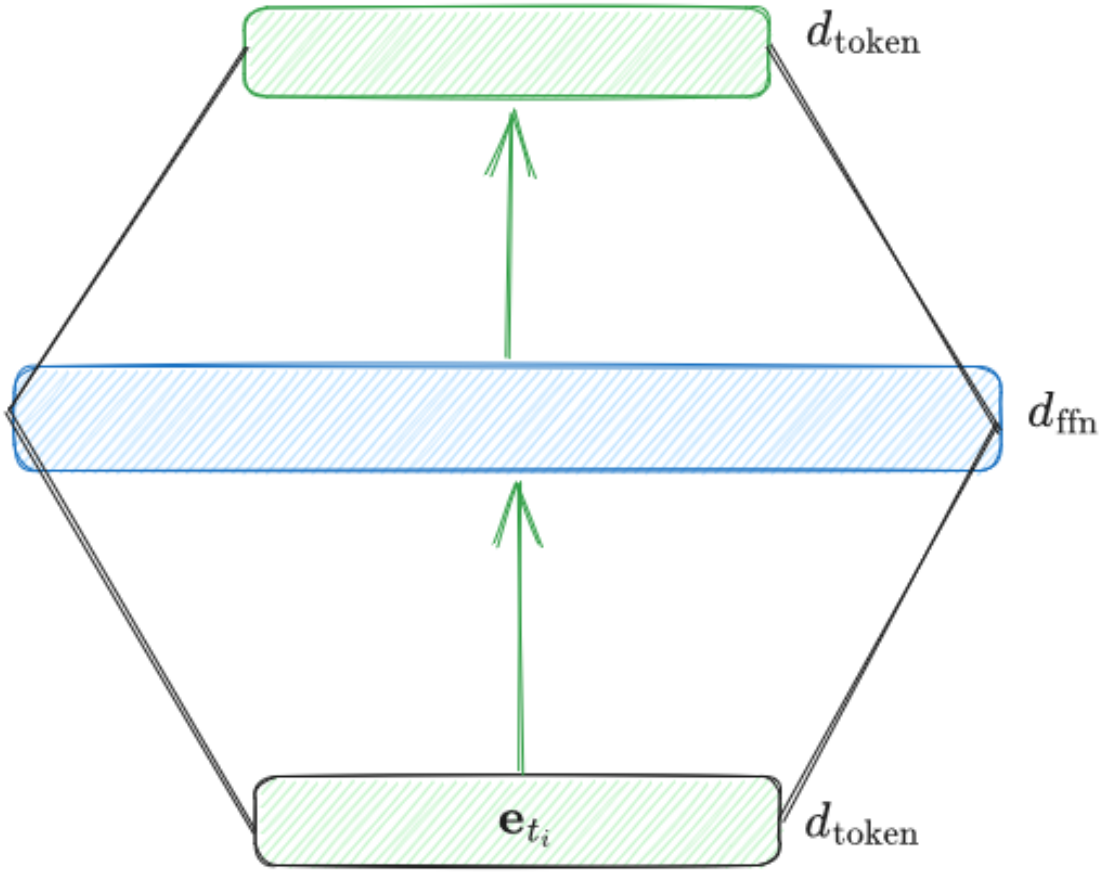
$$\mathbf{pe}_i = [0,0,\ldots,\underbrace{1}_{\text{indeks ke-}i},0,\ldots,0]\mathbf{W}^{pe} \in \mathbb{R}^{d_{\mathrm{token}}}, \tag{3.43}$$

$$\mathrm{pos}(t) = \mathbf{PE} = \begin{bmatrix} \mathbf{pe}_1 \\ \mathbf{pe}_2 \\ \vdots \\ \mathbf{pe}_L \end{bmatrix} \in \mathbb{R}^{L \times d_{\mathrm{token}}}. \tag{3.44}$$



Gambar 3.8: Ilustrasi dari *positional encoding* pada *transformer*. *Positional encoding* ditambahkan pada *token embedding* sebelum dijadikan masukan untuk *transformer*.

3.2.6 *Position-wise Feed-Forward Network*



Gambar 3.9: Ilustrasi *position-wise feed-forward network* pada *transformer*.

Position-wise Feed-Foward Network adalah *feed foward network* dengan dua kali transformasi linear dan sebuah fungsi aktivasi ReLU di antaranya. Gambar 3.9 menunjukkan ilustrasi dari *position-wise feed-forward network* dan Persamaan 3.45 hingga Persamaan 3.46 menunjukkan Transformasi yang dilakukan oleh *position-wise feed-forward network*.

$$\text{FFN}(\mathbf{X}) = \max(0, \mathbf{X}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2 \in \mathbb{R}^{L \times d_{\text{token}}} \tag{3.45}$$

$$\mathbf{W}_1 \in \mathbb{R}^{d_{\text{token}} \times d_{\text{ffn}}}, \mathbf{W}_2 \in \mathbb{R}^{d_{\text{ffn}} \times d_{\text{token}}}, \mathbf{b}_1 \in \mathbb{R}^{d_{\text{ffn}}}, \mathbf{b}_2 \in \mathbb{R}^{d_{\text{token}}}. \tag{3.46}$$

d_{ffn} adalah dimensi dari *feed forward network* yang digunakan. Vaswani et al. (2017) menggunakan $d_{\text{ffn}} = 2048$.

3.2.7 Koneksi Residu dan *Layer Normalization*

Pembaruan parameter model dilakukan pada semua *layer* secara serentak setiap iterasi *gradient descent*. Ketika parameter suatu *layer* mengalami pembaruan, distribusi dari *output* yang dihasilkan *layer* tersebut juga akan berubah pada iterasi selanjutnya. *Layer-layer* selanjutnya harus beradaptasi karena distribusi *input* dari *layer* tersebut berubah. Fenomena ini disebut *internal covariate shift* yang mengakibatkan proses pencarian parameter menjadi lebih lambat.

Layer Normalization berfungsi untuk mencegah masalah *internal covariate shift* di atas dengan membatasi distribusi nilai *output* – yang nantinya menjadi *input* pada *layer* selanjutnya – sehingga memiliki variansi 1 dan mean 0. Justifikasi lainnya di balik penggunaan *layer normalization* adalah variansi dari *input* untuk *self-attention layer* haruslah 1 (lihat Subbab 3.2.2), sehingga variansi dari bobot atensi $\text{Softmax}(\frac{\mathbf{E}\mathbf{W}^q(\mathbf{E}\mathbf{W}^k)^\top}{\sqrt{d_{\text{token}}}})$ akan 1 juga. Persamaan 3.47 hingga Persamaan 3.54 menunjukkan proses kerja dari *layer nor-*

malization.

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1,d_{\text{token}}} \\ x_{21} & x_{22} & \dots & x_{2,d_{\text{token}}} \\ \vdots & \vdots & \ddots & \vdots \\ x_{L1} & x_{L2} & \dots & x_{L,d_{\text{token}}} \end{bmatrix} \in \mathbb{R}^{L \times d_{\text{token}}} \tag{3.47}$$

$$\text{LayerNorm}(\mathbf{X}) = (\mathbf{X} - \boldsymbol{\mu}) \odot \frac{1}{\boldsymbol{\sigma}} \tag{3.48}$$

$$= \begin{bmatrix} \frac{x_{11}-\mu_1}{\sigma_1} & \frac{x_{12}-\mu_1}{\sigma_1} & \dots & \frac{x_{1,d_{\text{token}}}-\mu_1}{\sigma_1} \\ \frac{x_{21}-\mu_2}{\sigma_2} & \frac{x_{22}-\mu_2}{\sigma_2} & \dots & \frac{x_{2,d_{\text{token}}}-\mu_2}{\sigma_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{x_{L1}-\mu_L}{\sigma_L} & \frac{x_{L2}-\mu_L}{\sigma_L} & \dots & \frac{x_{L,d_{\text{token}}}-\mu_L}{\sigma_L} \end{bmatrix} \in \mathbb{R}^{L \times d_{\text{token}}} \tag{3.49}$$

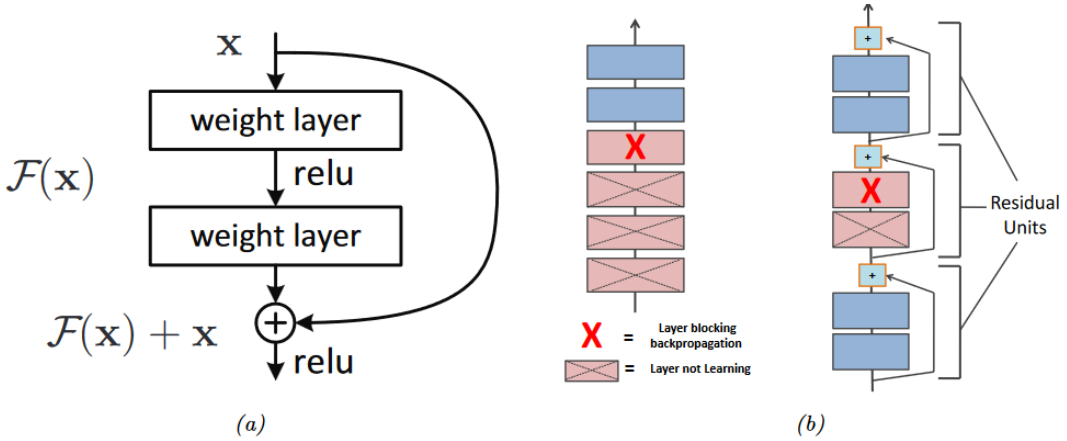
$$\boldsymbol{\mu} = \begin{bmatrix} \mu_1 & \dots & \mu_1 \\ \vdots & \ddots & \vdots \\ \mu_L & \dots & \mu_L \end{bmatrix} \in \mathbb{R}^{L \times d_{\text{token}}}, \tag{3.50}$$

$$\frac{1}{\boldsymbol{\sigma}} = \begin{bmatrix} \frac{1}{\sigma_1} & \dots & \frac{1}{\sigma_1} \\ \vdots & \ddots & \vdots \\ \frac{1}{\sigma_L} & \dots & \frac{1}{\sigma_L} \end{bmatrix} \in \mathbb{R}^{L \times d_{\text{token}}}, \tag{3.51}$$

$$\mu_i = \frac{1}{d_{\text{token}}} \sum_{j=1}^{d_{\text{token}}} x_{ij}, \quad i = 1, \dots, L, \tag{3.52}$$

$$\sigma_i = \sqrt{\frac{1}{d_{\text{token}}} \sum_{j=1}^{d_{\text{token}}} (x_{ij} - \mu_i)^2} \quad i = 1, \dots, L, \tag{3.53}$$

$$\odot = \text{element-wise product.} \tag{3.54}$$



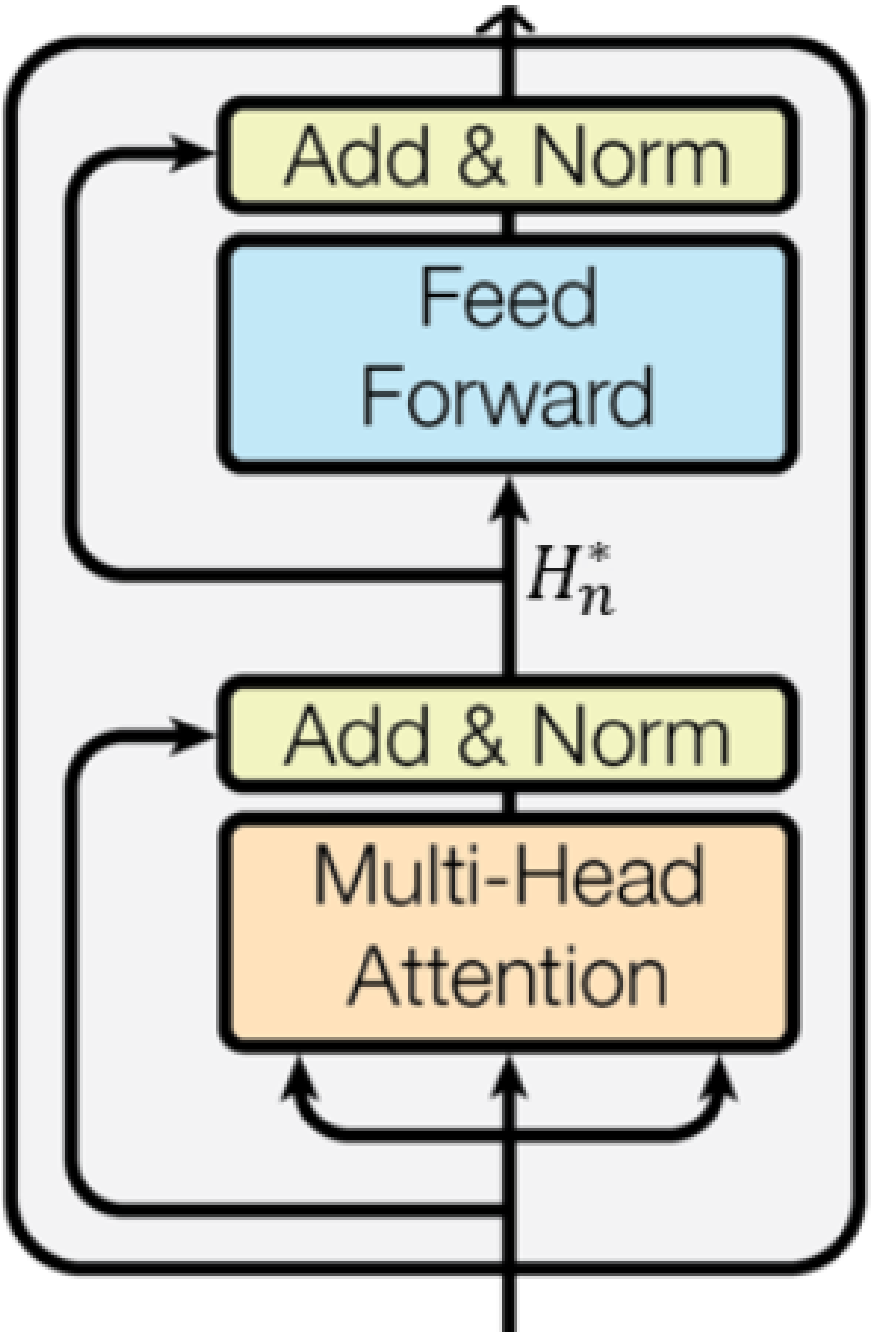
Gambar 3.10: Ilustrasi *layer normalization* pada *transformer*.

Koneksi Residu adalah koneksi yang menghubungkan *output* dari suatu *layer* dengan *input* dari *layer* selanjutnya. Koneksi residu digunakan untuk mengatasi masalah *vanishing gradient* yang terjadi pada *deep neural network* dengan memperbaiki *flow gradient* dari model. Persamaan matematis dari koneksi residu dijelaskan seperti pada Persamaan 3.55.

$$f_l'(\mathbf{x}) = f_l(\mathbf{x}) + \mathbf{x}, \tag{3.55}$$

dengan $f_l(\mathbf{x})$ adalah suatu *layer* atau kumpulan *layer* pada *deep neural network*. (3.56)

Pada *transformer*, *residual connection* digunakan sebelum *layer normalization* seperti pada Gambar 3.11.



Gambar 3.11: Ilustrasi koneksi residu.

3.2.8 Transformer Encoder

Dengan menggunakan *multi-head self-attention layer*, *position-wise feed-forward network layer*, dan *layer normalization* dan *residual connection* yang sudah dijelaskan sebelumnya, blok *encoder* pada *transformer encoder* dapat ditulis seperti pada Per-

samaan 3.62 hingga Persamaan 3.64.

$$\mathbf{X} \in \mathbb{R}^{L \times d_{\text{token}}}, \tag{3.57}$$

$$\text{EncoderBlock}(\mathbf{X}) = \mathbf{Z}_2, \tag{3.58}$$

$$\mathbf{Z}_2 = \text{LayerNorm}(\overbrace{\text{FFN}(\mathbf{Z}_1) + \mathbf{Z}_1}^{\text{Koneksi Residu}}), \tag{3.59}$$

$$\mathbf{Z}_1 = \text{LayerNorm}(\overbrace{\text{MHSA}_h(\mathbf{X}), +\mathbf{X}}^{\text{Koneksi Residu}}). \tag{3.60}$$

$$\tag{3.61}$$

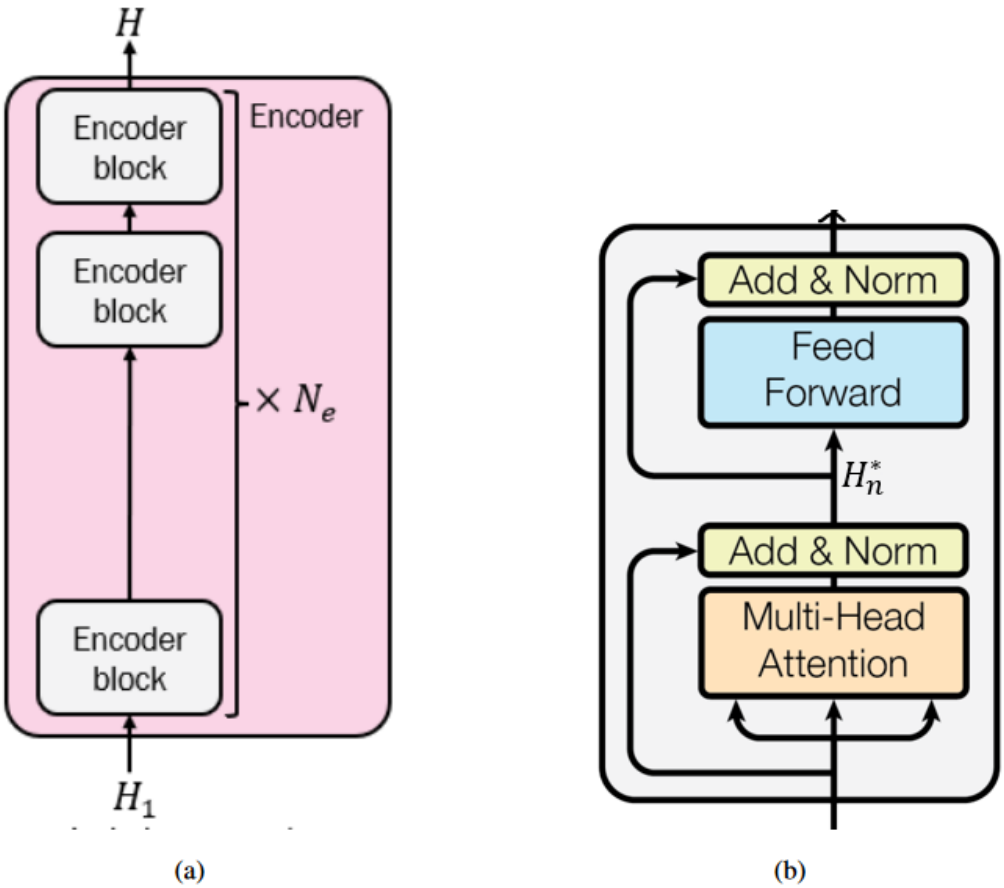
Terakhir, *transformer encoder* adalah komposisi dari beberapa blok *encoder*. Untuk *input* token $t = (t_1, t_2, \ldots, t_L)$, *transformer encoder* menghasilkan representasi vektor kontekstual dari setiap tokennya ditunjukkan pada Persamaan 3.62 hingga Persamaan 3.64.

$$t = (t_1, t_2, \ldots, t_L), \tag{3.62}$$

$$\mathbf{E} = \text{embed}(t) + \text{pos}(t), \tag{3.63}$$

$$\text{Encoder}(\mathbf{x}) = \text{EncoderBlock}_n(\text{EncoderBlock}_{n-1}(\ldots(\text{EncoderBlock}_1(\mathbf{X})))). \tag{3.64}$$

$$\tag{3.65}$$



Gambar 3.12: Ilustrasi *transformer encoder*.

3.3 *Bidirectional Encoder Representations from Transformers* (BERT)

Bidirectional Encoder Representations from Transformers (BERT) adalah *transformers encoder* yang telah dilatih sebelumnya (*pre-trained*) pada tugas *Masked Language Model* dan *Next Sentence Prediction* (Devlin et al., 2018). *Pre-training* BERT dilakukan dengan menggunakan korpus teks yang besar dan dilakukan secara *self-supervised* untuk mempelajari informasi umum tentang statistik bahasa. pengunaan BERT mengikuti prinsip *transfer learning*, yaitu model yang sudah dilatih sebelumnya pada tugas tertentu – dengan jumlah data yang besar – dapat digunakan untuk tugas lainnya dengan hanya menggunakan sedikit data latih. Model BERT tersusun atas 12 blok *encoder* dengan dimensi *token embedding* sebesar 768 dan 12 *attention heads* pada setiap blok *encoder*-nya.

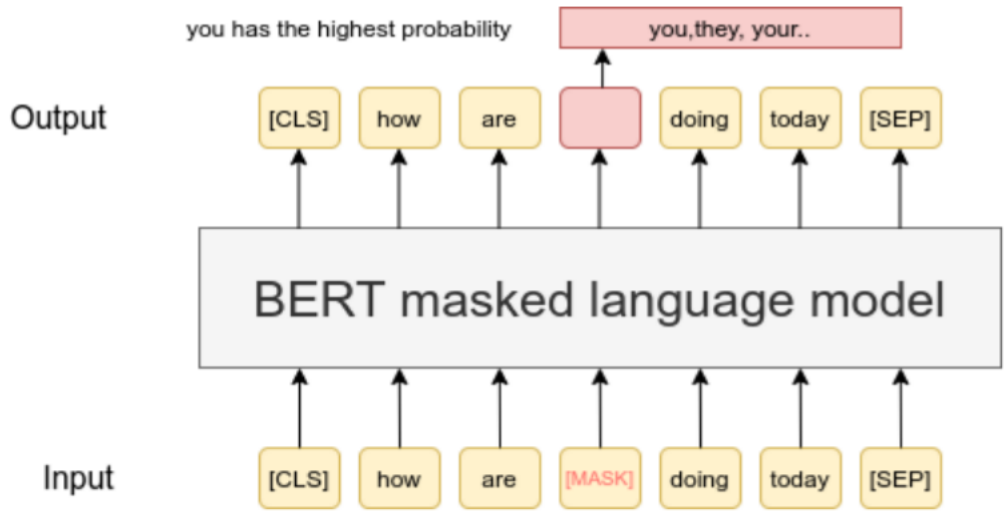
3.3.1 Representasi Input

Representasi *input* dari model BERT tidak hanya terdiri dari *token embedding* dan *positional encoding* seperti pada *transformer encoder* yang dijelaskan pada Subbab 3.2.8

3.3.2 *pre-traning* BERT

Pada tahap *pre-traning*, BERT dilatih pada dua tugas *self-supervised* dengan jumlah data yang besar, yaitu *Masked Language Model* dan *Next Sentence Prediction* yang masing-masing akan dijelaskan pada Subbab 3.3.2.1 dan Subbab 3.3.2.2. proses *pre-training* menggunakan paragraf-paragraf pada korpus teks yang besar, yaitu Wikipedia dengan 2.5 Miliar kata dan BookCorpus dengan 800 Juta kata.

3.3.2.1 *Masked Language Model* (MLM)



Gambar 3.13: Ilustrasi *Masked Language Modeling* (MLM) pada BERT. sebuah kata (token) secara acak di-hilangkan (*mask*) dan model diminta untuk menebak kata yang dihilangkan tersebut.

Tugas *Masked Language Model*(MLM) adalah tugas untuk memprediksi token yang dihilangkan (*masking*) pada suatu kalimat. Sebagai contoh pada kalimat `Let's make [MASK] chicken! [SEP] It [MASK] great with orange sauce`, model harus memprediksi token `fried` dan `tastes` pada token yang dihilangkan tersebut.

memprediksi kata yang dihilangkan memaksa *transformer* untuk memahami sintaks dan konteks dari kalimat tersebut. Sebagai contoh, model harus memahami bahwa kata

sifat `red` sering terletak sebelum kata benda seperti `house` atau `car`, tetapi tidak sebelum kata kerja seperti `shout`. Selain itu, tugas ini membuat model untuk memperoleh pemahaman umum tentang bahasa alami. Misalnya, setelah dilatih, model akan memberikan probabilitas yang lebih tinggi untuk kata `train` yang hilang dalam kalimat `[MASK] pulled into the station` daripada kata `chicken`.

selama proses pelatiahan MLM, sebanyak 15% dari semua token dipilih untuk dilakukan *masking*. 80% dari token yang terpilih akan diubah menjadi token `[MASK]`, 10% menjadi token acak, dan 10% lainnya adalah token yang sama.

3.3.2.2 *Next Sentence Prediction*

pada tugas *Next Sentence Prediction*, model BERT dapat dilatih untuk memprediksi apakah kalimat kedua dari pasangan kalimat adalah kalimat berikutnya. dengan kata lain, untuk pasangan kalimat (q,d), model BERT harus memprediksi apakah kalimat d adalah kalimat berikutnya dari kalimat q. *Input* dari tugas ini adalah `[CLS]` $t_{q_1}, t_{q_2}, \ldots, t_{q_n}$ `[SEP]` $t_{d_1}, t_{d_2}, \ldots, t_{d_m}$ `[SEP]` dengan t_{q_i} adalah token dari kalimat q dan t_{d_i} adalah token dari kalimat d dan *output* dari tugas ini adalah variabel biner yang menunjukkan apakah kalimat d adalah kalimat berikutnya dari kalimat q.

Selama proses pelatihan, setengah dari *input* tersebut adalah pasangan kalimat di mana kalimat kedua adalah kalimat berikutnya, dan setengah lainnya adalah kalimat yang diambil secara acak dari korpus sebagai kalimat kedua. Gambar **??** mengilustrasikan contoh dari tugas *next sentence prediction*.

3.3.3 BERT untuk Bahasa Indonesia (IndoBERT)

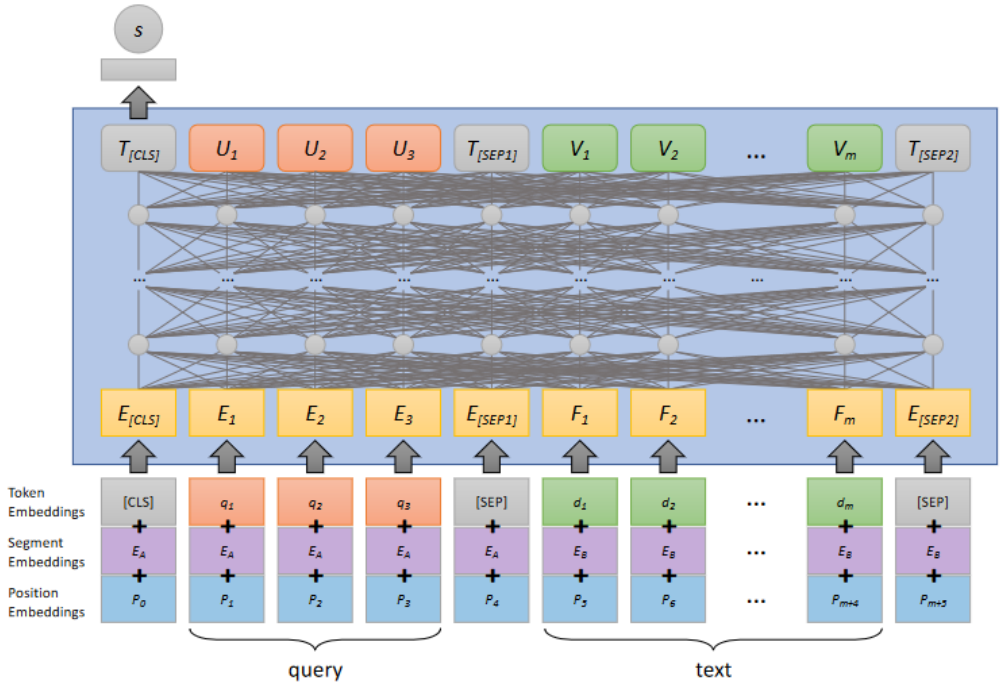
Pada penelitian ini, BERT yang digunakan adalah BERT untuk bahasa Indonesia yang sudah dilakukan *pre-training* sebelumnya oleh **?**. *Pre-training* BERT untuk bahasa Indonesia dilakukan dengan menggunakan korpus Wikipedia bahasa Indonesia dengan 74 Juta kata, artikel berita dari Kompas, Tempo, dan Liputan6 dengan 55 Juta kata, dan korpus web bahasa Indonesia dengan 90 Juta kata. model IndoBERT dilatih selama 2.4 Juta iterasi (180 epoch).

3.3.4 Penggunaan BERT untuk Pemeringkatan Teks

Bagian berikut akan mejelaskan penggunaan BERT untuk pemeringkatan teks. terdapat dua arsitektur yang digunakan, yaitu BERT_{CAT} dan BERT_{DOT} yang masing-masing akan dijelaskan pada Subbab 3.3.4.1 dan Subbab 3.3.4.2.

Model peringkat BERTCAT

3.3.4.1 BERT_{CAT}



Gambar 3.14: BERT_{CAT} mengambil kueri dan kandidat teks yang akan diberi skor sebagai *input* dan menggunakan BERT untuk klasifikasi relevansi. Penjumlahan elemen-wise dari token,*segment*, dan *positional embeddings* membentuk representasi vektor *input*. Setiap token input memiliki vektor kontekstual sebagai *output* model BERT. *Linear layer* menerima representasi akhir token [CLS] dan menghasilkan skor relevansi teks terkait dengan kueri (Lin et al., 2020).

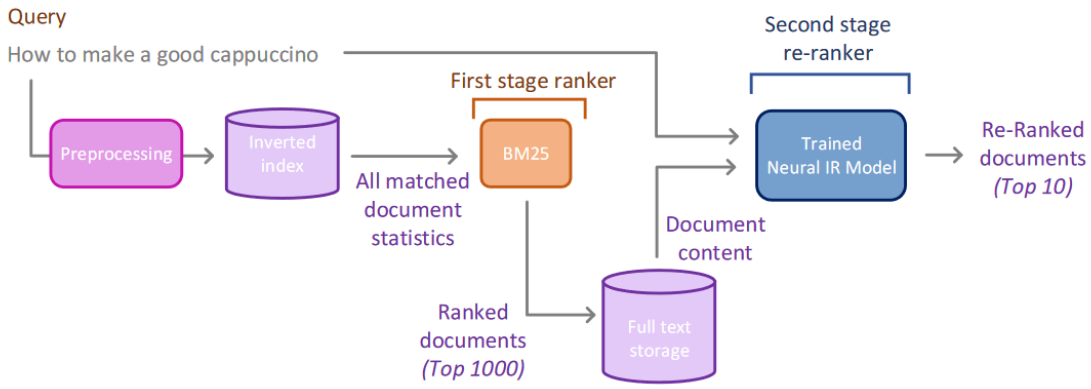
Salah satu cara pengunaan BERT untuk pemeringkatan Teks adalah dengan menggunakan BERT pada model untuk melakukan *soft classification* nilai relevansi dari pasangan (kueri, teks). Dengan kata lain, skor relevansi dari pasangan (kueri, teks) adalah probabilitas bahwa teks tersebut relevan dengan kueri seperti yang ditunjukkan pada Persamaan 3.66 hingga Persamaan 3.67.

$$\text{score}(q,d) = P(\text{relevance} = 1|q,d) = \sigma\left(\mathbf{h}_{[\text{CLS}]}\mathbf{W}^{\text{CLS}} + \mathbf{b}^{\text{CLS}}\right), \tag{3.66}$$

$$\mathbf{h}_{[\text{CLS}]} = \text{BERT}((\text{[CLS]}, q, \text{[SEP]}, d, \text{[SEP]}))_{[\text{CLS}]} \in \mathbb{R}^{d_{\text{token}}}, \tag{3.67}$$

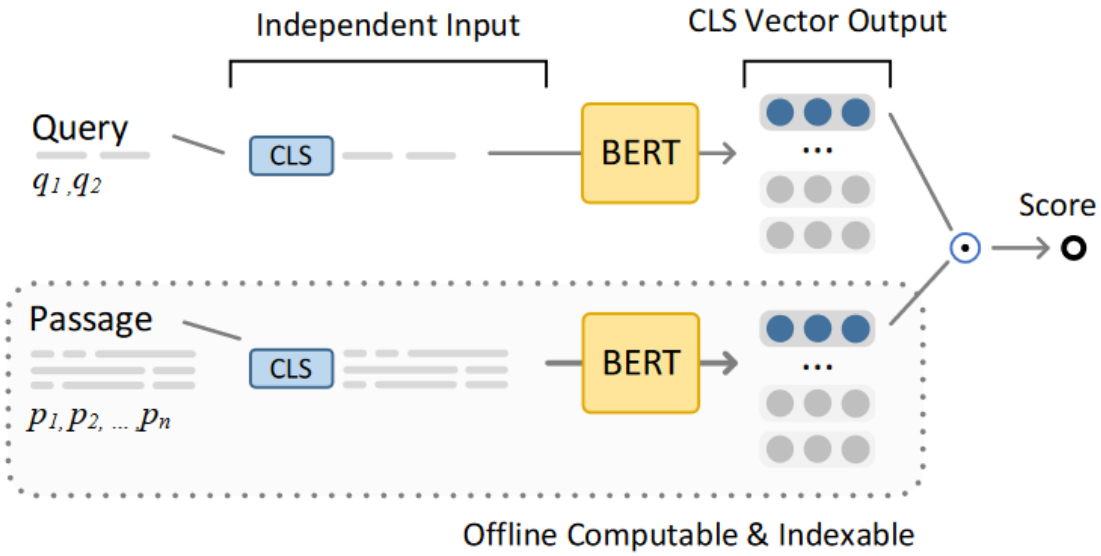
dengan $\mathbf{W}^{\text{CLS}} \in \mathbb{R}^{d_{\text{token}} \times 1}$ dan $\mathbf{b}^{\text{CLS}} \in \mathbb{R}$ adalah matriks bobot dan bias yang digunakan untuk melakukan *classification*, dan σ adalah fungsi sigmoid.

Untuk suatu kueri q dan kumpulan teks $D = \{d_1, d_2, \ldots, d_n\}$, perlu dilakukan perhitungan skor relevansi untuk setiap pasangan (q, d_i) dengan $i = 1, \ldots, n$ sebelum dilakukan pemeringkatan. hal ini menjadi masalah untuk jumlah dokumen yang besar karena setiap perhitungan skor relevansi dengan BERT_{CAT} membutuhkan waktu yang lama. Oleh karena itu, BERT_{CAT} biasanya digunakan sebagai *reranker* dari sistem pemeringkatan teks. teks yang akan diberikan skor relevansinya oleh BERT_{CAT} adalah teks yang sudah dipilih oleh sistem pemeringkatan teks yang lebih efisien seperti BM25 – biasanya $k = 100, 1000$ teks teratas dipilih oleh BM25. Gambar 3.15 mengilustrasikan arsitektur *retrieve* dan *rerank*.



Gambar 3.15: Arsitektur *retrieve and rerank*. *First-stage retrieval* dilakukan oleh BM25 dan *reranking* dilakukan oleh model *scoring* yang lebih kompleks seperti BERT_{CAT} (Hofstätter et al., 2021).

3.3.4.2 BERT_{DOT}



Gambar 3.16: BERT_{DOT} memetakan kueri dan kandidat teks ke dalam ruang vektor yang sama dan menghitung skor relevansi dengan melakukan *dot product* antara vektor representasi kontekstual dari kueri dan teks (Hofstätter et al., 2021).

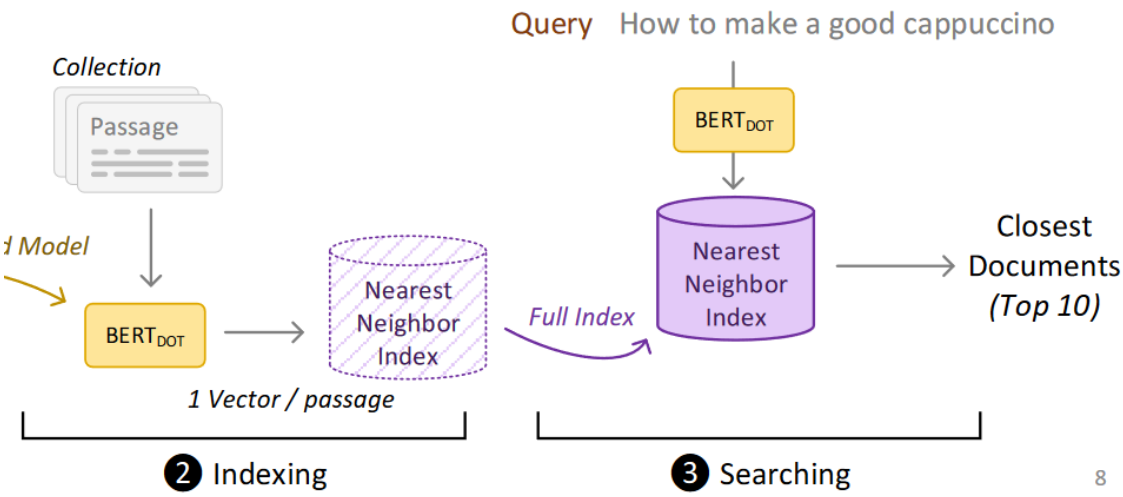
Berbeda dengan BERT_{CAT}, BERT_{DOT} tidak melakukan *soft classification* untuk setiap pasangan (kueri, teks). BERT_{DOT} menghitung skor relevansi dari pasangan (kueri, teks) dengan melakukan *dot product* antara vektor representasi kontekstual dari kueri dan teks seperti yang ditunjukkan pada Persamaan 3.68 hingga Persamaan 3.70.

$$\mathbf{q}_{[\text{CLS}]} = \text{BERT}(([\text{CLS}], q, [\text{SEP}]))_{[\text{CLS}]} \in \mathbb{R}^{d_{\text{token}}}, \tag{3.68}$$

$$\mathbf{d}_{[\text{CLS}]} = \text{BERT}(([\text{CLS}], d, [\text{SEP}]))_{[\text{CLS}]} \in \mathbb{R}^{d_{\text{token}}}, \tag{3.69}$$

$$\text{score}(q, d) = \mathbf{q}_{[\text{CLS}]}\mathbf{d}_{[\text{CLS}]}^{\top} \in \mathbb{R}. \tag{3.70}$$

Salah satu kelebihan dari BERT_{DOT} adalah vektor representasi dari setiap teks dapat dihitung terlebih dahulu dan disimpan dalam memori. Akibatnya, kita hanya perlu menghitung vektor dari kueri dan nilai *dot product* untuk setiap teksnya ketika akan dilakukan pemeringkatan – yang tentunya lebih efisien secara komputasi. Hal ini membuat BERT_{DOT} lebih cepat daripada BERT_{CAT} untuk melakukan pemeringkatan. Gambar 3.17 mengilustrasikan arsitektur pemeringkatan dengan BERT_{DOT}.



Gambar 3.17: Arsitektur pemeringkatan dengan BERT_{DOT}. Vektor representasi dari setiap teks dapat diindeks terlebih dahulu dan disimpan dalam memori (Hofstätter et al., 2021).

BAB 4
HASIL SIMULASI DAN PEMBAHASAN

Bab ini membahas mengenai proses *fine tuning* model *Bidirectional Encoder Representations from Transformers* (BERT) untuk mendapatkan model yang dapat digunakan untuk masalah pemeringkatan teks. Subbab4.1 menjelaskan mengenai spesifikasi perangkat keras dan perangkat lunak yang digunakan dalam penelitian. Selanjutnya, Subbab 4.2 menjelaskan mengenai tahapan simulasi yang dilakukan dalam penelitian. Informasi mengenai *dataset* latih dan uji dijelaskan pada Subbab 4.3. Subbab 4.4 menjelaskan lebih detail mengenai arsitektur model BERT, fungsi loss, serta konfigurasi *hyperparameter* yang digunakan dalam proses *fine tuning* model BERT. Terakhir, Subbab 4.5 menjelaskan mengenai evaluasi hasil *fine tuning* model BERT untuk pemeringkatan teks.

4.1 Spesifikasi Mesin dan Perangkat Lunak

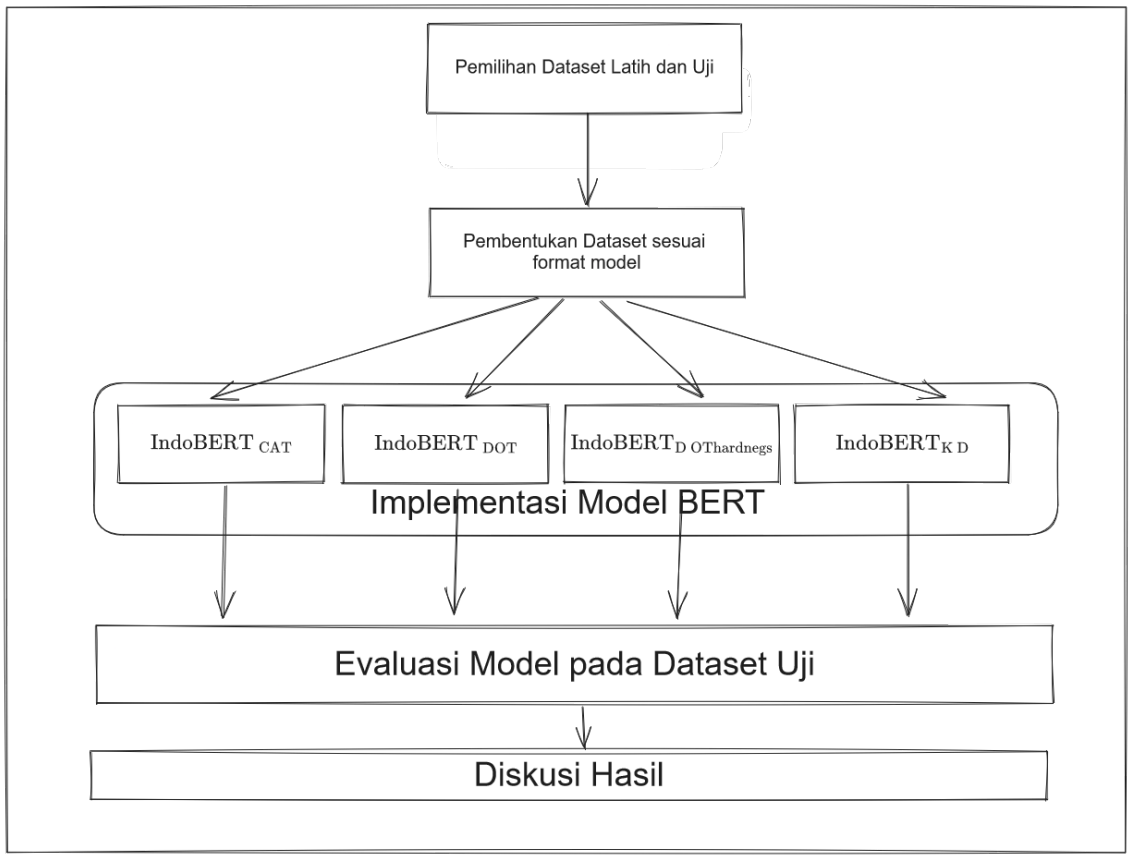
Proses *fine tuning* model BERT untuk pemeringkatan teks dilakukan menggunakan mesin dan perangkat lunak yang tertera pada Tabel 4.1.

Tabel 4.1: Spesifikasi Perangkat Lunak

CPU	AMD Ryzen 9 5950X 32-Core Processor
GPU	NVIDIA GeForce RTX 4090 24GB
Memori	64GB
Sistem Operasi	Ubuntu 20.04.2 LTS
Perangkat Lunak Pemrograman	Visual Studio Code 1.84.2
Bahasa Pemrograman	Python 3.8
Pustaka yang Digunakan	sentence-transformers 2.2.2 transformers 4.35.1 beir 2.0.0 gdown 4.7.1 torch 2.0.1+cu117

4.2 Tahapan Simulasi

Gambar 4.1 menunjukkan tahapan simulasi yang dilakukan dalam penelitian ini.



Gambar 4.1: Diagram Simulasi

Simulasi diawali dengan pengambilan data. Data yang digunakan adalah data pada penelitian Bonifacio, Campiotti, de Alencar Lotufo, dan Nogueira (2021), sebagai *dataset* latih, dan data pada penelitian X. Zhang, Ma, Shi, dan Lin (2021), X. Zhang et al. (2023) sebagai *dataset* ujinya. *Dataset* latih tidak dapat digunakan langsung untuk melatih model-model tersebut. Untuk setiap modelnya, diperlukan transformasi untuk mengubah bentuk dari *dataset* latih sehingga sesuai dengan formatnya. Transformasi *dataset* latih dan *hyperparameter* dari model akan dibahas lebih lanjut pada bagian Subbab 4.4. Selanjutnya, proses implementasi dan pelatihan model dilakukan. Setelah itu, Evaluasi dari setiap model pada *dataset* uji dilakukan. Terdapat satu model BM25 sebagai *baseline* untuk setiap modelnya. Terakhir, terdapat diskusi mengenai hasil evaluasi tersebut.

4.3 Data

Penelitian ini menggunakan satu *dataset* latih Mmarco train set Indonesia (Bonifacio et al., 2021) dan tiga *dataset* uji, yaitu Mmarco Dev set Indonesia, MrTyDi Test set In-

donesia (X. Zhang et al., 2021), dan Miracl Dev set Indonesia (X. Zhang et al., 2023). *Dataset* Miracl dan MrTyDi dipilih sebagai uji kemampuan *out-of-distibution* dari model yang dihasilkan. *Dataset* tersebut terdiri dari 3 file, yaitu *file* kueri, *file korpus* dan *file jugdements* yang telah dijelaskan pada Subbab 2.1.1. Tabel 4.2 menunjukkan informasi mengenai jumlah entri dari *file* kueri, *file korpus*, dan *file jugdements* dari setiap *dataset* yang digunakan dalam penelitian ini. Setiap *judgements* pada *dataset* adalah *jugdments* biner, yaitu bernilai 1 jika dokumen tersebut relevan dengan kueri dan 0 jika tidak relevan dengan kueri. Gambar **??** menunjukkan contoh *file jugdements* dari *dataset* Miracl Dev set Indonesia.

Tabel 4.2: Dataset Information

Dataset	Korpus	Kueri	Jugdements
Mmarco train set	8,841,823	1,010,916	532,761
Mmarco dev set	8,841,823	1,010,916	7,437
Mrtydi test set	1,469,399	829	961
Miracl dev set	1,446,315	960	9,668

4.4 Fine Tuning Model BERT

4.4.1 IndoBERT$_{\text{CAT}}$

Pada model IndoBERTCAT, arsitektur BERTCAT (lihat Subbab 3.3.4.1) digunakan untuk melakukan pemeringkatan teks. Selama proses pelatihan model, *dataset* yang digunakan berasal dari Mmarco train set dengan format (q,d,r) dengan q adalah kueri, d adalah dokumen, dan r adalah relevansi dokumen d terhadap kueri q. Pelatihan yang dilakukan adalah pelatihan untuk klasifikasi relevansi dokumen terhadap kueri. Perlu dicatat bahwa tidak ada contoh $r = 0$ dalam *dataset* Mmarco train set (lihat Subbab 2.1.1).

Untuk membentuk data latih dengan penilaian $r = 0$, pasangan $(q,d',0)$ ditambahkan dengan d' sebagai dokumen acak yang tidak relevan dengan kueri q. *Dataset* yang telah dibuat terdiri dari 500 ribu pasangan (q,d,r) dengan rasio 1:1 antara $r = 1$ dan $r = 0$. contoh dari *dataset* yang digunakan untuk pelatihan model IndoBERT$_{\text{CAT}}$ dapat ditemukan pada Tabel 4.3.

konfigurasi *hyperparameter* selama pelatiahan model indoBERT$_{\text{CAT}}$ dapat dilihat pada Tabel 4.4.

Tabel 4.3: Converted Table

Kueri	Teks	Relevansi
Berapa banyak kalori sehari yang hilang saat menyusui?	Tidak hanya menyusui lebih baik untuk bayi, namun penelitian juga mengatakan itu lebih baik bagi ibu. Menyusui membakar rata-rata 500 kalori sehari, dengan kisaran khas antara 200 hingga 600 kalori yang terbakar sehari. Diperkirakan produksi 1 oz. ASI membakar 20 kalori. Jumlah kalori yang terbakar tergantung pada seberapa banyak bayi makan. Menyusui kembar membakar dua kali lebih banyak daripada memberi makan hanya satu bayi. Dengan anak kembar, ibu mereka membakar 1000 kalori per hari. Membakar 500 kalori ekstra sehari akan menghasilkan satu pon penurunan berat badan mingguan.	1
Karakteristik iklim utama hutan hujan tropis	Kacang kola adalah buah dari pohon kola, genus (Cola) pohon yang berasal dari hutan hujan tropis Afrika.	0
Berapa lama pemulihan dari humerus yang patah?	Pemulihan Stroke. Stroke mempengaruhi setiap orang secara berbeda. Banyak penderita stroke yang terus membaik dalam waktu yang lama, kadang-kadang selama beberapa tahun. Pemulihan dari stroke melibatkan membuat perubahan dalam aspek fisik, sosial dan emosional hidup Anda. Anda akan membuat perubahan untuk mencegah stroke tambahan serta untuk memfasilitasi pemulihan seumur hidup Anda.	0

4.4.2 IndoBERT$_{\text{DOT}}$

Pada model IndoBERT$_{\text{DOT}}$, arsitektur BERT$_{\text{DOT}}$ (lihat Subbab 3.3.4.2) digunakan untuk melakukan pemeringkatan teks. fungsi loss yang digunakan untuk pelatihan model IndoBERT$_{\text{DOT}}$ adalah *N-pair loss*. Untuk kueri q, dokumen relevan d^+, dan kumpulan dokumen tidak relevan $\{d_i^-\}_{i=1}^{n-1}$ terhadap kueri q, *N-pair loss* didefinisikan sebagai berikut:

Tabel 4.4: Hyperparameter IndoBERT$_{\text{CAT}}$

Parameter	**Nilai**
Model pralatih	`indolem/indobert-base-uncased`
Total data	500,000
Batch Size	32
Total iterasi	78125 (5 epochs)
Optimizer	Adam dengan $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 1e-8$
Learning rate	2e-5
Learning rate warmup	Linear dengan 10% dari total iterasi
Fungsi loss	*Binary cross entropy*

$$L(q, d^+, \{d_i^-\}_{i=1}^{N-1}) = -\log \frac{\exp(\mathbf{h}_q^\top \mathbf{h}_d^+)}{\exp(\mathbf{h}_q^\top \mathbf{h}_d^+) + \sum_{i=1}^{n-1} \exp(\mathbf{h}_q^\top \mathbf{h}_i^-)}, \tag{4.1}$$

dengan keterangan sebagai berikut:

$$\mathbf{h}_q = \text{IndoBERT}_{\text{DOT}}((\text{[CLS]}, q, \text{[SEP]}))_{\text{[CLS]}}$$
$$\mathbf{h}_d^+ = \text{IndoBERT}_{\text{DOT}}(\text{[CLS]}, d^+, \text{[SEP]})_{\text{[CLS]}}$$
$$\mathbf{h}_i^- = \text{IndoBERT}_{\text{DOT}}(\text{[CLS]}, d_i^-, \text{[SEP]})_{\text{[CLS]}}$$

dataset Mmarco train set langsung dapat digunakan karena *dataset* tersebut sudah dalam bentuk (q, d^+). Kumpulan teks tak relevan $\{d_i^-\}_{i=1}^{n-1}$ dibentuk dengan menggunakan teks d pada *data point* yang lain pada batch yang sama. dengan begitu, nilai N pada *N-pair loss* adalah ukuran batch yang digunakan selama pelatihan model. Metode pemilihan dokumen negatif ini disebut dengan *in-batch negative sampling* (Karpukhin et al., 2020). Pada penelitian ini, digunakan seluruh *datapoint* pada *file jugdements* Mmarco train set – 532,761 *data point* – untuk membentuk *dataset* latih model IndoBERT$_{\text{DOT}}$.

Konfigurasi *hyperparameter* selama pelatiahan model indoBERT$_{\text{DOT}}$ dapat dilihat pada Tabel 4.5.

4.4.3 IndoBERT$_{\text{DOThardnegs}}$

Pada IndoBERT$_{\text{DOThardnegs}}$, arsitektur BERT$_{\text{DOT}}$ digunakan untuk melakukan pemeringkatan teks. fungsi loss yang digunakan untuk pelatihan model IndoBERT$_{\text{DOThardnegs}}$ adalah *N-pair loss* seperti IndoBERT$_{\text{DOT}}$. Perbedaan utama antara IndoBERT$_{\text{DOT}}$ dan

Tabel 4.5: *Hyperparameter* IndoBERT$_{\text{DOT}}$

Parameter	**Nilai**
Model pralatih	`indolem/indobert-base-uncased`
Batch Size	32
Total Iterasi	83243 (5 epochs)
Optimizer	Adam dengan $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 1e-8$
Learning rate	2e-5
Learning rate warmup	Linear dengan 10% dari total iterasi
Fungsi *loss*	*N-pair loss*

IndoBERT$_{\text{DOThardnegs}}$ adalah metode pemilihan dokumen negatif. Pada IndoBERT$_{\text{DOT}}$, dokumen negatif dipilih dari *data point* lain pada batch yang sama. Pada IndoBERT$_{\text{DOThardnegs}}$, dokumen negatif sudah terlebih dahulu dipilih dengan kriteria bahwa dokumen tersebut adalah dokumen yang tidak relevan dengan kueri q, tetapi pemeringkatan dengan BM25 berada di 100 teratas. Dengan kata lain, dokumen negatif adalah dokumen yang sulit dibedakan dengan dokumen positif menggunakan BM25. Dokumen *hard negative* ini diberikan oleh file `sentence-transformers/msmarco-hard-negatives` (Thakur, Reimers, Rücklé, Srivastava, & Gurevych, 2021) dan Tabel 4.6 menunjukkan contoh dari dokumen *file* tersebut. Nilai N yang dipilih pada penelitian ini adalah $N = 5$, dan jumlah data yang digunakan adalah 502.939 *data point*.

Tabel 4.6: Potongan dari *file* `sentence-transformers/msmarco-hard-negatives`. kolom qid berisikan id dari kueri, kolom *positive* adalah id dokumen positif, dan kolom *hard negative* adalah id dokumen yang sulit dibedakan dengan dokumen positif menggunakan BM25.

qid	*Positive*	*Hard Negative*
634	1668221	{ ”bm25”: [830424, 1985345, 1153265, 1638798, 2866545] }
492	875	{ ”bm25”: [1147445, 4859158, 7980570, 3409840, 6889336] }

Konfigurasi *hyperparameter* selama pelatiahan model indoBERT$_{\text{DOThardnegs}}$ dapat dilihat pada Tabel 4.7.

Tabel 4.7: *Hyperparameter* $\text{IndoBERT}_{\text{DOThardnegs}}$

Parameter	Nilai
Model pralatih	`indolem/indobert-base-uncased`
Total data	502,939
Batch Size	32
Total Iterasi	78585 (5 epochs)
Optimizer	Adam dengan $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 1e-8$
Learning rate	2e-5
Learning rate warmup	Linear dengan 10% dari total iterasi
Fungsi *loss*	*N-pair loss*

4.4.4 $\text{IndoBERT}_{\text{DOTMargin}}$

4.4.5 $\text{IndoBERT}_{\text{DOTKD}}$

$\text{IndoBERT}_{\text{KD}}$ dilatih dengan menggunakan prinsip *knowledge distillation*, yaitu proses *transfer* pengetahuan dari model yang sudah dilatih dengan baik (guru) ke model yang belum dilatih (murid). Model yang digunakan sebagai guru adalah model bahasa Inggris yang sudah dilatih dengan baik untuk melakukan pemeringkatan teks. Model Murid harus dipilih adalah model pralatih BERT multibahasa – model yang *pre-training*-nya dilakukan pada korpus multibahasa. Pemetaaan vektor dari model murid akan di-*align* dengan Pemetaan vektor dari model guru dengan fungsi *loss* berikut (Reimers & Gurevych, 2020):

$$L(s_i, t_i) = \left(|| \, M(s_i) - \hat{M}(s_i) \, || + || \, M(s_i) - \hat{M}(s_) \, || \right), \tag{4.2}$$

dengan keterangan sebagai berikut:

$M =$ model guru (bahasa Inggris)

$\hat{M} =$ model pralatih BERT murid (multibahasa)

$s_i =$ teks sumber (bahasa Inggris)

$t_i =$ teks target (bahasa Indonesia)

Tabel 4.8: Caption

Model	Mmarco Dev		MrTyDi Test		Miracl Dev	
	MRR@10	R@1000	MRR@10	R@1000	NCDG@10	R@1K
BM25 (Elastic Search)	.114	.642	.279	.858	.391	.971

4.5 Hasil Fine Tuning dan Evaluasi

4.5.1 Evaluasi BM25

4.5.2 Evaluasi IndoBERT$_{\mathbf{MEAN}}$

Tabel 4.9: Caption

Model	Mmarco Dev		MrTyDi Test		Miracl Dev	
	MRR@10	R@1000	MRR@10	R@1000	NCDG@10	R@1K
BM25 (Elastic Search)	.114	.642	.279	.858	.391	.971
IndoBERT$_{\text{MEAN}}$	.000	.000	.000	.000	.000	.000

4.5.3 Evaluasi IndoBERT$_{\mathbf{CAT}}$

Tabel 4.10: Caption

Model	Mmarco Dev		MrTyDi Test		Miracl Dev	
	MRR@10	R@1000	MRR@10	R@1000	NCDG@10	R@1K
BM25 (Elastic Search)	.114	.642	.279	.858	.391	.971
IndoBERT$_{\text{CAT}}$	.181	.642	.447	.858	.455	.971

4.5.4 Evaluasi IndoBERT$_{\mathbf{DOT}}$

Tabel 4.11: Caption

Model	Mmarco Dev		MrTyDi Test		Miracl Dev	
	MRR@10	R@1000	MRR@10	R@1000	NCDG@10	R@1K
BM25 (Elastic Search)	.114	.642	.279	.858	.391	.971
IndoBERT$_{\text{DOT}}$	.192	.847	.378	.936	.355	.920

4.5.5 Evaluasi IndoBERT$_{\text{DOThardnegs}}$

Tabel 4.12: Caption

Model	Mmarco Dev		MrTyDi Test		Miracl Dev	
	MRR@10	R@1000	MRR@10	R@1000	NCDG@10	R@1K
BM25 (Elastic Search)	.114	.642	.279	.858	.391	.971
IndoBERT$_{\text{DOThardnegs}}$	.232	.847	.471	.921	.397	.898

4.5.6 Evaluasi IndoBERT$_{\text{DOTMargin}}$

Tabel 4.13: Caption

Model	Mmarco Dev		MrTyDi Test		Miracl Dev	
	MRR@10	R@1000	MRR@10	R@1000	NCDG@10	R@1K
BM25 (Elastic Search)	.114	.642	.279	.858	.391	.971
IndoBERT$_{\text{DOTMargin}}$	.207	.799	.446	.929	.387	.899

4.5.7 Evaluasi IndoBERT$_{\text{KD}}$

Tabel 4.14: Caption

Model	Mmarco Dev		MrTyDi Test		Miracl Dev	
	MRR@10	R@1000	MRR@10	R@1000	NCDG@10	R@1K
BM25 (Elastic Search)	.114	.642	.279	.858	.391	.971
IndoBERT$_{\text{KD}}$	-	.803	.300	.761	-	-

4.5.8 Perbandingan Hasil Evaluasi

Tabel 4.15: Caption

Model	Mmarco Dev		MrTyDi Test		Miracl Dev	
	MRR@10	R@1000	MRR@10	R@1000	NCDG@10	R@1K
BM25 (Elastic Search)	.114	.642	.279	.858	.391	.971
$\text{IndoBERT}_{\text{MEAN}}$	.000	.000	.000	.000	.000	.000
$\text{IndoBERT}_{\text{CAT}}$	.181	.642	.447	.858	.455	.971
$\text{IndoBERT}_{\text{DOT}}$	.192	.847	.378	.936	.355	.920
$\text{IndoBERT}_{\text{DOTdnegs}}$	.232	.847	.471	.921	.397	.898
$\text{IndoBERT}_{\text{DOTMargin}}$	.207	.799	.446	.929	.387	.899
$\text{IndoBERT}_{\text{KD}}$	-	.803	.300	.761	-	-

Tabel 4.16: Caption

Model	Latensi (ms)	Memori(MB)
BM25 (Elastic Search)	6.55	800
$\text{IndoBERT}_{\text{DOT}}$	9.9	3072
$\text{IndoBERT}_{\text{CAT}}$	242	800

BAB 5
PENUTUP

Pada bab ini, Penulis akan memaparkan kesimpulan penelitian dan saran untuk penelitian berikutnya.

5.1 Kesimpulan

Berikut ini adalah kesimpulan terkait pekerjaan yang dilakukan dalam penelitian ini:

1. **Poin pertama**
   Penjelasan poin pertama.
2. **Poin kedua**
   Penjelasan poin kedua.

Tulis kalimat penutup di sini.

5.2 Saran

Berdasarkan hasil penelitian ini, berikut ini adalah saran untuk pengembangan penelitian berikutnya:

1. Saran 1.
2. Saran 2.

DAFTAR REFERENSI

Bahdanau, D., Cho, K., & Bengio, Y. (2016). *Neural machine translation by jointly learning to align and translate.*

Bonifacio, L. H., Campiotti, I., de Alencar Lotufo, R., & Nogueira, R. F. (2021). mmarco: A multilingual version of MS MARCO passage ranking dataset. *CoRR*, *abs/2108.13897*. Diakses dari https://arxiv.org/abs/2108.13897

Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2018). BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, *abs/1810.04805*. Diakses dari http://arxiv.org/abs/1810.04805

Geiger, A., Antic, B., & He, H. (2022). *Lecture: Deep learning, university of tübingen.* https://uni-tuebingen.de/fakultaeten/mathematisch-naturwissenschaftliche-fakultaet/fachbereiche/informatik/lehrstuehle/autonomous-vision/lectures/deep-learning/.

Hofstätter, S., Althammer, S., Sertkan, M., & Hanbury, A. (2021). *Advanced information retrieval 2021 & 2022.* Diakses dari https://github.com/sebastian-hofstaetter/teaching

Karpukhin, V., Oguz, B., Min, S., Lewis, P., Wu, L., Edunov, S., ... Yih, W.-t. (2020, November). Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 conference on empirical methods in natural language processing (emnlp)* (pp. 6769–6781). Online: Association for Computational Linguistics. Diakses dari https://www.aclweb.org/anthology/2020.emnlp-main.550 doi: 10.18653/v1/2020.emnlp-main.550

Lin, J., Nogueira, R. F., & Yates, A. (2020). Pretrained transformers for text ranking: BERT and beyond. *CoRR*, *abs/2010.06467*. Diakses dari https://arxiv.org/abs/2010.06467

pi tau. (2023). An even more annotated transformer. *pi-tau.github.io*. Diakses dari https://pi-tau.github.io/posts/transformer/ (Published on July 13, 2023)

Reimers, N., & Gurevych, I. (2020, 04). Making monolingual sentence embeddings multilingual using knowledge distillation. *arXiv preprint arXiv:2004.09813*. Diakses

dari http://arxiv.org/abs/2004.09813

Thakur, N., Reimers, N., Rücklé, A., Srivastava, A., & Gurevych, I. (2021). BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In *Thirty-fifth conference on neural information processing systems datasets and benchmarks track (round 2)*. Diakses dari https://openreview.net/forum?id=wCu6T5xFjeJ

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. In *Proceedings of the 31st international conference on neural information processing systems* (p. 6000–6010). Red Hook, NY, USA: Curran Associates Inc.

Weng, L. (2018). Attention? attention! *lilianweng.github.io*. Diakses dari https://lilianweng.github.io/posts/2018-06-24-attention/

Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2023). *Dive into deep learning*. Cambridge University Press. (https://D2L.ai)

Zhang, X., Ma, X., Shi, P., & Lin, J. (2021). Mr. TyDi: A multi-lingual benchmark for dense retrieval. *arXiv:2108.08787*.

Zhang, X., Thakur, N., Ogundepo, O., Kamalloo, E., Alfonso-Hermelo, D., Li, X., ... Lin, J. (2023, 09). MIRACL: A Multilingual Retrieval Dataset Covering 18 Diverse Languages. *Transactions of the Association for Computational Linguistics*, *11*, 1114-1131. Diakses dari https://doi.org/10.1162/tacl_a_00595 doi: 10.1162/tacl_a_00595

LAMPIRAN

LAMPIRAN 1: CHANGELOG

@todo

Silakan hapus lampiran ini ketika Anda mulai menggunakan *template*.

Template versi terbaru bisa didapatkan di `https://gitlab.com/ichlaffterlalu/latex-skripsi-ui-2017`. Daftar perubahan pada *template* hingga versi ini:

- versi 1.0.3 (3 Desember 2010):
  - *Template* Skripsi/Tesis sesuai ketentuan *formatting* tahun 2008.
  - Bisa diakses di `https://github.com/edom/uistyle`.
- versi 2.0.0 (29 Januari 2020):
  - *Template* Skripsi/Tesis sesuai ketentuan *formatting* tahun 2017.
  - Menggunakan BibTeX untuk sitasi, dengan format *default* sitasi IEEE.
  - *Template* kini bisa ditambahkan kode sumber dengan *code highlighting* untuk bahasa pemrograman populer seperti Java atau Python.
- versi 2.0.1 (8 Mei 2020):
  - Menambahkan dan menyesuaikan tutorial dari versi 1.0.3, beserta cara kontribusi ke template.
- versi 2.0.2 (14 September 2020):
  - Versi ini merupakan hasil *feedback* dari peserta skripsi di lab *Reliable Software Engineering* (RSE) Fasilkom UI, semester genap 2019/2020.
  - BibTeX kini menggunakan format sitasi APA secara *default*.
  - Penambahan tutorial untuk `longtable`, agar tabel bisa lebih dari 1 halaman dan header muncul di setiap halaman.
  - Menambahkan tutorial terkait penggunaan BibTeX dan konfigurasi *header/footer* untuk pencetakan bolak-balik.

  - Label ”Universitas Indonesia” kini berhasil muncul di halaman pertama tiap bab dan di bagian abstrak - daftar kode program.
  - *Hyphenation* kini menggunakan `babel` Bahasa Indonesia. Aktivasi dilakukan di `hype-indonesia.tex`.
  - Minor adjustment untuk konsistensi *license* dari template.

- versi 2.0.3 (15 September 2020):
  - Menambahkan kemampuan orientasi *landscape* beserta tutorialnya.
  - `\captionsource` telah diperbaiki agar bisa dipakai untuk `longtable`.
  - Daftar lampiran kini telah tersedia, lampiran sudah tidak masuk daftar isi lagi.
  - Nomor halaman pada lampiran dilanjutkan dari halaman terakhir konten (daftar referensi).
  - Kini sudah bisa menambahkan daftar isi baru untuk jenis objek tertentu (custom), seperti: ”Daftar Aturan Transformasi”. Sudah termasuk mekanisme *captioning* dan tutorialnya.
  - Perbaikan minor pada tutorial.

- versi 2.1.0 (8 September 2021):
  - Versi ini merupakan hasil *feedback* dari peserta skripsi dan tesis di lab *Reliable Software Engineering* (RSE) Fasilkom UI, semester genap 2020/2021.
  - Minor edit: ”Lembar Pengesahan”, dsb. di daftar isi menjadi all caps.
  - Experimental multi-language support (Chinese, Japanese, Korean).
  - Support untuk justifikasi dan word-wrapping pada tabel.
  - Penggunaan suffix ”(sambungan)” untuk tabel lintas halaman. Tambahan support suffix untuk `\captionsource`.

- versi 2.1.1 (7 Februari 2022):
  - Update struktur mengikuti fork template versi 1.0.3 di `https://github.com/rkkautsar/edom/ui-thesis-template`.
  - Support untuk simbol matematis `amsfonts`.

  - Kontribusi komunitas terkait improvement GitLab CI, atribusi, dan format sitasi APA bahasa Indonesia.
  - Perbaikan tutorial berdasarkan perubahan terbaru pada versi 2.1.0 dan 2.1.1.

- versi 2.1.2 (13 Agustus 2022):
  - Modifikasi penamaan beberapa berkas.
  - Perbaikan beberapa halaman depan (halaman persetujuan, halaman orisinalitas, dsb.).
  - Support untuk lembar pengesahan yang berbeda dengan format standar, seperti Laporan Kerja Praktik dan Disertasi.
  - Kontribusi komunitas terkait kesesuaian dengan format Tugas Akhir UI, kelengkapan dokumen, perbaikan format sitasi, dan *quality-of-life*.
  - Perbaikan tutorial.
- versi 2.1.3 (22 Februari 2023):
  - Dukungan untuk format Tugas Akhir Kelompok di Fasilkom UI.
  - Dukungan untuk format laporan Kampus Merdeka Mandiri di Fasilkom UI.
  - Minor bugfix: Perbaikan kapitalisasi variabel.
  - Quality-of-Life: Pengaturan kembali `config/settings.tex`.
  - Tutorial untuk beberapa *use case*.

LAMPIRAN 2: JUDUL LAMPIRAN 2

Lampiran hadir untuk menampung hal-hal yang dapat menunjang pemahaman terkait tugas akhir, namun akan mengganggu *flow* bacaan sekiranya dimasukkan ke dalam bacaan. Lampiran bisa saja berisi data-data tambahan, analisis tambahan, penjelasan istilah, tahapan-tahapan antara yang bukan menjadi fokus utama, atau pranala menuju halaman luar yang penting.

Subbab dari Lampiran 2

@todo

Isi subbab ini sesuai keperluan Anda. Anda bisa membuat lebih dari satu judul lampiran, dan tentunya lebih dari satu subbab.