# BAB II

# TINJAUAN PUSTAKA

## 2.1. Penelitian Terkait

### 2.1.1. *State-of-the-Art* BERT

BERT merupakan sebuah *pretrained language model*, sebuah arsitektur, dan juga dapat dikatakan sebagai sebuah metode *state-of-the-art* dalam dunia *Natural Language Processing*, yang diperkenalkan dan dikembangkan oleh Google pada akhir tahun 2018 [5]. BERT memiliki peranan yang cukup penting dalam banyak pekerjaan NLP seperti *Question Answering*, *Named Entity Recognition*, *Natural Language Inference*, *Text Classification*, dan masih banyak lainnya. BERT merupakan singkatan dari *Bidirectional Encoder Representation from Transformers*.

Kilas balik pada tahun 2017, Vaswani dan rekannya mempublikasikan sebuah *paper* yang berjudul *Attention Is All You Need* [6]. *Paper* tersebut berisi mengenai sebuah struktur berbasis *attention* untuk menangani masalah yang berhubungan dengan *sequence model*, seperti *machine translation*. Secara tradisional, *neural machine translation* pada umumnya menggunakan *Recurrent Neural Network* (RNN) atau *Convolutional Neural Network* (CNN) sebagai dasar dari *encoder* dan *decoder* model. Namun, model *transformer* yang berbasis *attention* pada penelitian ini, tidak lagi menggunakan tradisional RNN dan CNN. Model tersebut berjalan sepenuhnya secara paralel, hingga membuatnya menjadi model yang cepat dan juga memiliki performa yang meningkat.

### 2.1.2. Mekanisme *Attention*

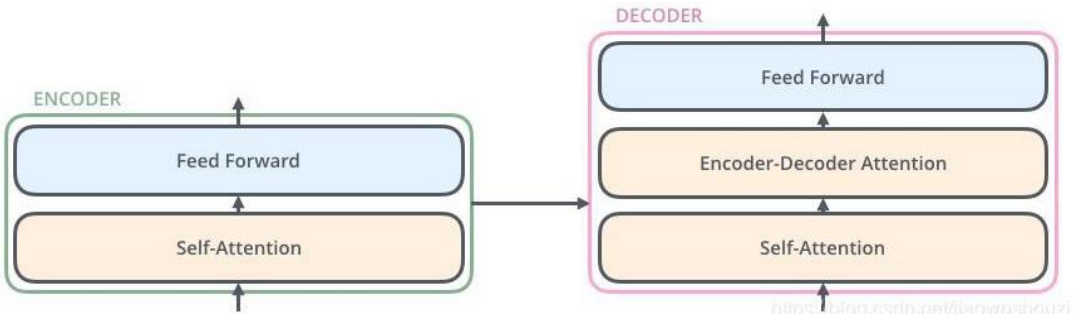
Mekanisme *attention* terdiri dari beberapa *hidden state* dari model, dan model tersebut memilih obyek mana yang akan diambil dari kumpulan *input sequnce*. Penting untuk mengetahui bagaimana cara kerja model seq2seq sebelum memasuki penjelasan *attention*. *Machine translation* secara tradisional berdasarkan sesuai dengan model seq2seq. Model tersebut terbagi menjadi sebuah *encoder* dan *decoder*, dan memiliki bagian RNN atau semacamnya seperti LSTM, GRU, dan lainnya. Vektor *encoder* adalah bagian akhir dari *hidden state* yang dibentuk dari bagian *encoder model*. Vektor ini bertujuan untuk mengenkapsulasi informasi untuk semua elemen *input* untuk membantu *decoder* membuat prediksi yang akurat. Hal tersebut menjadi seolah-olah *hidden state* insial dari bagian *decoder* merupakan bagian dari model. Masalah utama pada seq2seq model adalah pada saat seluruh objek masukan harus dikompres menjadi sebuah vektor yang memiliki ukuran pasti [7]. Jika objek atau dalam bentuk teks tersebut memiliki ukuran yang melebih dari yang seharusnya, hal tersebut menyebabkan kemungkinan untuk kehilangan informasi dari teks tersbut. *Attention* memiliki solusi dari masalah tersebut. Mekanisme *attention* mengatasi masalah ini dengan memungkinkan *decoder* untuk memeriksa kembali pada *hidden state* dari sumber *sequence*, kemudian menambahkan *weighted average* sebagai *input* tambahan pada *decoder*-nya. Dengan *attention*, model memilih konteks yang paling cocok dengan *node* yang sedang berjalan saat itu, sebagai *input* selama fase *decoder* berlangsung. Terdapat dua perbedaan di antara *attention* dan model tradisional seq2seq. Pertama, *encoder* menyediakan data yang lebih banyak pada *decoder*, dan

*encoder*-nya akan menyediakan *hidden state* dari semua *node* pada *decoder*, tidak hanya *hidden state* dari *node* terakhir dari *encoder*.

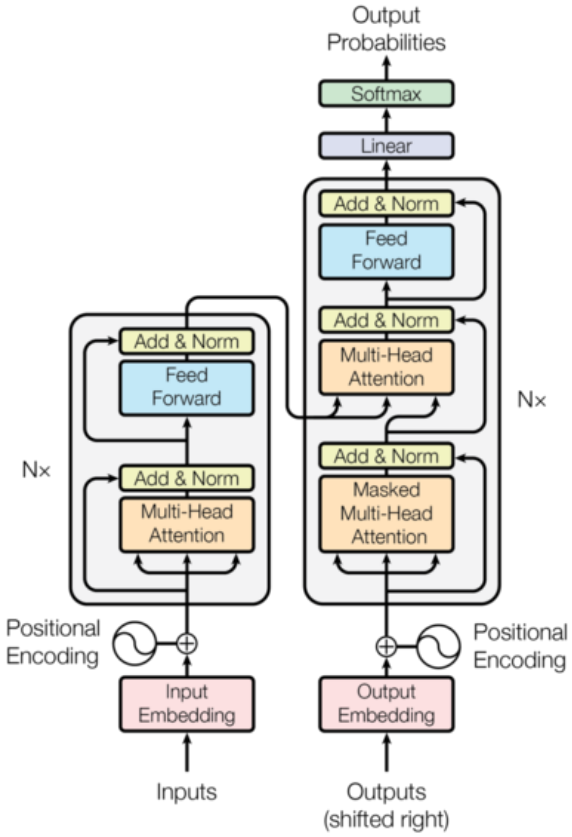
Kedua, *decoder* tidak secara langsung menggunakan *hidden state* yang disediakan oleh semua *encoder* sebagai *input*, namun mengadopsi sebuah mekanisme pilihan untuk memilih *hidden state* yang paling cocok dengan posisi saat itu. Untuk melakukannya, *decoder* menentukkan *hidden state* mana yang paling berhubungan dengan *node* tersebut dengan menghitung nilai dari setiap *hidden state* dan melakukan perhitungan *softmax* dari nilai tersebut, yang memungkinkan korelasi erat dari *hidden state* untuk memiliki nilai pecahan yang lebih besar, dan yang kurang relevan memiliki nilai pecahan yang lebih rendah. Kemudian, ia mengalikan setiap *hidden state* dengan nilai yang telah di-*softmax*, sehingga memperkuat *hidden state* dengan nilai tertinggi, dan melemahkan *hidden state* dengan nilai rendah. Penilaian ini dilakukan untuk setiap *time step* pada sisi *decoder* .

### 2.1.3. Mekanisme *Transformer*

Model *transformer* menggunakan arsitektur *encoder-decoder*. Pada *paper* BERT yang dipublikasikan oleh Google, jumlah *layer* pada *encoder* sama dengan jumlah *layer* pada *decoder*.




Gambar 1. Encoder dan decoder pada mekanisme attention [8]




Gambar 2. Arsitektur dari transformer [6]

*Encoder* terdiri dari dua bagian, *self-attention layer* dan sebuah *feed forward neural network*. *Self-attention* membantu suatu *node*, tidak hanya fokus pada satu kata, namun juga mendapatkan unsur semantik dari sebuah konteks. *Decoder* juga memiliki dua *layer network* yang sama seperti *encoder*, namun terdapat juga sebuah *attention layer* di antara dua *layer* tersebut untuk membantu suatu *node* untuk mendapatkan konten inti yang membutuhkan *attention*. Pada Gambar 2. merupakan illustrasi dari arsitektur *transformer*.

### 2.1.4. Mekanisme *Self-Attention* pada *Transformer*

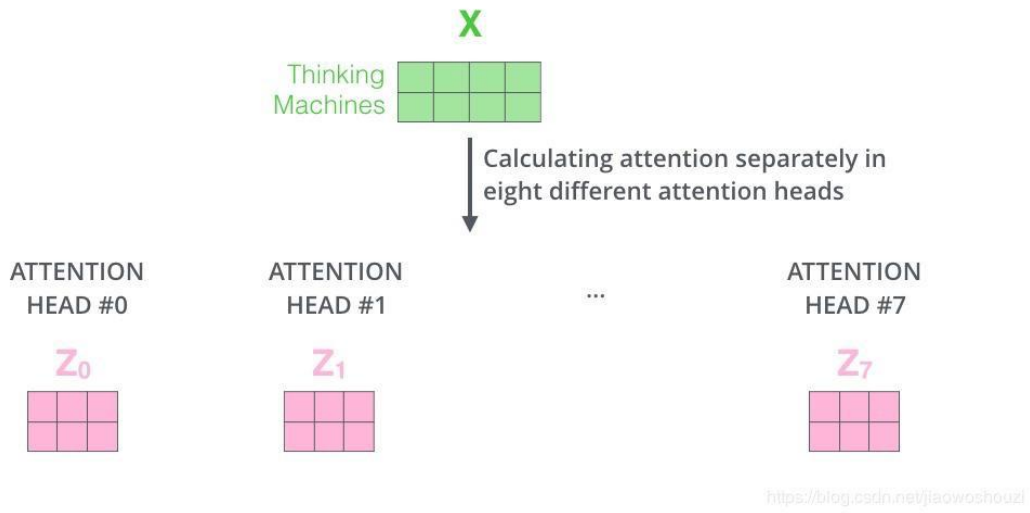
*Self-attention* merupakan cara dari *transformer* untuk mengubah pengartian dari beberapa kata yang saling berhubungan ke dalam kata yang sedang ditangani saat itu. Pertama, *self-attention* menghitung tiga vektor baru. Pada *paper* tersebut, dimensi dari vektor adalah 512. Ketiga vektor tersebut disebut *query*, *key*, dan *value*. Ketiga vektor tersebut dihasilkan dengan mengalikan vektor dari *word embedding* dan secara acak menginisialisasi sebuah *matrix* dengan dimensi 64.512, sedangkan *value* atau nilai tersebut akan diperbaharui selama proses *back-propagation* [6].

Selanjutnya, dilakukan perhitungan dari nilai pecahan dari *self-attention*, yang akan menentukan seberapa banyak *attention* yang digunakan pada *input sequence* lainnya disaat proses *encode* berlangsung. Metode perhitungan dari nilai pecahan tersebut dilakukan menggunakan vektor *query* dan *key*. Kemudian hasil dibagi dengan sebuah konstanta. Secara umum, nilai ini merupakan hasil akar dari dimensi pertama dari *matrix* yang telah disebutkan. Kemudian dilakukan perhitungan *softmax* pada seluruh nilai yang didapat. Hasilnya merupakan relevansi dari setiap kata dengan kata yang ada pada posisi saat itu. Secara natural, relevansi kata dari posisi tertentu tentunya memiliki nilai yang besar. Langkah terakhir adalah mengalikan nilai vektor dengan hasil *softmax* dan menjumlahkannya. Hasilnya adalah nilai dari *self-attention* pada *node* tersebut. Metode menentukan distribusi *weight* dengan derajat kemiripan antara *query* dan *key* disebut juga dengan *dot-product attention*.

### 2.1.5. Mekanisme Multi-headed Attention pada *Self-Attention*

Terdapat sebuah mekanisme lain pada *self-attention*, yang disebut sebagai *multi-headed attention*, yang tidak hanya menginisialisasi sebuah *matrix query*, *key* dan *value*. Melainkan, mekanisme tersebut menginisialisasi beberapa kelompok *matrix*, dan *transformer* menggunakan 8 kelompok sehingga hasil akhirnya berbentuk 8 *matrix*.

Pada *feed-forward neural network*, jumlah *matrix* yang dapat diterima hanyalah satu, jadi perlu dilakukan pengurangan dari 8 *matrix* menjadi 1. Caranya adalah, pertama menghubungkan ke 8 *matrix* hingga menjadi sebuah *matrix* yang besar, kemudian mengalikan *matrix* gabungan ini dengan sebuah *matrix* yang diinisialisasi secara acak, untuk mendapatkan *matrix* akhirnya.

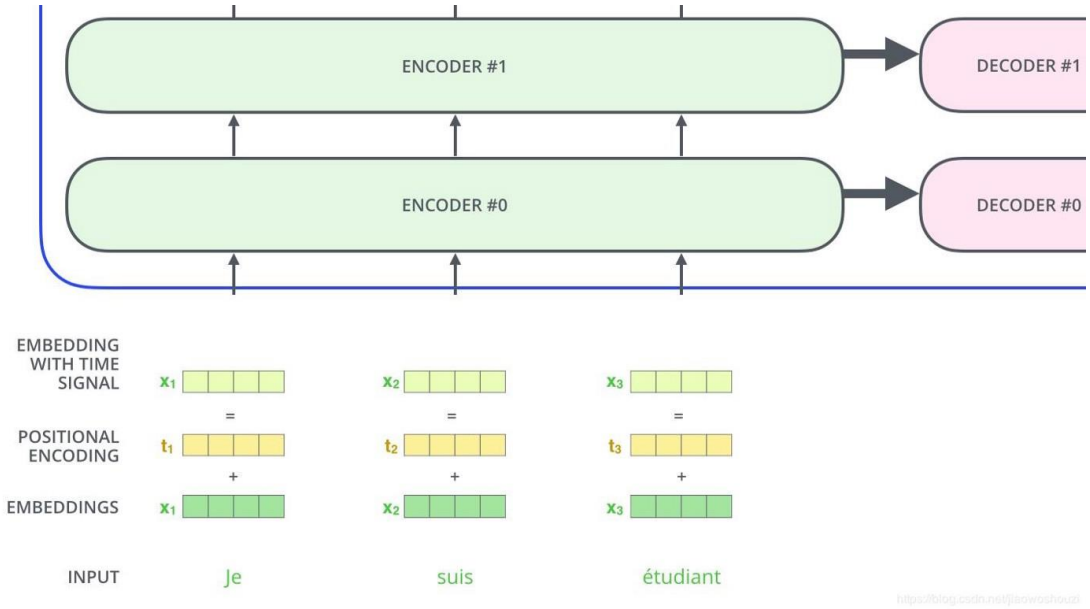



Gambar 3. Mekanisme *multi-headed attention* [9]

*Transformer* menggunakan *multi-head attention* dalam tiga acara berbeda. Pertama, pada *layer encoder-decoder attention*, *query* dihasilkan oleh *layer decoder* sebelumnya, dan *memory keys* dan *values* dihasilkan dari *output* dari *encoder*. Hal ini memungkinkan tiap posisi pada *decoder* untuk dapat menempatkan semua posisi dalam *input sequence*. Hal ini merupakan adaptasi dari mekanisme *encoder-decoder attention* yang ada pada model seq2seq. Kedua, pada *encoder* terdapat layer *self-attention*. Pada *layer self-attention*, seluruh *query*, *key*, dan *value*, dihasilkan dari satu tempat yang sama, yakni *output* dari *layer* sebelumnya pada *encoder*. Setiap posisi pada *encoder* dapat menempati semua posisi yang ada pada *layer* sebelumnya di *encoder*. Ketiga, *layer self-attention* pada *decoder* memungkinkan setiap posisi pada *decoder* untuk dapat menempati semua posisi hingga posisi tersebut.

### 2.1.6. Mekanisme *Positional Encoding*

Hal yang belum dilakukan sejauh ini adalah bagaimana cara untuk mengintepretasi urutan kata yang ada pada *input sequence* pada model *transformer*. Untuk mengatasi hal tersebut, *transformer* menambahkan sebuah vektor tambahan yaitu *Positional Encoding* pada *input* dari *layer-layer encoder* dan *decoder*. Dimensinya sama dengan dimensi pada *embedding*. Nilai dari *positional encoding* kemudian ditambahkan ke nilai dari *embedding* dan dikirim kepada *layer* selanjutnya sebagai *input*.

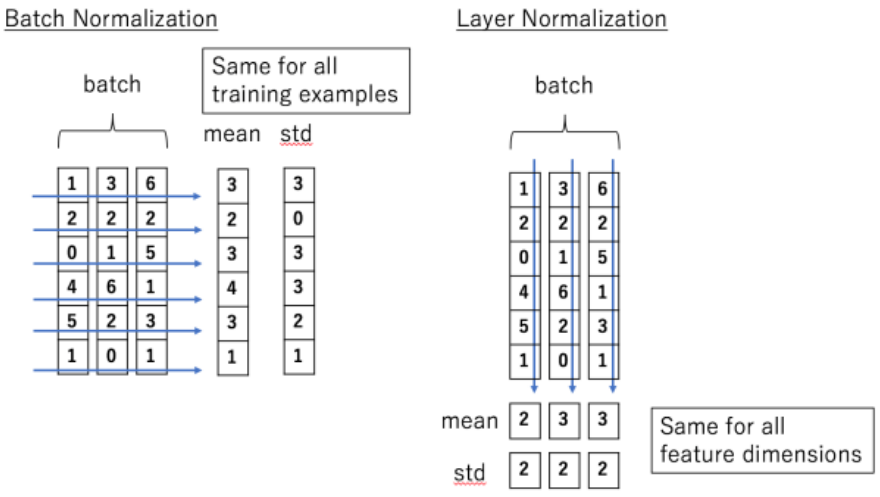



Gambar 4. Mekanisme *positional encoding* [9]

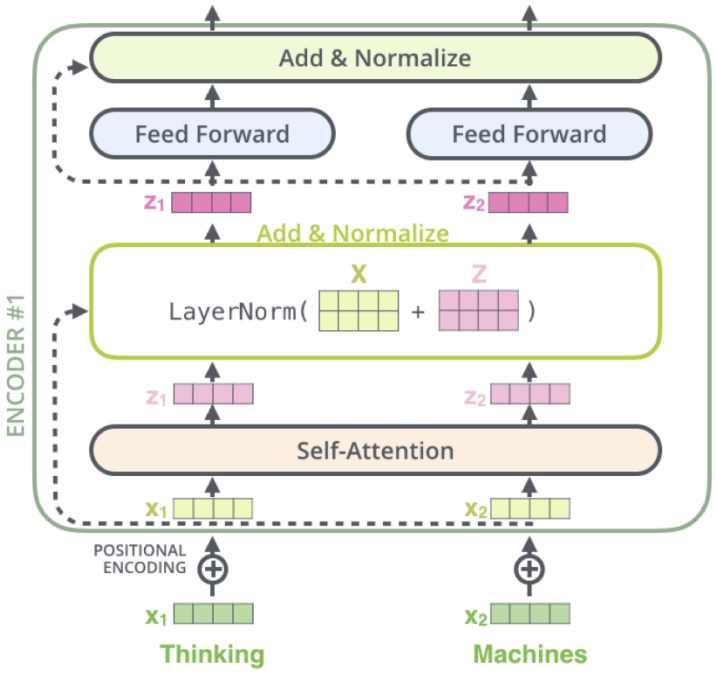
### 2.1.7. *Residual Connection* dan *Layer Normalization*

Pada *encoder* dan *decoder*, sebuah *residual connection* digunakan di sekitar setiap dua *sub-layers*, diikuti oleh *layer normalization*. *Residual connection* digunakan untuk memungkinkan *gradients* untuk mengalir melalui sebuah jaringan secara langsung, tanpa melewati fungsi *non-linear activation*. Fungsi *non-linear activation* dapat mengakibatkan *gradient* menghilang, bergantung pada *weight*-nya. *Residual connection* secara konseptual membentuk sebuah '*bus*' yang mengalir langsung melalui jaringan, dan sebaliknya, *gradient* juga dapat mengalir mundur. Normalisasi membantu dengan masalah yang disebut *internal covariate shift*. *Internal covariate shift* adalah *covariate shift* yang terjadi di dalam sebuah *neural network*. Hal ini terjadi karena selagi *network* di-*training* dan *weights* diperbaharui setiap saat, distribusi dari *output* dari *layer* tertentu pada *network* mengalami perubahan. Hal ini

membuat *layer* yang lebih tinggi untuk beradaptasi pada *drift* tersebut, yang juga memperlambat proses *learning*. Setelah normalisasi *input* pada *neural network*, perbedaan ekstrim dari skala *input features* tidak perlu dikhawatirkan lagi. Sebuah *batch* kecil atau *mini-batch* terdiri dari beberapa *examples* dengan jumlah *feature* yang sama. *Mini-batch* adalah kumpulan *matrix*, atau *tensor* jika tiap inputnya *multi-dimensional*, dimana satu *axis* sesuai dengan *batch* dan *axis* lainnya, atau kumpulan *axis*, sesuai dengan dimensi *features*. *Batch* normalisasi menormalisasi *input features* pada seluruh dimensi *batch* [10]. Fitur utama dari *layer normalization* adalah penormalisasian *input* dari seluruh *features*. Pada *batch* normalisasi, seluruh perhitungan dilakukan pada seluruh *batch* dan sama untuk setiap *example* yang ada pada *batch*. Sebagai perbandingan, *layer normalization* melakukan perhitungan pada seluruh tiap *features* dan *independent* terhadap *examples* lainnya.




Gambar 5. *Batch normalization* dan *layer normalization*




Gambar 6. *Residual connection* dan *layer normalization* [11]

Pada Gambar 6 adalah visualisasi pada saat *residual connection* dan *layer normalization* dijadikan satu.

### 2.1.8. Mekanisme *Decoder*

Pada arsitektur *transformer*, dapat dilihat bahwa bagian *decoder* mirip seperti bagian *encoder*, namun terdapat sebuah *masked multi-head attention* pada bagian bawah. *Mask* merupakan representasi dari hasil masking terhadap nilai tertentu, agar mereka tidak memberikan dampak terhadap parameter yang telah diperbaharui. Terdapat dua jenis *mask* pada model *transformer*, yaitu *padding mask* dan *sequence mask* [6]. *Padding mask* digunakan untuk semua *dot-product attention* yang telah di-*scale*, dan untuk *sequence mask* hanya digunakan untuk *decoder* pada *self-attention*.

Sebuah *padding mask* menyelesaikan suatu masalah dari sebuah *input sequence* yang menjadi ukuran variabel. Khususnya, apabila melakukan pad 0 (nol) setelah sebuah *sequence* yang lebih pendek. Namun apabila *input sequence* terlalu panjang, sedangkan konten pada bagian kiri di-*intercept*, lebihnya akan dibuang secara langsung [7]. Dikarenakan lokasi dari kontennya tidak merepresentasikan apapun sehingga mekanisme *attention* diciptakan untuk tidak mengabaikan lokasi. Pendekatan yang dilakukan yaitu dengan cara menambahkan sebuah *negative infinity* ke nilai dari posisi tersebut, agar probabilitas dari posisi tersebut akan mendekati nilai 0 setelah melewati proses *softmax*. *Padding mask* secara *actual* merupakan sebuah *tensor*, setiap nilai merupakan *Boolean*, dan nilai dari salah merupakan apa yang akan diproses.

Suatu *sequence mask* diciptakan untuk memastikan agar dekoder tidak dapat mengetahui informasi pada masa yang akan mendatang. Yaitu, untuk *sequence*, pada `time_step` t, *output* yang di-*decode* hanya bergantung pada *output* sebelum t, bukan *output* setelah t. Ini berlaku khusus untuk arsitektur *transformer* karena tidak memiliki RNN yang dapat melakukan *input sequence* secara sekuensial. Lalu pada tahap ini, *input* terhadap keseluruhan akan dilakukan. Jika tidak ada *mask*, maka *multi-head attention* akan mempertimbangkan seluruh decoder dari *input sequence* pada setiap posisi.

Untuk *self-attention* dari *decoder*, *dot-product attention* yang telah di-*scale* akan digunakan, serta *padding mask* dan *sequence mask* ditambahkan sebagai `attn_mask`. Pada kasus tertentu, `attn_mask` adalah sama dengan *padding mask*.

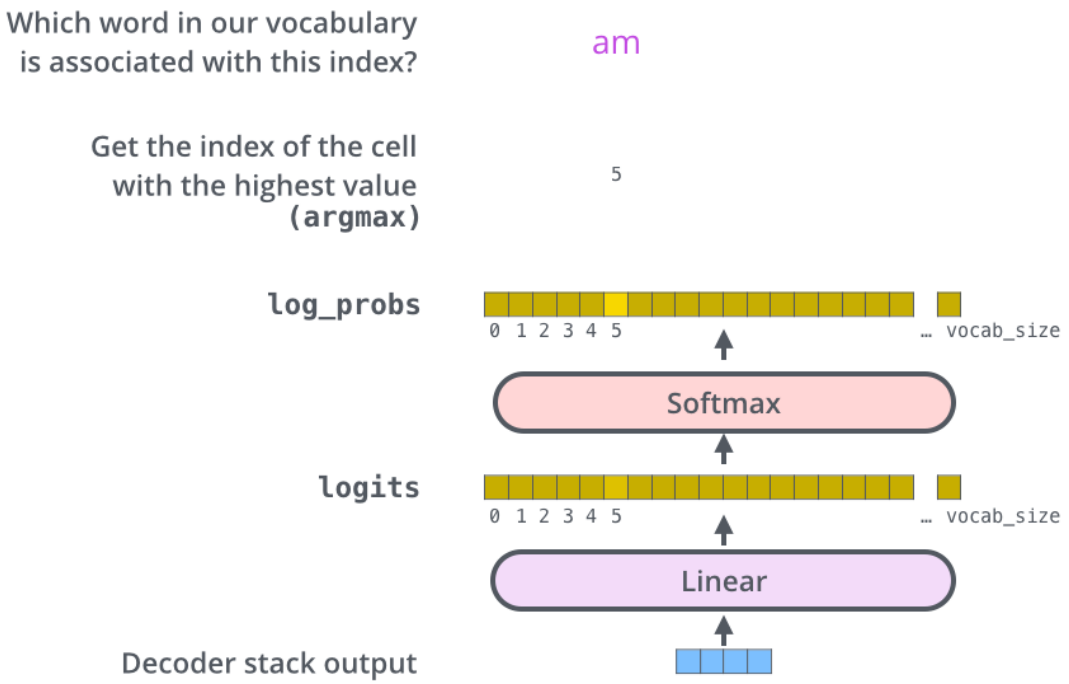
Salah satu detail lain adalah menggeser *input* dekoder ke kanan sebanyak satu posisi. Alasan dilakukannya hal ini adalah untuk menghindari model hanya mempelajari cara menyalin *input decoder* selama *training*, akan tetapi mempelajari *encoder sequence* yang telah diberikan dan *sequence* dekoder tertentu, yang telah diterima oleh model, sehingga model dapat memprediksi kata setelahnya. Apabila *sequence decoder* tidak digeser, maka model hanya belajar untuk menyalin *input decoder*, karena target kata untuk posisi i akan menjadi kata i dalam *input decoder*. Oleh karena itu, dengan menggeser *input decoder* sebannyak satu posisi, model perlu untuk memprediksi target kata untuk posisi i dengan hanya mengetahui kata 1, …, i – 1 dalam *sequence decoder*. Hal demikian mencegah model dari mempelajari *copy* / *paste task*. Posisi pertama dari *input* dekoder akan diisi dengan token *start-of-sentence*, karena pergeseran kanan tadi, maka posisi tersebut akan menjadi kosong. Demikian pula, menambahkan token *end-of-sentence* pada *input sequence* dekoder sebagai penanda akhir dari *sequence* tersebut dan hal tersebut juga ditambahkan pada target kalimat *output*.

### 2.1.9. *Output Layer*

Setelah lapisan *decoder* telah dieksekusi secara menyeluruh, *fully connected layer* dan *softmax layer* ditambahkan pada bagian akhir untuk memetakan vektor yang dihasilkan dari kosakata ke kata-kata.

*Linear Layer* merupakan *neural network* yang terhubung sepenuhnya sederhana yang memproyeksikan vektor yang dihasilkan oleh *stack of decoders*, menjadi vektor yang jauh lebih besar yang biasa disebut dengan *logits vector*. Asumsikan bahwa suatu

model mengetahui 10.000 kata-kata unik dari bahasa Inggris yang dipelajari dari *training dataset*-nya. Sehingga terbuatnya *logits vector* dengan lebar 10.000 kolom yang pada setiap kolom berkoresponden dengan skor kata yang unik. Itulah proses interpretasi *output* dari model yang diikuti oleh *Linear Layer*. Kemudian *softmax layer* mengubah skor menjadi probabilitas. Kolom dengan probabilitas tertinggi akan dipilih, dan kata yang terkait dengannya akan dihasilkan sebagai *output* untuk step tersebut [9].




Gambar 7. *Softmax* pada *decoder* [9]

### 2.1.10. *Bidirectional Encoder Representation from Transformer*

BERT merupakan model *deep learning neural network* dua arah yang berbasiskan arsitektur *Transformer*. Fitur utama dari BERT yang membuatnya berbeda dengan model *deep learning*-nya adalah penerapan *training transformer* dua arah (*bidirectional*) ke dalam *language model*. Hal demikian tentunya berbeda dengan

penelitian-penelitian sebelumnya yang melihat urutan teks baik dari kiri ke kanan atau kombinasi dari kiri ke kanan dan dari kanan ke kiri. Kini, BERT menggunakan teknik baru bernama *Masked Language Modeling* yang memungkinkan *training* dua arah dalam model yang sebelumnya tidak mungkin untuk dilakukan. Pada *default state*-nya, *Transformer* terdiri dari dua mekanisme terpisah, yaitu sebuah *encoder* yang membaca input teks dan sebuah *decoder* yang menghasilkan prediksi untuk tugas tersebut. Karena tujuan dari BERT merupakan untuk menghasilkan *language model*, sehingga hanya sebuah mekanisme *encoder* yang diperlukan.

Pada awalnya Google merilis dua versi seperti yang terlihat pada tulisan spesfikasi BERT dibawah paragraph ini. L merepresentasikan jumlah dari lapisan *transformer*, sedangkan H merepresentasikan dimensi dari *output*, dan A merepresentasikan jumlah dari *multi-headed attention*. Pada kedua versi, ukuran dari *feedforward* ditetapkan menjadi 4 *layer*.

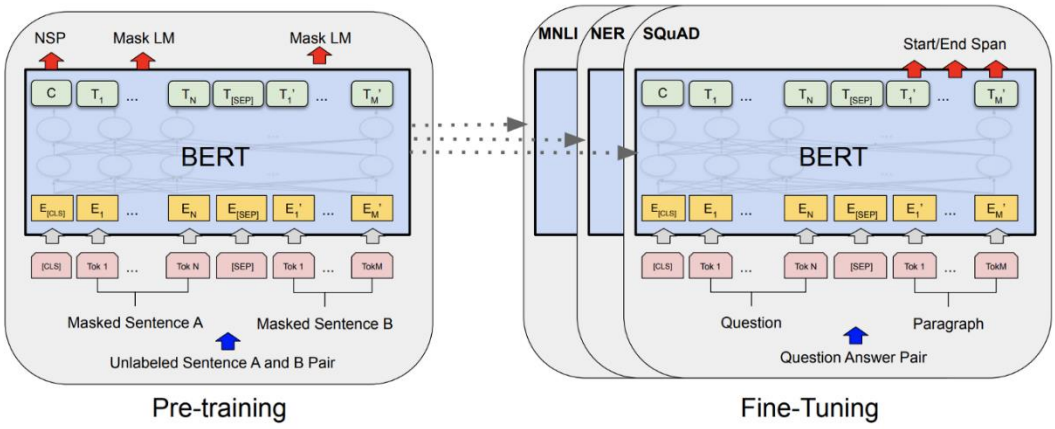
BERTBASE: L = 12, H = 768, A = 12, Total Parameter = 110M

BERTLARGE: L = 24, H = 1024, A = 16, Total Parameter = 340M

Terdapat dua fase penggunaan BERT: *pre-training* dan *fine-tuning*. Ketika *pre-training*, model di-*train* dengan data yang tidak berlabel. Untuk *fine-tuning*, model BERT pertama kali diinisialisasi dengan parameter *pre-trained*, dan seluruh parameter di-*finetune* dengan menggunakan data yang berlabel dari *downstream tasks*. Masing-masing *downstream task* memiliki *finetuned* model yang terpisah, meskipun mereka telah diinisialisasi dengan parameter *pre-trained* yang sama. Sebuah fitur unik dari

BERT adalah arsitekturnya yang *unified* atau terpadu terhadap berbagai *tasks* yang berbeda. Terdapat perbedaan yang kecil di antara arsitektur *pre-trained* dan arsitektur *final downstream*.

Fase *pre-training* dari BERT terdiri dari dua *unsupervised predictive tasks*, yaitu yang pertama merupakan *Masked Language Modelling* (MLM) sedangkan yang kedua merupakan *Next Sentence Prediction* [12].




Gambar 8. BERT *Pre-training* dan *Fine-tuning* [12]

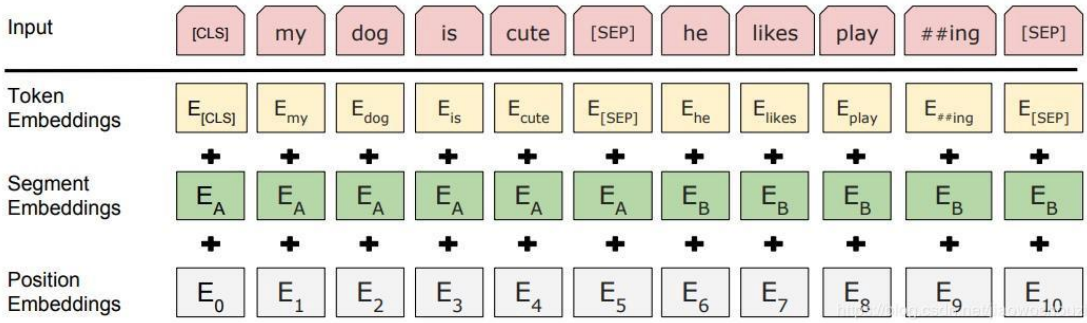
Karena BERT menggunakan fungsi dua arah dan efek dari mekanisme *multi-layer self-attention* yang digunakan oleh BERT, sehingga untuk *train* sebuah representasi *deep bidirectional*, sebagian persentase dari token *input* di-*masked* secara acak, lalu kemudian token yang telah di-*masked* tersebut akan diprediksi. *Final hidden vectors* yang terhubung terhadap *mask tokens* dimasukkan ke dalam *softmax output* melalui *vocabulary*, seperti pada *language model* standar. Tidak seperti model bahasa *pre-training left-to-right*, target MLM memungkinkan untuk membangun representasi

konteks dari sisi kiri dan sisi kanan, yang memungkinkan untuk melakukan *pre-train deep transformer* dua arah.

Untuk melakukan *training* sebuah model yang memahami relasi semantik antarkata, BERT juga melakukan *pre-train* terhadap pekerjaan *next sentence prediction* yang telah di-*binarized* sehingga dapat dengan mudah dihasilkan dari kumpulan teks apapun. Tujuan dari penambahan *pre-training* adalah dikarenakan banyak tugas NLP seperti QA dan NLI perlu memahami relasi antara dua kalimat, sehingga model *pre-trained* dapat beradaptasi dengan lebih baik terhadap berbagai *tasks* tersebut. Ketika melatih model BERT, *Masked Language Modelling* dan *Next Sentence Prediction* di-*train* bersama, dengan tujuan untuk meminimalkan *loss function* gabungan dari kedua pendekatan tersebut.

BERT tidak melihat kata sebagai token, namun BERT melihatnya sebagai WordPieces. Ini berarti bahwa sebuah kata dapat dipecah menjadi lebih dari satu sub-kata. Tokenisasi semacam ini sungguh bermanfaat ketika berhadapan dengan kata-kata *vocabulary*, dan memiliki kemungkinan lebih baik dalam merepresentasikan kata-kata yang rumit.

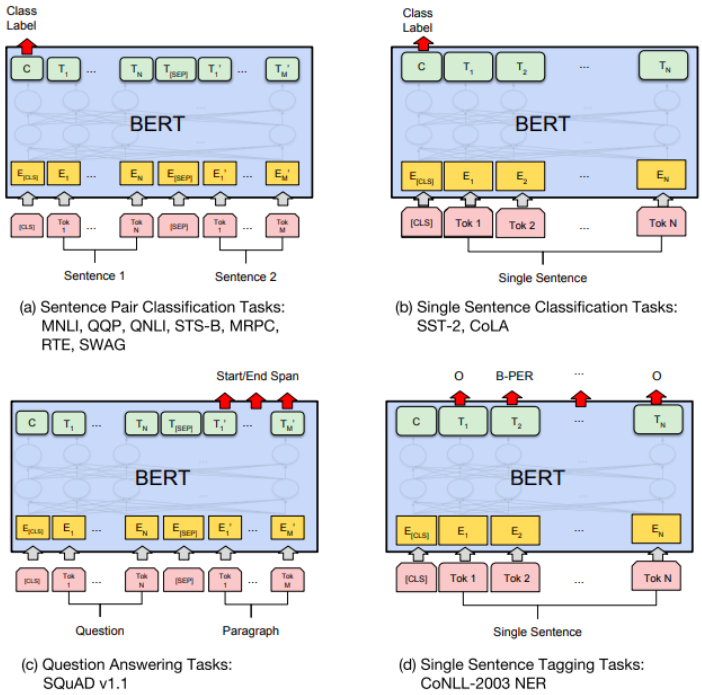
*Input* dari BERT dapat berupa satu kalimat atau sepasang kalimat dalam suatu urutan kata. Untuk kata tertentu, representasi *input*-nya dapat terdiri dari penjumlahan tiga bagian *Embedding*. Representasi visual dari *Embedding* akan diperlihatkan pada Gambar 9.




Gambar 9. Illustrasi dari embedding [13]

*Token Embeddings* merepresentasikan kata dalam bentuk vektor. Kata pertama merupakan *flag* CLS yang dapat digunakan untuk *subsequent classification tasks*. Untuk *non-classification tasks*, *flag* CLS-nya dapat diabaikan. *Segment Embeddings* digunakan untuk membedakan antara dua kalimat, karena *pre-training* tidak hanya merupakan *language model* namun juga sebagai *classification task* dengan dua kalimat sebagai *input*. *Position Embeddings* meng-*encode* urutan kata [13].

Untuk setiap *downstream* NLP *task*, pengguna model menentukan *inputs* & *outputs* yang spesifik dengan *task* terhadap BERT serta menyempurnakan seluruh parameter *end-to-end*. Pada *input*, kalimat A beserta kalimat B dari *pre-training* dapat dianalogikan dengan pasangan kalimat dalam parafrasa, pasangan *question-passage* dalam *question answering*, dan sebagainya. Sedangkan pada output, representasi token dimasukkan ke dalam *layer output* untuk *tasks* pada level token, seperti *sequence tagging* atau *question answering*, dan representasi [CLS] dimasukkan ke dalam lapisan *layer output* untuk klasifikasi, seperti *sentiment analysi*s. Dibandingkan dengan *pre-training*, *fine-tuning* relatif murah.




Gambar 10. Illustrasi *downstream task* BERT [12]

BERT dapat digunakan untuk beragam *language tasks*, dengan hanya dengan menambahkan *small layer* terhadap *core model* seperti :

1. *Classification tasks* seperti analisis sentimen dijalankan seperti *Next Sentence classification*, yaitu dengan menambahkan sebuah *classification layer* di atas output *Transformer* untuk token [CLS].
2. Pada *Question Answering tasks* (misalnya, SQuAD v1.1), perangkat lunak menerima pertanyaan yang berhubungan dengan sebuah *text sequence* dan diharuskan untuk mimilih jawaban dalam urutan tersebut. Dengan menggunakan BERT, suatu model Q&A dapat di-*train* dengan mempelajari dua vektor sebagai penanda dari awal dan akhir jawaban.

3. Pada *Named Entity Recognition* (NER), perangkat lunak menerima urutan teks dan diperlukan untuk menandai berbagai jenis dari entitas seperti orang, organisasi, tanggal dan lain-lain yang muncul dalam suatu teks. Dengan menggunakan BERT, sebuah model NER dapat di-*train* dengan cara memberikan vektor *output* dari setiap token ke dalam *classification layer* yang dapat memprediksi NER labelnya.

## 2.1. Natural Language Processing

*Natural language processing* merupakan sebuah cara atau metode komputasi yang bertujuan untuk menganalisis teks dengan menggunakan beberapa teori dan teknologi agar suatu teknologi dapat mengerti bahasa manusia atau bahasa natural pada beberapa peran atau aplikasi [14].

## 2.2. Google Cloud: Cloud Computing Services

Google *Cloud* merupakan sebuah rangkaian layanan komputasi pada *cloud* yang ditawarkan oleh Google. Infrastruktur yang ditawarkan Google *Cloud* adalah infrastruktur yang sama dengan yang digunakan Google secara internal untuk produk-produk untuk *end-user*, seperti Google *Search*, Google *Mail* dan Youtube. Google *Cloud Platform* (GCP), yang merupakan sarana dalam menggunakan produk Google *Cloud*, menyediakan *infrastructure-as-a-service*, *platform-as-a-service*, dan *environment* komputasi tanpa *server* atau *serverless* [15].

Layanan GCP dipilih peneliti dari antara layanan cloud lainnya dikarenakan GCP memberikan kredit secara cuma-cuma sebesar 300 dolar Amerika Serikat atau

setara dengan 5 juta rupiah. Peneliti telah melakukan kalkulasi awal dalam perhitungan penggunaan komputasi yang digunakan, dan kredit yang diberikan dapat menutupi biaya perkiraan. Terdapat beberapa *cloud service* yang digunakan oleh peneliti akan dijelaskan pada subbab-subbab berikut ini.

### 2.2.1. Google Compute Engine

Google *Compute Engine* merupakan mesin virtual yang mudah dikonfigurasikan yang berjalan pada pusat data Google dengan infrastruktur jaringan yang memiliki performa yang tinggi dan memiliki penyimpanan. Dengan *Compute Engine* pengguna dapat menggunakan mesin virtual yang sesuai dengan kebutuhannya baik untuk keperluan yang umum, atau *workload* yang dioptimisasi, dalam ukuran mesin yang dapat ditentukan secara khusus [16].

Terdapat beberapa macam mesin virtual yang ditawarkan dalam produk *Compute Engine* ini, yakni; *General purpose* (N2), *Compute Optimized* (C2), *Memory Optimized* (M2), *General purpose* (N2D). Pada penelitian ini peneliti memilih untuk menggunakan mesin virtual *general purpose* (N1) dikarenakan biayanya yang terjangkau dan spesifikasinya dapat memenuhi kebutuhan *training*.

### 2.2.2. Google Cloud Storage

Google *Cloud Storage* merupakan sebuah servis penyimpanan berkas secara daring untuk menyimpan dan mengakses data pada infrastruktur GCP. *Cloud Storage* memberikan performa dan skalabilitas *cloud* Google dengan keamanan dan kemudahan dalam *sharing* atau membukanya untuk dibagikan [17]. Berbeda dengan

Google *Drive* yang merupakan layanan penyimpanan juga, Google *Cloud Storage* memungkinkan pemakainya untuk mengintegrasikan penyimpanannya dengan layanan *cloud* lainnya. Peneliti menggunakan Cloud *Storage* untuk kepentingan penyimpanan *checkpoint* dari *training progress* agar jika proses diberhentikan, tidak perlu mengulang *training* dari awal.

### 2.2.3. Google Cloud TPU

TPU atau *Tensor Processing Unit* merupakan ASICs (*Application Specific Integrated Circuit*) yang dikembangkan khusus oleh Google, yang digunakan untuk mengakselerasi *workload* dari *machine learning*. TPU secara khusus dirancang berdasarkan pengalaman mendalam Google sebagai salah satu pemimpin dalam perkembangan *machine learning*.

TPU yang dipilih oleh peneliti adalah TPU tipe v2-8, karena memiliki performa yang cukup untuk menjalankan proses *training* dan merupakan TPU dengan harga yang paling terjangkau di antara tipe lainnya. Peneliti memilih TPU yang *preemptible*. TPU *preemptible* merupakan sebuah *Cloud* TPU yang dapat digunakan dengan harga yang jauh lebih rendah dibandingkan dengan TPU normal lainnya. Hal ini dapat terjadi dikarenakan *Cloud* TPU dapat mematikan atau *terminate* TPU *preemptible* ini sewaktu-waktu di saat ia membutuhkan akses pada TPU tersebut untuk kepentingan lainnya, tidak seperti TPU normal yang tidak akan di-*terminate* tanpa sepengetahuan pengguna TPU tersebut.

## 2.3. Python

Python merupakan bahasa pemrograman multifungsi yang berarti, tidak seperti JavaScript dan lainnya, Python dapat digunakan untuk banyak tipe pemrograman dan pengembangan perangkat lunak selain pengembangan web. Python pada umumnya lebih sering digunakan untuk tujuan *data science*, sebab kebanyakan fungsi pada Python mendukung pemrosesan data. Pada penelitian ini, versi python yang akan digunakan adalah python versi 3 dikarenakan library-library yang digunakan semua sudah mendukung Python versi 3.

## 2.4. TensorFlow

TensorFlow merupakan sebuah *open-source library* untuk komputasi numerik dan *machine learning* dalam skala besar. TensorFlow mengemas *machine learning* dan juga *deep learning* model beserta algoritmanya yang dapat digunakan untuk berbagai kebutuhan. TensorFlow menggunakan Python untuk menyediakan *front-end* API untuk membangun aplikasi dengan *framework* TensorFlow. TensorFlow mengeksekusi aplikasinya menggunakan bahasa pemrograman C++.

Peneliti menggunakan TensorFlow dalam melakukan pembuatan *pre-trained language model*. Hal ini dilakukan karena BERT dibangun dengan menggunakan *framework* TensorFlow dan banyak melibatkan penggunaan *tensor*.

## 2.5. PyTorch

Pytorch merupakan sebuah *open-source library* berbasis bahasa pemrograman Python, yang memanfaatkan kemampuan dari *Graphics Processing Unit* (GPU).

Pytorch juga merupakan salah satu *deep learning platform* untuk penelitian yang dibangun untuk menyediakan fleksibilitas dan kecepatan yang maksimum. PyTorch dikenal sebagai penyedia dari dua fitur paling *high-level* yakni; komputasi *tensor* dengan dukungan GPU *acceleration* yang kuat dan juga membangun *deep neural network* berdasarkan sistem *tape-based* Autograd.

PyTorch digunakan peneliti sebagai alat yang membantu dalam proses *fine-tuning*. Dalam proses tersebut, peneliti memilih PyTorch dibandingkan dengan Tensorflow dikarenakan fleksibilitas dan kemudahan yang ditawarkan oleh PyTorch.

### 2.5.1. HuggingFace

HuggingFace merupakan sebuah *open-source library* untuk bermacam-macam aplikasi dari NLP. HuggingFace menyediakan banyak model untuk kebutuhan NLP yang telah dikemas dan dapat secara langsung digunakan untuk pemodelan. Dikarenakan fleksibilitas dan kelengkapannya, peneliti menggunakan *library* yang disediakan oleh HuggingFace dalam proses *fine-tuning model*.

## 2.6. Sentencepiece

SentencePiece merupakan sebuah *language-independent subword tokenizer* dan juga *detokenizer* yang di desain untuk pemrosesan teks berbasis *neural*. Pada umumnya alat segmentasi *subword* mengasumsi bahwa masukannya telah di *tokenized* sebelumnya menjadi *word sequences*. Di sisi lain SentencePiece dapat melatih *subword models* langsung dari kalimat yang belum diolah, dimana memungkinkan pengguna untuk membuat sistem yang *end-to-end* dan *language-dependent*.

## 2.7. Scikit-learn

Scikit-learn merupakan sebuah *machine learning library* untuk bahasa pemrograman Python yang dapat digunakan secara gratis. Fitur-fitur yang dimiliki oleh Scikit-learn antara lain adalah bermacam-macam algoritma klasifikasi, regresi, dan *clustering* termasuk *support vector machine* (SVM), *random forests*, *gradient boosting*, k-*means*, dan DBSCAN. Scikit-learn di design untuk dapat dioperasikan bersama NumPy, *numerical* dan *scientific library* milik Python [18].

Peneliti menggunakan Scikit-learn untuk beberapa pemrosesan data seperti *splitting data*, dan lainnya. Tidak hanya itu, peneliti juga memanfaatkan fitur Scikit-learn dalam mengevaluasi. Fitur evaluasi tersebut dapat berupa *confusion matrix*, *accuracy*, F1 *score*, ROC-AUC *curve*, *cross validation*, dan lainnya.

## 2.8. Metrik Evaluasi

Pada penelitian ini metrik evaluasi dilakukan untuk mengukur performa dari model-model yang akan dibandingkan. Terdapat beberapa macam metrik evaluasi yang akan dijelaskan pada subbab-subbab di bawah ini.

### 2.8.1. K-Fold Cross Validation

Cross validation adalah prosedur *resampling* yang digunakan untuk mengevaluasi model *machine learning* pada *dataset* [19]. Prosedur ini memiliki parameter tunggal yang disebut k yang mengacu pada jumlah kelompok kombinasi dari data yang akan dibagi. Karena itu, prosedur ini sering disebut k-*fold cross-validation*.

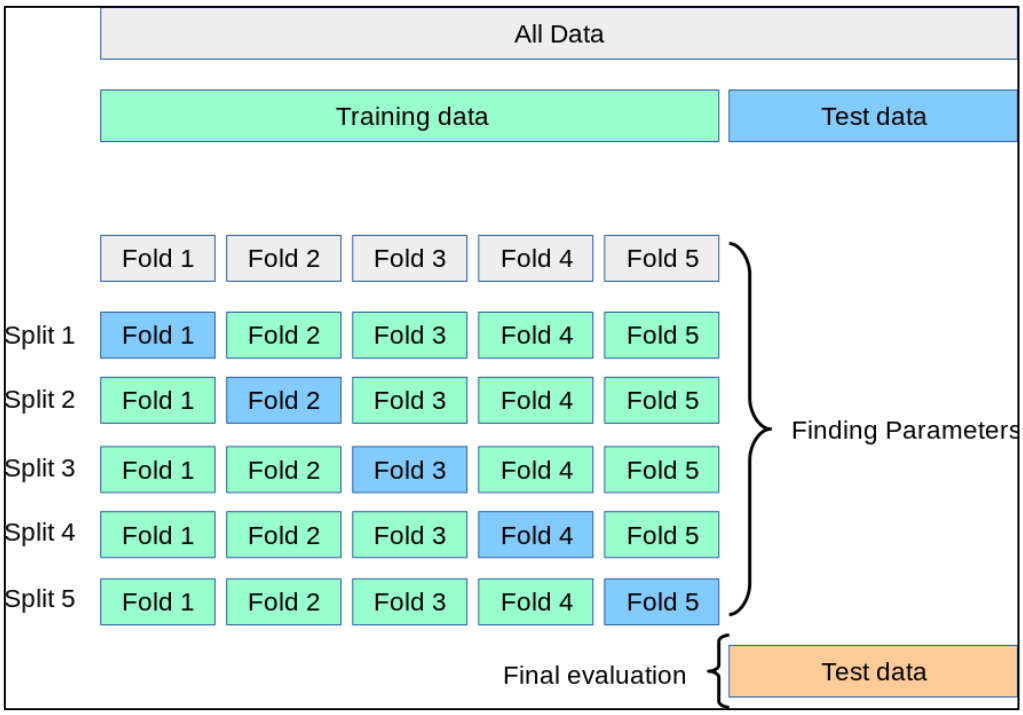
*Cross validation* umumnya digunakan dalam *machine learning* untuk mengevaluasi performa model *machine learning* dengan cara menggunakan kombinasi *dataset* yang telah di-*split* untuk memperkirakan bagaimana performa model secara umum, ketika digunakan untuk membuat prediksi pada *dataset* yang akan diujikan.

Ini adalah metode yang populer karena mudah dimengerti dan karena umumnya menghasilkan estimasi yang tidak terlalu bias atau tidak terlalu optimis dari performa model dibandingkan metode lain, seperti *train* / *test split* yang biasa.

Cara kerja dari K-*Fold cross validation* adalah;

1. Meng-*shuffle dataset* secara acak
2. Meng-*split dataset* ke dalam sebanyak k kelompok,
3. pada setiap kelompok yang unik, di-*split train* dan *test*, kemudian di-*fit* ke dalam model dan diuji dengan *test set*, simpan hasil evaluasi dan model yang telah di-*fit* di-*discard*.
4. Menyimpulkan performa model dari seluruh evaluasi yang didapatkan di setiap kelompok

Berikut adalah ilustrasi dari 5-*Fold Cross Validation* :




Gambar 11. Illustrasi *5-Fold Cross Validation* [20]

Pada penelitian ini, peneliti menggunakan fitur *stratify* pada pembagian *fold* dan juga *train-test split* agar perbandingan antar label tetap sama pada *train* maupun *test set* nya setelah di *split*.

### 2.8.2. Confusion Matrix

*Confusion Matrix* merupakan sebuah pengukuran performa khususnya pada masalah klasifikasi dimana keluaran dapat terdiri dari 2 kelas atau lebih. Matriks berbentuk tabel ini terdiri dari 4 kombinasi berbeda dari nilai yang diprediksi (*predicted*) dan nilai yang sebenarnya (*actual*) [21]. Illustrasi dari tabel ada pada Gambar 12.

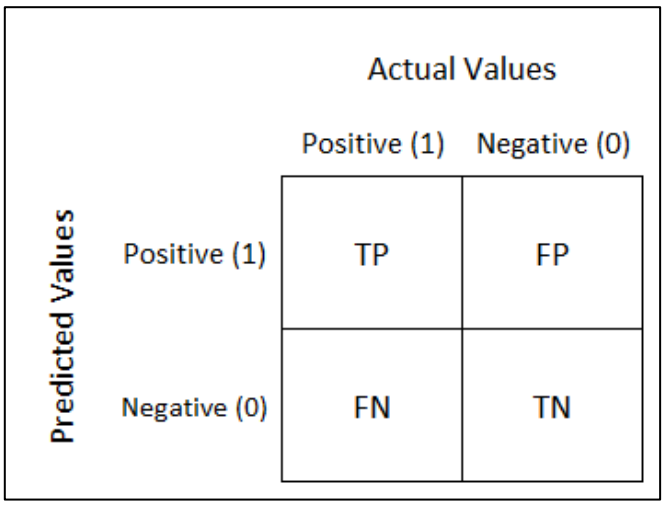
Matriks ini berperan penting dalam mengukur *recall*, *precision*, *accuracy*, F1 *score* dan juga kurva AUC-ROC. Berikut merupakan formula dari metrik-metrik tersebut:

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 \times precision \times recall}{precision + recall}$$

$$accuracy = \frac{TP + TN}{TP + FN + TN + FP}$$

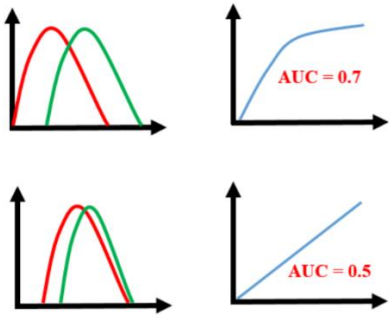



Gambar 12. Tabel dari *Confusion Matrix*

### 2.8.3. Kurva AUC-ROC

Kurva AUC-ROC merupakan tolak ukur yang dapat diandalkan dalam model prediksi *classification* dan merupakan metrik evaluasi yang paling penting dalam memeriksa performa dari model *classification*. Kurva AUC - ROC adalah pengukuran performa untuk masalah *classification* di berbagai pengaturan *thresholds*. ROC adalah kurva probabilitas dan AUC mewakili derajat atau ukuran dari sebuah keterpisahan. Ini menunjukkan seberapa mampu model membedakan antara kelasnya [22]. Semakin tinggi AUC berarti semakin baik model dalam membedakan antarkelas dan memprediksi pun menjadi lebih akurat. Dengan analogi, semakin tinggi nilai AUC, semakin baik modelnya membedakan antara peminjam yang baik dan peminjam yang berisiko.

Pada saat AUC bernilai 1 atau dapat dinyatakan dalam persen yaitu 100%, adalah kondisi ideal dimana model memiliki ukuran keterpisahan yang ideal, secara sempurna dapat membedakan antarkelasnya. Pada saat AUC bernilai di antara 1 dan 0.5, Sedangkan pada saat AUC bernilai 0.5 atau dapat dinyatakan dalam persen yaitu 50%, adalah kondisi terburuk model karena model tidak memiliki kemampuan diskriminasi sama sekali untuk membedakan antarkelasnya.




Gambar 13. Kurva AUC

### 2.8.4. Matthews Correlation Coefficient (MCC)

Matthew *correlation coefficient* merupakan sebuah pengkuran performa atau kualitas dari klasifikasi biner, yang dikenalkan oleh seorang *biochemist* bernama Brian W. Matthews pada tahun 1975. *Coefficient* ini memperhitungkan *true* dan *false positive* dan *negative* dan umumnya dianggap sebagai pengukuran seimbang yang dapat digunakan bahkan ketika kelas-kelasnya memiliki ukuran yang berbeda-beda.

Walaupun tidak ada cara sempurna untuk merepresentasikan *confusion matrix* dengan satu angka, MCC umumnya dianggap sebagai salah satu cara yang terbaik. Pengukuran lain, seperti seberapa banyak hasil prediksi yang benar seperti *accuracy*, tidak akan berguna dengan baik ketika kelas dari target label, memiliki ukuran yang sangat berbeda atau tidak seimbang. MCC dapat dihitung langsung dengan nilai yang ada pada *confusion matrix* dengan formula sebagai berikut:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + TP)(TN + FN)}}$$

Dalam persamaan ini, TP merupakan *true positive*, TN merupakan *true negatives*, FP merupakan *false positives*, dan FN merupakan *false negative*.

## 2.9. Simpulan Studi Literatur

Diketahui bahwa model *pretrained* BERT Multilingual di-*train* dengan korpus berukuran besar yang terdiri dari 104 bahasa [24]. Untuk pemakaian *pretrained* model tersebut juga tidak terdapat *language selector* yang berarti untuk penggunaan pada

kalimat dalam bahasa tertentu, kosakata yang digunakan untuk mentokenisasi memungkinkan adalah token dari bahasa lain. Hal tersebut dapat mengurangi kebenaran dalam representasi tiap subkata sebagai token pada saat representasi tersebut digunakan untuk kata dalam bahasa lain. Ukuran kosakata dari BERT Multilingual adalah 119.547 token untuk seluruh 104 bahasa sedangkan model *pretrained* BERT English memiliki 28.996 token, yang berarti tiap bahasanya tidak memiliki kosakata sebanyak dengan BERT yang di-*train* dengan satu spesifik bahasa.

Dikarenakan akan membangun model pretrained BERT dengan satu spesifik bahasa yaitu Bahasa Indonesia, maka konfigurasi yang digunakan akan disamakan dengan konfigurasi BERT English yang juga dibangun hanya dengan satu bahasa. Konfigurasi yang digunakan akan dijelaskan pada bab analisis dan perancangan sistem. Bahasa BERT yang telah di-*pretrain*, akan dibandingkan dengan Multilingual BERT melalui performa dari *downstream task* setelah proses *fine tuning* pada kedua model. Pada penelitian ini, peniliti diberi kesempatan untuk membuat model klasifikasi teks pada dataset *real world case* dari perusahaan yang tidak disebutkan namanya. Oleh karena itu, *downstream task* yang dipilih merupakan klasifikasi teks pada dataset *real world ca*se dan juga dataset *review playstore open-source* yang diunduh oleh peneliti. Kedua dataset merupakan dataset yang berbahasa Indonesia. Hasil evaluasi dari kedua dataset pada kedua model baik BERT Multilingual dan Bahasa BERT inilah yang akan digunakan sebagai metrik evaluasi pembanding diantara kedua model.