UNIVERSITAS INDONESIA

APLIKASI *BIDIRECTIONAL ENCODER REPRESENTATIONS FROM TRANSFORMERS* UNTUK PEMERINGKATAN TEKS BAHASA INDONESIA

SKRIPSI

CARLES OCTAVIANUS
2006568613

FAKULTAS FAKULTAS MATEMATIKA DAN ILMU PENGATAHUAN ALAM
PROGRAM STUDI MATEMATIKA
DEPOK
JANUARI 2024

UNIVERSITAS INDONESIA

APLIKASI *BIDIRECTIONAL ENCODER REPRESENTATIONS FROM TRANSFORMERS* UNTUK PEMERINGKATAN TEKS BAHASA INDONESIA

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

JANUARI 2024

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	:	**3 Januari 2024**

HALAMAN PENGESAHAN

Skripsi ini diajukan oleh	:	
Nama	:	Carles Octavianus
NPM	:	2006568613
Program Studi	:	Matematika
Judul Skripsi	:	Aplikasi *Bidirectional Encoder Representations from Transformers* untuk Pemeringkatan Teks Bahasa Indonesia

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 :

KATA PENGANTAR

Segala Puji dan Syukur penulis panjatkan kepada Tuhan Yang Maha Esa ats diberikan anugerah dan kesempatan sehingga penulis dapat menyelesaikan skripsi yang berjudul ”Aplikasi *Bidirectional Encoder Representations from Transformers* untuk Pemeringkatan Teks Bahasa Indonesia”. Penulisan skripsi ini dilakukan dalam rangka memenuhi salah satu syarat kelulusan dan gelar Sarjana Matematika pada Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Indonesia. Penyusunan skripsi ini didasari atas semangat, usaha dan doa kepada-Nya. Dalam proses Penyusunan skripsi, penulis juga tidak lepas dari bantuan orang sekitar, baik berupa dukungan, bimbingan, dan doa yang telah diberikan. Penulis juga mengucapkan terima kasih sebesar-besarnya kepada:

1. Sarini Abdullah S.Si., M.Stats., Ph.D., selaku dosen pembimbing yang banyak memberikan arahan, saran dan bantuan dalam menyelesaikan skripsi ini.
2. Orang tua penulis yang selalu memberikan doa, kasih sayang, serta dukungan berupa moril maupun materiil yang tak terhingga.
3. Bapak dan Ibu dosen dan staf pengajar Matematika Universitas Indonesia yang telah mengajarkan penulis berbagai macam ilmu.
4. Anthony, Antonius yang selalu menemani, mendukung dan memberikan semangat selama penyusunan skripsi.
5. Teman-teman penulis selama perkuliahan, yaitu Nicholas, Bravy, Owen, Gladys.
6. Komunitas *Machine learning* yang berkontribusi menyediakan sumber daya secara gratis dan terbuka sehingga membantu penulis dalam penelitian ini, baik dari dasar teori hingga tahap implementasi.
7. Pihak-pihak yang sudah membantu penulis dalam melakukan penelitian ini, menyusun skripsi, dna mendukung dalam dunia perkuliahan baik secara langsung maupun tidak langsung.

Akhir kata, penulis memohon maaf atas kekurangan dalam pengerjaan dan penulisan skripsi ini. Penulis menyadari bahwa skripsi ini masih belum sempurna. Oleh karena itu, kritik dan saran yang bersifat membangun sangat penulis harapkan. Semoga peneletian ini dapat bermanfaat bagi pengembangan ilmu pengetahuan kedepannya dan bagi pihak-pihak terkait.

Depok, 3 Januari 2024

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:

Aplikasi *Bidirectional Encoder Representations from Transformers* untuk Pemeringkatan Teks Bahasa Indonesia

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 : 3 Januari 2024
Yang menyatakan

(Carles Octavianus)

ABSTRAK

Nama	:	Carles Octavianus
Program Studi	:	Matematika
Judul	:	Aplikasi *Bidirectional Encoder Representations from Transformers* untuk Pemeringkatan Teks Bahasa Indonesia
Pembimbing	:	Sarini Abdullah S.Si., M.Stats., Ph.D.

Peningkatan jumlah data teks digital membuat manusia membutuhkan mekanisme untuk mengembalikan teks yang efektif dan efisien. Salah satu mekanisme untuk mengembalikan teks adalah dengan pemeringkatan teks. Tujuan dari pemeringkatan teks adalah menghasilkan daftar teks yang terurut berdasarkan relevansinya dalam menanggapi permintaan kueri pengguna. Pada penelitian ini, penulis menggunakan *Bidirectional Encoder Representations from Transformers* (BERT) untuk membangun model pemeringkatan teks berbahasa Indonesia. Penggunaan BERT memberikan peningkatan kualitas pemeringkatan teks bila dibandingkan dengan model *baseline* BM25. Peningkatan kualitas pemeringkatan teks tersebut dapat dilihat dari nilai metrik *recriprocal rank* (RR), *recall* (R), dan *normalized discounted cumulative gain* (NDCG).

Kata kunci:
IndoBERT, representasi teks, sistem temu balik, skoring teks

ABSTRACT

Name	:	Carles Octavianus
Study Program	:	Mathematics
Title	:	Bidirectional Encoder Representations from Transformers Application for Text Ranking in Indonesian
Counselor	:	Sarini Abdullah S.Si., M.Stats., Ph.D.

The increasing amount of digital text data requires humans to have mechanisms for retrieving text effectively and efficiently. One mechanism for text retrieval is text ranking. The goal of text ranking is to generate a list of texts sorted based on their relevance in responding to user query requests. In this study, the author uses Bidirectional Encoder Representations from Transformers (BERT) to build a text ranking model for the Indonesian language. The use of BERT improves the quality of text ranking compared to the baseline BM25 model. The improvement in the quality of text ranking can be seen from the values of the reciprocal rank (RR), recall (R), and normalized discounted cumulative gain (NDCG) metrics.

Key words:
IndoBERT, text representation, information retrieval system, text scoring

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR KODE PROGRAM

DAFTAR LAMPIRAN

BAB 1
PENDAHULUAN

1.1 Latar Belakang

Dalam era informasi digital, peningkatan jumlah data teks digital membuat manusia kesulitan untuk dapat memproses informasi secara efektif dan efisien. Untuk dapat memproses informasi dalam data teks, tahap pertama adalah melakukan penyimpanan data teks tersebut dengan efisien. Setelah itu, untuk dapat mengakses informasi yang ada didalam data teks, diperlukan suatu mekanisme untuk mengembalikan teks yang relevan ketika diperlukan. Jumlah data teks yang makin banyak membuat mekanisme pengembalian informasi menjadi makin penting. Pada tahun 2005 saja, Yahoo! telah mengindeks lebih dari 19.2 milyar dokumen (Liu, 2011) untuk Yahoo! *search engine*. Tanpa adanya mekanisme untuk mengembalikan teks yang efektif dan efisien, informasi yang melimpah tersebut akan tidak berguna.

Salah satu mekanisme untuk mengembalikan teks yang relevan adalah dengan melakukan pemeringkatan teks. Tujuan dari pemeringkatan teks adalah menghasilkan daftar teks yang terurut berdasarkan relevansinya dalam menanggapi permintaan kueri pengguna (Lin, Nogueira, & Yates, 2020). Aplikasi paling umum dari pemeringkatan teks adalah mesin pencarian, di mana mesin pencari (juga disebut sistem temu balik) menghasilkan daftar teks yang terurut seperti, halaman web, makalah ilmiah, artikel berita, tweet, berdasarkan relevansi terhadap permintaan pengguna. Dalam konteks ini, teks yang relevan adalah teks yang memiliki topik sesuai permintaan pengguna dan memenuhi kebutuhan informasi pengguna.

Salah satu algoritma pemeringkatan teks adalah *Best Match* 25 (BM25). BM25 yang dikembangkan oleh Robertson, Walker, Jones, Hancock-Beaulieu, dan Gatford (1994) adalah metode pemeringkatan teks berdasarkan statistik kata-kata pada kueri dan teks. Kata-kata antara kueri dan teks yang *match* akan memberikan kontribusi terhadap skor relevansi dari kueri dan teks tersebut. Sampai sekarang BM25 masih digunakan sebagai metode pemeringkatan teks baik pada penelitian akademis – sebagai *baseline*– maupun pada sistem komersial (Lin et al., 2020). Salah satu penyedia layanan pemeringkatan teks

dengan algoritma BM25 adalah Elastic search (`https://www.elastic.co/`). Meskipun populer, BM25 tetap memiliki kekurangan. Kekurangan BM25 akan tampak ketika kata pada kueri dan teks tidak ada yang *match*. Hal ini sering terjadi ketika pengguna menggunakan kata-kata yang berbeda untuk mendeskripsikan kebutuhan informasinya dengan kata-kata yang digunakan pada teks yang relevan.

Untuk mengatasi kekurangan tersebut, pemeringkatan tidak hanya dilakukan pada informasi statistik dari kueri dan dokumen saja. Dengan memanfaatkan data tambahan seperti *log* atau metadata dari teks, model *machine learning* dapat digunakan untuk memeringkatkan teks. Penggunaan *machine learning* untuk memeringkatan biasa dikenal sebagai topik *learning to rank*. Contoh konkrit penggunaan *machine learning* untuk pemeringkatan teks dapat ditemukan pada penelitian Abdillah, Murfi, dan Satria (2015). Kekurangan penggunaan *machine learning* untuk pemeringkatan adalah jumlah fitur tambahan yang dibutuhkan cukup banyak untuk mengimbangi kekurangan dari BM25, dan fitur tersebut biasanya dibuat secara manual (Lin et al., 2020).

Batasan yang dialami model *machine learning* pada era *learning to rank*, diatasi dengan menggunakan *deep learning*. *Deep learning* merupakan model komputasional yang belajar melakukan suatu tugas dengan men-*fitting* model dengan data dalam jumlah yang banyak. Dalam konteks pemeringakatan teks, metode *deep learning* menarik perhatian dengan dua alasan utama: Pertama, *deep learning* mengatasi permaslahan *missmatch* pada BM25 dengan representasi fitur yang kontinu.Kedua, *deep learning* menghilangkan kebutuhan akan fitur yang dibuat secara manual – yang merupakan tantangan besar dalam membangun model pemeringkatan teks dengan *machine learning* klasik (Hofstätter, Althammer, Sertkan, & Hanbury, 2021).

Model *Bidirectional Encoder Representations from Transformers* (BERT) adalah model pra-latih *deep learning* yang dikembangkan oleh Devlin, Chang, Lee, dan Toutanova (2018) untuk permasalahan bahasa alami. BERT menggunakan *transformer* (Vaswani et al., 2017) sebagai arsitektur dasarnya. Penggunaan 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. BERT telah menjadi *state-of-the-art* untuk berbagai permasalahan pemrosesan bahasa alami seperti *question answering*, *named entity recognition*, *sentiment analysis*, dan lain-lain.

Model BERT juga dapat digunakan untuk pemeringkatan teks. Model BERT per-

tama kali digunakan untuk pemeringkatan teks dilakukan oleh Nogueira dan Cho (2019). Model BERT hasil penelitian Nogueira dan Cho (2019) digunakan sebagai *classifier* nilai relevansi antara kueri dan teks yang dilatih pada *dataset* MS MARCO *passage retrieval* (Nguyen et al., 2016). Model BERT yang digunakan sebagai *classifier* dikenal sebagai BERT_{CAT} (BERT CONCAT). Di lain sisi, model BERT juga dapat digunakan untuk menghasilkan representasi vektor dari kueri dan teks. Representasi vektor tersebut dapat digunakan untuk menghitung skor relevansi antara kueri dan teks dengan fungsi *similarity* seperti *cosine similarity* ataupun *dot product* seperti yang ditunjukkan oleh Karpukhin et al. (2020); Reimers dan Gurevych (2019). model BERT yang digunakan untuk menghasilkan representasi vektor biasa disebut sebagai BERT_{DOT} (BERT DOT)

Model-model BERT yang sudah siap digunakan untuk pemeringkatan teks sangat berlimpah pada bahasa Inggris. Hal ini dapat dilihat pada *repository* model huggingface `sentence-transformers`. Namun, untuk bahasa Indonesia, model-model BERT yang siap digunakan untuk pemeringkatan teks masih sangat terbatas dan tidak ada penelitian yang mendokumentasikan performa model tersebut. Oleh karena itu, penelitian ini bertujuan mengembangkan model BERT untuk pemeringkatan teks berbahasa Indonesia dan mengukur performa model tersebut pada *dataset* pemeringkatan teks berbahasa Indonesia.

1.2 Rumusan Permasalahan

Berdasarkan latar belakang pada Subbab 1.1, rumusan permasalahan yang ingin dibahas pada penelitian ini adalah sebagai berikut:

1. Bagaimana pengaplikasian model BERT untuk pemeringkatan teks berbahasa Indonesia?
2. Bagaimana kinerja model BERT pada setiap *dataset* yang digunakan bila dibandingkan dengan model *baseline* BM25?

1.3 Tujuan Penelitian

Berdasarkan rumusan permasalahan pada Subbab 1.2, tujuan dari penelitian ini adalah sebagai berikut:

1. Membangun dan melatih kembali *(fine tuning)* model BERT untuk pemeringkatan teks berbahasa Indonesia.

2. Membandingkan kinerja model BERT pada setiap *dataset* yang digunakan bila dibandingkan dengan model *baseline* BM25.

1.4 Metodologi Penelitian

Metodologi yang dilakukan pada peneletian ini adalah sebagai berikut:

1. Studi literatur

   Penelitian dimulai dengan sebuah studi literatur terkait model dan konsep yang perlu dipahami. Topik yang dipelajari antara lain, pemeringkatan teks, metrik untuk pemeringkatan teks, *transformer*, *Bidirectional Encoder Representations from Transformers* (BERT) untuk pemeringkatan teks dan *representation learning*. Studi literatur dijalankan dengan membaca buku dan penelitian terdahulu. Hasil dari studi ini digunakan sebagai landasan teori pada penelitian.

2. Pengumpulan data

   Dataset yang digunakan pada penelitian ini adalah *dataset* Mmarco *train set* bahasa Indonesia (Bonifacio, Campiotti, de Alencar Lotufo, & Nogueira, 2021) untuk melatih kembali (*fine tuning*) model BERT, *dataset* Mmarco *dev set* bahasa Indonesia (Bonifacio et al., 2021), MrTyDi *dev set* bahasa Indonesia (X. Zhang, Ma, Shi, & Lin, 2021), dan Miracl *dev set* bahasa Indonesia (X. Zhang et al., 2023) untuk menguji performa model yang dihasilkan.

3. Implementasi

   Implementasi dilakukan dengan bahasa pemrograman python. Tahapan implementasi terdiri atas, persiapan data, pelatihan model, dan pengevaluasian model yang dihasilkan.

4. Analisis hasil dan diskusi

   Penelitian difokuskan pada perbandingan kinerja model BERT yang dihasilkan dengan model *baseline* BM25. Analisis dilakukan dengan membandingkan nilai metrik *recriprocal rank* (RR), *recall* (R), dan *normalized discounted cumulative gain* (NDCG).

1.5 Batasan Permasalahan

Batasan-batasan permasalahan pada penelitian ini adalah sebagai berikut:

1. *Dataset* yang digunakan untuk melatih kembali (*fine tuning*) model BERT adalah *dataset* Mmarco *train set* bahasa Indonesia (Bonifacio et al., 2021).

2. *Dataset* yang digunakan untuk mengukur performa model adalah *dataset* Mmarco *dev set* bahasa Indonesia (Bonifacio et al., 2021) untuk *in-domain test* serta Mr-TyDi *dev set* bahasa Indonesia (X. Zhang et al., 2021), dan Miracl *dev set* bahasa Indonesia (X. Zhang et al., 2023) untuk *out-of-domain test*.

3. Kinerja model diamati dengan metrik *recriprocal rank* (RR), *recall* (R), dan *normalized discounted cumulative gain* (NDCG).

BAB 2
LANDASAN TEORI

2.1 Masalah Pemeringkatan Teks

Permasalahan pemeringkatan teks adalah permasalahan untuk menentukan urutan teks yang paling relevan dengan kueri q yang diberikan. Dalam bahasa yang lebih formal, diberikan kueri q dan himpunan teks terbatas $\mathcal{D} = \{d_1, d_2, ..., d_n\}$, keluaran yang diinginkan dari permasalahan ini adalah barisan teks $D_k = (d_{i_1}, d_{i_2}, ..., d_{i_k})$ yang merupakan k teks yang paling relevan dengan kueri q. Selain itu, biasanya nilai k akan lebih kecil dari banyaknya teks yang ada, sehingga permasalahan pemeringkatan teks 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 teks yang ingin di-*retreive* oleh sebuah sistem pemeringkatan teks. 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 teks dari *file* korpus. performa dari sistem pemeringkatan teks akan diukur dengan mengambil k teks dari *file* korpus untuk setiap kueri pada *file* kueri. 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 teks pada *file* korpus. Pada *file* judgements terdapat 3 kolom, yaitu id kueri, id teks, dan relevansi r antara kueri dan teks tersebut. Pasangan $(\text{kueri}, \text{teks})$ yang relevan akan memiliki nilai $r > 0$ dan nilai r yang makin besar menunjukkan tingkat relevansi yang makin tinggi. Selain itu, pasangan $(\text{kueri}, \text{teks})$ yang tidak relevan akan memiliki nilai $r = 0$ dan biasanya pasangan $(\text{kueri}, \text{teks})$ 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*.

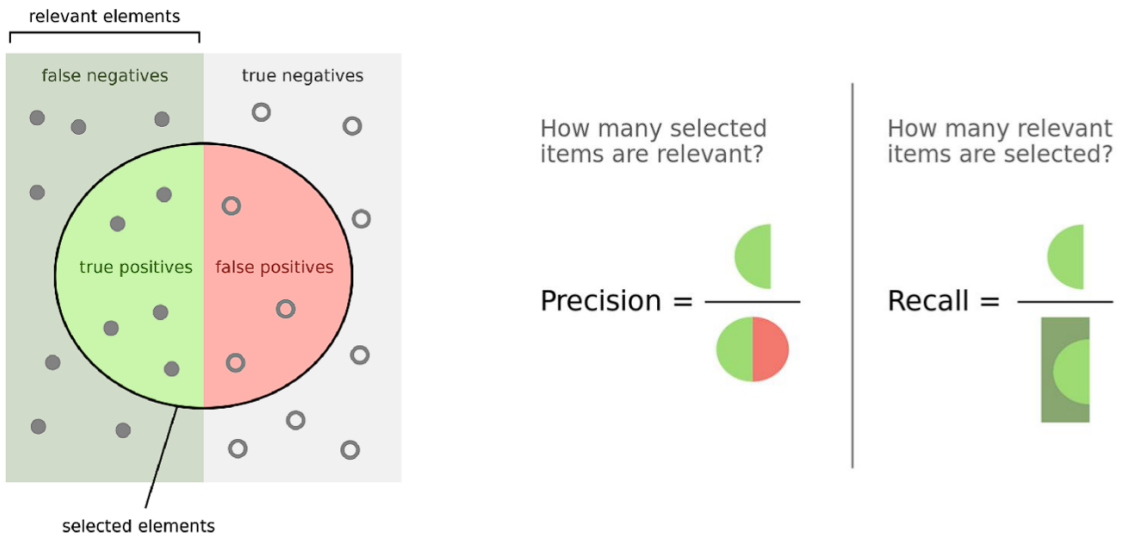
Tabel 2.3: Potongan *file* judgements *dataset* Miracl.

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 teks 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 teks relevan yang diambil oleh sistem terhadap seluruh teks yang relevan dengan kueri q. Sedangkan nilai presisi dihitung sebagai rasio teks relevan yang diambil oleh sistem terhadap seluruh teks yang diambil oleh sistem.
Sumber: (Hofstätter et al., 2021).

Presisi dan *recall* adalah metrik yang paling sederhana untuk mengukur kemampuan dari suatu sistem pemeringkatan teks. *Recall* mengukur kemampuan sistem dalam mengembalikan semua teks yang relevan dengan kueri q dari himpunan teks $\mathcal{D}$, sedangkan presisi mengukur kemampuan sistem dalam mengembalikan teks yang relevan dengan kueri q dari himpunan teks $\mathcal{D}$. Untuk suatu kueri q, kumpulan teks $\mathcal{D} = \{d_1, d_2, ..., d_n\}$, dan barisan k teks 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 2.2.

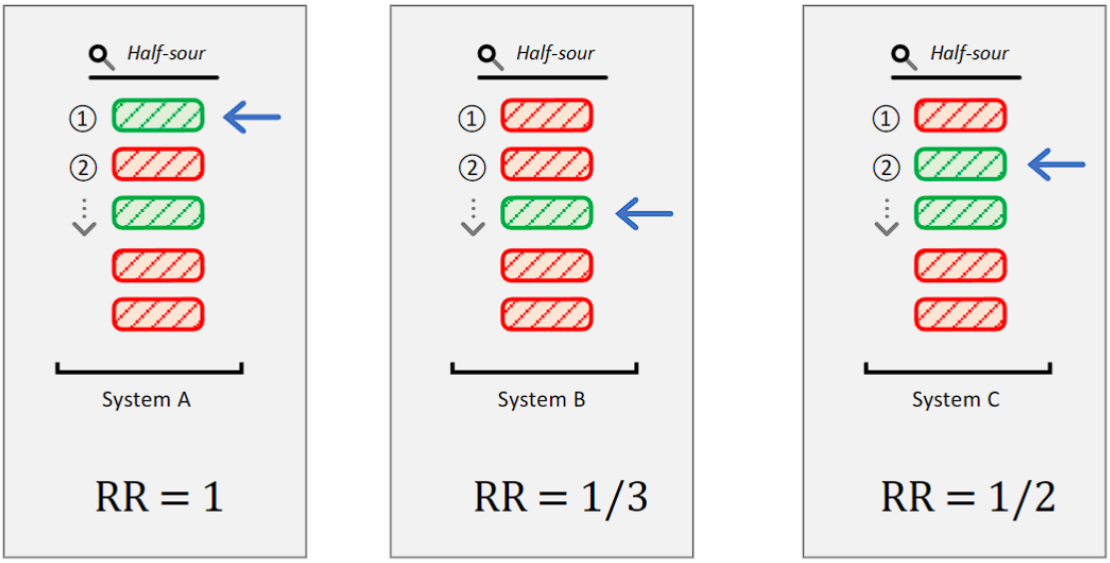
$$\text{recall}(q, D_k)@\text{k} = \frac{\sum_{d \in D_k} \text{rel}(q,d)}{\sum_{d \in \mathcal{D}} \text{rel}(q,d)} \in [0,1], \tag{2.1}$$

$$\text{precision}(q, D_k)@\text{k} = \frac{\sum_{d \in D_k} \text{rel}(q,d)}{|D_k|} \in [0,1], \tag{2.2}$$

$$\text{rel}(q,d) = \begin{cases} 1 & \text{jika } r > 1 \\ 0 & \text{jika } r = 0 \end{cases}. \tag{2.3}$$

Sebagai contoh, jika terdapat 10 teks yang relevan dengan kueri q, dan sistem mengembalikan $k = 100$ teks, namun hanya terdapat 5 teks 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* (RR). Kotak berwarnakan hijau menunjukkan teks yang relevan dengan kueri q dan kotak berwarna merah menunjukkan teks yang tidak relevan dengan kueri q. Nilai RR pada sistem A, B, dan C berturut-turut adalah 1, 0.33, dan 0.5 karena posisi dari teks yang relevan pertama adalah 1, 3, dan 2.
Sumber: (Hofstätter et al., 2021).

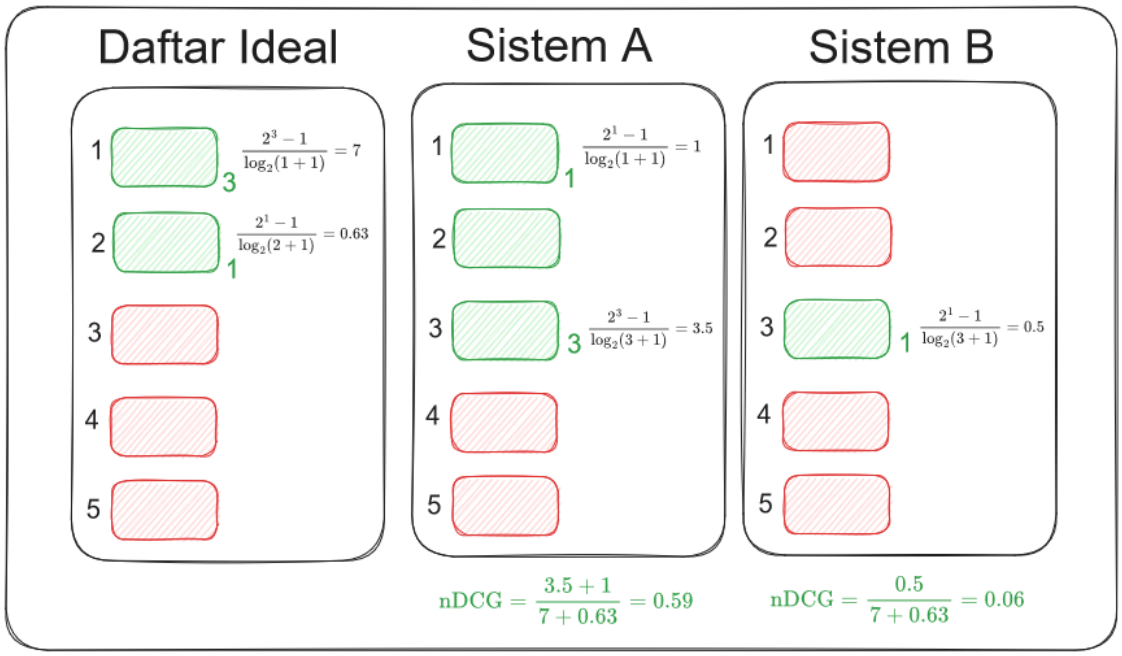
Metrik lainnya yang sering digunakan untuk mengukur performa sistem pemeringkatan adalah *reciprocal rank* (RR). Metrik RR menitikberatkan pada peringkat dari teks relevan pertama dengan kueri q. Persamaan 2.4 hingga Persamaan 2.5 menunjukkan cara menghitung RR dari suatu kueri q dan barisan k teks yang diambil oleh sistem.

$$\text{RR}(q,D_k)@\text{k} = \begin{cases} \frac{1}{\text{FirstRank}(q,D_k)} & \text{jika } \exists d \in D_k \text{ dengan } \text{rel}(q,d) = 1 \\ 0 & \text{jika } \forall d \in D_k,\ \text{rel}(q,d) = 0 \end{cases} \in [0,1], \quad (2.4)$$

$$\text{FirstRank}(q,D_k) = \text{posisi teks relevan pertama } d \in D_k \text{ dengan } \text{rel}(q,d) = 1. \quad (2.5)$$

Gambar 2.2 mengilustrasikan metrik RR. Pada gambar tersebut, nilai RR dari sistem A adalah 1 $(\frac{1}{1})$ karena posisi dari teks 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 teks yang relevan pertama adalah 3 dan 2. Selain itu, jika tidak terdapat teks 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: Ilustrasi perhitungan NDCG. Kotak berwarna hijau menunjukkan teks yang relevan dengan kueri q dan kotak berwarna merah menunjukkan teks yang tidak relevan dengan kueri q serta nilai disebelah kotak berwarnakan hijau menunjukkan *judgements* r. Nilai NDCG dari sistem A, B, adalah rasio antara DCG dari sistem tersebut dengan DCG dari sistem ideal.
Sumber: (Hofstätter et al., 2021), telah diolah kembali.

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 r yang tak biner. Persamaan 2.6 hingga Persamaan 2.9 menunjukkan cara menghitung NDCG dari suatu kueri q dan barisan k teks yang diambil oleh sistem.

$$\text{nDCG}(q, D_k)@\text{k} = \frac{\text{DCG}(q, D_k)@\text{k}}{\text{DCG}(q, D_k^{\text{ideal}})@\text{k}} \in [0, 1], \tag{2.6}$$

$$\text{DCG}(q, D_k)@\text{k} = \sum_{d \in D_k} \frac{2^{\text{rel}(q,d)} - 1}{\log_2(\text{rank}(d, D_k) + 1)}, \tag{2.7}$$

$$\text{rank}(d, D_k) = \text{Posisi } d \text{ dalam } D_k, \tag{2.8}$$

$$\text{rel}(q, d) = r. \tag{2.9}$$

Perhitungan *discounted cumulative gain* (DCG) pada Persamaan 2.7 dapat dijelaskan menjadi dua faktor berikut:

1. Faktor $2^{\text{rel}(q,d)} - 1$ menunjukkan bahwa teks yang lebih relevan akan memiliki nilai yang lebih tinggi dari teks yang kurang relevan untuk posisi teks yang sama.

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

Nilai dari NDCG pada Persamaan 2.6 adalah nilai DCG pada barisan teks D_k yang dinormalisasi oleh nilai DCG pada barisan teks ideal D_k^{ideal}. Barisan teks ideal D_k^{ideal} adalah barisan teks yang diurutkan berdasarkan relevansinya dengan kueri q.

2.2 Pemeringkatan Teks dengan Statistik

Untuk mengambil k teks dari kumpulan $\mathcal{D}$ diperlukan suatu fungsi skor $\text{score}(q,d,\mathcal{D})$ yang mengukur relevansi antara kueri q dan teks d. Dengan mencari skor antara q terhadap semua teks pada $\mathcal{D}$, barisan teks $D_k = (d_{i_1}, d_{i_2}, \ldots, d_{i_k})$ dipilih sehingga $\text{score}(q,d_{i_1},\mathcal{D}) \geq \text{score}(q,d_{i_2},\mathcal{D}) \geq \cdots \geq \text{score}(q,d_{i_k},\mathcal{D})$ adalah k teks dengan skor tertinggi. Bagian ini menjelaskan dua fungsi skor statistik 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 dan tingkat *rarity* kata dalam kumpulan teks. Selanjutnya, Subbab 2.2.2 membahas fungsi skor statistik yang menjadi *baseline* dalam penelitian ini.

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

Fungsi skor TF-IDF adalah fungsi skor statistik yang menghitung $\text{score}(q,d,\mathcal{D})$ antara kueri q dan teks d dengan menghitung frekuensi kemunculan kata dan tingkat *rarity* kata. Untuk suatu kueri q, misalkan $T_q = \{t_1, t_2, \ldots, t_{L_1}\}$ adalah himpunan kata yang terdapat pada kueri q. Misalkan juga $T_d = \{t_1, t_2, \ldots, t_n\}$ adalah himpunan kata yang terdapat pada teks d. Nilai skor antara q dan d diberikan oleh persamaan Persamaan 2.10 sampai

Persamaan 2.15.

$$\text{score}(q,d,\mathcal{D}) = \sum_{t \in T_q \cap T_d} \text{TF-IDF}(t,d,\mathcal{D}) \tag{2.10}$$

$$\text{TF-IDF}(t,d,\mathcal{D}) = \text{tf}(t,d) \times \text{idf}(t,\mathcal{D}) \tag{2.11}$$

$$\text{tf}(t,d) = \frac{\text{Count}(t,d)}{|d|} \tag{2.12}$$

$$\text{Count}(t,d) = \text{jumlah kemunculan } t \text{ dalam } d \tag{2.13}$$

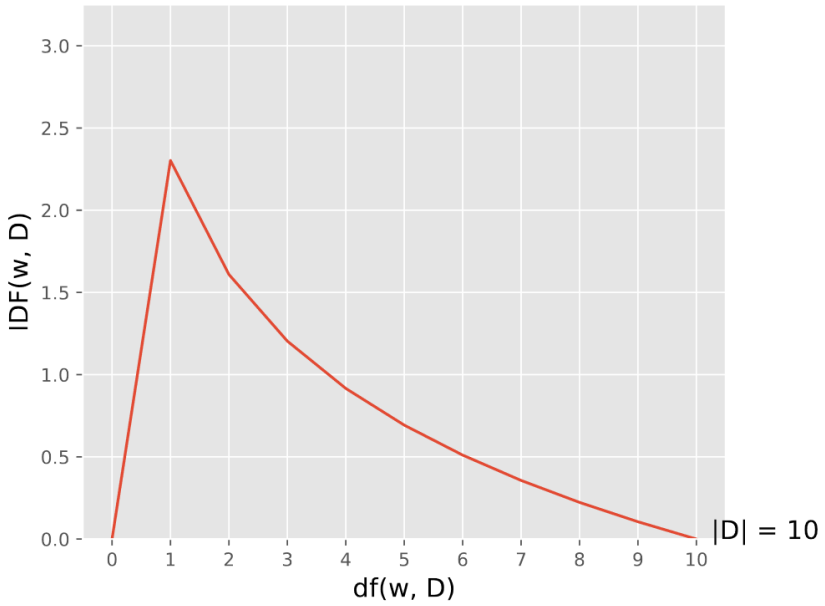
$$\text{idf}(t,\mathcal{D}) = \begin{cases} \log_2\left(\frac{|\mathcal{D}|}{\text{df}(t,\mathcal{D})}\right) & \text{jika df}(t,\mathcal{D}) > 0 \\ 0 & \text{jika df}(t,\mathcal{D}) = 0 \end{cases} \tag{2.14}$$

$$\text{df}(t,\mathcal{D}) = \text{jumlah teks pada } \mathcal{D} \text{ yang mengandung } t \tag{2.15}$$

Skor untuk pasangan (q,d) dihitung dengan menjumlahkan skor TF-IDF dari setiap kata yang terdapat pada kueri q dan teks 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 2.10 dapat dijelaskan menjadi dua faktor utama berikut:

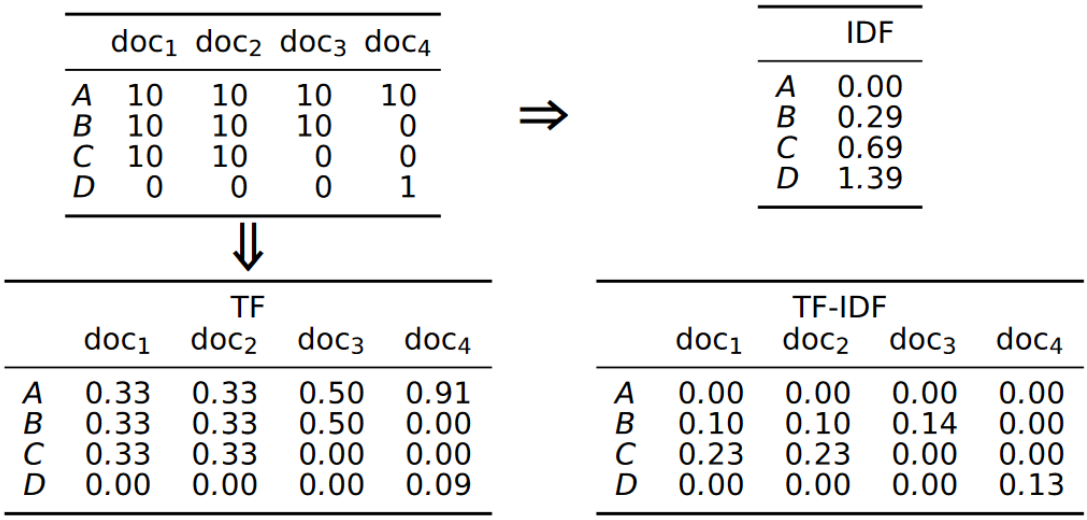
1. Faktor $\text{tf}(t,d)$ menunjukkan bahwa nilai TF-IDF meningkat seiring dengan bertambahnya frekuensi kemunculan kata t pada teks d.

2. Faktor $\text{idf}(t,\mathcal{D})$ menunjukkan bahwa nilai TF-IDF meningkat seiring dengan *rarity* dari kata t pada himpunan teks $\mathcal{D}$. Akibatnya, kata yang jarang muncul pada himpunan teks $\mathcal{D}$ yang muncul pada suatu teks tertentu akan menghasilkan skor yang tinggi. Sementara itu, kata-kata yang sering muncul pada koleksi teks $\mathcal{D}$ memiliki nilai *downgraded*. Gambar 2.4 menunjukkan grafik dari fungsi idf.



Gambar 2.4: Grafik dari fungsi idf. Nilai idf menurun seiring dengan bertambahnya nilai df$(t, \mathcal{D})$.
Sumber: (Potts et al., 2023).

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 teks dan bisa diabaikan. Di sisi lain, kata-kata yang muncul secara berlebihan dalam satu teks tetapi jarang muncul dalam teks lainnya akan menghasilkan nilai 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 2.5 menunjukkan contoh perhitungan skor TF-IDF untuk suatu kumpulan teks.



Gambar 2.5: Ilustrasi perhitungan skor TF-IDF untuk suatu kumpulan teks.
Sumber: (Potts et al., 2023).

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 2.17) daripada $\text{tf}(q,d)$ (Persamaan 2.11). 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 2.16 hingga Persamaan 2.20 menunjukkan cara menghitung skor relevansi dari suatu kueri q dan teks d.

$$\text{score}(q,d,\mathcal{D}) = \sum_{t \in T_q \cap T_d} \text{BM25}(t,d,\mathcal{D}), \tag{2.16}$$

$$\text{BM25}(t,d,\mathcal{D}) = \text{idf}_{\text{BM25}}(t,\mathcal{D}) \times \text{score}_{\text{BM25}}(q,d,\mathcal{D}), \tag{2.17}$$

$$\text{score}_{\text{BM25}}(t,d) = \frac{\text{tf}(t,d) \times (k_1+1)}{\text{tf}(t,d) + k_1 \times (1 - b + b \times \frac{|d|}{\text{avgdl}})}, \tag{2.18}$$

$$\text{idf}_{\text{BM25}}(t,\mathcal{D}) = \log\left(1 + \frac{|\mathcal{D}| - \text{df}(t,\mathcal{D}) + 0.5}{\text{df}(t,\mathcal{D}) + 0.5}\right), \tag{2.19}$$

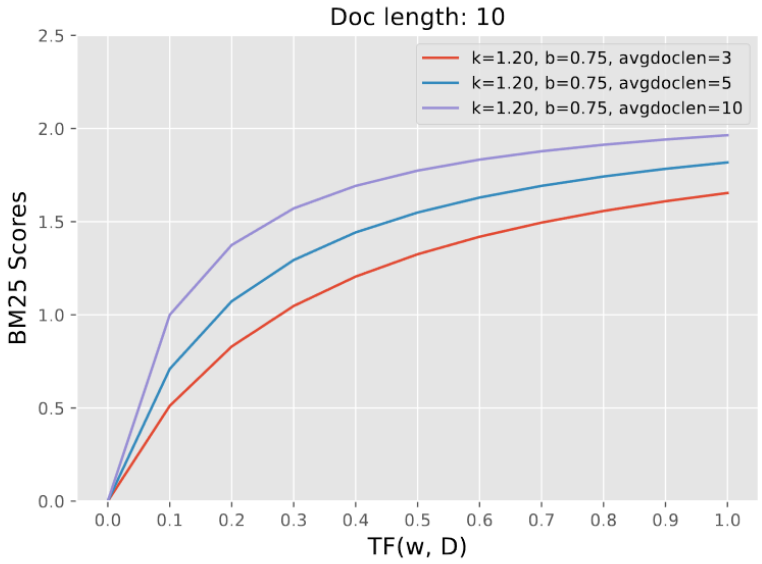
$$\text{avgdl} = \text{rata-rata panjang teks pada koleksi } \mathcal{D}. \tag{2.20}$$

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

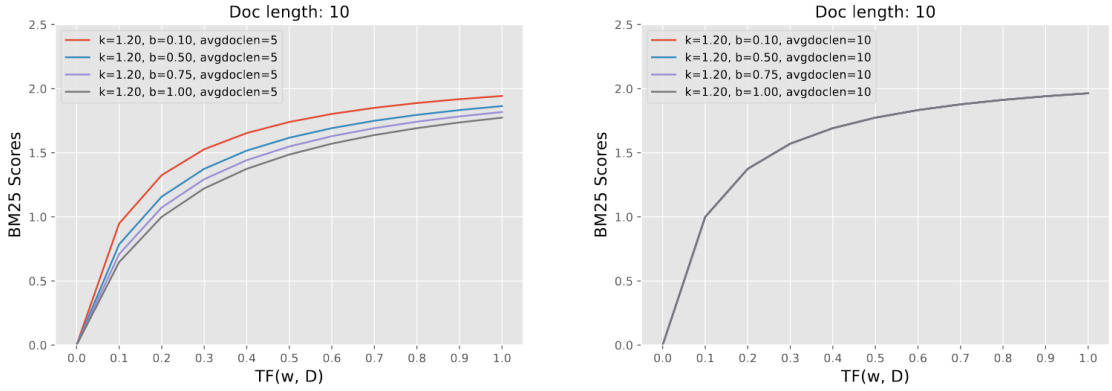
1. Faktor $\frac{|\mathrm{d}|}{\mathrm{avgdl}}$ pada $\frac{\mathrm{tf}(t,d)\times(k_1+1)}{\mathrm{tf}(t,d)+k_1\times\left(1-b+b\times\frac{|d|}{\mathrm{avgdl}}\right)}$ men-*penalize* skor pada teks yang panjangnya lebih besar dari rata-rata panjang teks pada himpunan teks $\mathcal{D}$. Gambar 2.6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

2. Nilai b menentukan seberapa besar efek dari faktor $\frac{|d|}{\mathrm{avgdl}}$ terhadap skor yang dihasilkan. Gambar 2.7 menunjukkan efek dari perbedaan nilai b terhadap skor yang dihasilkan. Untuk $\frac{|d|}{\mathrm{avgdl}} = 1$, faktor b tidak memiliki pengaruh terhadap skor. Nilai b yang umum dipilih berada pada rentang $[0.5, 0.8]$ (Hofstätter et al., 2021).

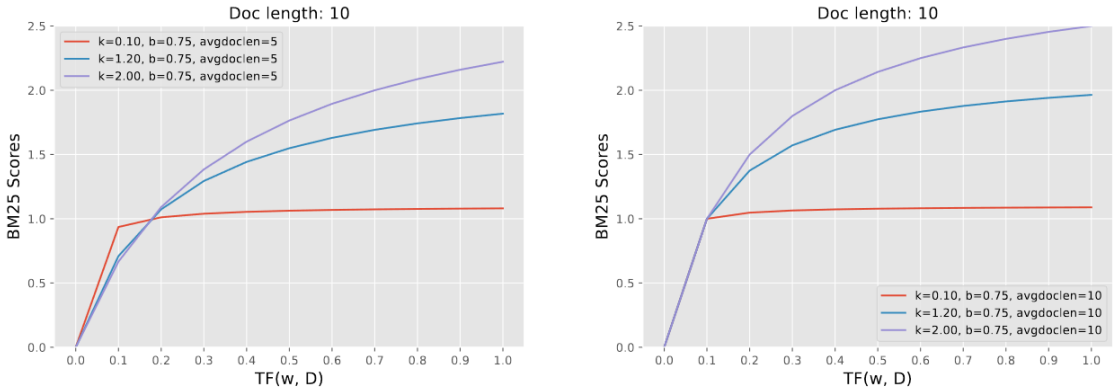
3. Nilai k_1 men-*penalize* kemunculan kata t pada teks d yang berlebih. Gambar 2.8 menunjukkan efek dari perbedaan nilai k_1 terhadap skor yang dihasilkan. untuk nilai k_1 yang ekstrim, nilai $\mathrm{score}_{\mathrm{BM25}}(t,d)$ hanya menjadi indikator saja dari kemunculan kata t pada teks d. Nilai k_1 yang umum dipilih berada pada rentang $[1.2, 2.0]$ (Hofstätter et al., 2021).



Gambar 2.6: Kumpulan grafik dari fungsi $\mathrm{score}_{\mathrm{BM25}}(t,d)$ dengan perbedaan nilai avgdl.
Sumber: (Potts et al., 2023).

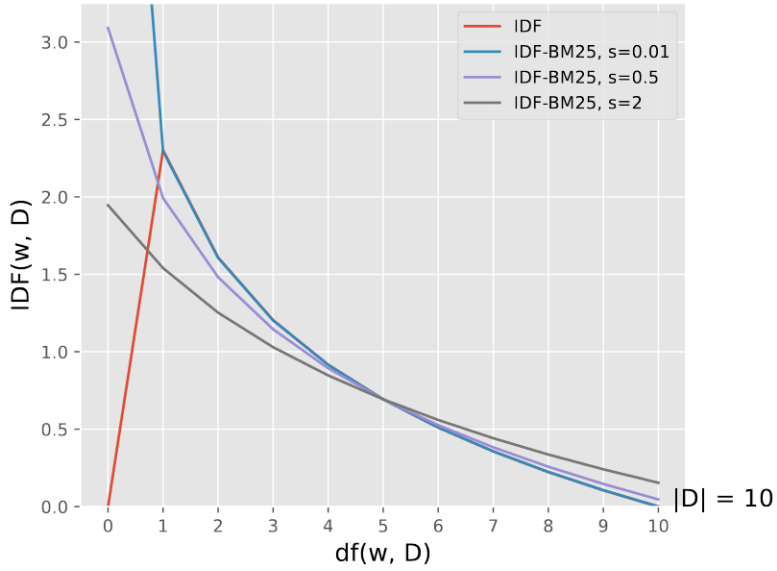


Gambar 2.7: Kumpulan grafik dari fungsi $\text{score}_{\text{BM25}}(t,d)$ dengan perbedaan nilai b.
Sumber: (Potts et al., 2023).



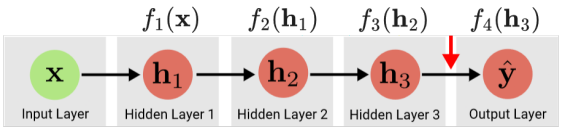
Gambar 2.8: Kumpulan grafik dari fungsi $\text{score}_{\text{BM25}}(t,d)$ dengan perbedaan nilai k_1.
Sumber: (Potts et al., 2023).

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 teks $\mathcal{D}$ – semata-mata untuk konsistensi dengan asumsi bahwa kata t yang tidak muncul pada himpunan teks $\mathcal{D}$ memiliki nilai idf (*rarity*) 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.



Gambar 2.9: Perbedaan antara idf_{BM25} dan idf.
Sumber: (Potts et al., 2023).

2.3 *Deep Learning*



Gambar 2.10: Ilustrasi dari *Directed Acyclic Graph* (DAG) pada arsitektur *deep learning feed-forward neural network* (FFN).
Sumber: (Geiger et al., 2022), telah diolah kembali.

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* $\hat{\mathbf{y}}$ melalui serangkaian fungsi f_1, f_2, f_3, f_4 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_4 adalah *output* dari model. Model pada

Gambar 2.10 dapat ditulis sebagai Persamaan 2.21.

$$\hat{\mathbf{y}} = f_{\text{model}}(\mathbf{x};\boldsymbol{\theta}) = f_4(f_3(f_2(f_1(\mathbf{x};\boldsymbol{\theta}_1);\boldsymbol{\theta}_2);\boldsymbol{\theta}_3);\boldsymbol{\theta}_4), \tag{2.21}$$

dengan $\boldsymbol{\theta} = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_3, \boldsymbol{\theta}_4\}$ adalah parameter dari model.

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* (parameter yang dipilih prior proses pelatihan dilakukan) 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 2.22 hingga Persamaan 2.24 menunjukkan arsitektur MLP untuk permasalahan regresi dengan L *layer* dan fungsi aktivasi ϕ.

$$f_{\text{model}}(\mathbf{x};\boldsymbol{\theta}) = f_L(f_{L-1}(\ldots f_1(\mathbf{x}))\ldots), \tag{2.22}$$

$$f_L(\mathbf{x}) = \mathbf{x}\mathbf{W}_L + \mathbf{b}_L \in \mathbb{R}^{d_L}, \tag{2.23}$$

$$f_l(\mathbf{x};\mathbf{W}_l,\mathbf{b_l}) = \phi(\mathbf{x}\mathbf{W}_l + \mathbf{b}_l) \in \mathbb{R}^{d_l}, \quad l = 1,2,\ldots,L-1, \tag{2.24}$$

dengan keterangan sebagai berikut:

$\phi(\mathbf{x}) =$ fungsi aktivitasi non-linear,

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

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

$\mathbf{b}_l =$ vektor bias $\in \mathbb{R}^{d_l}$.

Untuk Permasalahan Klasifikasi Biner dengan *input* $\mathbf{x} \in \mathbb{R}^{d_0}$ dan *output* $y \in \{0,1\}$, Persamaan 2.25 hingga Persamaan 2.29 menunjukkan arsitektur MLP untuk permasa-

lahan klasifikasi biner.

$$f_{\text{model}}(\mathbf{x};\boldsymbol{\theta}) = f_L(f_{L-1}(\dots f_1(\mathbf{x}))\dots), \tag{2.25}$$

$$f_L(\mathbf{x}) = \sigma(\mathbf{x}\mathbf{W}_L + \mathbf{b}_L), \tag{2.26}$$

$$\sigma(x) = \frac{1}{1+e^x} \in (0,1), \tag{2.27}$$

$$\text{decision}(\mathbf{x};\boldsymbol{\theta}) = \begin{cases} 1 & \text{jika } f_{\text{model}}(\mathbf{x};\boldsymbol{\theta}) \geq \text{threshold} \\ 0 & \text{jika } f_{\text{model}}(\mathbf{x};\boldsymbol{\theta}) < \text{threshold}, \end{cases} \tag{2.28}$$

$$\text{threshold} \in [0,1]. \tag{2.29}$$

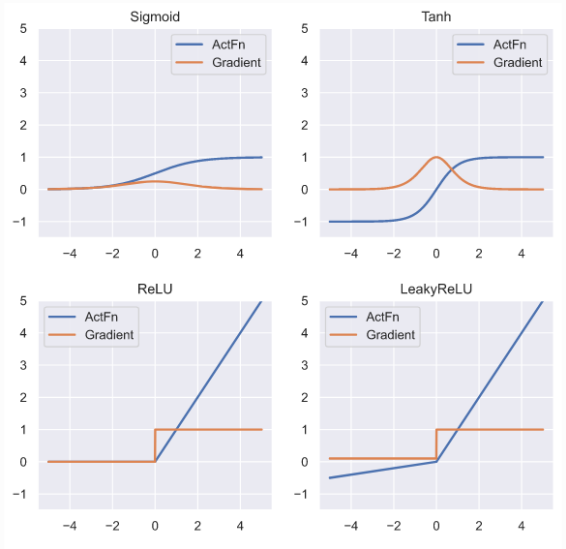
Perbedaan utama antara MLP untuk permasalahan regresi dan klasifikasi adalah fungsi aktivasi pada *output layer*. Pada permasalahan regresi, fungsi aktivasi pada *output layer* adalah fungsi identitas, sedangkan pada permasalahan klasifikasi, fungsi aktivasi pada *output layer* adalah fungsi *sigmoid*. 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 2.29 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: Grafik dari fungsi aktivasi pada Tabel 2.4 dan turunannya.
Sumber: (Lippe, 2022).

2.3.3 Fungsi *Loss*

Misalkan $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$ 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 2.38. Penurunan fungsi *loss* BCE dengan mengikuti prinsip MLE yang akan dijelaskan pada bagian berikut.

Misalkan $y_i \mid \mathbf{x}$ mengikuti distribusi bernoulli dengan parameter $\mathrm{p} = f_{\text{model}}(\mathbf{x}; \boldsymbol{\theta})$ yang saling independen antara satu sama lainnya. Persamaan 2.30 menunjukkan definisi dari $y_i \mid \mathbf{x}$.

$$y_i \mid \mathbf{x} \overset{\text{iid}}{\sim} \text{Bernoulli}(f_{\text{model}}(\mathbf{x}; \boldsymbol{\theta})), \tag{2.30}$$

$$p(y_i \mid \mathbf{x}) = f_{\text{model}}(\mathbf{x}; \boldsymbol{\theta})^{y_i} (1 - f_{\text{model}}(\mathbf{x}; \boldsymbol{\theta}))^{1-y_i}. \tag{2.31}$$

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

$$\mathcal{L}(\boldsymbol{\theta}) = \prod_{i=1}^{N} p(y_i \mid \mathbf{x}_i; \boldsymbol{\theta}). \tag{2.32}$$

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

$$\boldsymbol{\theta}_{\text{MLE}} = \arg\max_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}). \tag{2.33}$$

Untuk mempermudah perhitungan, fungsi *likelihood* diubah menjadi negatif *log-likelihood* $\ell(\boldsymbol{\theta})$, sehingga permasalahan optimasi dapat ditulis seperti Persamaan 2.34 hingga Persamaan 2.36.

$$\ell(\boldsymbol{\theta}) = -\log \mathcal{L}(\boldsymbol{\theta}), \tag{2.34}$$

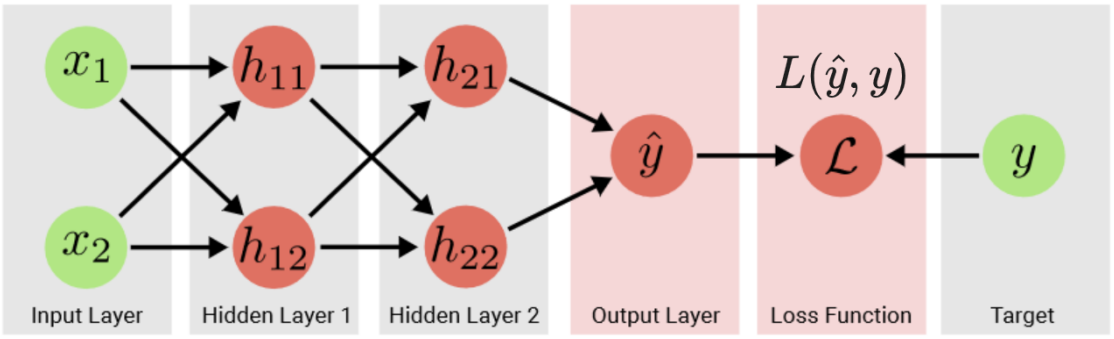
$$= -\sum_{i=1}^{N} \log \left(p(y_i \mid \mathbf{x}_i; \boldsymbol{\theta}) \right), \tag{2.35}$$

$$\boldsymbol{\theta}_{\text{MLE}} = \arg\min_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}). \tag{2.36}$$

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 2.38. Gambar 2.12 mengilustrasikan *directed acyclic graph* (DAG) dari model ketika proses pelatihan dilakukan.

$$\boldsymbol{\theta}_{\text{MLE}} = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} \underbrace{-y_i \log \left(f_{\text{model}}(\mathbf{x}_i; \boldsymbol{\theta}) \right) - (1 - y_i) \log \left(1 - f_{\text{model}}(\mathbf{x}_i; \boldsymbol{\theta}) \right)}_{\text{Binary Cross Entropy Loss } L(y_i, f_{\text{model}}(\mathbf{x}_i; \boldsymbol{\theta}))}, \tag{2.37}$$

$$L(y_i, f_{\text{model}}(\mathbf{x}_i; \boldsymbol{\theta})) = -y_i \log \left(f_{\text{model}}(\mathbf{x}_i; \boldsymbol{\theta}) \right) - (1 - y_i) \log \left(1 - f_{\text{model}}(\mathbf{x}_i; \boldsymbol{\theta}) \right). \tag{2.38}$$



Gambar 2.12: Ilustrasi dari *Directed Acyclic Graph* (DAG) pada model *deep learning* ketika proses pelatihan dilakukan.
Sumber: (Geiger et al., 2022), telah diolah kembali.

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 *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 $\boldsymbol{\theta}$ di-*update* secara iteratif dengan mengikuti arah negatif dari *gradient* $\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta})$ yang menunjukkan arah dari penurunan fungsi *loss* $\ell(\boldsymbol{\theta})$. Untuk kumpulan data $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$, Persamaan 2.39 menunjukkan algoritma *gradient descent* untuk mencari nilai $\boldsymbol{\theta}$.

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \frac{1}{n} \sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} L(y_i, f_{\text{model}}(\mathbf{x}_i; \boldsymbol{\theta}^{(t)})), \tag{2.39}$$

dengan $\eta \in \mathbb{R}^+$ adalah *learning rate* yang menentukan seberapa besar perubahan pada $\boldsymbol{\theta}$ pada setiap iterasi.

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 ma-

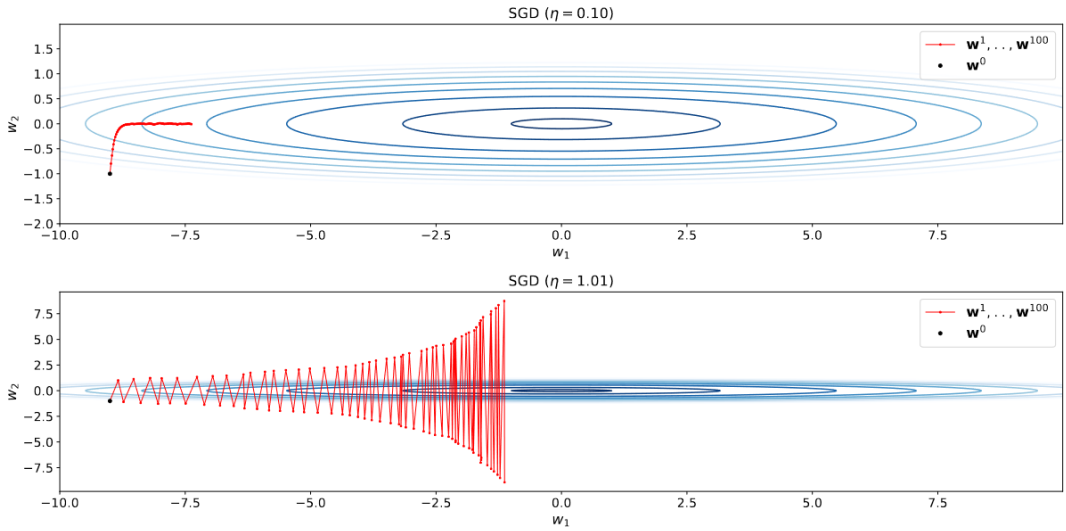
hal dan diperlukan memori yang besar untuk menyimpan gradien dari semua data pada *dataset* latih. Untuk mengatasi masalah tersebut, digunakan metode *stochastic gradient descent* (SGD) dimana setiap *update* dari parameter $\boldsymbol{\theta}$ dihitung dengan mengambil rata-rata *gradient* dari sebagian data pada *dataset* $\mathcal{B} \subseteq \mathcal{D}$. Persamaan 2.42 menunjukkan algoritma *stochastic gradient descent*.

$$\mathcal{B} = \{(\mathbf{x}_{i_1}, y_{i_1}), (\mathbf{x}_{i_2}, y_{i_2}), \ldots, (\mathbf{x}_{i_b}, y_{i_b})\} \subseteq \mathcal{D}, |\ \mathcal{B}\ | \ll |\ \mathcal{D}\ |, \tag{2.40}$$

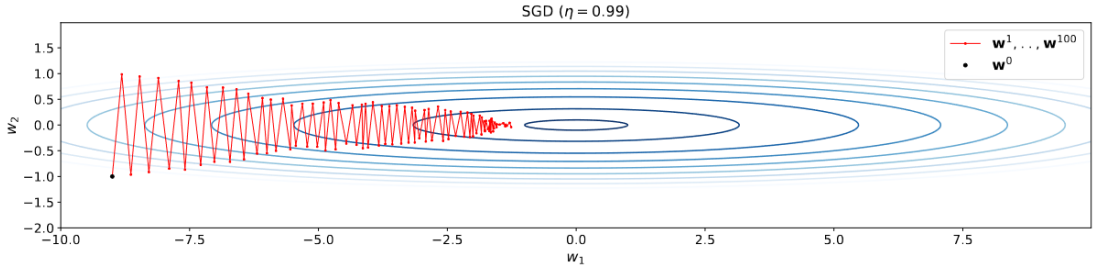
$$\nabla_{\boldsymbol{\theta}} L_{\mathcal{B}}(\boldsymbol{\theta}) = \frac{1}{b} \sum_{i=1}^{b} \nabla_{\boldsymbol{\theta}} L(y_i, f_{\text{model}}(\mathbf{x}_i; \boldsymbol{\theta})), \tag{2.41}$$

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}^{(t)}). \tag{2.42}$$

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: Seratus iterasi pertama dari pembaruan parameter $\boldsymbol{\theta} = \{w_1, w_2\}$ dengan *learning rate* yang terlalu kecil dan terlalu besar.
Sumber: (Geiger et al., 2022).



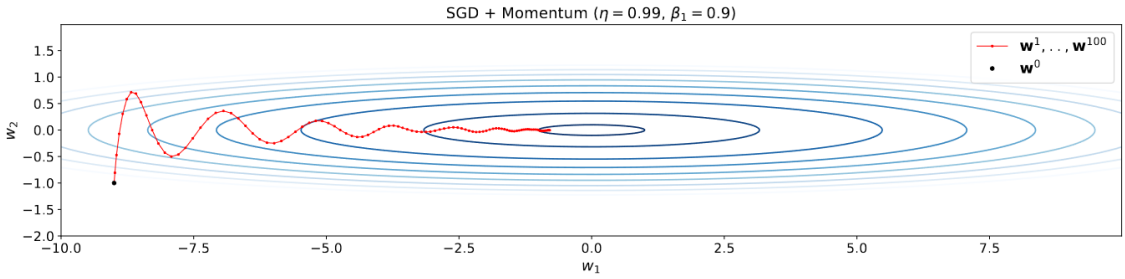
Gambar 2.14: Seratus iterasi pertama dari pembaruan parameter $\boldsymbol{\theta} = \{w_1, w_2\}$ dengan *learning rate* yang baik.
Sumber: (Geiger et al., 2022).

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 $\boldsymbol{\theta}$ adalah *exponential moving average* dari gradien pada iterasi sekarang dan gradien pada iterasi sebelumnya. Persamaan 2.43 menunjukkan algoritma *stochastic gradient descent* dengan momentum dan Gambar 2.15 mengilustrasikan pembaruan parameter $\boldsymbol{\theta}$ dengan momentum.

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \mathbf{m}^{(t+1)}, \tag{2.43}$$

$$\mathbf{m}^{(t+1)} = \beta_1 \mathbf{m}^{(t)} + (1 - \beta_1) \nabla_{\boldsymbol{\theta}} L_{\mathcal{B}}(\boldsymbol{\theta}^{(t)}), \tag{2.44}$$

dengan $\beta_1 \in [0, 1)$ adalah *momentum* yang mengatur seberapa besar pengaruh gradien pada iterasi sebelumnya terhadap gradien pada iterasi sekarang.



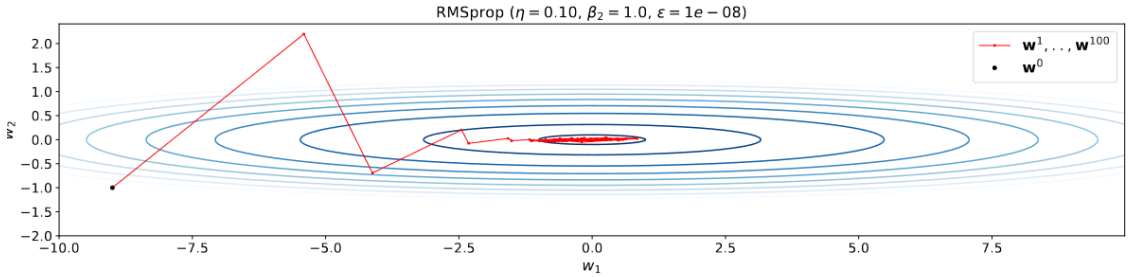
Gambar 2.15: Ilustrasi dari pembaruan parameter $\boldsymbol{\theta} = \{w_1, w_2\}$ dengan *stochastic gradient descent* dengan momentum.
Sumber: (Geiger et al., 2022).

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 2.45 menunjukkan algoritma *stochastic gradient descent* dengan *adaptive learning rate* dan Gambar 2.16 menngilustrasikan pembaruan parameter $\boldsymbol{\theta}$ dengan *adaptive learning rate*.

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \frac{\eta \nabla_{\boldsymbol{\theta}} L_{\mathcal{B}}(\boldsymbol{\theta}^{(t)})}{\sqrt{\mathbf{v}^{(t+1)}} + \varepsilon}, \tag{2.45}$$

$$\mathbf{v}^{(t+1)} = \beta_2 \mathbf{v}^{(t)} + (1-\beta_2)\left(\nabla_{\boldsymbol{\theta}} L_{\mathcal{B}}(\boldsymbol{\theta}^{(t)}) \odot \nabla_{\boldsymbol{\theta}} L_{\mathcal{B}}(\boldsymbol{\theta}^{(t)})\right), \tag{2.46}$$

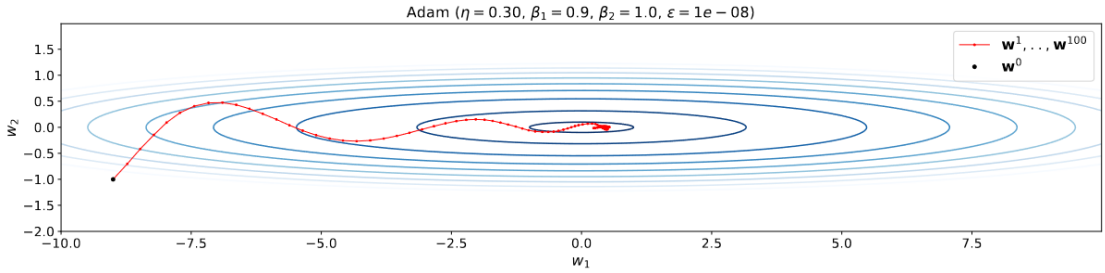
dengan $\beta_2 \in [0,1)$ dan $\odot$ adalah operasi perkalian *element-wise* antara dua matriks atau vektor.



Gambar 2.16: Ilustrasi dari pembaruan parameter $\boldsymbol{\theta} = \{w_1, w_2\}$ dengan *adaptive learning rate*.
Sumber: (Geiger et al., 2022).

Faktor ε yang ditambahkan pada Persamaan 2.45 digunakan untuk menghindari pembagian dengan nol pada awal iterasi karena inisialiasi awal vektor $\mathbf{v}^{(0)}$ adalah nol.

Terakhir, metode optimasi *Adaptive Moment Estimation* (Adam) menggabungkan metode *stochastic gradient descent* dengan momentum dan *adaptive learning rate*. Persamaan 2.47 menunjukkan algoritma *stochastic gradient descent* dengan Adam dan Gambar 2.17 mengilustrasikan pembaruan parameter $\boldsymbol{\theta}$ dengan Adam. Persamaan 2.47 menunjukkan persamaan dari metode optimasi Adam dan Gambar 2.17 mengilustrasikan pembaruan parameter $\boldsymbol{\theta}$ dengan Adam.



Gambar 2.17: Ilustrasi dari pembaruan parameter $\boldsymbol{\theta} = \{w_1, w_2\}$ dengan Adam.
Sumber: (Geiger et al., 2022).

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \frac{\eta \hat{\mathbf{m}}^{(t+1)}}{\sqrt{\hat{\mathbf{v}}^{(t+1)}} + \varepsilon}, \tag{2.47}$$

$$\hat{\mathbf{m}}^{(t+1)} = \frac{\mathbf{m}^{(t+1)}}{1-\beta_1}, \tag{2.48}$$

$$\hat{\mathbf{v}}^{(t+1)} = \frac{\mathbf{v}^{(t+1)}}{1-\beta_2}, \tag{2.49}$$

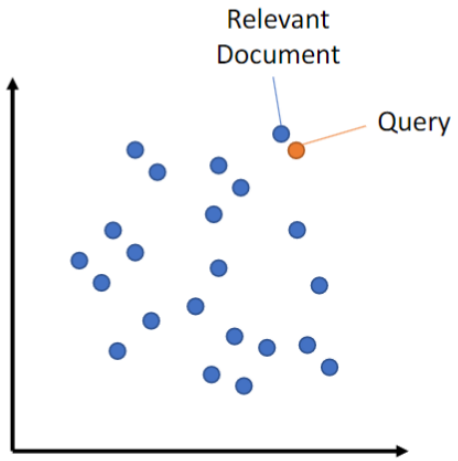
$$\mathbf{m}^{(t+1)} = \beta_1 \mathbf{m}^{(t)} + (1-\beta_1) \nabla_{\boldsymbol{\theta}} L_{\mathcal{B}}(\boldsymbol{\theta}^{(t)}), \tag{2.50}$$

$$\mathbf{v}^{(t+1)} = \beta_2 \mathbf{v}^{(t)} + (1-\beta_2) \left(\nabla_{\boldsymbol{\theta}} L_{\mathcal{B}}(\boldsymbol{\theta}^{(t)}) \odot \nabla_{\boldsymbol{\theta}} L_{\mathcal{B}}(\boldsymbol{\theta}^{(t)}) \right). \tag{2.51}$$

Alasan dilakukan pembagian dengan $(1-\beta_1)$ dan $(1-\beta_2)$ Persamaan 2.48 dan Persamaan 2.49 adalah untuk menghilangkan bias pada *momentum* dan *running variance* pada awal iterasi.

2.4 Pembelajaran Representasi

Pembelajaran representasi adalah proses pembelajaran model *machine learning* atau *deep learning* $f_{\text{model}}(x) : \mathcal{X} \to \mathbb{R}^n$ yang memetakan *high dimensional input* $x \in \mathcal{X}$ ke dalam ruang vektor $\mathbb{R}^n$. Pemetaan yang diharapkan dari model $f_{\text{model}}(x)$ diharapkan dapat mengenkode informasi yang terkandung pada x ke dalam vektor $\mathbf{R}^n$. Salah satu contoh informasi yang dapat dienkode adalah jarak antara dua kalimat x_1 dan x_2 yang memiliki kesamaan semantik atau sintaksis akan lebih dekat daripada jarak antara dua kalimat x_1 dan x_3 yang tidak memiliki kesamaan sama sekali pada ruang vektor. Gambar 2.18 mengilustrasikan pemetaan *input* menjadi vektor.



Gambar 2.18: Ilustrasi dari Pemetaan *input* menjadi vektor. *input* yang memiliki kesamaan semantik atau sintaksis akan lebih dekat daripada *input* yang tidak memiliki kesamaan.
Sumber: `https://www.sbert.net`

2.4.1 Fungsi *Loss* pada Pembelajaran Representasi

Fungsi loss pada pembelajaran representasi untuk permasalahan pemeringkatan teks biasanya disebut sebagai *ranking loss*. Meminimalkan fungsi *ranking loss* berarti memastikan bahwa *input-input* yang serupa berada lebih dekat daripada *input-input* yang tidak mirip. Sebagian besar fungsi loss pada pembelajaran representasi tidak memerlukan label kelas.

Fungsi loss yang digunakan pada penelitian ini adalah *N-pair loss* (van den Oord, Li, & Vinyals, 2018). *N-pair loss* dapat ditinjau sebagai klasifikasi multikelas dengan N-1 kelas berupa kelas negatif (kelas yang tidak mirip dengan *input* yang diberikan) dan 1 kelas positif (kelas yang mirip dengan *input* yang diberikan). Untuk suatu *input* x dan *input* positif x^+ dan kumpulan *input* negatif $\{x_i^-\}_{i=1}^{N-1}$, *N-pair loss* dapat ditulis seperti pada Persamaan 2.52.

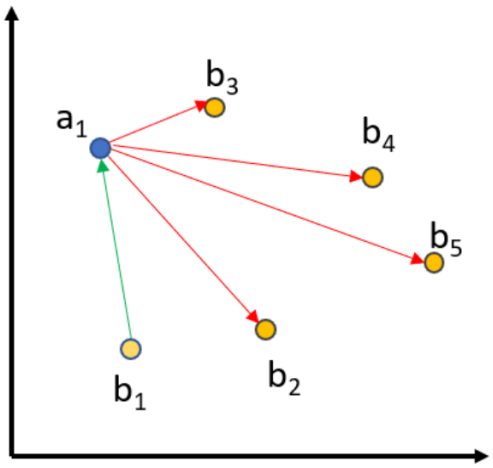
$$L(x,x^+,\{x_i^-\}_{i=1}^{N-1}) = \\ -\log \frac{\exp(\text{sim}((f_{\text{model}}(x),f_{\text{model}}(x^+)))}{\sum_{i=1}^{N-1}\exp(\text{sim}(f_{\text{model}}(x),f_{\text{model}}(x_i^-))) + \exp(\text{sim}(f_{\text{model}}(x),f_{\text{model}}(x^+)))}, \quad (2.52)$$

dengan sim adalah fungsi yang mengukur keserupaan antara dua vektor (fungsi jarak). Pada penelitian ini, fungsi sim yang digunakan adalah fungsi *dot product* sehingga Per-

samaan 2.52 dapat ditulis ulang seperti pada Persamaan 2.53.

$$L(x,x^+,\{x_i^-\}_{i=1}^{N-1}) = -\log \frac{\exp(f_{\text{model}}(x)^\top f_{\text{model}}(x^+))}{\sum_{i=1}^{N-1}\exp(f_{\text{model}}(x)^\top f_{\text{model}}(x_i^-)) + \exp(f_{\text{model}}(x)^\top f_{\text{model}}(x^+))}. \tag{2.53}$$

Gambar 2.19 mengilustrasikan fungsi *N-pair loss*. Untuk pasangan teks yang relevan (a,b_1), tujuannya adalah untuk meminimalkan jarak (fungsi sim) antara a dan b_1 sehingga jarak tersebut lebih kecil dibandingkan dengan jarak antara a dan b_i yang lain.



Gambar 2.19: Ilustrasi fungsi objektif *N-pair loss*. Untuk pasangan teks yang relevan (a,b_1), tujuannya adalah untuk meminimalkan jarak antara a dan b_1 sehingga jarak tersebut lebih kecil dibandingkan dengan jarak antara a dan b_i yang lain.
Sumber: `https://www.sbert.net/`

BAB 3
BIDIRECTIONAL ENCODER REPRESENTATIONS FROM TRANSFORMERS 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 menunjukkan bagaimana mekanisme *attention* dilakukan.

$$\text{Attention}(\mathbf{q}, \mathbf{K}, \mathbf{V}) = [\alpha_1, \alpha_2, \ldots, \alpha_n]\mathbf{V} \in \mathbb{R}^{d_v}, \tag{3.1}$$

dengan keterangan sebagai berikut:

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

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

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

dan $f_{attn}(\mathbf{q}, \mathbf{k})$ 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

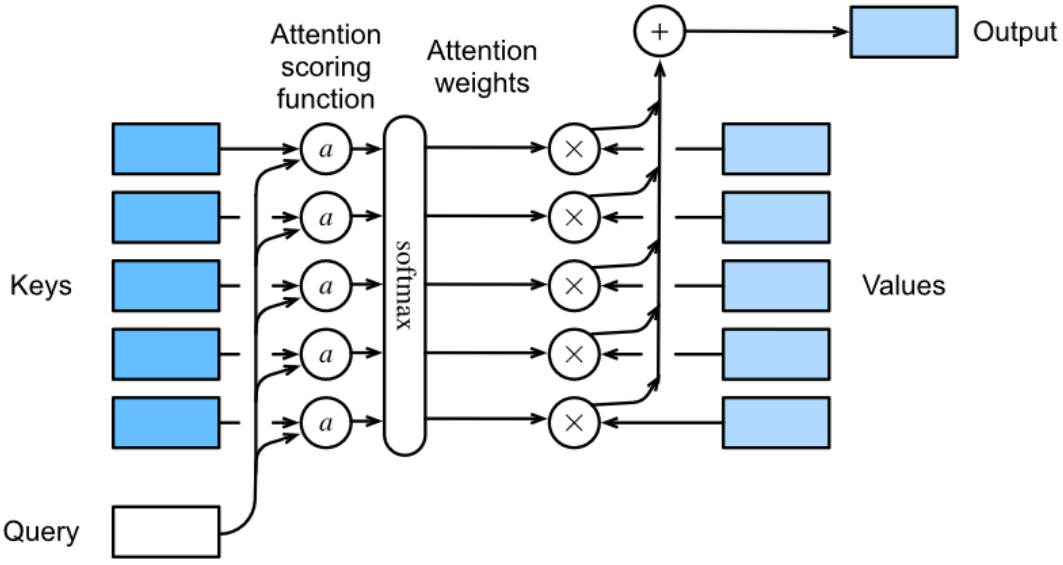
$f_{\text{attn}}(\mathbf{q},\mathbf{k}) = \mathbf{q}\cdot\mathbf{k}$, nilai dari Attention$(\mathbf{q},\mathbf{K},\mathbf{V})$ adalah $[0,1]$, karena nilai maksimal f_{attn} terjadi ketika $\mathbf{k} = [1,2]$.

Mekansime *attention* pada Persamaan 3.1 disebut sebagai *hard attention* karena hanya satu vektor nilai $\mathbf{v}_i$ yang dipilih dari sekumpulan vektor nilai $\mathbf{V}$. Berbeda dengan *hard attention*, *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 3.2 menunjukkan bagaimana mekanisme *soft attention* dilakukan.

$$\text{Attention}(\mathbf{q},\mathbf{K},\mathbf{V}) = [\alpha_1,\alpha_2,\ldots,\alpha_n]\mathbf{V} \in \mathbb{R}^{d_v}, \tag{3.2}$$

dengan nilai α_i yang dihitung seperti persamaan berikut:

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



Gambar 3.1: Ilustrasi dari mekanisme *soft attention*.
Sumber: (A. Zhang et al., 2023).

Dengan rata-rata terbobot dari $\mathbf{V}$, transformasi *soft attention* terturunkan. Hal ini merupakan syarat *fundamental* yang harus dimiliki oleh sebuah model *deep learning*.

Sebagai contoh, hasil dari 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 $Q = \{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_m\}$, mekanisme *attention* untuk setiap triplet $(\mathbf{q}_i, \mathbf{K}, \mathbf{V})$ dapat dihitung secara bersamaan seperti yang ditunjukkan pada Persamaan 3.3.

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathbf{AV} \in \mathbb{R}^{m \times d_v}, \tag{3.3}$$

dengan:

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

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

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

dan 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

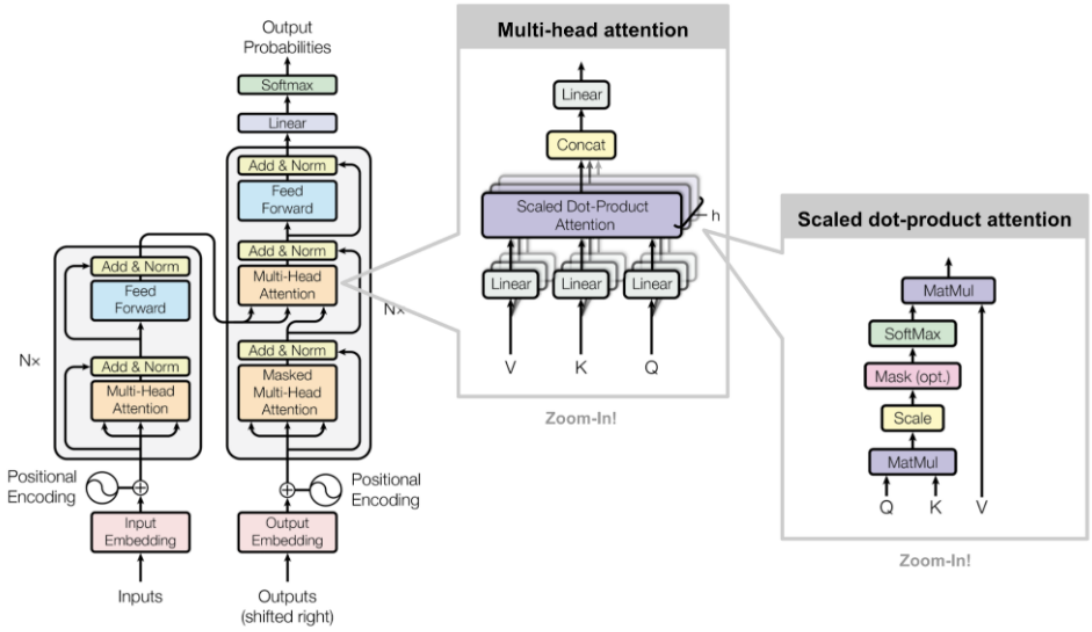
Mekanisme *attention* yang dilakukan oleh Vaswani et al. (2017) merupakan mekanisme *attention* parametrik. Pada mekanisme *attention* parametrik, nilai f_{attn} antar vektor kueri $\mathbf{q}$ dan $\mathbf{v}$ dibandingkan pada ruang vektor yang akan dipelajari (*learned embedding space*) daripada ruang 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 3.4

$$f_{attn}(\mathbf{q}\mathbf{W}^q, \mathbf{k}\mathbf{W}^k) = (\mathbf{q}\mathbf{W}^q + \mathbf{k}\mathbf{W}^k)\mathbf{W}^{\text{out}} \in \mathbb{R}, \tag{3.4}$$

dengan $\mathbf{W}^q \in \mathbb{R}^{d_q \times d_{\text{attn}}}, \mathbf{W}^k \in \mathbb{R}^{d_k \times d_{\text{attn}}}, \mathbf{W}_{\text{out}} \in \mathbb{R}^{d_{\text{attn}} \times 1}$ adalah matriks parameter 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 **q** dan **k** di ruang vektor yang dipelajari (*learned embedding space*). Persamaan 3.5 menunjukkan bagaimana *dot-product attention* dihitung.

$$f_{attn}(\mathbf{q}\mathbf{W}^q, \mathbf{k}\mathbf{W}^k) = (\mathbf{q}\mathbf{W}^q)(\mathbf{k}\mathbf{W}^k)^\top. \tag{3.5}$$

3.2 *Transformer*



Gambar 3.2: Arsitektur *transformer*.
Sumber: (Weng, 2018).

Transformer merupakan arsitektur *deep learning* yang pertama kali diperkenalkan oleh Vaswani et al. (2017). Awalnya *transformer* 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):

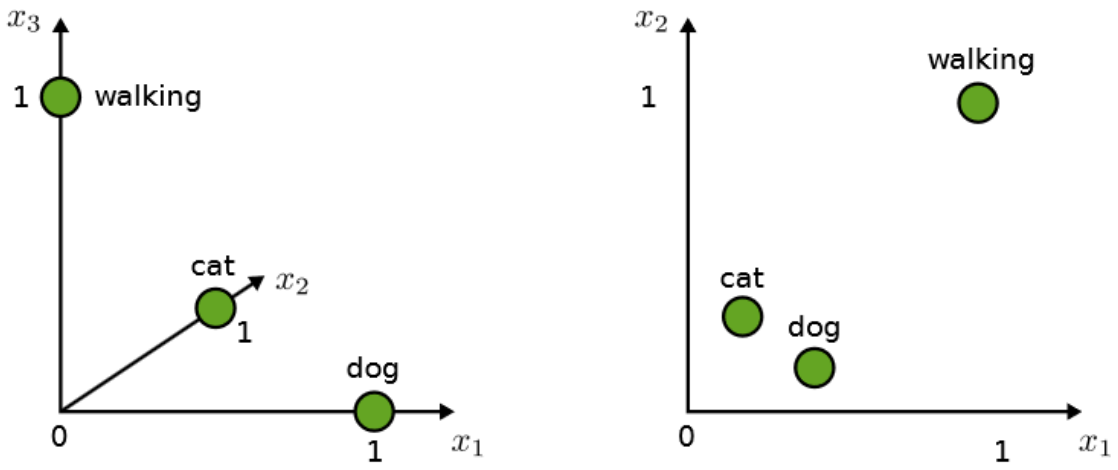
1. *Encoder self-attention*: menggunakan barisan *input* yang berupa barisan token atau kata sebagai masukan untuk menghasilkan barisan representasi kontekstual, berupa vektor, dari *input*. Setiap representasi token tersebut memiliki ketergantungan dengan token lainnya dari barisan *input*.

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. *Encoder-Decoder 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* terdiri dari pasangan *encoder-decoder*. Aristektur dari *transformer* dapat dilihat pada Gambar 3.2. 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. pada permasalahan tersebut, *output* dari lapisan *encoder* yang digunakan sebagai masukan untuk suatu *classifier*. Subbab 3.2.1 hingga Subbab 3.2.8 menjelaskan arsitektur model *transformer encoder* dan berbagai mekanisme yang menyusun model *transformer*.

3.2.1 *Token Embedding (Input Embedding)*



Gambar 3.3: Ilustrasi dari representasi token. Gambar kiri menunjukkan representasi token dengan *one-hot encoding*, sedangkan gambar kanan menunjukkan representasi token dengan *token embedding*.
Sumber: (Geiger et al., 2022).

Perlu diingat kembali bahwa *input* dari fungsi Attention (dan tentunya *transformer*) adalah barisan vektor. Jika mekanisme *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 $\mathbf{oh}_{i_j} = [0, \ldots, 0, 1, 0, \ldots, 0] \in \{0,1\}^{|\mathcal{T}|}$, 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*, representasi yang digunakan adalah vektor padat dengan dimensi d_{token} 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 3.6 hingga Persamaan 3.7 menunjukkan bagaimana representasi vektor dari barisan suatu barisan token $t = (t_{i_1}, t_{i_2}, \ldots, t_{i_L})$ dihitung.

$$\text{Embed}(t) = \mathbf{E}_t = \begin{bmatrix} \mathbf{e}_{i_1} \\ \mathbf{e}_{i_2} \\ \vdots \\ \mathbf{e}_{i_L} \end{bmatrix} \in \mathbb{R}^{L \times d_{\text{token}}}, \tag{3.6}$$

$$\mathbf{e}_{i_j} = \mathbf{oh}_{i_j} \mathbf{E}_{\mathcal{T}} \in \mathbb{R}^{d_{\text{token}}}. \tag{3.7}$$

Gambar 3.3 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, biasanya 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 **q** dan vektor kunci **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 $\sqrt{d_{\text{attn}}}$, seperti yang ditunjukkan Persamaan 3.8.

$$f_{attn}(\mathbf{q}\mathbf{W}^q, \mathbf{k}\mathbf{W}^k) = \frac{\mathbf{q}\mathbf{W}^q (\mathbf{k}\mathbf{W}^k)^\top}{\sqrt{d_{\text{attn}}}} \in \mathbb{R}, \tag{3.8}$$

dengan $\mathbf{W}^q \in \mathbb{R}^{d_q \times d_{\text{attn}}}, \mathbf{W}^k \in \mathbb{R}^{d_k \times d_{\text{attn}}}$ adalah matriks parameter dan d_{attn} adalah dimensi dari *learned embedding space* yang digunakan untuk perhitungan nilai atensi.

Pembagian dengan $\sqrt{d_{\text{attn}}}$ 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 $\sqrt{d_{\text{attn}}}$, variansi dari nilai atensi akan memiliki faktor tambahan $\sigma^2 d_{\text{attn}}$, seperti yang ditunjukkan

pada Persamaan 3.9 hingga Persamaan 3.10.

$$\mathbf{q}\mathbf{W}^q \sim \mathcal{N}(0, \sigma^2) \text{ dan } \mathbf{k}\mathbf{W}^k \sim \mathcal{N}(0, \sigma^2), \tag{3.9}$$

$$\text{Var}(\mathbf{q}\mathbf{W}^q(\mathbf{k}\mathbf{W}^k)^\top) = \sum_{i=1}^{d_{\text{attn}}} \text{Var}\left((\mathbf{q}\mathbf{W}^q)_i((\mathbf{k}\mathbf{W}^k)_i^\top\right) = \sigma^4 d_{\text{attn}}. \tag{3.10}$$

Akibatnya, untuk nilai d_{attn} yang cukup besar, akan terdapat satu elemen atensi acak $(\mathbf{q}\mathbf{W}^q(\mathbf{k}\mathbf{W}^k)^\top)_i$ sehingga $\mid (\mathbf{q}\mathbf{W}^q(\mathbf{k}\mathbf{W}^k)^\top)_i \mid \gg \mid (\mathbf{q}\mathbf{W}^q(\mathbf{k}\mathbf{W}^k)^\top)_j \mid$ 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.

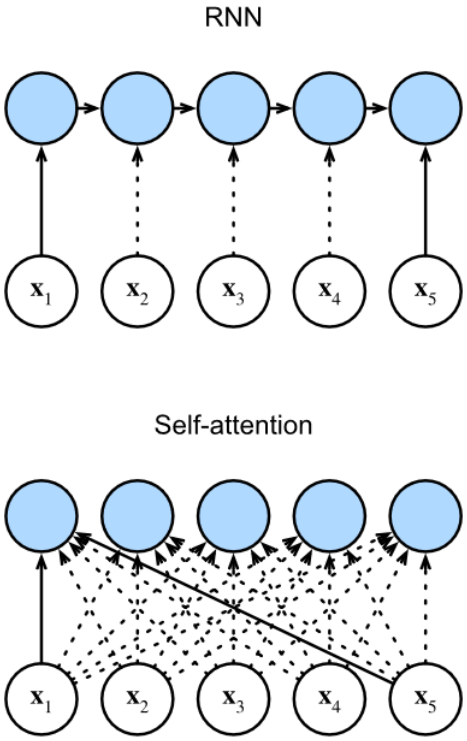
Dengan *scaled dot product attention*, tidak ada faktor d_{attn} pada variansi nilai atensi. faktor σ^4 pada Persamaan 3.11 tidak menjadi masalah karena dengan *normalization layer* yang dijelaskan pada Subbab 3.2.7 mengakibatkan $\sigma^2 \approx 1$ sehingga $\sigma^4 \approx \sigma^2 \approx 1$.

$$\text{Var}\left(\frac{\mathbf{q}\mathbf{W}^q(\mathbf{k}\mathbf{W}^k)^\top}{\sqrt{d_{\text{attn}}}}\right) = \frac{\sigma^4 d_{\text{attn}}}{d_{\text{attn}}} = \sigma^4 \tag{3.11}$$

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 3.12.

$$\text{Attention}(\mathbf{Q}\mathbf{W}^q, \mathbf{K}\mathbf{W}^k, \mathbf{V}) = \text{Softmax}(\frac{\mathbf{Q}\mathbf{W}^q(\mathbf{K}\mathbf{W}^k)^\top}{\sqrt{d_{\text{attn}}}})\mathbf{V} \in \mathbb{R}^{m \times d_v}. \tag{3.12}$$

3.2.3 *Self-Attention*



Gambar 3.4: 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.
Sumber: (A. Zhang et al., 2023).

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 3.4 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.

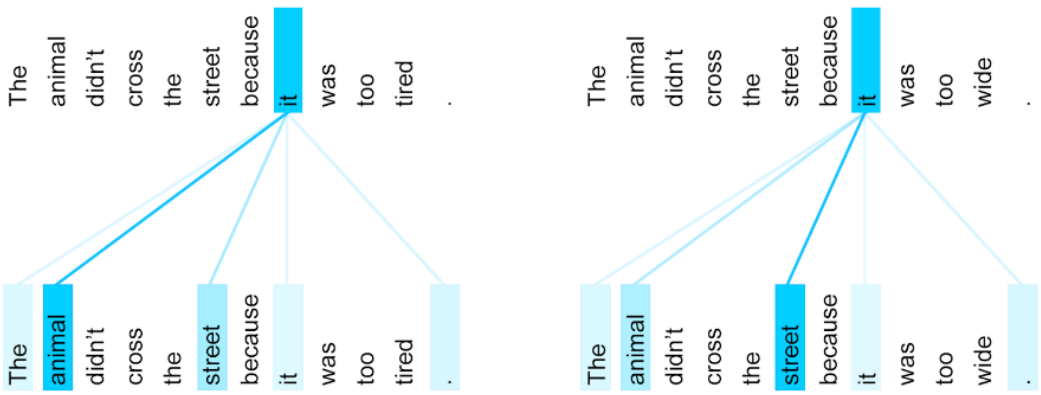
Perhitungan *self-attention* pada *transformer* yang digunakan adalah *scaled dot product attention* yang telah dijelaskan pada Subbab 3.2.2. Pada *self-attention*, kumpulan vektor kueri **Q**, vektor kunci **K**, dan vektor nilai **V** adalah vektor yang sama, yaitu *embedding* dari token **E** yang 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 3.13 menunjukkan bagaimana *self-attention* dihitung.

$$\begin{aligned}\text{Self-Attention}(\mathbf{E}) &= \text{Attention}(\mathbf{E}\mathbf{W}^q, \mathbf{E}\mathbf{W}^k, \mathbf{E}\mathbf{W}^v) \\ &= \text{Softmax}(\frac{\mathbf{E}\mathbf{W}^q(\mathbf{E}\mathbf{W}^k)^\top}{\sqrt{d_{\text{token}}}})(\mathbf{E}\mathbf{W}^v) \in \mathbb{R}^{L \times d_{\text{token}}},\end{aligned} \tag{3.13}$$

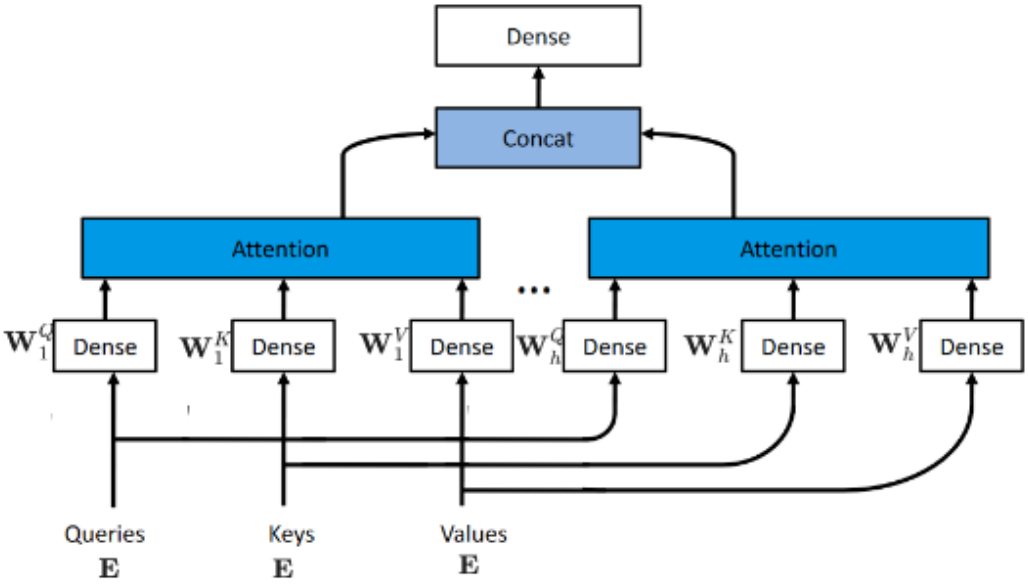
dengan $\mathbf{W}^q, \mathbf{W}^k, \in \mathbb{R}^{d_{\text{token}} \times d_{\text{token}}}, \mathbf{W}^v \in \mathbb{R}^{d_{\text{token}} \times d_{\text{token}}}$ adalah matriks bobot.

Self-attention dapat dikonsepsikan sebagai proses pembentukan representasi token yang kontekstual. Untuk setiap tokennya, *self-attention* menghitung keserupaan antara token $\mathbf{E}\mathbf{W}^q$ 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 3.5 adalah contoh dari *self-attention* yang menghasilkan representasi vektor kontekstual pada token `it`. Pada Gambar 3.5 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 3.5 memiliki bobot atensi yang tinggi terhadap token *street*.



Gambar 3.5: Ilustrasi *self-attention* dalam menghasilkan representasi vektor kontekstual dari barisan token. Representasi vektor dari token `it` akan bergantung terhadap barisan token *input*.
Sumber: (Murphy, 2022)

3.2.4 *Multi-Head Self-Attention*



Gambar 3.6: Ilustrasi *multi-head self-attention* pada *transformer*. *Multi-head self-attention* menghitung *self-attention* sebanyak h kali pada subruang yang berbeda.
Sumber: (Murphy, 2022), telah diolah kembali.

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 3.14 menunjukkan bagaimana *multi-head self-attention* dihitung.

$$\text{MHSA}(\mathbf{E}) = [\text{head}_1|\ldots|\text{head}_h]\mathbf{W}^O \in \mathbb{R}^{L\times d_{\text{token}}}, \tag{3.14}$$

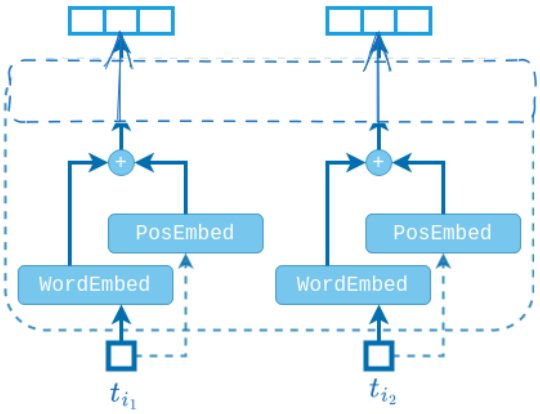
$$\text{head}_i = \text{Self-Attention}_i(\mathbf{E}) = \text{Softmax}(\frac{\mathbf{E}\mathbf{W}_i^q(\mathbf{E}\mathbf{W}_i^k)^\top}{\sqrt{d_{\text{token}}/h}})\mathbf{E}\mathbf{W}_i^v \in \mathbb{R}^{L\times \frac{d_{\text{token}}}{h}}, \tag{3.15}$$

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

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 de-

ngan 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*



Gambar 3.7: Ilustrasi dari *positional encoding* pada *transformer*. *Positional encoding* ditambahkan pada *token embedding* sebelum dijadikan *input* untuk *transformer*.
Sumber: (pi tau, 2023), telah diolah kembali.

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 3.16 terpenuhi.

$$\text{Self-Attention}(\mathbf{E}\mathbf{P}_\pi) = \text{Self-Attention}(\mathbf{E})\mathbf{P}_\pi \tag{3.16}$$

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 ditun-

jukkan pada Persamaan 3.17.

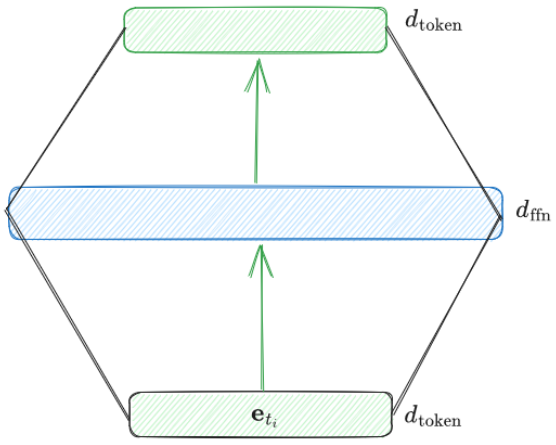
$$\text{PE}_{\text{pos},i} = \begin{cases} \sin\left(\frac{pos}{10000^{i/d_{\text{token}}}}\right) & \text{jika } i \bmod 2 = 0, \\ \cos\left(\frac{pos}{10000^{(i-1)/d_{\text{token}}}}\right) & \text{lainnya.} \end{cases} \tag{3.17}$$

Berbeda dengan Vaswani et al. (2017), Devlin et al. (2018) menggunakan matriks parameter $\mathbf{W}^{pe} \in \mathbb{R}^{L_{\max} \times d_{\text{token}}}$ untuk menghitung matriks *positional encoding* $\mathbf{PE} \in \mathbb{R}^{L \times d_{\text{token}}}$ seperti yang ditunjukkan pada Persamaan 3.18 hingga Persamaan 3.19. Kekurangan dari pendekatan ini adalah model tidak dapat melakukan inferensi pada barisan token yang lebih panjang dari $L_{\max}$. Gambar 3.7 mengilustrasikan *positional encoding* pada *transformer*.

$$\mathbf{pe}_i = [0, 0, \ldots, \underbrace{1}_{\text{indeks ke-}i}, 0, \ldots, 0]\mathbf{W}^{pe} \in \mathbb{R}^{d_{\text{token}}}, \tag{3.18}$$

$$\text{pos}(t) = \mathbf{PE} = \begin{bmatrix} \mathbf{pe}_1 \\ \mathbf{pe}_2 \\ \vdots \\ \mathbf{pe}_L \end{bmatrix} \in \mathbb{R}^{L \times d_{\text{token}}}. \tag{3.19}$$

3.2.6 *Position-wise Feed-Forward Network*



Gambar 3.8: 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 3.8 menunjukkan ilustrasi dari *position-wise feed-forward network* dan Persamaan 3.20 menunjukkan Transfor-

masi yang dilakukan oleh *position-wise feed-forward network*.

$$\text{FFN}(\mathbf{X}) = \max(0, \mathbf{X}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2 \in \mathbb{R}^{L \times d_{\text{token}}}, \tag{3.20}$$

dengan $\mathbf{W}_1 \in \mathbb{R}^{d_{\text{token}} \times d_{\text{ffn}}}, \mathbf{W}_2 \in \mathbb{R}^{d_{\text{ffn}} \times d_{\text{token}}}, \mathbf{b}_1 \in \mathbb{R}^{d_{\text{ffn}}}, \mathbf{b}_2 \in \mathbb{R}^{d_{\text{token}}}$ adalah matriks bobot dan bias. d_{ffn} adalah dimensi dari *feed forward network* yang digunakan.

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 tiap baris bobot atensi $\text{Softmax}(\frac{\mathbf{E}\mathbf{W}^q(\mathbf{E}\mathbf{W}^k)^\top}{\sqrt{d_{\text{token}}}})$ akan 1 juga. Persamaan 3.21 menunjukkan proses kerja dari *layer normalization*.

$$\begin{aligned}\text{LayerNorm}(\mathbf{X}) &= (\mathbf{X} - \boldsymbol{\mu}) \odot \frac{1}{\boldsymbol{\sigma}} \\ &= \begin{bmatrix} \frac{x_{11}-\mu_1}{\sigma_1} & \frac{x_{12}-\mu_1}{\sigma_1} & \cdots & \frac{x_{1,d_{\text{token}}}-\mu_1}{\sigma_1} \\ \frac{x_{21}-\mu_2}{\sigma_2} & \frac{x_{22}-\mu_2}{\sigma_2} & \cdots & \frac{x_{2,d_{\text{token}}}-\mu_2}{\sigma_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{x_{L1}-\mu_L}{\sigma_L} & \frac{x_{L2}-\mu_L}{\sigma_L} & \cdots & \frac{x_{L,d_{\text{token}}}-\mu_L}{\sigma_L} \end{bmatrix} \in \mathbb{R}^{L \times d_{\text{token}}},\end{aligned} \tag{3.21}$$

dengan keterangan berikut:

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

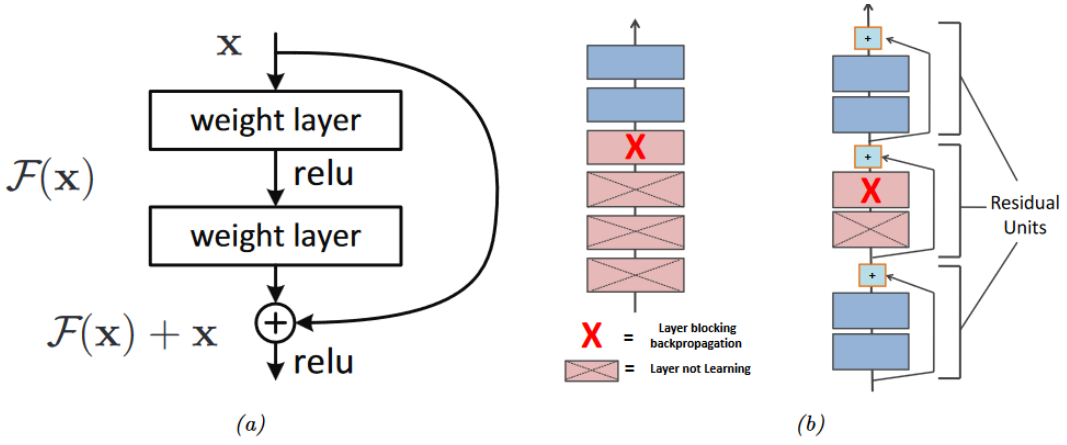
$$\boldsymbol{\mu} = \begin{bmatrix} \mu_1 & \dots & \mu_1 \\ \vdots & \ddots & \vdots \\ \mu_L & \dots & \mu_L \end{bmatrix} \in \mathbb{R}^{L \times d_{\text{token}}},$$

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

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

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

$\odot$ = *element-wise product.*



Gambar 3.9: Ilustrasi koneksi residu.
Sumber: (Murphy, 2022)

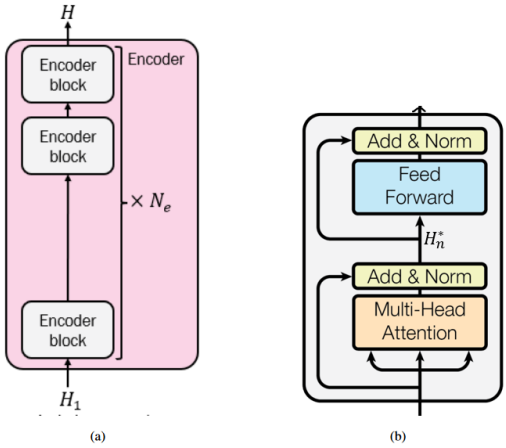
Koneksi Residu adalah koneksi yang menghubungkan *output* dari suatu *layer* dengan *input* dari *layer* selanjutnya. Koneksi residu digunakan untuk mengatasi masalah *van-*

ishing gradient yang terjadi pada *deep neural network* dengan memperbaiki *flow gradient* dari model. Persamaan matematis dari koneksi residu dijelaskan seperti pada Persamaan 3.22.

$$f_l'(\mathbf{x}) = f_l(\mathbf{x}) + \mathbf{x}, \tag{3.22}$$

dengan $f_l(\mathbf{x})$ adalah suatu *layer* atau kumpulan *layer* pada *deep neural network*. Pada *transformer*, *residual connection* digunakan sebelum *layer normalization*.

3.2.8 *Transformer Encoder*



Gambar 3.10: Ilustrasi *transformer encoder*. (a) *transformer encoder*, (b) *encoder* blok.

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 Persamaan 3.26 hingga Persamaan 3.27.

$$\text{EncoderBlock}(\mathbf{X}) = \mathbf{Z}_2 \in \mathbb{R}^{L \times d_{\text{token}}}, \tag{3.23}$$

$$\mathbf{Z}_2 = \text{LayerNorm}(\overbrace{\text{FFN}(\mathbf{Z}_1) + \mathbf{Z}_1}^{\text{Koneksi Residu}}), \tag{3.24}$$

$$\mathbf{Z}_1 = \text{LayerNorm}(\overbrace{\text{MHSA}_h(\mathbf{X}), +\mathbf{X}}^{\text{Koneksi Residu}}), \tag{3.25}$$

dengan $\mathbf{X} \in \mathbb{R}^{L \times d_{\text{token}}}$ adalah *input* dari blok *transformer*.

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 3.26 hingga Persamaan 3.27.

$$\mathbf{E} = \text{embed}(t) + \text{pos}(t) \in \mathbb{R}^{L \times d_{\text{token}}}, \tag{3.26}$$

$$\text{Encoder}(\mathbf{E}) = \text{EncoderBlock}_n(\text{EncoderBlock}_{n-1}(\ldots(\text{EncoderBlock}_1(\mathbf{E})))) \in \mathbb{R}^{L \times d_{\text{token}}}. \tag{3.27}$$

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 dan mempelajari representasi vektor kata yang kontekstual. 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 $\text{BERT}_{\text{BASE}}$ tersusun atas 12 blok *encoder* dengan dimensi *token embedding* d_{token} sebesar 768, 12 *attention heads* pada setiap blok *encoder*-nya, 110 juta parameter dan panjang barisan token maksimal L_{max} sebesar 512.

3.3.1 Representasi Input

Model BERT menghasilkan representasi vektor kontekstual dari token dengan *transformer encoder*. *Input* dari *transformer encoder* berupa vektor numerik sehingga teks harus diubah mengikuti format tersebut. *Input* dari model BERT adalah penjumlahan dari *token embedding* (lihat Subbab 3.2.1), *segment embedding*, dan *position embedding* (lihat Subbab 3.2.5).

Token embedding didapatkan dengan terlebih dahulu memecah teks menjadi kumpulan token. Proses pemecahan teks ini dinamakan proses *tokenizing*. *Tokenizing* pada BERT dilakukan dengan algoritma *WordPiece* yang dikembangkan oleh Wu et al. (2016).

Dalam proses *tokenizing*, teks dipecah menjadi kata atau subkata – kata-kata yang tidak terdaftar pada kosakata yang dipecah menjadi subkata. Sebagai contoh, misalkan

kata `etiopia` tidak terdaftar pada kosakata, kata tersebut akan dipecah menjadi subkata `eti` dan `##opia`, dengan subkata `eti` dan `pia` terdaftar pada kosakata. Simbol `##` menandakan bahwa subkata tersebut merupakan subkata dari kata sebelumnya. Untuk suatu *input* berupa teks seperti `apa nama ibukota etiopia`, proses *tokenizing* mengubah teks tersebut menjadi barisan token (`apa`, `nama`, `ibu`, `kota`, `eti`, `##opia`).

Setelah proses *tokenizing*, token khusus `[CLS]` dan `[SEP]` ditambahkan di awal dan akhir barisan token sehingga barisan token menjadi (`[CLS]`, `apa`, `nama`, `ibu`, `kota`, `eti`, `##opia`, `[SEP]`). Token `[CLS]` digunakan sebagai representasi dari seluruh kalimat dan token `[SEP]` digunakan sebagai pemisah antar kalimat.

Selanjutnya, setiap token pada barisan token dipetakan ke vektor numerik seperti yang dijelaskan pada Subbab 3.2.1. Hasil *token embedding* berupa vektor numerik yang dapat merepresentasikan suatu kata. Untuk menyimpan representasi vektor dari seluruh kosakata, BERT menggunakan matriks *token embedding* yang dipelajari ketika proses *pre-training* yang berukuran $|\mathcal{T}| \times d_{\text{token}}$ dengan $|\mathcal{T}|$ adalah ukuran kosakata. Tabel 3.1 menunjukkan ilustrasi dari pemetaan *token embedding* pada BERT.

Tabel 3.1: Ilustrasi pemetaan *token embedding* pada BERT.

Token	**Token ID**	***Token Embedding***
saya	0	[-0.0124, -0.0556, -0.0235, ..., -0.0168, -0.0401, -0.0107]
apa	1	[-0.0112, -0.054, -0.0245, ..., -0.0168, -0.0401, -0.0107]
kamu	2	[-0.0124, -0.0556, -0.0235, ..., -0.0168, -0.0401, -0.0107]
⋮	⋮	⋮
opia	508	[-0.0124, -0.0556, -0.0235, ..., -0.0168, -0.0401, -0.0107]
nasi	509	[-0.0218, -0.0556, -0.0135, ..., -0.0043, -0.0151, -0.0249]
⋮	⋮	⋮
eti	5000	[-0.0124, -0.0556, -0.0235, ..., -0.0168, -0.0401, -0.0107]
⋮	⋮	⋮

Segment embedding digunakan sebagai penanda bagian yang berbeda pada suatu barisan token. Sebagai contoh, pada permasalahan *question answering* (*QA*), *input* pertanyaan direpresentasikan oleh barisan token *A* dan paragraf konteks direpresentasikan oleh barisan token *B*. Setiap *segment* dipisahkan oleh token khusus `[SEP]`. Setiap token yang berada pada *segment* yang sama akan memiliki vektor numerik yang identik pada *segment embedding*. Untuk menyimpan representasi vektor dari setiap *segment*, BERT

menggunakan matriks *segment embedding* yang dipelajari ketika proses *pre-training* yang berukuran $2 \times d_{\text{token}}$. Tabel 3.2 menunjukkan ilustrasi dari pemetaan *segment embedding* pada BERT.

Tabel 3.2: Ilustrasi pemetaan *segment embedding* pada BERT.

Segment	***Segment Embedding***
0	[0.0004, 0.011, 0.0037, ..., -0.0066, -0.0034, -0.0086]
1	[0.0011, -0.003, -0.0032, ..., 0.0047, -0.0052, -0.0112]

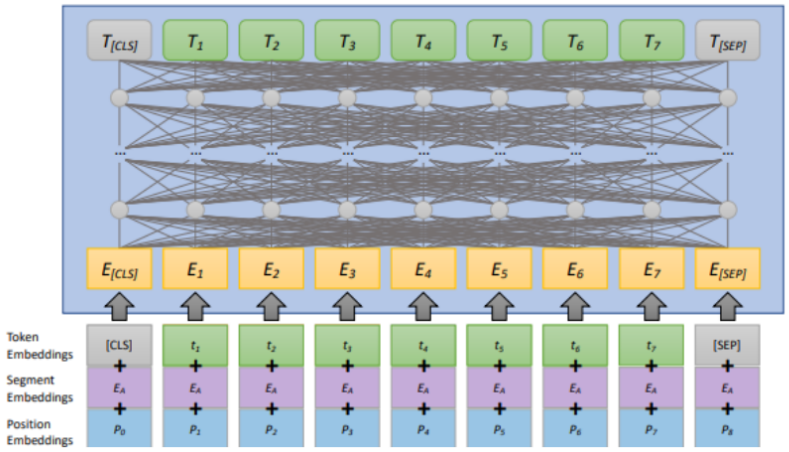
Positional embedding digunakan untuk menghilangkan sifat permutasi *equivariant* pada barisan token yang telah dijelaskan pada Subbab 3.2.5. Untuk menyimpan representasi vektor dari setiap posisi token, BERT menggunakan matriks *positional embedding* yang dipelajari ketika proses *pre-training* yang berukuran $L_{\text{max}} \times d_{\text{token}}$, dengan L_{max} adalah panjang maksimal barisan token. Tabel 3.3 menunjukkan ilustrasi dari pemetaan *positional embedding* pada BERT.

Tabel 3.3: Ilustrasi pemetaan *positional embedding* pada BERT.

Posisi Token	***Position Embedding***
0	[0.0175, -0.0256, -0.0366, ..., 0.0, 0.0007, 0.0154]
1	[0.0078, 0.0023, -0.0194, ..., 0.0289, 0.0298, -0.0053]
2	[-0.0113, -0.002, -0.0116, ..., 0.0149, 0.0187, -0.0073]
509	[0.0174, 0.0035, -0.0096, ..., 0.003, 0.0004, -0.0269]
510	[0.0217, -0.006, 0.0147, ..., -0.0056, -0.0126, -0.0281]
511	[0.0026, -0.0233, 0.0055, ..., 0.0175, 0.0275, -0.0777]

Input dari model BERT adalah hasil penjumlahan dari *token embedding*, *segment embedding*, dan *positional embedding*. Pada penelitian ini, Proses pembentukan *input embedding* tersebut diasumsikan menjadi bagian dari fungsi BERT. Oleh karena itu, fungsi BERT didefinisikan sebagai fungsi yang menerima suatu barisan token ($\texttt{[CLS]}, t_1, t_2, \ldots, t_L, \texttt{[SEP]}$) dan menghasilkan barisan vektor kontekstual ($\mathbf{T}_{\texttt{[CLS]}}, \mathbf{T}_{t_1}, \mathbf{T}_{t_2}, \ldots, \mathbf{T}_{t_L}, \mathbf{T}_{\texttt{[SEP]}}$) seperti yang ditunjukkan pada Persamaan 3.28 dan Gambar 3.11.

$$\text{BERT}((\texttt{[CLS]}, t_1, t_2, \ldots, t_L, \texttt{[SEP]})) = (\mathbf{T}_{\texttt{[CLS]}}, \mathbf{T}_{t_1}, \mathbf{T}_{t_2}, \ldots, \mathbf{T}_{t_L}, \mathbf{T}_{\texttt{[SEP]}}). \tag{3.28}$$

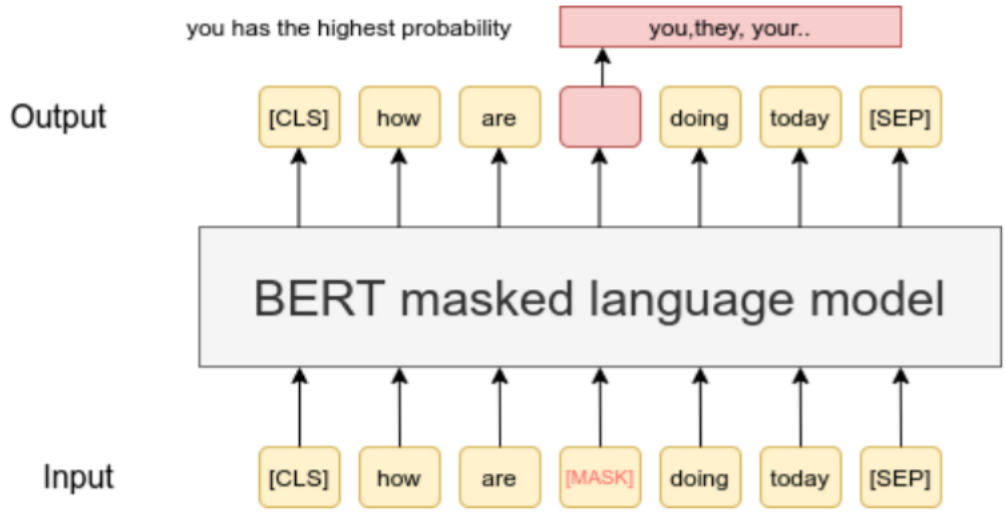


Gambar 3.11: Ilustrasi dari representasi *input* pada BERT. Barisan kata diubah menjadi *token*, *segment*, dan *positional embedding*. Jumlahan *embedding* ini menghasilkan *embedding input*, yang melewati 12 blok *transformer encoder*. Representasi kontekstual vektor kata diambil dari blok terakhir.
Sumber: (Lin et al., 2020).

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 3.12: 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 (`[MASK]`) 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`.

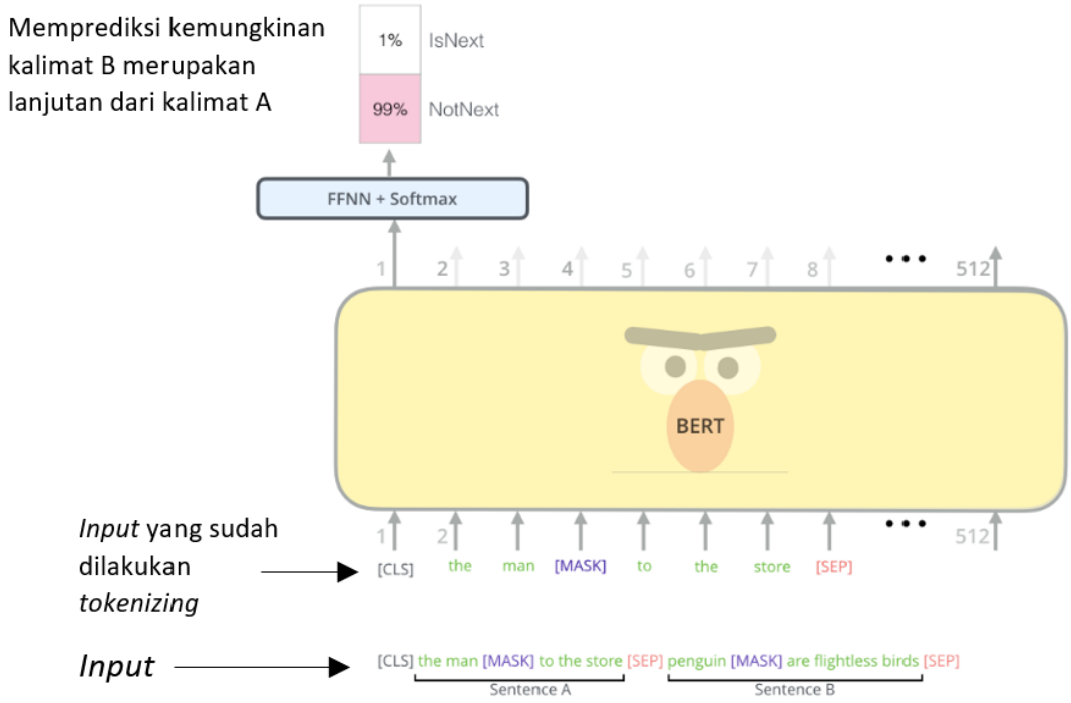
Selama proses pelatiahan MLM, sebanyak 15% dari semua token dipilih untuk dilakukan *masking*. 80% dari token yang terpilih akan diubah menjadi token `[MASK]`, 10% menjadi token acak, dan 10% lainnya adalah token yang sama.

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. *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 3.13 mengilustrasikan contoh dari tugas *next sentence prediction*.



Gambar 3.13: Ilustrasi *next sentence prediction* pada BERT. Model diminta untuk memprediksi apakah kalimat kedua adalah kalimat berikutnya dari kalimat pertama.

3.3.3 BERT untuk Bahasa Indonesia (IndoBERT)

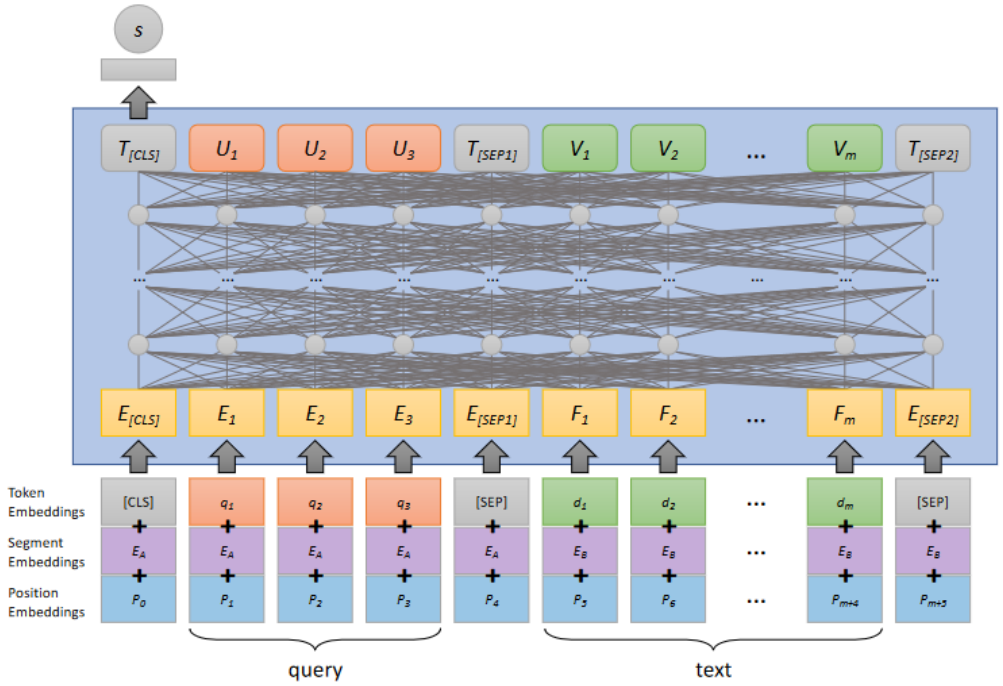
Pada penelitian ini, BERT yang digunakan adalah BERT untuk bahasa Indonesia yang sudah dilakukan *pre-training* sebelumnya oleh Koto, Rahimi, Lau, dan Baldwin (2020). *Pre-training* BERT untuk bahasa Indonesia dilakukan dengan menggunakan korpus Wikipedia bahasa Indonesia dengan 74 Juta kata, artikel 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). Informasi lebih lan-

jut mengenai model IndoBERT dapat dilihat pada laman `https://huggingface.co/indolem/indobert-base-uncased`.

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.

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 melakukan 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.
Sumber: (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 Per-

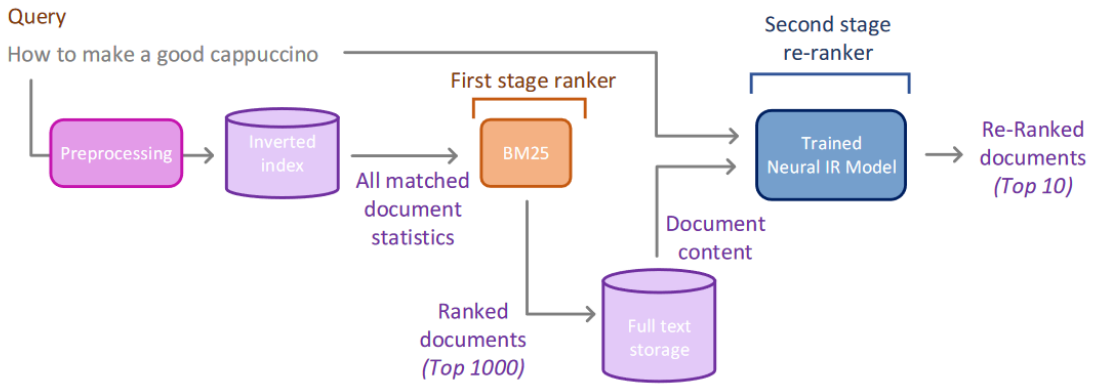
samaan 3.29 hingga Persamaan 3.30.

$$\text{score}(q,d) = P(\text{relevance} = 1|q,d) = \sigma\left(\mathbf{h}_{[\text{CLS}]}\mathbf{W}^{\text{CLS}} + \mathbf{b}^{\text{CLS}}\right) \in (0,1), \tag{3.29}$$

$$\mathbf{h}_{\texttt{[CLS]}} = \text{BERT}((\texttt{[CLS]}, q, \texttt{[SEP]}, d, \texttt{[SEP]}))_{\texttt{[CLS]}} \in \mathbb{R}^{d_{\text{token}}}, \tag{3.30}$$

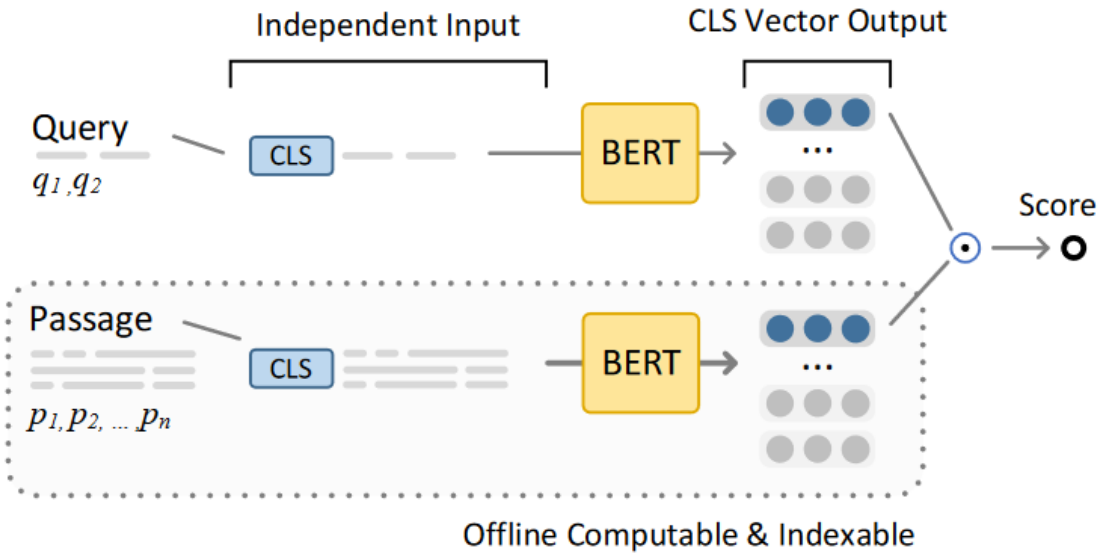
dengan $\mathbf{W}^{\text{CLS}} \in \mathbb{R}^{d_{\text{token}} \times 1}$ dan $\mathbf{b}^{\text{CLS}} \in \mathbb{R}$ adalah matriks bobot dan bias yang digunakan untuk melakukan *classification*, dan σ adalah fungsi sigmoid.

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}.
Sumber: (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.
Sumber: (Hofstätter et al., 2021).

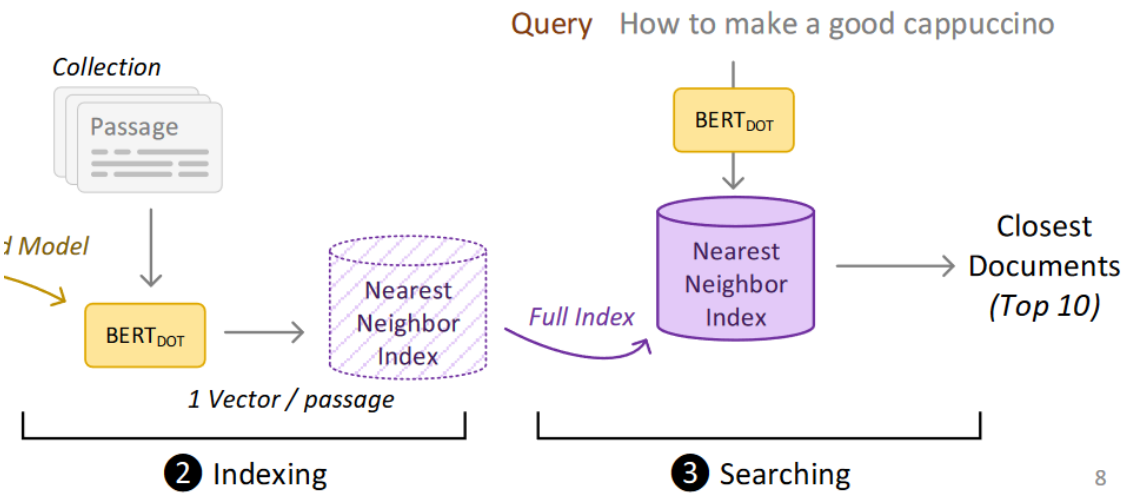
Berbeda dengan BERT_{CAT}, BERT_{DOT} tidak melakukan *soft classification* untuk setiap pasangan (kueri, teks). BERT_{DOT} menghitung skor relevansi dari pasangan (kueri, teks) dengan melakukan *dot product* antara vektor representasi kontekstual dari kueri dan teks seperti yang ditunjukkan pada Persamaan 3.31 hingga Persamaan 3.33.

$$\text{score}(q,d) = \mathbf{q}_{\texttt{[CLS]}} \cdot \mathbf{d}_{\texttt{[CLS]}} \in \mathbb{R}, \tag{3.31}$$

$$\mathbf{q}_{\texttt{[CLS]}} = \text{BERT}((\texttt{[CLS]}, q, \texttt{[SEP]}))_{\texttt{[CLS]}} \in \mathbb{R}^{d_{\text{token}}}, \tag{3.32}$$

$$\mathbf{d}_{\texttt{[CLS]}} = \text{BERT}((\texttt{[CLS]}, d, \texttt{[SEP]}))_{\texttt{[CLS]}} \in \mathbb{R}^{d_{\text{token}}}. \tag{3.33}$$

Salah satu kelebihan dari BERT_{DOT} adalah vektor representasi $\mathbf{d}_{\texttt{[CLS]}}$ dari teks dapat diindeks terlebih dahulu dan disimpan dalam memori. Akibatnya, kita hanya perlu menghitung $\mathbf{q}_{\texttt{[CLS]}}$ sekali, dan dapat menghitung skor relevansi untuk setiap pasangan (kueri, teks) dengan melakukan *dot product* antara $\mathbf{q}_{\texttt{[CLS]}}$ dan $\mathbf{d}_{\texttt{[CLS]}}$ yang sudah disimpan dalam memori sebelumnya. 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 dan skor relevansi dapat dihitung dengan melakukan *dot product* antara vektor representasi kueri dan teks.
Sumber: (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 pada masalah pemeringkatan teks. Subbab 4.1 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

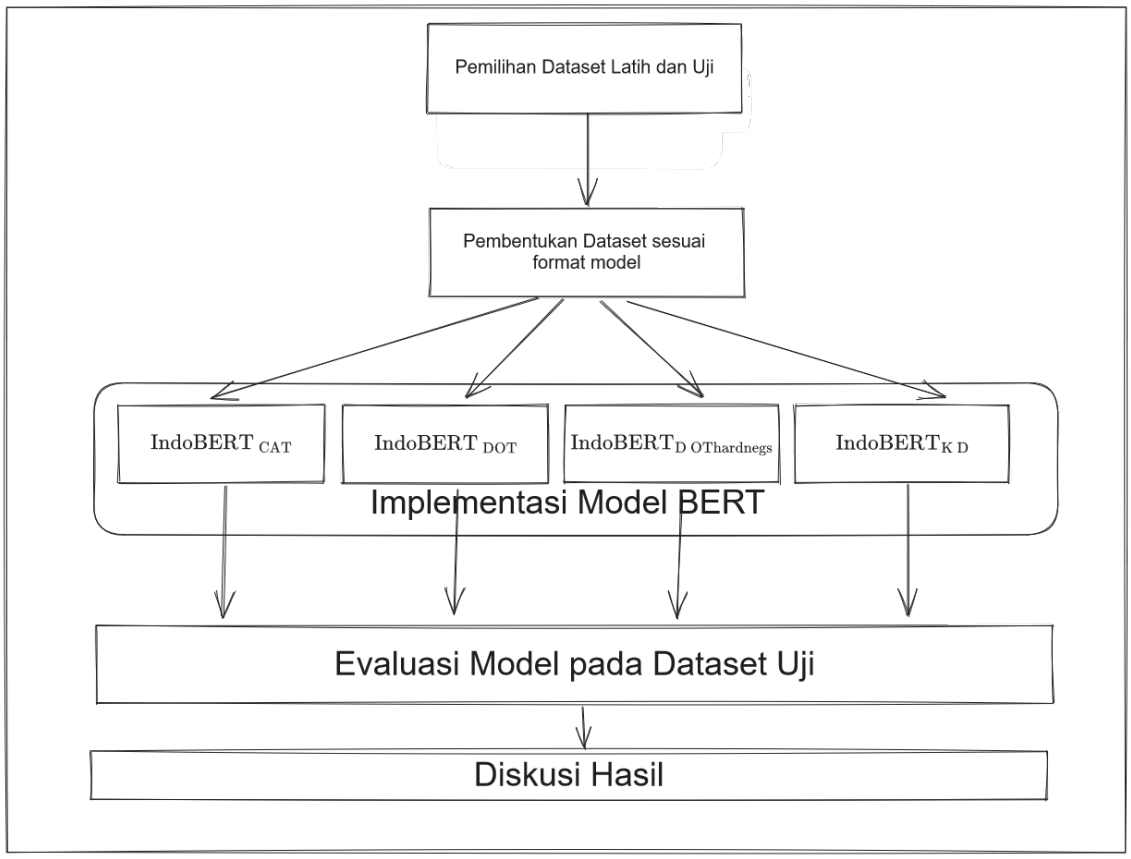
Proses *fine tuning* model BERT untuk pemeringkatan teks dilakukan menggunakan mesin dan perangkat lunak yang tertera pada Tabel 4.1.

Tabel 4.1: Spesifikasi perangkat lunak yang digunakan pada penelitian ini.

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 et al. (2021), sebagai *dataset* latih, dan data pada penelitian X. Zhang et al. (2021), X. Zhang et al. (2023) sebagai *dataset* uji. *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, akan dilakukan evaluasi setiap model pada *dataset* uji. Terdapat satu model BM25 sebagai *baseline* untuk membandingkan hasil evaluasi dari model-model yang dilatih. Terakhir, terdapat diskusi mengenai hasil evaluasi tersebut.

4.3 Data

Penelitian ini menggunakan satu *dataset* latih mMarco *train set* bahasa Indonesia (Bonifacio et al., 2021) dan tiga *dataset* uji, yaitu mMarco *dev set* bahasa Indonesia, MrTyDi Test

set Indonesia (X. Zhang et al., 2021), dan Miracl *dev set* bahasa Indonesia (X. Zhang et al., 2023). *Dataset* Miracl dan MrTyDi dipilih sebagai uji kemampuan *out-of-distibution* dari model yang dihasilkan. Setiap *dataset* 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.

Tabel 4.2: Tabel Informasi untuk Setiap *Dataset*. Kolom *Korpus* menunjukkan jumlah entri pada *file korpus*, kolom *Kueri* menunjukkan jumlah entri pada *file kueri*, dan kolom *Jugdements* menunjukkan jumlah entri pada *file jugdements* (pasangan kueri dan teks dengan nilai relevansi).
Sumber: (Tsai et al., 2019)

Dataset	**Korpus**	**Kueri**	**Jugdements**
mMarco train set	8,841,823	502.939	532,761
mMarco dev set	8,841,823	6980	7,437
Mrtydi test set	1,469,399	829	961
Miracl dev set	1,446,315	960	9,668

Setiap *judgements* pada *dataset* adalah *jugdgements* biner, yaitu bernilai 1 jika teks tersebut relevan dengan kueri dan 0 jika tidak relevan dengan kueri. Tabel 4.3 hingga Tabel 4.5 menunjukkan contoh dari *file* korpus, *file* kueri, dan *file jugdements* pada *dataset* Miracl *dev set*.

Tabel 4.3: 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.

Tabel 4.4: 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?

Tabel 4.5: Potongan *file* judgements *dataset* Miracl.

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

4.4 *Fine Tuning* Model BERT

Bagian ini menjelaskan mengenai konfigurasi *hyperparameter* yang digunakan pada setiap model yang dikerjakan pada peneletian ini. Terdapat empat model yang dikerjakan pada penelitian ini, yaitu IndoBERT$_{\text{CAT}}$, IndoBERT$_{\text{DOT}}$, IndoBERT$_{\text{DOThardnegs}}$, dan IndoBERT$_{\text{DOTKD}}$. Keempat model tersebut merupakan model BERT yang dilatih dengan menggunakan *dataset* mMarco *train set* dengan prosedur yang berbeda-beda.

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

Pada model IndoBERT$_{\text{CAT}}$, arsitektur BERT$_{\text{CAT}}$ (lihat Subbab 3.3.4.1) digunakan untuk melakukan pemeringkatan teks. Pengunaan BERT sebagai *classifier* nilai relevansi pasangan kueri dan teks pertama kali diperkenalkan oleh Nogueira dan Cho (2019). Proses pelatihan model menggunakan *dataset* yang digunakan berasal dari mMarco *train set* dengan format (q,d,r) dengan q adalah kueri, d adalah teks, dan r adalah relevansi teks d terhadap kueri q. 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 teks 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 $r = 0$. Potongan *dataset* yang digunakan untuk pelatihan model IndoBERT$_{\text{CAT}}$ dapat ditemukan pada Tabel 4.6.

Tabel 4.6: Potongan *dataset* yang digunakan untuk pelatihan model IndoBERT$_{\text{CAT}}$.

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

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

Tabel 4.7: *Hyperparameter* yang digunakan untuk *fine tuning* 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 selama 10% dari total iterasi
Fungsi loss	*Binary cross entropy*

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, teks relevan d^+, dan kumpulan teks tidak relevan $\{d_i^-\}_{i=1}^{N-1}$ terhadap kueri q, *N-pair loss* dihitung sebagai berikut:

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

dengan keterangan sebagai berikut:

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

Pasangan (q, d^+) diambil langsung dari *file judgements* pada mMarco *train set*. Kumpulan teks tak relevan $\{d_i^-\}_{i=1}^{N-1}$ dibentuk dengan menggunakan teks d pada *data point* yang berbeda pada *batch* yang sama. Nilai N pada *N-pair loss* adalah ukuran *batch* yang digunakan selama pelatihan model. Metode pemilihan teks 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 4.8.

Tabel 4.8: *Hyperparameter* yang digunakan untuk *fine tuning* IndoBERT$_{\text{DOT}}$.

Parameter	**Nilai**
Model pralatih	`indolem/indobert-base-uncased`
Total data	532,761
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 selama 10% dari total iterasi
Fungsi *loss*	*N-pair loss*

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 IndoBERT$_{\text{DOThardnegs}}$ pada metode pemilihan teks negatif. Pada IndoBERT$_{\text{DOThardnegs}}$, teks negatif sudah terlebih dahulu dipilih dengan kriteria bahwa teks tersebut adalah teks yang tidak relevan dengan kueri q, tetapi pemeringkatan dengan BM25 berada di 100 teratas. Dengan kata lain, teks negatif adalah teks yang sulit dibedakan dengan teks positif ketika menggunakan BM25. Pentingnya pemilihan teks *negative* yang baik untuk fungsi loss *N-pair loss* ditunjukkan pada penelitian Qu et al. (2020); Xiong et al. (2020). Nilai N yang dipilih pada penelitian ini adalah $N = 5$, dan jumlah data yang digunakan adalah 502.939 *data point*.

Tabel 4.9: Potongan *file hard negative*. Kolom qid berisikan id dari kueri, kolom *positive* adalah id teks positif, dan kolom *hard negative* adalah id teks yang sulit dibedakan dengan teks positif menggunakan BM25.
Sumber: `https://huggingface.co/datasets/carles-undergrad-thesis/mmarco-hardnegs-bm25`

qid	*Positive*	*Hard Negative*
1185869	0	[2942572, 5154062, 2942571, 5154065, 3870084]
1185868	16	[6821177, 1641650, 1641656, 1641659, 1203539]
597651	49	[6398884, 162755, 1838949, 1391482, 7818305]

Konfigurasi *hyperparameter* selama pelatiahan model indoBERT$_{\text{DOThardnegs}}$ dapat dilihat pada Tabel 4.10.

Tabel 4.10: *Hyperparameter* yang digunakan untuk *fine tuning* 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 selama 10% dari total iterasi
Fungsi *loss*	*N-pair loss*

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

IndoBERT$_{\text{DOTKD}}$ dilatih dengan menggunakan prinsip *knowledge distillation*, yaitu proses *transfer* pengetahuan dari model yang sudah dilatih dengan baik (guru) ke model yang belum dilatih (murid). Pelatihan dengan *knowledge distillation* untuk menghasilkan representasi kalimat yang baik diperkenalkan oleh Reimers dan Gurevych (2020) pada permasalahan pemahaman bahasa alami lainnya. Pada peneletian ini, model yang digunakan sebagai guru adalah model bahasa Inggris yang sudah dilatih dengan baik untuk

melakukan pemeringkatan teks. Model Murid yang dapat dipilih adalah model pralatih BERT multibahasa – model yang *pre-training*-nya dilakukan pada korpus multibahasa seperti mBERT (*mulitingual* BERT). Pemetaaan vektor dari model murid akan di-*align* dengan Pemetaan vektor dari model guru dengan fungsi *loss* berikut:

$$L(s_i,t_i) = \left((|| M(s_i) - \hat{M}(s_i) ||)^2 + (|| M(s_i) - \hat{M}(t_i) ||)^2 \right), \tag{4.2}$$

dengan keterangan sebagai berikut:

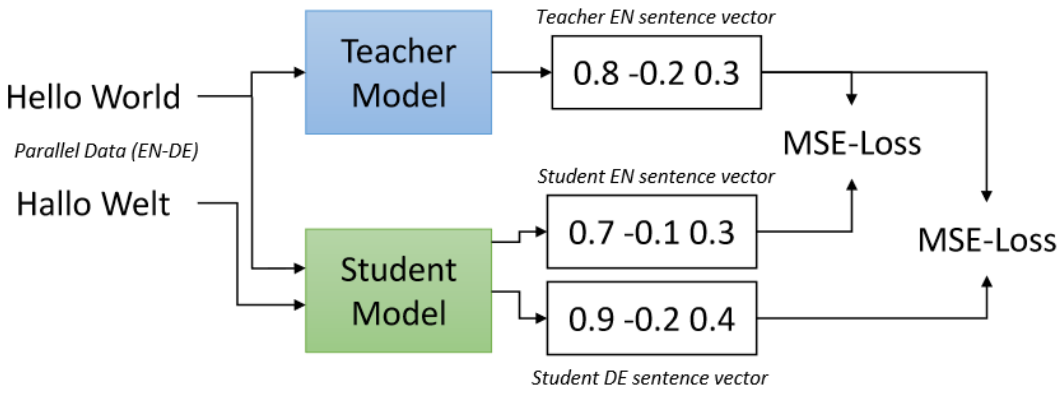
M = pemeataan vektor oleh model guru,

$\hat{M}$ = pemetaan vektor oleh model murid,

s_i = teks sumber (bahasa Inggris),

t_i = teks target (bahasa Indonesia).

Gambar 4.2 menunjukkan ilustrasi dari proses pelatihan dengan *knowledge distillation*. 500 ribu kueri dan 500 ribu korpus dari mMarco *train set* Indonesia di-*align* dengan terjemahannya seperti yang ditunjukkan pada Tabel 4.11.



Gambar 4.2: Ilustrasi dari pelatihan model IndoBERT$_{\text{DOTKD}}$ dengan *knowledge distillation*. Kalimat paralel diberikan sebagai *input* pada model guru dan model murid. vektor yang dihasilkan oleh model guru dan model murid di-*align* menggunakan fungsi *loss mean squared error*.
Sumber: (Reimers & Gurevych, 2020)

Tabel 4.11: Potongan dari *dataset* yang digunakan untuk pelatihan model IndoBERT$_{KD}$.
Sumber: `carles-undergrad-thesis/en-id-parallel-sentences`

text_en	**txt_id**
Defining alcoholism as a disease is associated with Jellinek	Mendefinisikan alkoholisme sebagai penyakit dikaitkan dengan Jellinek
ECT is a treatment that is used for	ECT adalah pengobatan yang digunakan untuk
Ebolavirus is an enveloped virus, which means	Ebolavirus adalah virus yang diselimuti, yang berarti
How much does Cambridge Manor cost per month	Berapa biaya Cambridge Manor per bulan?

Konfigurasi *hyperparameter* selama pelatiahan model indoBERT$_{KD}$ dapat dilihat pada Tabel 4.12.

Tabel 4.12: *Hyperparameter* yang digunakan untuk *fine tuning* IndoBERT$_{DOTKD}$.

Parameter	**Nilai**
Model guru	`sentence-transformers/msmarco-bert-base-dot-v5`
Model murid	`bert-base-multilingual-uncased`
Total data	1,000,000
Batch Size	64
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 selama 10% dari total iterasi
Fungsi *loss*	*Mean squared error*

4.5 Evaluasi Model

Subbab ini membahas perihal hasil dari *Fine tuning* dan evaluasi dari keempat model. BM25 digunakan sebagai *baseline* untuk membandingkan evaluasi setiap model. BM25 yang digunakan adalah *software Elastic search* (`https://www.elastic.co/elasticsearch/`) untuk bahasa Indonesia dengan konfigurasi parameter *default*.

Metrik yang digunakan pada *datasets* mMarco *dev set* dan MrTyDi *test set* adalah *Reciprocal Rank* pada 10 teks teratas (RR@10) dan *Recall* pada 1000 teks teratas (R@1000). Berbeda dengan mMarco dan MrTyDi, Metrik yang digunakan pada *dataset* Miracl *dev set* adalah *Normalized Discounted Cumulative Gain* pada 10 teks teratas (NDCG@10) dan *Recall* pada 1000 teks teratas (R@1000).

4.5.1 Evaluasi IndoBERT$_{\text{CAT}}$

Tabel 4.13: Evaluasi model IndoBERT$_{\text{CAT}}$ pada *dataset* mMarco *dev set*, MrTyDi *test set*, dan Miracl *dev set*.

Model	mMarco Dev		MrTyDi Test		Miracl Dev	
	RR@10	R@1000	RR@10	R@1000	NDCG@10	R@1000
BM25	.114	**.642**	.279	**.858**	.391	**.971**
BM25+IndoBERT$_{\text{CAT}}$	**.181**	**.642**	**.447**	**.858**	**.455**	**.971**

Tabel 4.13 menujukkan evaluasi dan perbandingan antara model IndoBERT$_{\text{CAT}}$ dan BM25. Perlu diingat kembali bahwa model IndoBERT$_{\text{CAT}}$ digunakan sebagai *reranker* (lihat Subbab 3.3.4.1) untuk memperbaiki hasil dari BM25. Efeknya nilai *Recall* dari model IndoBERT$_{\text{CAT}}$ sama dengan nilai *Recall* dari BM25. Namun, nilai *Reciprocal Rank* dari model IndoBERT$_{\text{CAT}}$ lebih tinggi dari BM25. Hal ini menunjukkan bahwa model IndoBERT$_{\text{CAT}}$ lebih baik dalam memberikan urutan teks yang direkomendasikan oleh BM25.

nilai RR@10 dari model IndoBERT$_{\text{CAT}}$ pada dataset mMarco *dev set* meningkat sebesar .67 poin (58%) dibandingkan dengan BM25. Peningkatan yang sama juga terjadi pada nilai RR@10 MrTyDi *test set* dan nilai NDCG@10 Miracl *dev set* dengan peningkatan sebesar .168 poin (60%) dan .064 poin (16%) masing-masingnya.

4.5.2 Evaluasi IndoBERT$_{\text{DOT}}$

Tabel 4.14: Evaluasi model IndoBERT$_{\text{DOT}}$ pada *dataset* mMarco *dev set*, MrTyDi *test set*, dan Miracl *dev set*.

Model	mMarco Dev		MrTyDi Test		Miracl Dev	
	RR@10	R@1000	RR@10	R@1000	NDCG@10	R@1000
BM25	.114	.642	.279	.858	**.391**	**.971**
IndoBERT$_{\text{DOT}}$	**.192**	**.847**	**.378**	**.936**	.355	.920

Tabel 4.14 menunjukkan evaluasi dan perbandingan antara model IndoBERT$_{\text{DOT}}$ dan BM25. Peningkatan terjadi pada nilai RR@10 dan R@1000 pada *dataset* mMarco *dev set* dan MrTyDi *test set*. Nilai RR@10 pada mMarco *dev set* meningkat sebesar .078 poin (68%) dan pada MrTyDi *test set* sebesar .099 poin (35%). Nilai R@1000 pada mMarco *dev set* meningkat sebesar .205 poin (32%) dan pada MrTyDi *test set* sebesar .078 poin (9%). Sementara itu, nilai NDCG@10 pada Miracl *dev set* menurun sebesar .036 poin (-9%) dan nilai R@1000 juga menurun sebesar .051 poin (-5%).

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

Tabel 4.15: Evaluasi model IndoBERT$_{\text{DOThardnegs}}$ pada *dataset* mMarco *dev set*, MrTyDi *test set*, dan Miracl *dev set*.

Model	mMarco Dev		MrTyDi Test		Miracl Dev	
	RR@10	R@1000	RR@10	R@1000	NDCG@10	R@1000
BM25	.114	.642	.279	.858	.391	**.971**
IndoBERT$_{\text{DOThardnegs}}$	**.232**	**.847**	**.471**	**.921**	**.397**	.898

Tabel 4.15 menunjukkan evaluasi dan perbandingan antara model IndoBERT$_{\text{DOThardnegs}}$ dan BM25. Peningkatan terjadi pada nilai RR@10 dan R@1000 pada *dataset* mMarco *dev set* dan MrTyDi *test set*. Nilai RR@10 pada mMarco *dev set* meningkat sebesar .118 poin (103%) dan pada MrTyDi *test set* sebesar .192 poin (69%). Nilai R@1000 pada mMarco *dev set* juga meningkat sebesar .205 poin (32%) dan pada MrTyDi *test set* sebesar .063 poin (7%). Sementara itu, nilai NDCG@10 pada Miracl *dev set* menurun sebesar .006 poin (-1%) dan nilai R@1000 juga menurun sebesar .073 poin (-8%).

4.5.4 Evaluasi IndoBERT$_{KD}$

Tabel 4.16: Evaluasi model IndoBERT$_{KD}$ pada *dataset* mMarco *dev set*, MrTyDi *test set*, dan Miracl *dev set*.

Model	mMarco Dev		MrTyDi Test		Miracl Dev	
	RR@10	R@1000	RR@10	R@1000	NDCG@10	R@1000
BM25	.114	.642	.279	.858	**.391**	**.971**
IndoBERT$_{DOTKD}$	**.235**	**.867**	**.393**	**.882**	.374	.871

Tabel 4.16 menunjukkan evaluasi dan perbandingan antara model IndoBERT$_{DOTKD}$ dengan BM25. Peningkatan Terjadi pada nilai RR@10 dan R@1000 pada *dataset* mMarco *dev set* dan MrTyDi *test set*. Nilai RR@10 pada mMarco *dev set* meningkat sebesar .121 poin (106%) dan pada MrTyDi *test set* sebesar .114 poin (41%). Nilai R@1000 pada mMarco *dev set* juga meningkat sebesar .225 poin (37%) dan pada MrTyDi *test set* sebesar .024 poin (3%). Sementara itu, nilai NDCG@10 pada Miracl *dev set* menurun sebesar .017 poin (-4%) dan nilai R@1000 juga menurun sebesar .100 poin (-10%).

4.6 Diskusi Hasil

Tabel 4.17: Evaluasi dari model IndoBERT$_{CAT}$, IndoBERT$_{DOT}$, IndoBERT$_{DOThardnegs}$, dan IndoBERT$_{DOTKD}$ pada *dataset* mMarco *dev set*, MrTyDi *test set*, dan Miracl *dev set*.

Model	mMarco Dev		MrTyDi Test		Miracl Dev	
	RR@10	R@1000	RR@10	R@1000	NDCG@10	R@1000
BM25	.114	.642	.279	.858	.391	**.971**
BM25+IndoBERT$_{CAT}$	.181	.642	.447	.858	**.455**	**.971**
IndoBERT$_{DOT}$	.192	.847	.378	**.936**	.355	.920
IndoBERT$_{DOThardnegs}$	.232	.847	**.471**	.921	.397	.898
IndoBERT$_{DOTKD}$	**.235**	**.867**	.393	.882	.374	.871

Tabel 4.17 menunjukkan evaluasi dari model IndoBERT$_{CAT}$, IndoBERT$_{DOT}$, IndoBERT$_{DOThardnegs}$, dan IndoBERT$_{DOTKD}$ pada *dataset* mMarco *dev set*, MrTyDi *test set*, dan Miracl *dev set*. Perhatikan bahwa model IndoBERT$_{CAT}$ adalah model yang *robust* dibandingkan dengan model lainnya – Peningkatan metrik terjadi disetiap *dataset* yang digunakan. Hal ini dapat dikaitkan dengan fakta bahwa model dengan

arsitektur BERT_{CAT} dapat mengaitkan kata-kata pada kueri dengan kata-kata pada teks ketika proses pemberian skor. Namun, meskipun $\text{IndoBERT}_{\text{CAT}}$ adalah model yang paling *robust*, peningkatan setiap metrik tidaklah terlalu besar seperti model yang berasitekturkan BERT_{DOT}. Dugaan yang dapat penulis berikan adalah proses pelatihan tersebut membutuhkan waktu yang lebih lama untuk menghasilkan fungsi skoring yang baik karena cara melatih model BERT_{CAT} yang berupa klasifikasi relevansi antara kueri dan teks. Pelatihan yang dilakukan pada model $\text{IndoBERT}_{\text{CAT}}$ memiliki kelebihan dan kekurangan tersendiri. Kelebihan yang didapat adalah nilai skor antar kueri dan teks memiliki makna, meskipun berdiri sendiri. Nilai skor antara kueri dan dokumen menunjukkan nilai relevansinya antara kueri dan teks tersebut, hal ini kontras dengan model BERT_{DOT} yang hanya menghasilkan skor antara kueri dan teks, tanpa memiliki makna. Di lain sisi, kekurangannya adalah tugas klasifikasi relevansi tidak secara langsung melatih dengan tujuan yang diharapkan. Pada proses pemeringkatan teks, skor antara kueri dan teks tidak perlu memiliki makna, yang diinginkan hanyalah skor antara kueri dan teks yang relevan lebih tinggi dibandingkan skor antara kueri dan teks tidak relevan.

Berbeda dengan arsitektur BERT_{CAT}, arsitektur BERT_{DOT} dilatih dengan tujuan yang diharapkan secara langsung dengan *N-pair loss*. Hal ini dapat dilihat dari hasil evaluasi pada Tabel 4.17 dimana model $\text{IndoBERT}_{\text{DOT}}$ dan $\text{IndoBERT}_{\text{DOThardnegs}}$ memiliki peningkatan yang lebih besar dibandingkan model $\text{IndoBERT}_{\text{CAT}}$ pada mMarco *dev set* – *in-domain dataset*. Pemilihan *hard negative* teks pada model $\text{IndoBERT}_{\text{DOThardnegs}}$ juga meningkatkan performa model. Hal ini dapat dilihat pada evaluasi di *dataset* lainnya.

Skor yang dihasilkan oleh model $\text{IndoBERT}_{\text{DOT}}$ dan $\text{IndoBERT}_{\text{DOThardnegs}}$ tidak memiliki makna jika berdiri sendiri. Skor antara kueri dan teks hanya digunakan untuk membandingkan skor antara kueri dan teks lainnya. Skor antara kueri dan teks tidak dapat digunakan untuk mengetahui nilai relevansi antara kueri dan teks tersebut.

Model $\text{IndoBERT}_{\text{DOTKD}}$ memiliki performa yang terbaik pada *in-domain dataset* mMarco *dev set*. Namun performa tersebut tidaklah sebaik model $\text{IndoBERT}_{\text{DOThardnegs}}$ pada *out-of-domain dataset* MrTyDi *test set* dan Miracl *dev set*. Pelatihan dengan meng-*align* pemetaan vektor antara model murid dan model guru tidak baik untuk mengeneralisasi teks dan kueri yang tidak ada pada *dataset* pelatihan.

Tabel 4.18 menunjukkan latensi dan memori yang digunakan oleh model berasitekturkan BERT_{DOT}, BERT_{CAT}, dan BM25. Latensi dan memori diukur pada *hardware* yang

Tabel 4.18: *Benchmark* model BERT_{DOT} dan BERT_{CAT}, dan BM25 pada *dataset* mMarco *dev set*. Latensi dan memori diukur pada *hardware* yang sama dengan yang digunakan pada pelatihan model.

Model	Latensi (ms)	Memori(MB)
BM25 (Elastic Search)	**6.55** (CPU)	**800**
BERT_{DOT}	9.9 (GPU)	3072
BM25+BERT_{CAT}	242 (GPU)	**800**

sama dengan yang digunakan pada pelatihan model. Model berasitekturkan BERT_{DOT} Memiliki latensi yang hampir sama dengan BM25 dan Latensi yang lebih baik dibandingkan dengan model BERT_{CAT}. Hal ini dapat disebabkan model fungsi skor pada BERT_{DOT} hanyalah berupa perkalian titik dan vektor teks dapat dihitung (*indexing*) terlebih dahulu sebelum melakukan pemeringkatan teks. Bandingkan dengan model BERT_{CAT} yang memiliki fungsi skor yang kompleks dan membutuhkan waktu yang lama untuk menghitung skor antara kueri dan teks, meskipun hanya memeringkatkan (*reranking*) 1000 teks. Sementara itu, model BERT_{DOT} memerlukan memori yang lebih banyak karena representasi vektor padat dengan dimensi 768 harus disimpan terlebih dahulu. Model BERT_{CAT} hanya memerlukan memori yang sama dengan BM25.

Pada penelitian ini, interpretasi dari model tidaklah dibahas secara mendetail. paragraf berikut akan membahas sekilas mengenai interpretasi model BERT_{DOT} dan BERT_{CAT}.

Pada model BERT_{CAT}, interpretasi dapat dilakukan dengan *integrated gradients* (Sundararajan, Taly, & Yan, 2017). *Integrated Gradients* menghitung kontribusi dari setiap fitur (kata) pada hasil prediksi. Gambar 4.3 menunjukkan contoh interpretasi dari model BERT_{CAT} dengan *software* Captum.

Legend: Negative Neutral Positive

True Label	Predicted Label	Attribution Label	Attribution Score	Word Importance
1	1 (0.99)	1	1.83	[CLS] kapan komando divisi ix banteng dibentuk ? [SEP] komando divisi ix banteng adalah suatu komando militer yang dibentuk pada masa perang kemerdekaan (1945 - 1950) di sumatera tengah yang wilayah operasinya meliputi empat provinsi sekarang , ya itu sumatera barat , riau , jambi dan kepulauan riau . komando divisi ix banteng mempunyai pasukan yang banyak karena adanya sekolah pendidikan opsi ##r di bukittinggi yang pendiriannya diprakarsai oleh para perwira didikan jepang dimasa pendudukan , bahkan salah satu pasukannya yaitu resimen 6 dianggap sebagai pasukan terbaik di sumatera . [SEP]
1	1 (0.99)	1	1.96	[CLS] apakah nama ibukota eti ##opia ? [SEP] kini eti ##opia merupakan negara berbentuk republik dan mengambil bagian secara aktif dal ##a , aktivitas - aktivitas kerja sama internasional . ibukota ##nya add ##is aba ##ba merupakan pusat administrasi kesatuan afrika (au) . [SEP]
1	1 (0.99)	1	1.11	[CLS] apakah yang dimaksud dengan revolusi ? [SEP] menurut kamus besar bahasa indonesia revolusi / re · vo · lu · si / / revolusi / n * 1 perubahan ketatanegaraan (pemerintahan atau keadaan sosial) yg dilakukan dn ##g kekerasan (spt dn ##g perlawanan bersenjata) [SEP]
1	1 (1.00)	1	1.23	[CLS] apa akar dari semua kejahatan [SEP] secara pribadi , saya percaya akar dari semua kejahatan adalah ke ##ego ##isan dan ##ke ##ego ##isan diungkapkan dalam segala cara . sebaliknya adalah cinta yang mengorbankan diri sendiri . seseorang juga dapat berbicara tentang akar segala kejahatan dalam hal dusta . [SEP]

Gambar 4.3: Interpretasi dari model BERT_{CAT} dengan *integrated gradients*. Kata dengan warna hijau berarti kata tersebut berkontribusi positif terhadap hasil prediksi. Di lain sisi, kata yang berwarna merah berarti kata tersebut berkontribusi negatif terhadap hasil prediksi.

Model berasitekturkan BERT_{DOT} jauh lebih sulit untuk menginterpretasikannya. Hal

yang dapat dilakukan adalah dengan menghitung nilai *importance* kata dengan menghitung hasil kali titik representasi vektor dari teks dengan representasi vektor masing-masing kata pada teks tersebut. Dengan hal ini, dapat nilai *importance* dari kata dapat diurutkan. Tabel 4.19 menunjukkan kueri, teks dan 5 kata penting dari kueri dan teks tersebut.

Tabel 4.19: Interpretasi dari model BERT_{DOT} dengan menghitung hasil kali titik antara vektor teks dengan vektor masing-masing kata pada teks tersebut. Hanya 5 kata dengan nilai *importance* tertinggi yang ditunjukkan.

Kueri	**5 Kata Penting**	**Teks**	**5 Kata Penting**
Kapan Petrus Lombardus lahir?	[lahir, petrus, kapan, lombardus, ?]	Petrus Lombardus mungkin dilahirkan di Novara; atau kemungkinan lainnya adalah di Lumellogno[7] (saat itu sebuah komune pedesaan, sekarang menjadi bagian dari Provinsi Novara, Piemonte), di barat laut Italia, dari suatu keluarga miskin.[8] Kelahirannya diperkirakan antara tahun 1095-1100.	[petrus, dilahirkan, kelahirannya, lombardus, keluarga]
Dimana Jamie Richard Vardy lahir?	[lahir, richard, jamie, ?, dimana]	Jamie Richard Vardy (lahir dengan nama Gill; 11 January 1987) adalah pemain sepak bola Inggris yang bermain di klub Premiere League Leicester City dan tim nasional Inggris. Ia bermain sebagai striker, namun juga bisa bermain di sayap	[gill, lahir, richard, tim, sepak]

BAB 5
PENUTUP

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

5.1 Kesimpulan

Implementasi model *Bidirectional Encoder Representations from Transformers* (BERT) untuk pemeringkatan teks berbahasa Indonesia telah dilakukan pada Subbab 4. Berikut ini adalah kesimpulan terkait pekerjaan yang dilakukan dalam penelitian ini:

1. Berdasarkan penjelasan pada Subbab 3 dan implementasi pada Subbab 4 telah ditunjukkan dua cara penggunaan BERT untuk pemeringakatan teks, yaitu BERT sebagai *soft classifier* dari nilai relevansi (kueri, teks) dan BERT sebagai pemetaan teks ke dalam ruang vektor dengan nilai skor relevansi dihitung dengan fungsi *similarity* seperti jarak kosinus dan *dot product*.

2. Tabel 4.13 hingga Tabel 4.16 menunjukkan bahwa model BERT yang dilatih kembali (*fine tuning*) pada *dataset* Mmarco *train set* menghasilkan skor yang lebih baik dibandingkan dengan model *baseline* BM25 pada dua *dataset* uji Mmarco *dev set* dan MrTyDi *dev set*. Pada *dataset* Miracl *dev set*, hanya IndoBERT$_{\text{CAT}}$ yang menghasilkan skor yang lebih baik dibandingkan dengan model *baseline* BM25.

5.2 Saran

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

1. Pelatihan model BERT dapat dilakukan dengan *dataset* yang lebih beragam.

2. Memperbanyak *dataset* uji untuk pemeringkatan teks, sehingga dapat dilakukan analisis yang lebih mendalam terhadap setiap model yang dihasilkan.

3. Menambah jumlah model *baseline* untuk pemeringkatan teks. Beberapa model yang dapat ditambahkan adalah TF-IDF, Word2Vec, ELMo, dan arsitektur *non-transformer* seperti LSTM dan CNN.

DAFTAR REFERENSI

Abdillah, A. A., Murfi, H., & Satria, Y. (2015). Uji kinerja learning to rank dengan metode support vector regression. *IndoMS Journal on Industrial and Applied Mathematics*, *2*(1), 15–25.

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

Koto, F., Rahimi, A., Lau, J. H., & Baldwin, T. (2020). Indolem and indobert: A benchmark dataset and pre-trained language model for indonesian NLP. *CoRR*, *abs/2011.00677*. Diakses dari https://arxiv.org/abs/2011.00677

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

Lippe, P. (2022). *UvA Deep Learning Tutorials.* https://uvadlc-notebooks.readthedocs.io/en/latest/.

Liu, T.-Y. (2011). *Learning to rank for information retrieval* (1st ed.). Springer Publishing Company, Incorporated.

Murphy, K. P. (2022). *Probabilistic machine learning: An introduction*. MIT Press. Diakses dari probml.ai

Nguyen, T., Rosenberg, M., Song, X., Gao, J., Tiwary, S., Majumder, R., & Deng, L. (2016). MS MARCO: A human generated machine reading comprehension dataset. *CoRR*, *abs/1611.09268*. Diakses dari http://arxiv.org/abs/1611.09268

Nogueira, R. F., & Cho, K. (2019). Passage re-ranking with BERT. *CoRR*, *abs/1901.04085*. Diakses dari http://arxiv.org/abs/1901.04085

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)

Potts, C., Ethayarajh, K., Karamcheti, S., Lee, M., Li, S., Li, X. L., ... Ògúnremí, T. (2023). *Cs224u: Natural language understanding.* https://web.stanford.edu/class/cs224u/index.html.

Qu, Y., Ding, Y., Liu, J., Liu, K., Ren, R., Zhao, X., ... Wang, H. (2020). Rocketqa: An optimized training approach to dense passage retrieval for open-domain question answering. *CoRR*, *abs/2010.08191*. Diakses dari https://arxiv.org/abs/2010.08191

Reimers, N., & Gurevych, I. (2019, 11). Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 conference on empirical methods in natural language processing.* Association for Computational Linguistics. Diakses dari https://arxiv.org/abs/1908.10084

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

Robertson, S. E., Walker, S., Jones, S., Hancock-Beaulieu, M., & Gatford, M. (1994). Okapi at trec-3. In *Text retrieval conference.* Diakses dari https://api.semanticscholar.org/CorpusID:3946054

Sundararajan, M., Taly, A., & Yan, Q. (2017). Axiomatic attribution for deep networks.

CoRR, *abs/1703.01365*. Diakses dari http://arxiv.org/abs/1703.01365

Tsai, Y. H., Bai, S., Yamada, M., Morency, L., & Salakhutdinov, R. (2019). Transformer dissection: An unified understanding for transformer's attention via the lens of kernel. *CoRR*, *abs/1908.11775*. Diakses dari http://arxiv.org/abs/1908.11775

van den Oord, A., Li, Y., & Vinyals, O. (2018). Representation learning with contrastive predictive coding. *CoRR*, *abs/1807.03748*. Diakses dari http://arxiv.org/abs/1807.03748

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/

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., ... Dean, J. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, *abs/1609.08144*. Diakses dari http://arxiv.org/abs/1609.08144

Xiong, L., Xiong, C., Li, Y., Tang, K., Liu, J., Bennett, P. N., ... Overwijk, A. (2020). Approximate nearest neighbor negative contrastive learning for dense text retrieval. *CoRR*, *abs/2007.00808*. Diakses dari https://arxiv.org/abs/2007.00808

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: KODE SIMULASI

1. Repositori kode: `https://github.com/carlesoctav/beir-skripsi`
2. Repositori model dan data: `https://huggingface.co/carles-undergrad-thesis`
3. Repositori data (*raw*): `https://drive.google.com/drive/folders/1l_fbqJSn2AR8f-g1QnANl5czm2aLO0rO`

```
from time import time
from beir import util, LoggingHandler
from beir.retrieval import models
from beir.datasets.data_loader import GenericDataLoader
from beir.retrieval.evaluation import EvaluateRetrieval
from beir.retrieval.search.dense import DenseRetrievalExactSearch as DRES

import logging
import pathlib, os
import random
from pyprojroot import here


logging.basicConfig(format='%(asctime)s - %(message)s',
                    datefmt='%Y-%m-%d %H:%M:%S',
                    level=logging.INFO,
                    handlers=[LoggingHandler()])


corpus_path = str(here('datasets/miracl/corpus.jsonl'))
query_path = str(here('datasets/miracl/queries.jsonl'))
qrels_path = str(here('datasets/miracl/dev.tsv'))

corpus, queries, qrels = GenericDataLoader(
    corpus_file=corpus_path,
    query_file=query_path,
    qrels_file=qrels_path).load_custom()



model =
    DRES(models.SentenceBERT("carles-undergrad-thesis/indobert-mmarco-hardnegs-bm25"),
    batch_size=128)
retriever = EvaluateRetrieval(model, score_function="dot")
```

```
33
34
35 start_time = time()
36 results = retriever.retrieve(corpus, queries)
37 end_time = time()
38 print("Time taken to retrieve: {:.2f} seconds".format(end_time - start_time))
39
40
41 logging.info("Retriever evaluation for k in: {}".format(retriever.k_values))
42 ndcg, _map, recall, precision = retriever.evaluate(qrels, results, retriever.k_values)
43
44 mrr = retriever.evaluate_custom(qrels, results, retriever.k_values, metric="mrr")
45 recall_cap = retriever.evaluate_custom(qrels, results, retriever.k_values,
       metric="r_cap")
46 hole = retriever.evaluate_custom(qrels, results, retriever.k_values, metric="hole")
47
48 top_k = 10
49
50 query_id, ranking_scores = random.choice(list(results.items()))
51 scores_sorted = sorted(ranking_scores.items(), key=lambda item: item[1], reverse=True)
52 logging.info("Query : %s\n" % queries[query_id])
53
54 for rank in range(top_k):
55     doc_id = scores_sorted[rank][0]
56     logging.info("Rank %d: %s [%s] - %s\n" % (rank+1, doc_id,
       corpus[doc_id].get("title"), corpus[doc_id].get("text")))
```

Kode 1: Kode untuk mengevaluasi BERT_{DOT}

```
from beir import util, LoggingHandler
from beir.datasets.data_loader import GenericDataLoader
from beir.retrieval.evaluation import EvaluateRetrieval
from beir.retrieval.search.lexical import BM25Search as BM25
from beir.reranking.models import CrossEncoder
from beir.reranking import Rerank

import pathlib, os
import logging
import random
from pyprojroot import here


logging.basicConfig(format='%(asctime)s - %(message)s',
                    datefmt='%Y-%m-%d %H:%M:%S',
                    level=logging.INFO,
                    handlers=[LoggingHandler()])


corpus_path = str(here('datasets/mrtydi/indonesian/corpus.jsonl'))
query_path = str(here('datasets/mrtydi/indonesian/queries.jsonl'))
qrels_path = str(here('datasets/mrtydi/indonesian/qrels/dev.tsv'))

corpus, queries, qrels = GenericDataLoader(
    corpus_file=corpus_path,
    query_file=query_path,
    qrels_file=qrels_path).load_custom()

#### Provide parameters for Elasticsearch
hostname = "localhost" #localhost
index_name = "mrtydi-indo" # trec-covid
initialize = True # False
language = "indonesian"

model = BM25(index_name=index_name, hostname=hostname,
    initialize=initialize,language=language)
retriever = EvaluateRetrieval(model)

results = retriever.retrieve(corpus, queries)

cross_encoder_model =
    CrossEncoder('carles-undergrad-thesis/indobert-crossencoder-mmarco', max_length =
    512)

reranker = Rerank(cross_encoder_model, batch_size=256)

rerank_results = reranker.rerank(corpus, queries, results, top_k=100)

ndcg, _map, recall, precision = EvaluateRetrieval.evaluate(qrels, rerank_results,
    retriever.k_values)
```

```
47
48 mrr = retriever.evaluate_custom(qrels, rerank_results, retriever.k_values, metric="mrr")
49 recall_cap = retriever.evaluate_custom(qrels, rerank_results, retriever.k_values,
       metric="r_cap")
50 hole = retriever.evaluate_custom(qrels, rerank_results, retriever.k_values,
       metric="hole")
51
52 top_k = 10
53
54 query_id, ranking_scores = random.choice(list(rerank_results.items()))
55 scores_sorted = sorted(ranking_scores.items(), key=lambda item: item[1], reverse=True)
56 logging.info("Query : %s\n" % queries[query_id])
57
58 for rank in range(top_k):
59     doc_id = scores_sorted[rank][0]
60     logging.info("Rank %d: %s [%s] - %s\n" % (rank+1, doc_id,
       corpus[doc_id].get("title"), corpus[doc_id].get("text")))
```

Kode 2: Kode untuk mengevaluasi BERT_{CAT}

```
from beir import util, LoggingHandler
from beir.datasets.data_loader import GenericDataLoader
from beir.retrieval.evaluation import EvaluateRetrieval
from beir.retrieval.search.lexical import BM25Search as BM25

import pathlib, os, random
import logging
from pyprojroot import here

logging.basicConfig(format='%(asctime)s - %(message)s',
                    datefmt='%Y-%m-%d %H:%M:%S',
                    level=logging.INFO,
                    handlers=[LoggingHandler()])



corpus_path = str(here('datasets/miracl/corpus.jsonl'))
query_path = str(here('datasets/miracl/queries.jsonl'))
qrels_path = str(here('datasets/miracl/dev.tsv'))

corpus, queries, qrels = GenericDataLoader(
    corpus_file=corpus_path,
    query_file=query_path,
    qrels_file=qrels_path).load_custom()


hostname = "localhost"
index_name = "miracl-indo"

initialize = True

language = "indonesian"

number_of_shards = 1
model = BM25(index_name=index_name, hostname=hostname, language=language,
    initialize=initialize, number_of_shards=number_of_shards)
retriever = EvaluateRetrieval(model)

results = retriever.retrieve(corpus, queries)

logging.info("Retriever evaluation for k in: {}".format(retriever.k_values))
ndcg, _map, recall, precision = retriever.evaluate(qrels, results, retriever.k_values)

mrr = retriever.evaluate_custom(qrels, results, retriever.k_values, metric="mrr")
recall_cap = retriever.evaluate_custom(qrels, results, retriever.k_values,
    metric="r_cap")
hole = retriever.evaluate_custom(qrels, results, retriever.k_values, metric="hole")


query_id, scores_dict = random.choice(list(results.items()))
```

```
logging.info("Query : %s\n" % queries[query_id])

scores = sorted(scores_dict.items(), key=lambda item: item[1], reverse=True)
for rank in range(10):
    doc_id = scores[rank][0]
    logging.info("Doc %d: %s [%s] - %s\n" % (rank+1, doc_id,
    corpus[doc_id].get("title"), corpus[doc_id].get("text")))
```

Kode 3: Kode untuk mengevaluasi BM25

```python
import torch
import argparse
from datasets import load_dataset
from transformers import (
    Trainer,
    AutoModelForSequenceClassification,
    AutoTokenizer,
    AutoConfig,
    TrainingArguments,
)


if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--output_dir", required=True)
    parser.add_argument(
        "--model", default="indolem/indobert-base-uncased", type=str, required=False
    )

    parser.add_argument("--learning_rate", default=2e-5, type=float, required=False)
    parser.add_argument("--batch_size", default=16, type=int, required=False)
    parser.add_argument("--max_samples", default=250_000, type=int, required=False)
    args = parser.parse_args()

    tokenizer = AutoTokenizer.from_pretrained(args.model)
    config = AutoConfig.from_pretrained(args.model)
    config.num_labels = 1
    config.problem_type = "multi_label_classification"
    model = AutoModelForSequenceClassification.from_pretrained(
        args.model, config=config
    )

    dataset = load_dataset("carles-undergrad-thesis/indo-mmarco-500k")["train"]
    if args.max_samples:
        dataset = dataset.select(range(args.max_samples))

    def split_examples(batch):
        queries = []
        passages = []
        labels = []
        for label in ["positive", "negative"]:
            for (query, passage) in zip(batch["query"], batch[label]):
                queries.append(query)
                passages.append(passage)
                labels.append(int(label == "positive"))
        return {"query": queries, "passage": passages, "label": labels}

    dataset = dataset.map(
        split_examples, batched=True, remove_columns=["positive", "negative"]
    )
```

```
51
52      def tokenize(batch):
53          tokenized = tokenizer(
54              batch["query"],
55              batch["passage"],
56              padding=True,
57              truncation="only_second",
58              max_length=512,
59          )
60          tokenized["labels"] = [[float(label)] for label in batch["label"]]
61          return tokenized
62
63      dataset = dataset.map(
64          tokenize, batched=True, remove_columns=["query", "passage", "label"]
65      )
66      dataset.set_format("torch")
67
68      print(len(dataset))
69
70      training_args = TrainingArguments(
71          output_dir=args.output_dir,
72          # fp16=True,
73          # fp16_backend="amp",
74          per_device_train_batch_size=args.batch_size,
75          logging_steps=10_000,
76          warmup_steps=0.1*len(dataset)*args.batch_size,
77          save_total_limit=1,
78          num_train_epochs=1,
79
80      )
81      trainer = Trainer(
82          model,
83          training_args,
84          train_dataset=dataset,
85          tokenizer=tokenizer,
86      )
87
88      train_result = trainer.train()
89      trainer.save_model()
```

Kode 4: Kode untuk melatih IndoBERT$_{\text{CAT}}$

```python
from sentence_transformers import losses, models, SentenceTransformer
from beir import util, LoggingHandler
from beir.datasets.data_loader import GenericDataLoader
from beir.retrieval.train import TrainRetriever
import pathlib, os
import logging
from pyprojroot import here
import torch

logging.basicConfig(format='%(asctime)s - %(message)s',
                    datefmt='%Y-%m-%d %H:%M:%S',
                    level=logging.INFO,
                    handlers=[LoggingHandler()])

dataset = "mmarco"
corpus_path = str(here('datasets/mmarco/indonesian/corpus.jsonl'))
query_path = str(here('datasets/mmarco/indonesian/queries.jsonl'))
qrels_path = str(here('datasets/mmarco/indonesian/qrels/dev.tsv'))

corpus, queries, qrels = GenericDataLoader(
    corpus_file=corpus_path,
    query_file=query_path,
    qrels_file=qrels_path).load_custom()

model_name = "indolem/indobert-base-uncased"
word_embedding_model = models.Transformer(model_name, max_seq_length=256)
pooling_model = models.Pooling(word_embedding_model.get_word_embedding_dimension(),
    pooling_mode = "cls")
model = SentenceTransformer(modules=[word_embedding_model, pooling_model])


retriever = TrainRetriever(model=model, batch_size=32)

train_samples = retriever.load_train(corpus, queries, qrels)
train_dataloader = retriever.prepare_train(train_samples, shuffle=True)

train_loss = losses.MultipleNegativesRankingLoss(model=retriever.model, scale=1.0,
    similarity_fct=util.dot_score)

ir_evaluator = retriever.load_dummy_evaluator()

model_save_path = os.path.join(pathlib.Path(__file__).parent.absolute(), "output",
    "{}-v1-{}".format(model_name, dataset))
os.makedirs(model_save_path, exist_ok=True)


num_epochs = 5
evaluation_steps = 1_000_000
warmup_steps = int(len(train_samples) * num_epochs / retriever.batch_size * 0.1)

```

```
retriever.fit(train_objectives=[(train_dataloader, train_loss)],
                evaluator=ir_evaluator,
                epochs=num_epochs,
                output_path=model_save_path,
                warmup_steps=warmup_steps,
                evaluation_steps=evaluation_steps,
                use_amp=True)


# model.save_to_hub("st-indobert-mmarco-v1", "carles-undergrad-thesis")
```

Kode 5: Kode untuk melatih IndoBERT$_{\text{DOT}}$

```
import tensorflow as tf
from transformers import TFXLMRobertaModel, AutoTokenizer, TFAutoModel
from datasets import load_dataset
from datetime import datetime
import logging
from pyprojroot.here import here
import os

class mean_pooling_layer(tf.keras.layers.Layer):
    def __init__(self):
        super(mean_pooling_layer, self).__init__()

    def call(self, inputs):
        token_embeddings = inputs[0]
        attention_mask = inputs[1]
        input_mask_expanded = tf.cast(
            tf.broadcast_to(tf.expand_dims(attention_mask, -1),
    tf.shape(token_embeddings)),
            tf.float32
        )

        embeddings = tf.math.reduce_sum(token_embeddings * input_mask_expanded, axis=1)
    / tf.clip_by_value(tf.math.reduce_sum(input_mask_expanded, axis=1), 1e-9,
    tf.float32.max)
        return embeddings


    def get_config(self):
        config = super(mean_pooling_layer, self).get_config()
        return config


def create_model():
    base_student_model =
    TFAutoModel.from_pretrained("distilbert-base-multilingual-cased",from_pt=True)
    input_ids_en = tf.keras.layers.Input(shape=(256,),name='input_ids_en',
    dtype=tf.int32)
    attention_mask_en = tf.keras.layers.Input(shape=(256,), name='attention_mask_en',
    dtype=tf.int32)
    input_ids_id = tf.keras.layers.Input(shape=(256,),name='input_ids_id',
    dtype=tf.int32)
    attention_mask_id = tf.keras.layers.Input(shape=(256,), name='attention_mask_id',
    dtype=tf.int32)
    mean_pooling = mean_pooling_layer()

    output_en = base_student_model.distilbert(input_ids_en,
    attention_mask=attention_mask_en).last_hidden_state[:,0,:]
    output_id = base_student_model.distilbert(input_ids_id,
    attention_mask=attention_mask_id).last_hidden_state[:,0,:]

```

```
    student_model = tf.keras.Model(inputs=[input_ids_en, attention_mask_en, input_ids_id, attention_mask_id], outputs=[output_en, output_id])
    print(student_model.summary())
    return student_model

class sentence_translation_metric(tf.keras.callbacks.Callback):
    def on_epoch_end(self,epoch,logs):
        embeddings_en, embeddings_id = self.model.predict(val_dataset, verbose=1)
        # get the embeddings
        # compute the cosine similarity between the two
        #normalize the embeddings
        similarity_matrix = tf.matmul(embeddings_en, embeddings_id, transpose_b=True)
        print(f"==>> similarity_matrix: {similarity_matrix}")
        # get the mean similarity
        correct_en_id = 0
        for i in range(similarity_matrix.shape[0]):
            if tf.math.argmax(similarity_matrix[i]) == i:
                correct_en_id += 1

        similarity_matrix_T = tf.transpose(similarity_matrix)
        correct_id_en = 0
        for i in range(similarity_matrix_T.shape[0]):
            if tf.math.argmax(similarity_matrix_T[i]) == i:
                correct_id_en += 1

        acc_en_id = correct_en_id / similarity_matrix.shape[0]
        acc_id_en = correct_id_en / similarity_matrix_T.shape[0]
        avg_acc = (acc_en_id + acc_id_en) / 2
        print(f"translation accuracy from english to indonesian = {acc_en_id}")
        print(f"translation accuracy from indonesian to english = {acc_id_en}")
        print(f"average translation accuracy = {avg_acc}")

        logs["val_acc_en_id"] = acc_en_id
        logs["val_acc_id_en"] = acc_id_en
        logs["val_avg_acc"] = avg_acc


class ConstantScheduler(tf.keras.optimizers.schedules.LearningRateSchedule):
  def __init__(self, max_lr, warmup_steps=5000):
    super().__init__()
    self.max_lr = tf.cast(max_lr, tf.float32)
    self.warmup_steps = warmup_steps

  def __call__(self, step):
    step = tf.cast(step, tf.float32)
    condition  = tf.cond(step < self.warmup_steps, lambda: step / self.warmup_steps, lambda: 1.0)
    return self.max_lr * condition


```

```
if __name__ == "__main__":
    num_data = 0
    dataset = load_dataset("carles-undergrad-thesis/en-id-parallel-sentences-embedding")

    dataset_1 = dataset["train"]

    # for split in dataset:
    #     dataset_1 = concatenate_datasets([dataset_1, dataset[split]])


    batch_size = 512
    dataset = dataset_1.train_test_split(test_size=0.01, shuffle=True)
    train_dataset = dataset["train"]
    val_dataset = dataset["test"]

    print(f"==>> val_dataset.shape: {val_dataset.shape}")

    train_dataset = train_dataset.to_tf_dataset(
        columns=["input_ids_en", "attention_mask_en", "input_ids_id",
    "attention_mask_id"],
        label_cols="target_embedding",
        batch_size=batch_size,
    ).unbatch()

    val_dataset = val_dataset.to_tf_dataset(
        columns=["input_ids_en", "attention_mask_en", "input_ids_id",
    "attention_mask_id"],
        label_cols="target_embedding",
        batch_size=batch_size,
    ).unbatch()

    #check feature
    print(train_dataset.element_spec)
    print(val_dataset.element_spec)

    train_dataset = train_dataset.batch(batch_size, drop_remainder=True).cache()
    val_dataset = val_dataset.batch(batch_size, drop_remainder=True).cache()


    warm_up_steps = 5_000_000 / batch_size *0.1
    learning_rate = ConstantScheduler(2e-5, warmup_steps= warm_up_steps)

    optimizer = tf.keras.optimizers.Adam(learning_rate, beta_1=0.9, beta_2=0.98,
                                         epsilon=1e-9)


    loss = tf.keras.losses.MeanSquaredError()

    date_time = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")

```

```
output_path = here(f"disk/model/{date_time}/model.h5")

if not os.path.exists(here("disk/model")):
    os.makedirs(here("disk/model"))

model_checkpoint = tf.keras.callbacks.ModelCheckpoint(
                filepath = output_path,
                save_weights_only = True,
                monitor = "val_avg_acc",
                mode = 'auto',
                verbose = 1,
                save_best_only = True,
                initial_value_threshold = 0.5,
                )


# tensor_board  = tf.keras.callbacks.TensorBoard(
#                   log_dir = "gs://dicoding-capstone/output/logs/"+date_time
# )

if not os.path.exists(here("disk/performance_logs")):
    os.makedirs(here("disk/performance_logs"))


csv_logger = tf.keras.callbacks.CSVLogger(
                filename = here(f"disk/performance_logs/log-{date_time}.csv"),
                separator = ",",
                append = False
)


callbacks = [sentence_translation_metric(), model_checkpoint, csv_logger]


cluster_resolver = tf.distribute.cluster_resolver.TPUClusterResolver("local")
tf.config.experimental_connect_to_cluster(cluster_resolver)
tf.tpu.experimental.initialize_tpu_system(cluster_resolver)
strategy = tf.distribute.TPUStrategy(cluster_resolver)

with strategy.scope():
    student_model = create_model()
    student_model.compile(optimizer=optimizer, loss=loss)

student_model.fit(train_dataset, epochs=5, validation_data=val_dataset,
callbacks=callbacks)

last_epoch_save = here(f"disk/model/last_epoch/{date_time}.h5")

if not os.path.exists(here("disk/model/last_epoch")):
    os.makedirs(here("disk/model/last_epoch"))
```

```
186
187     student_model.save_weights(last_epoch_save)
```

Kode 6: Kode untuk melatih IndoBERT$_{\text{DOTKD}}$

```
from sentence_transformers import SentenceTransformer, models, losses, InputExample
from beir import util, LoggingHandler
from beir.datasets.data_loader import GenericDataLoader
from beir.retrieval.train import TrainRetriever
from torch.utils.data import Dataset
from tqdm.autonotebook import tqdm
import pathlib, os, gzip, json
import logging
import random
from pyprojroot import here

logging.basicConfig(format='%(asctime)s - %(message)s',
                    datefmt='%Y-%m-%d %H:%M:%S',
                    level=logging.INFO,
                    handlers=[LoggingHandler()])


dataset = "mmarco"
corpus_path = str(here('datasets/mmarco/indonesian/corpus.jsonl'))
query_path = str(here('datasets/mmarco/indonesian/queries.jsonl'))
qrels_path = str(here('datasets/mmarco/indonesian/qrels/train.tsv'))

corpus, queries, _ = GenericDataLoader(
    corpus_file=corpus_path,
    query_file=query_path,
    qrels_file=qrels_path).load_custom()


train_batch_size = 32
max_seq_length = 250
num_negs = 5

triplets_url = "https://sbert.net/datasets/msmarco-hard-negatives.jsonl.gz"
msmarco_triplets_filepath = os.path.join("datasets", "msmarco-hard-negatives.jsonl.gz")
if not os.path.isfile(msmarco_triplets_filepath):
    util.download_url(triplets_url, msmarco_triplets_filepath)

logging.info("Loading MSMARCO hard-negatives...")

train_queries = {}
missing_bm25_counter = 0
with gzip.open(msmarco_triplets_filepath, 'rt', encoding='utf8') as fIn:
    for line in tqdm(fIn):
        data = json.loads(line)
        pos_pids = [item['pid'] for item in data['pos']]
        qid = data['qid']
        neg_pids = set()
        try:
            for item in data['neg']["bm25"]:
                pid = item['pid']
```

```
                neg_pids.add(pid)

                if len(neg_pids) >= num_negs:
                    break
        except:
            missing_bm25_counter += 1
            continue

        if len(pos_pids) > 0 and len(neg_pids) > 0:
            train_queries[qid] = {'query': queries[qid], 'pos': pos_pids, 'hard_neg':
    list(neg_pids)}

print("Missing BM25 counter: {}".format(missing_bm25_counter))
logging.info("Train queries: {}".format(len(train_queries)))

# We create a custom MSMARCO dataset that returns triplets (query, positive, negative)
# on-the-fly based on the information from the mined-hard-negatives jsonl file.

class MSMARCODataset(Dataset):
    def __init__(self, queries, corpus):
        self.queries = queries
        self.queries_ids = list(queries.keys())
        self.corpus = corpus

        for qid in self.queries:
            self.queries[qid]['pos'] = list(self.queries[qid]['pos'])
            self.queries[qid]['hard_neg'] = list(self.queries[qid]['hard_neg'])
            random.shuffle(self.queries[qid]['hard_neg'])

    def __getitem__(self, item):
        query = self.queries[self.queries_ids[item]]
        query_text = query['query']

        pos_id = query['pos'].pop(0)
        pos_text = self.corpus[pos_id]["text"]
        query['pos'].append(pos_id)

        neg_id = query['hard_neg'].pop(0)
        neg_text = self.corpus[neg_id]["text"]
        query['hard_neg'].append(neg_id)

        return InputExample(texts=[query_text, pos_text, neg_text])

    def __len__(self):
        return len(self.queries)

model_name = "indolem/indobert-base-uncased"
word_embedding_model = models.Transformer(model_name, max_seq_length=max_seq_length)
pooling_model = models.Pooling(word_embedding_model.get_word_embedding_dimension(),
    pooling_mode = "cls")
```

```
model = SentenceTransformer(modules=[word_embedding_model, pooling_model])


retriever = TrainRetriever(model=model, batch_size=train_batch_size)
train_dataset = MSMARCODataset(train_queries, corpus=corpus)
train_dataloader = retriever.prepare_train(train_dataset, shuffle=True,
    dataset_present=True)
train_loss = losses.MultipleNegativesRankingLoss(model=retriever.model,
    similarity_fct=util.dot_score, scale=1)
ir_evaluator = retriever.load_dummy_evaluator()

model_save_path = os.path.join(pathlib.Path(__file__).parent.absolute(), "output",
    "{}-hardnegs-{}".format(model_name, dataset))
os.makedirs(model_save_path, exist_ok=True)


num_epochs = 5
evaluation_steps = 10000
warmup_steps = int(0.1 * num_epochs * len(train_dataset) / train_batch_size)

retriever.fit(train_objectives=[(train_dataloader, train_loss)],
                epochs=num_epochs,
                output_path=model_save_path,
                evaluation_steps=evaluation_steps,
                use_amp=True)


model.save_to_hub("indobert-mmarco-hardnegs-bm25", "carles-undergrad-thesis")
```

Kode 7: Kode untuk melatih IndoBERT$_{\text{DOThardnegs}}$ dengan *hard negatives*

LAMPIRAN 2: *OUTPUT* DARI SIMULASI

Terlampir contoh *output* pengevaluasian dari suatu model, Informasi lebih lanjut dapat dilihat di `https://github.com/carlesoctav/beir-skripsi`.

```
100%|████████████████████████████████████████████████████████████|
8841823/8841823 [00:43<00:00, 2045[107/1908]
2023-10-20 10:45:58 - Loaded 8841823 DEV Documents.
2023-10-20 10:45:58 - Doc Example: {'text': 'Kehadiran komunikasi di
tengah pikiran ilmiah sama pentingnya dengan kebe
rhasilan Proyek Manhattan seperti kecerdasan ilmiah. Kehadiran komunikasi
di tengah pikiran ilmiah sama pentingnya den
gan keberhasilan Proyek Manhattan seperti kecerdasan ilmiah. Satu -
satunya awan yang menggantung di atas prestasi men
gesankan para peneliti dan insinyur atom adalah apa sebenarnya tujuan
kesuksesan mereka; ratusan ribu nyawa yang tidak
 bersalah dilenyapkan.', 'title': ''}
2023-10-20 10:45:58 - Loading Queries...
2023-10-20 10:46:01 - Loaded 6980 DEV Queries.
2023-10-20 10:46:01 - Query Example: Berapa tahun William Bradford
melayani sebagai gubernur koloni Plymouth?
2023-10-20 10:46:01 - Activating Elasticsearch....
2023-10-20 10:46:01 - Elastic Search Credentials: {'hostname':
'localhost', 'index_name': 'mmarco-indo', 'keys': {'tit
le': 'title', 'body': 'txt'}, 'timeout': 100, 'retry_on_timeout': True,
'maxsize': 24, 'number_of_shards': 'default',
'language': 'indonesian'}
que:   0%|
| 0/55 [00:00<?, ?it/s]
/usr/local/lib/python3.8/dist-
packages/elasticsearch/connection/base.py:190:
ElasticsearchDeprecationWarning: Elastics
earch built-in security features are not enabled. Without authentication,
your cluster could be accessible to anyone.
See
https://www.elastic.co/guide/en/elasticsearch/reference/7.17/security-
minimal-setup.html to enable security.
  warnings.warn(message, category=ElasticsearchDeprecationWarning)
que:
100%|████████████████████████████████████████████████████████████
████████| 55/55 [01:01<00:00,  1.12s/it]
2023-10-20 10:47:05 - Use pytorch device: cuda
2023-10-20 10:47:45 - Starting To Rerank Top-1000....
Batches:   2%|▉
| 475/26727 [02:21<2:31:56,  2.88it/s]
Batches:   2%|▉
| 476/26727 [02:21<2:20:09,  3.12it/s]
Batches:
100%|████████████████████████████████████████████████████████████|
26727/26727 [2:11:06<00:00,  3.40it/s]
2023-10-20 13:01:31 - For evaluation, we ignore identical query and
document ids (default), please explicitly set ``ig

2023-10-20 10:47:05 - Use pytorch device: cuda
[87/1908]
2023-10-20 10:47:45 - Starting To Rerank Top-1000....
Batches:   2%|▉
| 475/26727 [02:21<2:31:56,  2.88it/s]
```

```
Batches:   2%|▉
| 476/26727 [02:21<2:20:09,  3.12it/s]
Batches:
100%|████████████████████████████████████████████████████████████|
26727/26727 [2:11:06<00:00,  3.40it/s]
2023-10-20 13:01:31 - For evaluation, we ignore identical query and
document ids (default), please explicitly set ``ig
nore_identical_ids=False`` to ignore this.
2023-10-20 13:01:36 -

2023-10-20 13:01:36 - NDCG@1: 0.1052
2023-10-20 13:01:36 - NDCG@3: 0.1667
2023-10-20 13:01:36 - NDCG@5: 0.1940
2023-10-20 13:01:36 - NDCG@10: 0.2243
2023-10-20 13:01:36 - NDCG@100: 0.2707
2023-10-20 13:01:36 - NDCG@1000: 0.2786
2023-10-20 13:01:36 -

2023-10-20 13:01:36 - MAP@1: 0.1019
2023-10-20 13:01:36 - MAP@3: 0.1500
2023-10-20 13:01:36 - MAP@5: 0.1652
2023-10-20 13:01:36 - MAP@10: 0.1778
2023-10-20 13:01:36 - MAP@100: 0.1874
2023-10-20 13:01:36 - MAP@1000: 0.1878
2023-10-20 13:01:36 -

2023-10-20 13:01:36 - Recall@1: 0.1019

2023-10-20 13:01:36 - Recall@1: 0.1019
[62/1908]
2023-10-20 13:01:36 - Recall@3: 0.2105
2023-10-20 13:01:36 - Recall@5: 0.2765
2023-10-20 13:01:36 - Recall@10: 0.3688
2023-10-20 13:01:36 - Recall@100: 0.5829
2023-10-20 13:01:36 - Recall@1000: 0.6421
2023-10-20 13:01:36 -

2023-10-20 13:01:36 - P@1: 0.1052
2023-10-20 13:01:36 - P@3: 0.0731
2023-10-20 13:01:36 - P@5: 0.0576
2023-10-20 13:01:36 - P@10: 0.0386
2023-10-20 13:01:36 - P@100: 0.0062
2023-10-20 13:01:36 - P@1000: 0.0007
2023-10-20 13:01:36 -

2023-10-20 13:01:39 - MRR@1: 0.1047
2023-10-20 13:01:39 - MRR@3: 0.1538
2023-10-20 13:01:39 - MRR@5: 0.1689
2023-10-20 13:01:39 - MRR@10: 0.1814
2023-10-20 13:01:39 - MRR@100: 0.1908
2023-10-20 13:01:39 - MRR@1000: 0.1912
2023-10-20 13:01:40 -

2023-10-20 13:01:43 - R_cap@1: 0.1047
```

```
2023-10-20 13:01:43 - R_cap@3: 0.2101

2023-10-20 13:01:43 - R_cap@5: 0.2760
[36/1908]
2023-10-20 13:01:43 - R_cap@10: 0.3680
2023-10-20 13:01:43 - R_cap@100: 0.5817
2023-10-20 13:01:43 - R_cap@1000: 0.6408
2023-10-20 13:01:43 -

2023-10-20 13:01:47 - Hole@1: 0.8920
2023-10-20 13:01:47 - Hole@3: 0.9235
2023-10-20 13:01:47 - Hole@5: 0.9386
2023-10-20 13:01:47 - Hole@10: 0.9564
2023-10-20 13:01:47 - Hole@100: 0.9816
2023-10-20 13:01:47 - Hole@1000: 0.9786
2023-10-20 13:01:47 - Query : Apa akar dari semua kejahatan

2023-10-20 13:01:47 - Rank 1: 7213594 [] - Secara pribadi, saya percaya
akar dari semua kejahatan adalah keegoisan dan
 keegoisan diungkapkan dalam segala cara. Sebaliknya adalah cinta yang
mengorbankan diri sendiri. Seseorang juga dapat
 berbicara tentang akar segala kejahatan dalam hal dusta. ♪ And to my
mind the root of evil is to deny Godvine reveal
♪

2023-10-20 13:01:47 - Rank 2: 7213591 [] - Akar dari semua setan berasal
dari manusia. Manusia itu sendiri adalah akar
 dari semua kejahatan. Sekarang saya tidak mencoba untuk mengatakan bahwa
setiap pria, wanita, dan anak secara inheren
 jahat tetapi tidak dapat dibantah bahwa bahkan tidak akan ada konsep
kejahatan jika bukan untuk manusia.

2023-10-20 13:01:47 - Rank 3: 5451019 [] - """Ya, karena cinta akan uang
adalah akar dari semua kejahatan,"" artinya,
bukan berarti setiap kejahatan harus berasal dari ""cinta uang,"" tetapi
bahwa tidak ada yang dapat dibayangkan kejaha
tan yang dapat terjadi pada anak laki-laki dan perempuan dari laki-laki
yang tidak mungkin musim semi dari keserakahan

2023-10-20 13:01:47 - Rank 3: 5451019 [] - """Ya, karena cinta akan uang
adalah akar dari semua kejahatan,"" [13/1908]
bukan berarti setiap kejahatan harus berasal dari ""cinta uang,"" tetapi
bahwa tidak ada yang dapat dibayangkan kejaha
tan yang dapat terjadi pada anak laki-laki dan perempuan dari laki-laki
yang tidak mungkin musim semi dari keserakahan
 cinta emas dan kekayaan."

2023-10-20 13:01:47 - Rank 4: 8380731 [] - Contoh Kalimat & Contoh. 1
Simone Weil: Kejahatan adalah akar misteri, rasa
 sakit adalah akar dari pengetahuan. Jadi menurutmu uang adalah akar dari
semua kejahatan. Apakah Anda pernah bertanya
 apa akar dari semua uang. 3 Med Yones: Dalam tradisi agama, cinta akan
uang adalah akar segala kejahatan. Dalam ekono
mi, kelangkaan adalah akar dari semua kejahatan.
```

2023-10-20 13:01:47 - Rank 5: 3611440 [] - Simone Weil: Kejahatan adalah akar misteri, rasa sakit adalah akar dari pen
getahuan. Jadi menurutmu uang adalah akar dari semua kejahatan. Apakah Anda pernah bertanya apa akar dari semua uang.
Med Yones: Dalam tradisi agama, cinta akan uang adalah akar segala kejahatan.

2023-10-20 13:01:47 - Rank 6: 1599590 [] - "Ringkasan Cepat. Kata akar Latin yang berarti ""buruk"" atau ""jahat."" Ak
ar ini adalah kata yang berasal dari banyak kosakata bahasa Inggris, termasuk mal, mal treat, dan mal ice. Anda dapat
mengingat bahwa mal berarti ""buruk"" melalui fungsi mal, atau ""buruk"" bekerja bagian, dan bahwa itu berarti ""jahat
"" melalui es mal, atau sengaja ""jahat"" dilakukan untuk yang lain."

2023-10-20 13:01:47 - Rank 7: 4715621 [] - akar akar akar: akar utama tanaman, yang langsung mengalir ke bumi hingga k
edalaman yang cukup dalam tanpa membelah. Akar kejahatan, akar akar dari yang berkembang kejahatan masyarakat modern,
adalah ide laba. Salah satunya adalah rasa hormat terhadap otoritas, yang hilang adalah akar dari Bolshevism.

2023-10-20 13:01:47 - Rank 8: 7867039 [] - Apa akar segala macam kejahatan? Jawaban 1 Ada kepercayaan bahwa uang adala
h akar dari semua jahat Jawaban 2 1 Timotius 6:10 - Karena cinta uang adalah akar dari segala macam kejahatan, untuk y
ang beberapa ha... telah menyimpang dari iman dalam keserakahan mereka, dan menusuk diri melalui dengan banyak penderi

2023-10-20 13:01:47 - Rank 9: 5451020 [] - 10. Cinta akan uang bukan uang itu sendiri, tapi cinta itu sendiri, keingin
an untuk menjadi kaya (1Ti 6:9) adalah akar (Ellicott dan Middleton: bukan sebagai English Version, 'akar') dari semua
 kejahatan. (So the Greek plural). Orang yang paling kaya mungkin tidak kaya dalam arti yang buruk; yang termiskin mun
gkin ingin menjadi begitu (Ps 62:10).

2023-10-20 13:01:47 - Rank 10: 583029 [] - "Ringkasan Cepat. Kata akar Latin yang berarti ""buruk"" atau ""jahat."" Ak
ar ini adalah asal usul dari banyak kosakata bahasa Inggris, termasuk kata - kata yang salah bentuk, perlakuan yang sa
lah, dan niat jahat. Anda dapat mengingat bahwa mal berarti ""buruk"" melalui kerusakan, atau ""buruk"" bagian kerja,
dan bahwa itu berarti ""jahat"" melalui kedengkian, atau sengaja ""jahat"" dilakukan untuk yang lain."