

BAB II

TINJAUAN PUSTAKA

Pada bab ini akan dijelaskan seluruh landasan-landasan teori yang berhubungan dengan penelitian, yaitu tentang metode-metode yang digunakan dalam pembangunan aplikasi pada penelitian, serta teori-teori pendukung lainnya.

2.1 *Text Classification*

Text Classification adalah proses pemberian *tag* atau kategori ke teks menurut isinya. Klasifikasi teks dapat digunakan untuk mengatur, menyusun, dan mengkategorikan hampir semua hal. Misalnya, artikel berita yang dapat diatur berdasarkan topik. Klasifikasi teks dapat dilakukan dengan dua cara berbeda: klasifikasi manual dan otomatis. Cara yang pertama, annotator manusia menafsirkan konten teks dan mengkategorikannya. Yang terakhir menerapkan pembelajaran mesin, pemrosesan bahasa alami, dan teknik lainnya untuk secara otomatis mengklasifikasikan teks dengan cara yang lebih cepat dan lebih hemat biaya. Dengan menggunakan pembelajaran mesin klasifikasi teks menjadibagian yang penting dari kehidupan karena memungkinkan untuk dengan mudah mendapatkan informasi yang diinginkan dengan waktu yang cepat. *Text Classification* dapat diotomatisasikan dengan menggunakan *Natural Language Processing* (NLP) yang dapat secara otomatis menganalisis teks dan kemudian menetapkan satu set label atau kategori berdasarkan teks yang dianalisis. Dengan di otomatisasi *text classification* dapat diimplementasikan kedalam banyak hal contoh nya klasifikasi artikel.

2.2 *Natural Language Processisng (NLP)*

Natural Language Processing arau NLP adalah salah satu cabang dari ilmu *Artificial Intelligence* (AI) yang berfokus di pelatihan komputer untuk dapat memahami teks dengan bahasa yang digunakan manusia. NLP menggabungkan linguistik komputasi dengan model statistik, pembelajaran mesin (*machine learning*), dan pembelajaran mendalam (*deep learning*). Teknologi tersebut memungkinkan komputer untuk memproses bahasa manusia dalam bentuk teks atau data suara dan untuk memhami makna sepenuh nya, lengkap dengan maksud dan sentimen penulis atau pembicara.

Salah satu kegunaan NLP adalah dapat melatih program komputer untuk mengklasifikan teks, menerjemahkan teks dari satu bahasa ke bahasa lainnya, menanggapi perintah lisan dan lainnya. Beberapa penerapan NLP diantaranya. Perintah di gawai yang dioperasikan dengan perintah lisan, asisten digital, perangkat lunak dikte ucapan ke teks, *chatbot consumer service*, dan lainnya.

2.3 *Metode experimental research design*

Metode penelitian dalam eksperimen pembelajaran mesin adalah hal yang sangat penting karena akurasi dan keandalan hasil dipengaruhi oleh metode penelitian yang digunakan. Pendekatan umum yang digunakan dalam melakukan penelitian pembelajaran mesin adalah sebagai berikut: *Data collection*, *Data pre-processing*, *Model training*, *model testing*, *model evaluation*. Langkah-langkah ini membantu peneliti untuk mendapatkan hasil yang terbaik. Visualisasi langkah-langkah *experimental research design* dirangkum dalam Gambar 2.1 (Kamiri and Mariga, 2021).



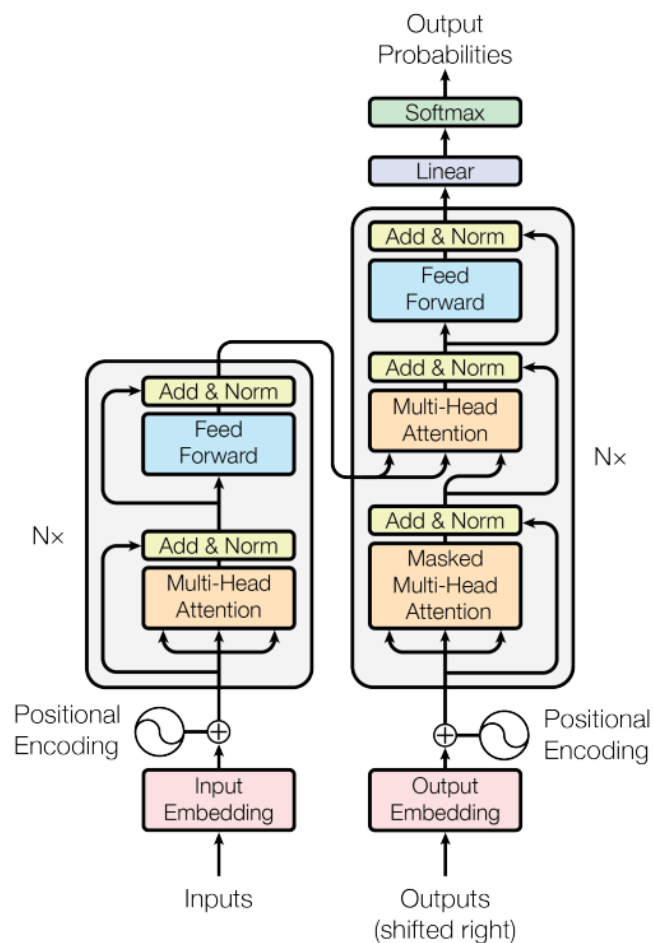
Gambar 2.1 Pendekatan umum penelitian *machine learning* (Kamiri and Mariga, 2021).

2.4 Transformers

Transformers adalah suatu teknik yang diusulkan untuk menyingkirkan arsitektur berulang dengan mengandalkan mekanisme perhatian (*Self-attention mechanism*). *Self-attention mechanism* adalah mekanisme yang mempelajari hubungan kontekstual antara kata-kata dalam teks (Vaswani *et al.*, 2017). *Transformers* mulanya digunakan sebagai model yang berfungsi untuk melakukan proses translasi bahasa. *Transformers* sendiri terdiri atas dua komponen, yaitu *encoder* dan *decoder*, penjelasan *Encoder* dan *Decoder* adalah sebagai berikut.

1. *Encoder*, berfungsi untuk membaca seluruh *input* teks. *Encoder* terdiri dari tumpukan (*stack*) $N = 6$ dengan lapisan (*layer*) yang identik. Setiap lapisan mempunyai dua lapisan bawah (*sub-layer*) yaitu lapisan *self-attention layer* dan lapisan *feed-forward neural network*. Dengan *self-attention layer*, *encoder* dapat membantu *transformer* untuk memahami kata dengan menggunakan konteks semantik dari kata tersebut.
2. *Decoder*, berfungsi untuk menghasilkan *output* yang berupa hasil prediksi. *Decoder* juga terdiri dari tumpukan (*stack*) $N = 6$ dengan lapisan (*layer*) yang identik. Setiap lapisan (*layer*) terdiri dari dua lapisan bawah (*sub-layer*) seperti yang ada pada *encoder*, dengan tambahan lapisan ketiga yaitu *attention layer* di antara dua lapisan tersebut untuk membantu *transformer*

saat ini mendapatkan *key content* yang membutuhkan *attention* dengan melakukan *multi-head attention* pada *output* dari *encoder*. Sama dengan di *encoder*, *self-attention layer* di *decoder* membuat setiap posisi di *decoder* dapat menangani semua posisi sebelumnya dan posisi saat itu. Gambar 2.2 mengvisualisasikan *encoder* dan *decoder* pada *transformers*.



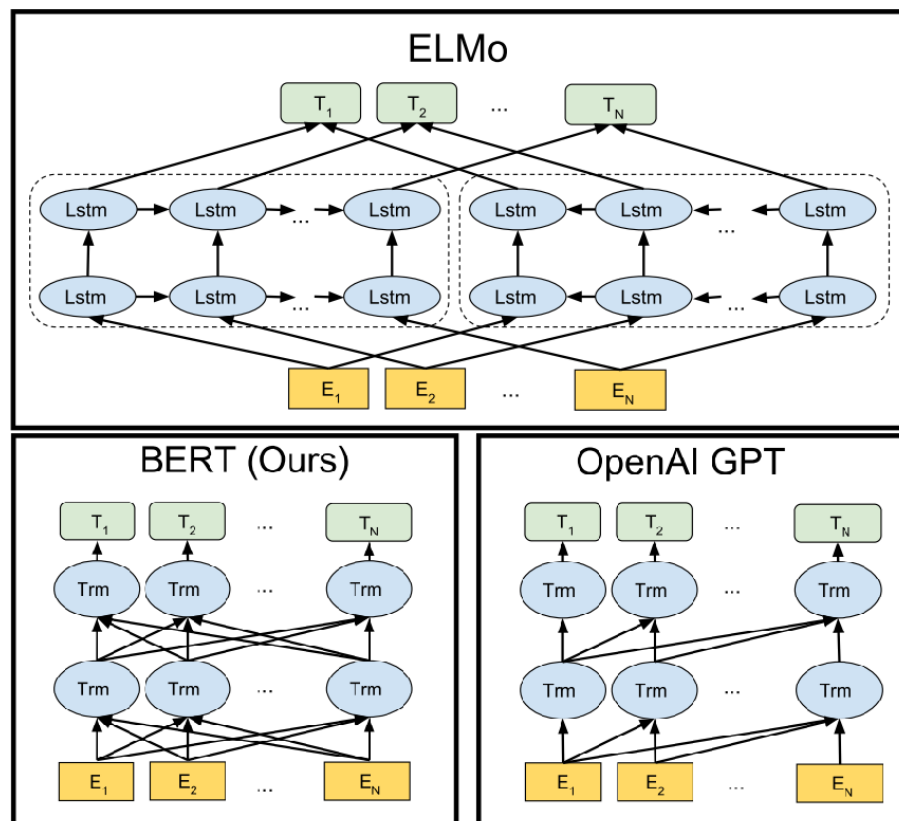
Gambar 2.2 Visualisasi *encoder* dan *Decoder* pada *Transformer* (Vaswani *et al.*, 2017)

2.5 Bidirectional Encoder Representations from Transformers (BERT)

Bidirectional Encoder Representations from Transformers (BERT) adalah sebuah algoritma yang dikembangkan oleh para peneliti di Google AI Language

pada tahun 2018. Pada Oktober 2019 BERT sendiri untuk pertama kali diperkenalkan oleh Google. BERT sendiri dikembangkan berdasarkan teknik-teknik pada deep learning serta dari beberapa model penelitian sebelumnya, seperti ELMo, OpenAI GPT dan *Transformer*. Model dari BERT sendiri dikembangkan dari *Transformer* (Devlin *et al.*, 2019).

Berbeda dari model-model yang sebelumnya menerapkan pembelajaran searah. Dimana model akan membaca teks dari kiri ke kanan, kanan ke kiri atau gabungan dari keduanya. BERT menerapkan pembelajaran secara dua arah secara bersamaan (*bidirectional*), yang memungkinkan model membaca teks dari kiri ke kanan dan kanan ke kiri secara bersamaan. Gambar 2.3 memvisualisasikan perbedaan BERT dengan arsitektur OpenAI GPT dan ELMo lainnya.



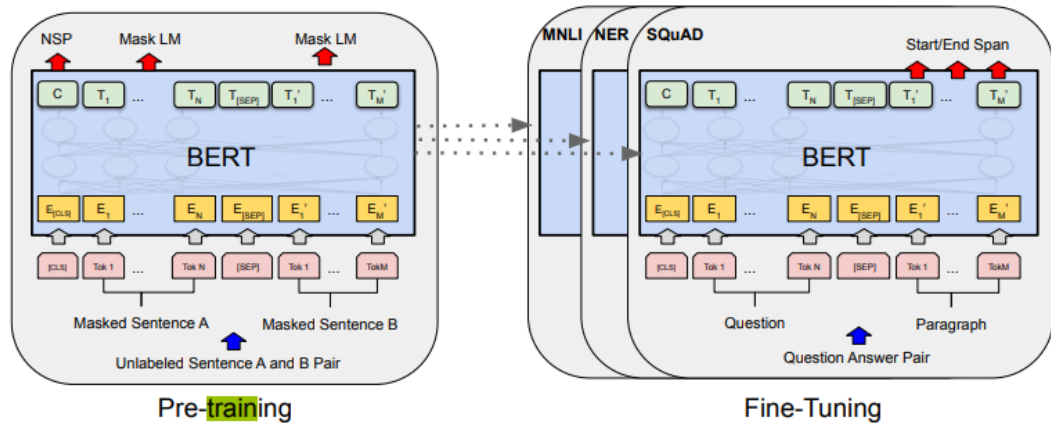
Gambar 2.3 Visualisasi BERT, OpenAI GPT dan ELMo (Devlin *et al.*, 2019)

BERT adalah arsitektur model *pre-trained* yang dirancang dengan fitur untuk mempertimbangkan konteks kata dengan membaca kata yang bersebelahan dengan kata yang dipelajari. Dengan kemampuan itu bert dapat meningkatkan hasil model *Natural Language Processing* (NLP), seperti klasifikasi teks. Kemampuan BERT yang bisa mengekstraksi lebih banyak informasi dari teks dengan pembelajaran kontekstual. Pada implementasinya BERT dapat digunakan untuk menyelesaikan berbagai permasalahan yang terkait dengan NLP. Beberapa tugas yang bisa dikerjakan BERT dengan baik antara lain adalah *text encoding*, *similarity retrieval*, *text summarization*, *question answering* dan *text classification*. Dalam melakukan berbagai tugas tersebut, model harus dilatih dengan cara khusus pada model BERT. Pelatihan khusus pada BERT untuk tugas tertentu ini dinamakan *fine-tuning*.

Arsitektur yang digunakan oleh BERT adalah *multi-layer bidirectional Transformer*, yang dimana arsitektur ini serupa dengan arsitektur *Transformer*, akan tetapi ada sedikit perbedaan dimana BERT hanya menggunakan bagian *encoder*-nya saja. Meskipun BERT tidak menggunakan *decoder* hasilnya tetap optimal, hal ini dikarenakan BERT menambah jumlah *stack* dari *encoder*, terdapat dua ukuran model yang ada pada BERT, yaitu BERT_{BASE} dan BERT_{LARGE}. Kedua ukuran model BERT ini memiliki banyak lapisan *encoder* atau dinamakan *transformer blocks*. BERT_{BASE} memiliki *encoder* dengan 12 *layers*, 12 *self-attention heads*, *hidden size* sebesar 768, dan 110M *parameters*. Sedangkan BERT_{LARGE} terdapat 24 *layers*, 16 *self-attention heads*, *hidden size* sebesar 1024, dan 340M *parameters*. BERT_{BASE} dilatih selama 4 hari menggunakan 4 cloud TPUs sedangkan BERT_{LARGE} membutuhkan 4 hari menggunakan 16 TPUs (Devlin *et al.*,

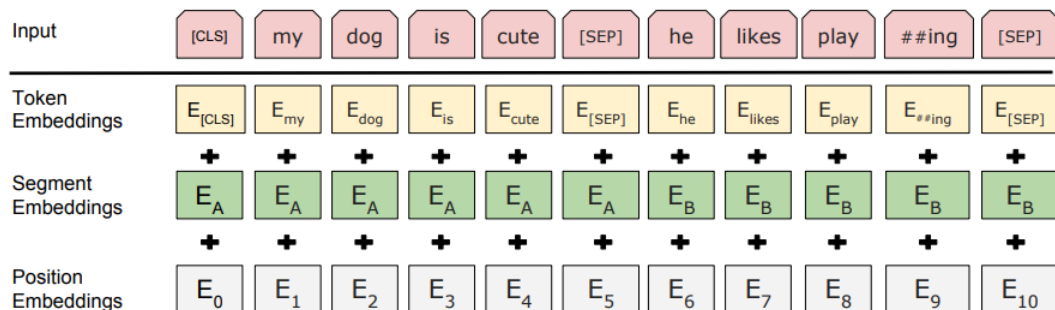
2019). Input teks yang digunakan oleh model BERT, akan diproses terlebih dahulu menggunakan tokenization. Terdapat beberapa syarat yang harus dipenuhi ketika menggunakan yaitu menambahkan token spesial di awal dan di akhir kalimat, penambahan token *padding* untuk kalimat yang panjangnya tidak maksimum untuk memastikan semua kalimat memiliki panjang yang sama. Token yang ditambahkan di awal kalimat adalah token [CLS] atau *Classifier-token*, token ini menandakan mulainya kalimat atau mulainya klasifikasi. Sementara token yang menandai akhir kalimat adalah [SEP] atau *Seperation-token* (Devlin *et al.*, 2019).

Pendekatan yang dilakukan oleh BERT bisa dibagi menjadi dua tahap, yaitu *pre-training* dan *fine-tuning*. Pada *pre-training*, model akan dilatih pada sejumlah besar teks yang tak berlabel dan akan melalui beberapa tugas, seperti masked language modelling dan prediksi kalimat. Tahapan *pre-training* ini memiliki fungsi untuk membuat model mempelajari serta memahami konteks dari bahasa yang digunakan. Setelah melakukan *pre-training*, model akan masuk ke tahap *fine-tuning* yang dapat disesuaikan dengan tugas tertentu secara spesifik. Pada tahap *fine-tuning*, model akan diinisiasi pada parameter yang telah dilatih sebelumnya sebelum akhirnya akan disesuaikan dengan menggunakan data yang berlabel. Pada tahap *fine-tuning*, model akan diinisiasi pada parameter yang telah dilatih sebelumnya sebelum akhirnya akan disesuaikan dengan menggunakan data yang berlabel (Sun *et al.*, 2019). Prosedur *pre-training* dan *fine-tuning* bisa dilihat di Gambar 2.4.



Gambar 2.4 Prosedur *pre-training* dan *fine tuning*

Dalam membantu BERT menangani berbagai tugas, representasi input digunakan untuk mewakili input berupa kalimat pada *token sequence*. Setiap token tersebut ketika dibuat representasinya akan memiliki tiga *embeddings*. Ketiga *embeddings* ini adalah *token-embedding*, *segment embedding* dan *positional embedding*. Ketiga embedding ini mengubah teks agar bisa dibaca oleh BERT. Representasi input pada BERT sendiri merupakan hasil akumulasi dari ketiga embedding tersebut (Devlin *et al.*, 2019). Visualisasi fungsi dari ketiga embedding pada representasi input ini diperlihatkan pada Gambar 2.5. Ketiga token ini memiliki fungsinya masing-masing pada model BERT.



Gambar 2.5 Visualisasi fungsi *token-embedding*, *segment embedding* dan *positional embedding*

1. *Token Embedding* merupakan layer yang pertama akan dilewati, *layer* ini akan menghasilkan representasi vektor dari tiap input. Pertama input akan memasuki tahap tokenisasi dengan memanfaatkan *WordPiece embedding*. Hal ini bertujuan untuk mengubah input menjadi bentuk token dan memberi *id* untuk tiap token. Setelah mendapatkan *id*, berikutnya token akan diubah menjadi bentuk representasi tokennya. Pada tahap ini juga setiap input akan diberikan token (CLS) dan token (SEP) sebagai penanda, penambahan token ini berfungsi sebagai representasi input untuk tugas klasifikasi dan memisahkan input teks. BERT menggunakan *WordPiece embeddings* dengan 30.000 token *vocabulary*.
2. *Segment Embedding* ini ditambahkan pada input token untuk membedakan tiap kalimat dan mengetahui urutan kalimat. Karena BERT memungkinkan untuk menerima input berupa pasangan kalimat, *layer* ini dibutuhkan untuk membedakan input tersebut. Layer ini hanya memiliki dua representasi, A untuk token pada kalimat pertama dan B untuk token pada kalimat kedua.
3. *Positional Embedding* merupakan layer yang terakhir ditambahkan pada input token. *Layer* ini ditambahkan pada tiap input token untuk mengetahui informasi terkait posisi token pada urutan. Informasi ini dibutuhkan oleh BERT melakukan pemahaman terkait konteks dari sebuah kata dalam kalimat. Sehingga ketika ada dua kata yang serupa, BERT akan dapat memahami konteks dari kalimat yang dimaksud.

BERT dapat dilatih untuk memahami sebuah bahasa dan dapat pula disempurnakan (*fine-tune*) untuk mempelajari tugas-tugas tertentu. Model BERT

dapat digunakan dengan dua cara, yaitu *pre-training* dan *fine-tuning*. Tahap pertama yaitu *pre-training* adalah tahap dimana BERT dibuat untuk memahami dan mempelajari bahasa dan konteksnya. Tahap kedua adalah *fine-tuning* dimana BERT dilatih dengan *dataset* khusus untuk pekerjaan yang spesifik dengan *dataset* yang dipelajari (Devlin *et al.*, 2019).

2.5.1 Pre-Train

Pre-train atau pelatihan pada BERT dilakukan menggunakan dua tugas unsupervised yang dijalankan secara bersamaan. Kedua fungsi untuk proses pelatihan tersebut merupakan Masked Language Model dan Next Sentence Prediction. Penggunaan kedua fungsi tersebut dilakukan untuk menggantikan decoder dalam proses pembelajaran. Karena pada model BERT sendiri hanya menggunakan encoder tanpa decoder (Devlin *et al.*, 2019). Untuk korpus *pre-train*, digunakan BooksCorpus sebanyak 800 juta kata dan Wikipedia bahasa Inggris 2.500 juta kata (Devlin *et al.*, 2019).

1. *Masked Language Model* (MLM)

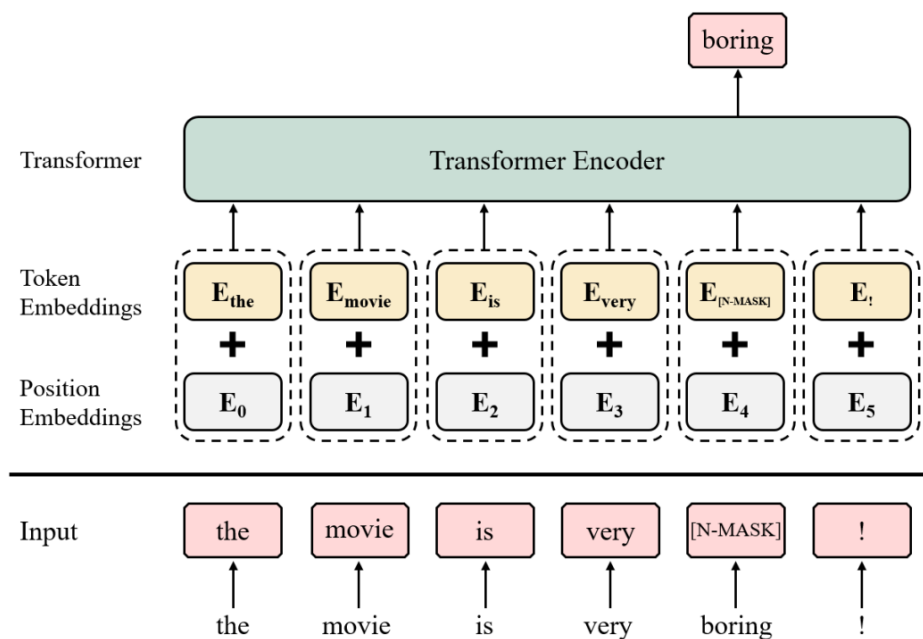
Language modelling adalah pekerjaan untuk memprediksi kata berikutnya yang diberikan urutan kata atau kalimat. Dalam *masked language modelling* alih-alih memprediksi setiap token berikutnya, akan ada persentase token input yang ditutupi secara acak dan hanya token ditutupi yang diprediksi.

Model dua arah (*Bi-directional*) lebih kuat daripada model satu arah (*Uni-directional*). Tetapi dalam model berlapis-lapis, model *Bi-directional* tidak berfungsi karena lapisan bawah memperlihatkan informasi dan memungkinkan token yang ditutupi untuk melihat dirinya sendiri di lapisan selanjutnya.

Implementasi *masked language modelling* dalam BERT, kata yang ditutupi tidak semuanya diubah menjadi token [MASK] karena nantinya token [MASK] tidak akan pernah terlihat sebelum fine tuning. Oleh karena itu 15% token dipilih secara acak.

- a) 80% dari token yang dipilih acak akan diubah menjadi token [MASK]
- b) 10% dari token yang dipilih acak akan diubah dengan kata lain secara acak
- c) 10% dari token yang dipilih secara acak tidak diubah

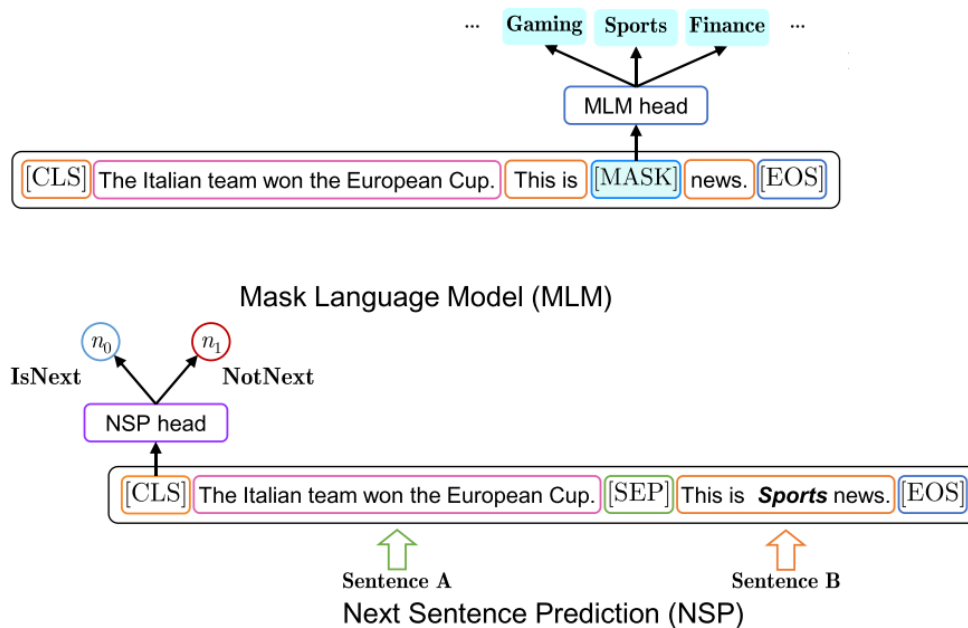
Keuntungan dari menggunakan model ini adalah encoder dari Transformer tidak mengetahui kata mana yang harus diprediksi atau diganti secara acak. Hal ini memaksa model untuk tetap melakukan representasi konteks distribusional pada tiap token input. Karena penggantian kata secara acak hanya dilakukan pada 1,5% dari keseluruhan token, membuat hal ini tidak mengganggu tahap pemahaman model. Visualisasi *masked language modelling* ada pada Gambar 2.6.



Gambar 2.6 Visualisasi *masked language modelling*

2. Next Sentence Prediction

Pada proses pembelajaran BERT, model dapat menerima input berupa pasangan kalimat. Model akan diminta untuk melakukan prediksi apakah kalimat kedua merupakan lanjutan dari kalimat pertama atau bukan. Selama tahap pembelajaran, model akan menerima dua input. Pertama 50% dari input merupakan pasangan kalimat, dimana kalimat kedua adalah kalimat lanjutan dari kalimat pertama. Kedua 50% dari input berasal dari kalimat yang diambil secara acak dari corpus sebagai pengganti dari kalimat kedua. Visualisasi *next sentence prediction* dan *mask language model* dapat dilihat di Gambar 2.7.



Gambar 2.7 Visualisasi Visualisasi *next sentence prediction* dan *mask language model* (Sun et al., 2021)

2.5.2 Fine-Tuning

Fine-tuning, merupakan tahap pemindahan kemampuan dari model yang sudah dilatih ke model baru. Model baru pada *fine-tuning* dapat disesuaikan dengan

tugas tertentu, semisal klasifikasi teks. Pada klasifikasi teks, BERT hanya akan menggunakan token (CLS) dalam melakukan tugas klasifikasi. Karena penggunaan token ini sudah dapat merepresentasikan keseluruhan kalimat. BERT akan menyesuaikan nilai dari token [CLS] dengan *dataset* yang ada, tahap ini disebut dengan *sentence similarity*. Adapun tahapan yang dilakukan pada tahap *fine-tuning* adalah sebagai berikut:

1. Tokenisasi dilakukan pada input teks. Hal ini dilakukan untuk memecah kata dari input teks menjadi token-token tertentu sesuai dengan kamus BERT serta menambahkan token spesial. Jika kata tersebut tidak memiliki token, maka BERT akan memecah kata tersebut menjadi sub kata yang memiliki token.
2. Token hasil proses tokenisasi akan diubah menjadi ke dalam bentuk *embedding*.
3. Tahap pembelajaran terjadi pada *encoder* sesuai dengan tahapan pada *Transformer*.
4. Pada layer terakhir hanya output dari token [CLS] saja yang digunakan untuk melakukan klasifikasi.

2.6 Hyperparameter

Hyperparameter adalah pengaturan yang digunakan untuk mengontrol tingkah laku dari algoritma pembelajaran pada *machine learning*. Setiap algoritma pasti memiliki nilai dari *hyperparameter* yang akan diatur sebelum proses pelatihan dilakukan sebab nilai sebuah *hyperparameter* biasanya tidak didapatkan dari proses pembelajaran. Dengan menggunakan berbagai konfigurasi *hyperparameter* bisa didapatkan konfigurasi yang paling sesuai (Wu, Perin and Picek, 2020).

Dalam *machine learning*, *hyperparameter tuning* adalah tantangan dalam memilih kumpulan *hyperparameter* yang sesuai untuk algoritma pembelajaran. *Hyperparameter tuning* adalah nilai untuk parameter yang digunakan untuk mempengaruhi proses pembelajaran. Selain itu, faktor-faktor lain, seperti bobot simpul juga dipelajari. Untuk menggeneralisasi pola data yang beragam, model *machine learning* yang sama akan memerlukan batasan, bobot, atau kecepatan pembelajaran yang berbeda.

2.6.1 Batch Size

Batch size adalah jumlah sampel data yang biasanya melewati jaringan saraf pada satu waktu. *Batch size* menentukan jumlah sampel yang harus dikerjakan sebelum memperbarui parameter model internal. Perbandingan *Batch* dan *Batch size* adalah dimana *batch* merupakan kelompok yang digunakan untuk menggabungkan satu atau lebih sampel data. Sedangkan *Batch size* adalah jumlah total sampel data training yang ada di setiap batch (Brownlee, 2018).

Semakin kecil nilai batch size, maka proses *training* yang dilakukan akan memakan memori yang semakin sedikit. Begitu pula sebaliknya, semakin besar nilai *batch size*, maka proses *training* yang dilakukan akan memakan memori yang semakin banyak. Contoh nilai *batch size* yaitu 16, 32, 64, 128, 256, dan 512.

2.6.2 Learning-rate

Learning-rate adalah *hyperparameter* yang mengontrol seberapa banyak perubahan model dalam menanggapi estimasi kesalahan setiap kali bobot model diperbarui. Memilih *learning rate* merupakan hal yang menantang karena nilai

yang terlalu kecil dapat mengakibatkan proses pelatihan yang lama, sedangkan nilai yang terlalu besar dapat mengakibatkan pembelajaran set bobot sub-optimal terlalu cepat atau proses pelatihan yang tidak stabil .

Learning rate mungkin merupakan *hyperparameter* yang paling penting ketika mengkonfigurasi *neural network*. Oleh karena itu, sangat penting untuk mengetahui bagaimana cara menyelidiki efek *learning rate* pada kinerja model (Kingma and Ba, 2015).

2.6.3 *Epoch*

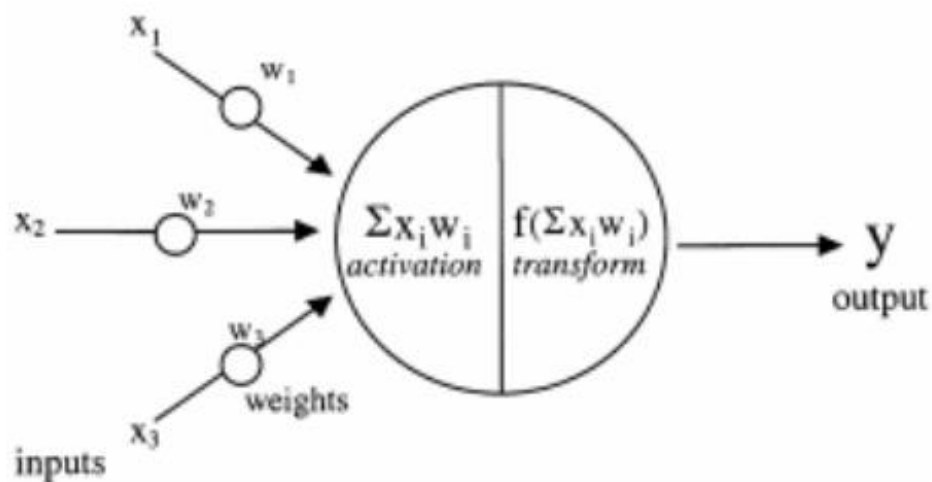
Epoch adalah ketika seluruh dataset sudah melalui proses *training* pada *Neural Network* sampai dikembalikan ke awal untuk sekali putaran, karena satu *Epoch* terlalu besar untuk dimasukkan (*feeding*) kedalam komputer maka dari itu kita perlu membaginya kedalam satuan kecil (*batches*).

Semakin besar nilai *epoch*, maka semakin baik tingkat akurasi yang dihasilkan pada proses *training* yang artinya proses *training* yang dilakukan akan semakin lama. Perlu diingat juga bahwa jika nilai *epoch* terlalu besar maka akan menyebabkan terjadinya *overfitting*, begitu pula sebaliknya jika nilai *epoch* terlalu kecil maka akan menyebabkan terjadinya *underfitting*. Setiap dataset akan memerlukan nilai *Epoch* yang berbeda-beda oleh sebab itu diperlukan percobaan beberapa kali untuk mendapatkan nilai yang optimal (Brownlee, 2018).

2.6.4 *Activation Function*

Dalam penyusunannya, *deep learning* melibatkan fungsi aktivasi pada arsitekturnya. Fungsi aktivasi adalah fungsi-fungsi yang akan menghitung

masukan dan bias pada *neural network* dengan melibatkan bobot yang ditentukan sebelumnya. Fungsi ini juga yang akan menentukan *neuron-neuron* yang perlu diaktifkan pada proses tersebut. Fungsi-fungsi ini akan memanipulasi data yang masuk dan menghasilkan *output* sesuai yang diinginkan. Fungsi-fungsi ini akan memanipulasi data yang masuk dan menghasilkan output sesuai yang diinginkan. Gambar 2.8 menunjukkan visualisasi bagaimana input yang masuk dioperasikan dengan variabel w atau *weight* lalu diaplikasikan ke fungsi aktivasi dan ditransformasikan hingga pada akhirnya menjadi keluaran y (Nwankpa *et al.*, 2018).



Gambar 2.8 visualisasi *activation function* (Nwankpa *et al.*, 2018)

Fungsi aktivasi memiliki beberapa jenis yang penggunaannya disesuaikan dengan kebutuhan dan jenis *output* yang diharapkan. *Output* dari *neuron* pada *deep learning* dapat dikontrol dengan pemilihan fungsi aktivasi yang tepat. Oleh karena itu, pemilihan fungsi aktivasi yang tepat dapat membuat model memberikan hasil yang terbaik.

2.6.5 *Optimizer*

Optimizer atau pengoptimal adalah sebuah metode yang digunakan untuk meminimalkan loss function atau untuk memaksimalkan efisiensi dari pemodelan. *Optimizer* merupakan fungsi matematika yang bergantung pada parameter model yang dapat dipelajari, yaitu bobot (*weights*) & bias (*biases*). *Optimizer* membantu mengetahui cara mengubah bobot dan *learning-rate* dari jaringan syaraf (*neural network*) untuk mengurangi kerugian (*losses*) (Sun, 2019). *Optimizer* terdiri dari beberapa jenis diantaranya RMSProp, SGD, Adam, Nadam, RAdam, dan lain-lain. Untuk mengetahui *optimizer* terbaik dengan tujuan mendapatkan model yang optimal, dalam penelitian ini akan dilakukan uji perbandingan dengan 3 jenis *optimizer* diantaranya:

A. **Adam (*Adaptive Moment Estimation*)**

Adam adalah salah satu teknik optimisasi adaptif yang dapat digunakan sebagai pengganti dari *classical stochastic gradient descent* untuk memperbarui bobot secara iteratif berdasarkan data *training*. Adam bisa dikatakan sebagai kombinasi dari RMSprop dan *stochastic gradient descent* dengan momentum. Adam pertama kali diperkenalkan ke publik pada tahun 2015 oleh Diedrik Kingma dari OpenAI dan Jimmy Ba dari University of Toronto (Kingma and Ba, 2015). Adam adalah metode optimisasi dengan *learning-rate* yang adaptif, dimana adam menghitung *learning-rate* individu untuk parameter yang berbeda. Adam menggunakan estimasi gradien momen pertama dan kedua untuk mengadaptasi *learning-rate* untuk setiap bobot jaringan saraf, karena itu adam adalah singkatan dari *adaptive moment estimation*.

Adam berbeda dari *classical stochastic gradient descent*. *Stochastic gradient descent* menggunakan *single learning-rate (alpha)* untuk semua pembaruan bobot dan *learning-rate* tidak berubah selama *training*. *Learning-rate* dipertahankan untuk setiap bobot jaringan (*parameter*) dan diadaptasi secara terpisah saat *learning* berkembang. Metode menghitung *learning-rate* adaptif individu untuk *parameter* yang berbeda dari perkiraan momen pertama dan kedua dari gradien. Adam bisa digambarkan sebagai gabungan dari kelebihan dua ekstensi dari *stochastic gradient descent*. Secara khusus:

1. *Adaptive gradient* (AdaGrad) yang mempertahankan *learning-rate* per-*parameter* yang meningkatkan kinerja pada masalah dengan gradien yang menyebar.
2. *Root mean square propagation* (RMSProp) yang juga mempertahankan *learning-rate* per-*parameter* yang berdasarkan rata-rata besaran gradien terbaru untuk bobot. Artinya, algoritma berfungsi dengan baik pada masalah yang tidak konstan seperti jika ada banyak *noise*.

Adam lebih banyak digunakan dalam natural language (NLP) dan speech serta lebih aman saat digunakan untuk inisialisasi bobot dan cepat dalam model belajar. Persamaan 2.1 sampai 2.5 merupakan perhitungan untuk setiap bobot yang diperbarui (Kingma and Ba, 2015).

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \quad (2.1)$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \quad (2.2)$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \quad (2.3)$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \quad (2.4)$$

$$w_t = w_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}} \quad (2.5)$$

Keterangan:

η : tingkat pembelajaran awal (*learning-rate*)

ϵ : sebuah angka kecil yang digunakan untuk mencegah pembagian dengan 0
(nilainya 1e-8 atau 1e-10)

β_1 dan β_2 : Exponential decay rates (nilainya masing-masing 0.9 dan 0.999)

B. RMSprop

RMSProp, *root mean square propagation*, adalah algoritma/metode optimasi yang dirancang untuk pelatihan *Artificial Neural Network* (ANN). Dan ini adalah algoritma yang tidak dipublikasikan yang pertama kali diusulkan dalam kursus Coursera. “Neural Network for Machine Learning” kuliah ke-enam oleh Geoff Hinton. RMSProp terletak di ranah metode tingkat pembelajaran adaptif, yang semakin populer dalam beberapa tahun terakhir karena merupakan perpanjangan dari algoritma Stochastic Gradient Descent (SGD), metode momentum, dan dasar dari algoritma Adam. RMSProp pengaplikasian *stochastic technology for mini-batch gradient descent*.

RMSprop mirip dengan algoritma *gradient descent* dengan momentum. RMSprop membatasi osilasi dalam arah vertikal. Oleh karena itu, RMSprop dapat meningkatkan kecepatan belajar dan algoritma dapat mengambil langkah yang lebih besar dalam arah horizontal yang konvergen lebih cepat. Perbedaan antara

RMSprop dan penurunan gradien adalah pada bagaimana gradien dihitung. RMSprop adalah pengoptimal yang memanfaatkan besarnya gradien terbaru untuk menormalkan gradient, yang menjaga rata-rata bergerak di atas gradien root mean square oleh karena itu disebut dengan Rms. $f'(\theta_t)$ menjadi turunan dari loss sehubungan dengan parameter pada langkah waktu t . Dalam bentuk dasarnya, diberikan tingkat langkah α dan istilah peluruhan γ dilakukan persamaan berikut.

$$r_t = (1 - \gamma) f'(\theta_t)^2 + \gamma r_t - 1 \quad (2.6)$$

$$v_{t+1} = \frac{\alpha}{\sqrt{r_t}} f'(\theta_t) \quad (2.7)$$

$$\theta_{t+1} = \theta_t - v_{t+1} \quad (2.8)$$

Keterangan:

α : tingkat pembelajaran awal (*learning-rate*)

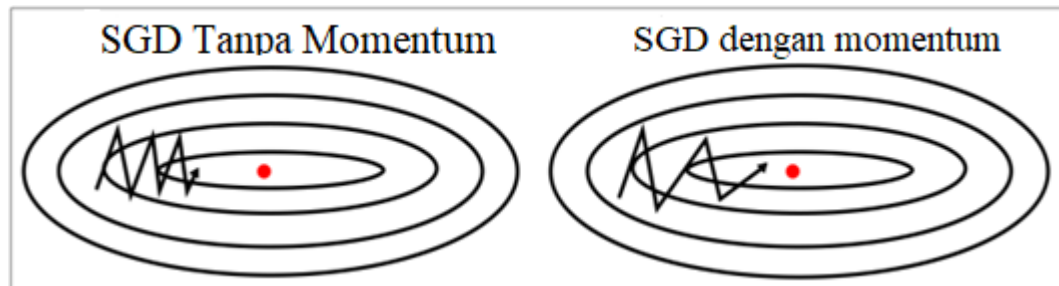
r_t : rata-rata eksponensial dari kotak gradien

θ_t : gradien pada waktu t sepanjang v_j

C. Nadam

Algoritma *Nesterov-accelerated Adaptive Moment Estimation*, atau Nadam, adalah perpanjangan dari algoritma optimisasi *Adaptive Movement Estimation* (Adam) untuk menambahkan *Nesterov Accelerated Gradient* (NAG) atau *Nesterov momentum*, yang merupakan jenis momentum yang ditingkatkan (Dozat, 2016). Momentum itu seperti bola yang menggelinding menuruni bukit. Bola akan mendapatkan momentum saat menggelinding menuruni bukit. Momentum membantu mempercepat *Gradient Descent* (GD) ketika memiliki permukaan yang

melengkung lebih curam ke satu arah daripada ke arah lain. Itu juga meredam osilasi seperti yang ditunjukkan pada Gambar 2.9.



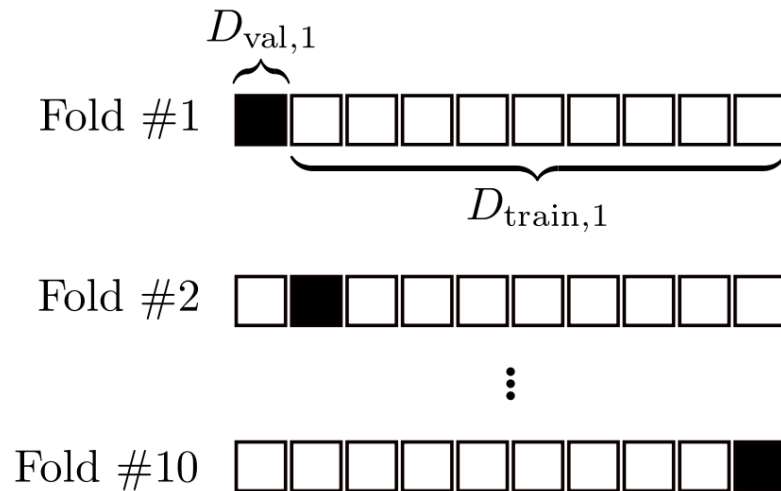
Gambar 2.9 Perbandingan algoritma SGD tanpa momentum dan dengan momentum (Du, 2019)

2.7 Cross Validation

Cross validation adalah salah satu metode *resampling* data yang diterapkan dalam menilai atau mengevaluasi kemampuan prediktif dari model yang dibangun dan mencegah terjadinya *overfitting* pada model yang dibangun. Secara teknis, *cross validation* akan memisahkan *dataset* yang akan digunakan untuk *training* dan *testing* model yang dibangun. *Dataset* tersebut dipisah sehingga data yang digunakan untuk melakukan *training* tidak akan dipakai kembali ketika model melakukan *testing* terhadap hasil *training*. Salah satu jenis dari *cross validation* yang sering digunakan adalah *k-fold cross-validation*. *K-fold cross validation* akan mempartisi data sejumlah k partisi dan mengulangi eksperimen sebesar k kali. Tujuannya agar semua data bergantian dalam evaluasi sehingga dapat diperoleh hasil akurasi yang maksimal (Berrar, 2018).

K-fold Cross-validation adalah penerapan *cross validation* dengan membagi data menjadi sebanyak k subset. Subset tersebut kemudian dibagi menjadi data

untuk *learning* dan validasi. Sebagai contoh, nilai k yang dipilih adalah 10. Gambar 2.10 adalah visualisasi cara kerja dari *10-fold cross-validation*.



Gambar 2.10 visualisasi cara kerja dari *10-fold cross-validation* (Berrar, 2018)

Pada visualisasi *10-fold cross-validation* tersebut, *dataset* yang digunakan untuk *training* adalah sebanyak $k-1$ subset dan satu subset digunakan untuk melakukan validasi terhadap hasil *training* dari model. D_{val} ditandai untuk subset *testing*, sedangkan D_{train} ditandai sebagai subset yang akan dipakai untuk *training*. D_{val} akan digunakan untuk mengetes hasil *training* dari D_{train} . Prosedur tersebut akan berulang sampai semua subset pernah menjadi D_{val} .

2.8 Confusion Matrix

Confusion matrix adalah salah satu metode dalam *machine learning* yang digunakan untuk melakukan evaluasi pada model. Pada *confusion matrix*, kolom mewakili hasil prediksi, sementara baris mewakili nilai sebenarnya. Misalkan untuk memprediksi kasus dimana terdapat dua kelas kata, yaitu positif dan negatif. Maka akan terdapat 4 kelas pada perhitungan *confusion matrix*. Kelas positif

diklasifikasikan menjadi dua, sebagai *true positive* *false positive*. Kelas negatif diklasifikasikan menjadi dua, sebagai *true negative* dan *false negative* (Sokolova and Lapalme, 2009).

Tabel 2.1 *Confusion matrix*

		<i>Actual Class</i>	
		<i>Positive (P)</i>	<i>Negative (N)</i>
<i>Predicted Class</i>	<i>Positive (P)</i>	<i>True Positive (TP)</i>	<i>False Positive (FP)</i>
	<i>Negative (N)</i>	<i>False Negative (FN)</i>	<i>True Negative (TN)</i>

Keterangan:

1. *True Positive (TP)* merupakan banyaknya data yang kelas aktualnya adalah kelas positif dengan kelas prediksinya merupakan kelas positif (benar dengan nilai positif).
2. *True Negative (TN)* merupakan banyaknya data yang kelas aktualnya adalah kelas negatif dengan kelas prediksinya merupakan kelas negatif (benar dengan nilai negatif).
3. *False Positive (FP)* merupakan banyaknya data yang kelas aktualnya adalah kelas negatif dengan kelas prediksinya merupakan kelas positif (salah dengan nilai positif).
4. *False Negative (FN)* merupakan banyaknya data yang kelas aktualnya adalah kelas positif dengan kelas prediksinya merupakan kelas negatif (salah dengan nilai negatif).

Dengan menggunakan confusion matrix, maka dapat dihitung *performance metrics* yang dapat mengukur kinerja model yang telah dibuat. Beberapa *performance metrics* antara lain *accuracy*, *precision*, *recall* dan *f-score* atau *f-measure*.

1. *Accuracy* adalah keakuratan dari model dalam mengklasifikasikan data dengan benar dan berdasarkan nilai prediksi dan nilai sebenarnya. *Accuracy* menghitung rasio prediksi benar dengan keseluruhan data. Namun nilai *accuracy* memiliki beberapa kelemahan seperti kurang informatif dan *bias*, sehingga perlu digunakannya metrik pengukuran lain untuk membantu dalam pengukuran performa model yang dibuat, yaitu nilai *precision*, *recall*, dan *f1-score*. *Accuracy* dapat dihitung menggunakan persamaan 2.9.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \quad (2.9)$$

2. *Precision* adalah keakuratan dari data yang diminta dengan hasil prediksi oleh model. *Precision* adalah nilai dari rasio prediksi benar positif terhadap keseluruhan hasil yang diprediksi positif. Nilai *precision* digunakan untuk mengukur pola positif yang diprediksi dengan benar (*True Positive*) dari total pola prediksi dalam kelas positif. Pada multiclass classification ada dua jenis *precision* yaitu mikro(μ) dan makro(M) dengan perhitungan nilai *precision* dapat dilihat pada Persamaan 2.10 dan 2.11.

$$Precision_{\mu} = \frac{\sum_{i=1}^l TP_i}{\sum_{i=1}^l (TP_i + FP_i)} \quad (2.10)$$

$$Precision_M = \frac{\sum_{i=1}^l \frac{TP_i}{TP_i + FP_i}}{l} \quad (2.11)$$

3. Recall adalah kemampuan dari model untuk menemukan kembali sebuah informasi. Recall adalah nilai rasio prediksi benar positif terhadap keseluruhan data yang benar positif. Nilai *recall* digunakan untuk mengukur rasio prediksi benar positif dibandingkan dengan keseluruhan data yang benar positif. Pada *multiclass classification* ada dua jenis *recall* yaitu mikro(μ) dan makro(M) dengan perhitungan nilai *recall* dapat dilihat pada Persamaan 2.12 dan Persamaan 2.13.

$$Recall_{\mu} = \frac{\sum_{i=1}^l TP_i}{\sum_{i=1}^l (TP_i + FN_i)} \quad (2.12)$$

$$Recall_M = \frac{\sum_{i=1}^l \frac{TP_i}{TP_i + FN_i}}{l} \quad (2.13)$$

4. Nilai *f1-score* merupakan perbandingan rata-rata *precision* dan *recall* yang dibobotkan. Perhitungan nilai *f1-score* dapat dilihat pada Persamaan 2.14 dan Persamaan 2.15.

$$F1\ Score_{\mu} = 2 \times \frac{Precision_{\mu} \times Recall_{\mu}}{Precision_{\mu} + Recall_{\mu}} \quad (2.14)$$

$$F1\ Score_M = 2 \times \frac{Precision_M \times Recall_M}{Precision_M + Recall_M} \quad (2.15)$$

Dimana seluruh rentang nilai pada keempat perhitungan di atas, terdiri dari 0 sampai 1. Hasil yang lebih baik akan ditandai dengan nilai yang lebih mendekati nilai 1 dan sebaliknya (Haddi, Liu and Shi, 2013).

2.9 Streamlit

Streamlit adalah sebuah *framework* aplikasi yang bersifat *open source* dalam bahasa Python (Streamlit Inc., 2022). Streamlit dibuat untuk memudahkan

pengembangan aplikasi web dalam bidang *data science* dan *machine learning*. Dengan menggunakan streamlit, pengembangan aplikasi web dapat dilakukan dengan mudah dan cepat karena hanya perlu menggunakan bahasa Python. Arsitektur streamlit memungkinkan pengembang untuk membuat aplikasi dengan menggunakan alur yang sama seperti membuat skrip Python. *Framework* ini juga kompatibel dengan *library-library* Python sehingga lebih memudahkan dalam pengembangan aplikasi *data science*.

Layaknya program yang dibuat menggunakan bahasa Python, aplikasi web yang dikembangkan menggunakan streamlit dapat melakukan proses impor data, transformasi data, visualisasi data, hingga melakukan prediksi data seperti membuat untuk melakukan klasifikasi judul artikel berbahasa Inggris. Proses analisis dengan menggunakan streamlit bersifat interaktif dalam artian pengguna aplikasi dapat memasukkan nilai-nilai yang diinginkan dengan menggunakan widget dan hasil akan langsung tampil dalam *website* tersebut.

2.10 Unified Modeling Language (UML)

Unified Modeling Language atau yang biasa disingkat menjadi UML adalah bahasa standar yang digunakan untuk melakukan perancangan perangkat lunak (Booch et al., 1999). UML berfungsi untuk membantu tahapan perancangan sistem perangkat lunak dengan memvisualisasikan, menspesifikasikan, membangun, dan mendokumentasikan artefak dari sistem tersebut. Visualisasi pada UML berisikan simbol-simbol dan notasi yang masing-masing memiliki makna tersendiri. Dengan ini, saat pengembang sistem perangkat lunak memvisualisasikan model yang akan dikembangkan dengan menggunakan UML, pengembang lainnya dapat

menginterpretasikan model tersebut dengan mudah. Menspesifikasikan dalam UML memiliki arti untuk membangun model yang tepat, tidak ambigu, dan lengkap. UML dapat juga digunakan untuk memetakan model ke bahasa pemrograman bahkan ke table pada sebuah database relasional. UML juga berfungsi untuk mendokumentasikan rancangan perangkat lunak seperti arsitektur sistem beserta detail-detailnya.


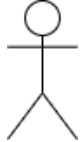
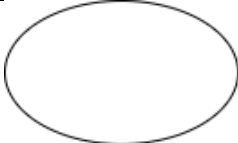




Terdapat tiga elemen penting dalam UML, yaitu *things*, *relationships*, dan diagram. *Things* dalam UML merupakan elemen-elemen yang merepresentasikan node dalam diagram, seperti *class*, *package*, *interface*, *use case*, dan lain sebagainya. *Relationship* dalam UML, seperti artinya secara harafiah, merupakan elemen yang menggambarkan hubungan antar *things*. Elemen-elemen hubungan ini berbentuk panah atau garis yang dapat melambangkan ketergantungan, asosiasi, generalisasi, maupun realisasi. Elemen terakhir pada UML adalah diagram yang merupakan presentasi grafis yang berisikan elemen-elemen *things* yang terhubung dengan elemen-elemen hubungan. Diagram dibuat sebagai visualisasi sistem agar lebih mudah dimengerti. Terdapat berbagai macam diagram dalam UML, diantaranya adalah *use case diagram* dan *activity diagram*.

2.10.1 Use Case Diagram

Use case diagram adalah diagram yang berfungsi untuk pemodelan perilaku sebuah sistem, subsistem, atau kelas dengan menggambarkan relasi antara sistem tersebut dengan aktor dalam sebuah *use case* (Booch et al., 1999). Diagram ini biasanya digunakan untuk memodelkan konteks sebuah sistem dengan cara menetapkan aktor-aktor beserta peranannya.

Diagram ini menggambarkan keseluruhan sistem secara umum sehingga elemen-elemen di dalamnya tidak banyak. Elemen-elemen pada *use case diagram* dapat dilihat pada Tabel 2.2 berikut.



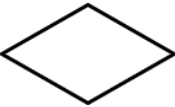


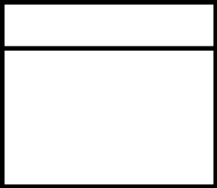
Tabel 2.2 Elemen-elemen *Use Case Diagram*

No	Elemen	Simbol	Keterangan
1	Sistem		Batasan-batasan proses yang sudah dideskripsikan dalam sebuah sistem
2	Aktor		Elemen pemicu sistem. Dapat berupa orang, mesin, ataupun sistem lain
3	<i>Use Case</i>		Potongan proses yang merupakan bagian dari sistem
4	<i>Association</i>		Menggambarkan interaksi antara use case dan aktor
5	<i>Dependency Include</i>		Menghubungkan dua use case jika satu di antaranya membutuhkan use case yang lainnya
6	<i>Dependency Extend</i>		Menghubungkan dua use case jika satu di antaranya terkadang akan memanggil use case yang lainnya tergantung pada kondisi
7	<i>Generalization</i>		Menggambarkan pewarisan antara dua aktor atau use case dimana salah satu mewarisi properti ke aktor atau use case yang lainnya

2.10.2 Activity Diagram

Activity diagram adalah sebuah diagram yang digunakan untuk pemodelan aspek dinamik dari sebuah sistem dengan melibatkan pemodelan langkah sekuensial pada sebuah proses komputasi (Booch et al., 1999). Pada dasarnya, diagram ini adalah sebuah *flowchart* yang menggambarkan alur control dari satu aktivitas ke aktivitas lainnya dengan beberapa elemen tambahan yang dapat digunakan pada kasus-kasus tertentu. Elemen-elemen yang digunakan di dalam *activity diagram* dapat dilihat pada Tabel 2.3 berikut.

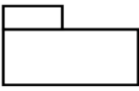
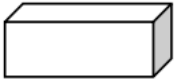


Tabel 2.3 Elemen-elemen *Activity Diagram*

No	Elemen	Simbol	Keterangan
1	Status Awal / <i>initial state</i>		Menandai dimulainya aktivitas
2	Aktivitas / <i>action state</i>		Aktivitas yang dilakukan sistem. Aktivitas biasanya diawali dengan kata kerja
3	Percabangan / <i>Decision / Sequential branch</i>		Percabangan dimana ada pilihan aktivitas yang lebih dari satu
4	Penggabungan / <i>Concurrent join</i>		Penggabungan dimana yang mana lebih dari satu aktivitas lalu digabungkan jadi satu
5	Status Akhir		Menandai berakhirnya aktivitas
6	<i>Swimlane</i>		Swimlane memisahkan organisasi bisnis yang bertanggung jawab terhadap aktivitas yang terjadi

2.10.3 Deployment Diagram

Deployment diagram menunjukkan konfigurasi dari *run time processing nodes* dan juga komponen yang terdapat di dalamnya. Diagram ini digunakan untuk memodelkan bagaimana sistem berjalan saat tahap *deployment*. Sering kali melibatkan pemodelan topologi dari perangkat keras yang mengeksekusi sistem. Penjelasan dari simbol-simbol *deployment diagram* dapat dilihat pada Tabel 2.4.

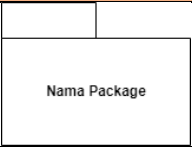
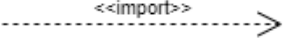
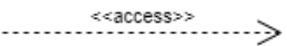
Tabel 2.4 Elemen-elemen *Sequence Diagram*

No	Elemen	Simbol	Keterangan
1	<i>Package</i>		<i>Package</i> merupakan simbol bungkusan dari satu atau lebih komponen
2	<i>Node</i>		Merepresentasikan <i>hardware</i> atau <i>execution environment</i> .
3	<i>Dependency</i>		Ketergantungan antar komponen, arah panah mengarah pada komponen yang dipakai
4	<i>Link</i>		Relasi antar komponen

2.10.4 Package Diagram

Package dalam UML adalah elemen yang mengorganisir elemen menjadi berkelompok layaknya sebuah folder (Booch et al., 1999). Sebuah *package* dapat memiliki elemen-elemen tersendiri seperti kelas, *interface*, komponen, *node*, kolaborasi, *use case*, diagram, bahkan *package* lain. Sedangkan *package diagram* merupakan diagram yang digunakan untuk memvisualisasikan *package* dalam suatu sistem serta relasinya. Terdapat dua jenis relasi antar *package*, yaitu dependensi *import* dan *access*. Tabel 2.5 berikut menjelaskan mengenai elemen-elemen pada *package diagram* beserta simbol dan penjelasannya.

Tabel 2.5 5Elemen-elemen *Package Diagram*

No	Elemen	Simbol	Keterangan
1	<i>Package</i>		Sekelompok elemen-elemen model
2	<i>Import</i>		Suatu dependensi yang mengindikasikan isi tujuan paket secara umum yang ditambahkan ke dalam sumber paket
3	<i>Access</i>		Suatu dependensi yang mengindikasikan isi tujuan paket secara umum yang bisa digunakan pada nama sumber paket

2.11 Penelitian Terkait

Penelitian yang berkaitan dengan NLP, BERT, dan *dataset* yang digunakan telah banyak dilakukan sebelumnya. Penelitian-penelitian terkait dapat dijadikan referensi dalam penelitian dan sebagai pembandingan hasil evaluasi model yang dibangun. Pada penelitian oleh Mulahuwaisha dkk. melakukan klasifikasi judul berita bahasa inggris menggunakan k-Nearest Neighbors (kNN), Support Vector Machine (SVM), Decision Tree (DT), dan Long Short-Term Memory (LSTM) dengan dataset *news aggregator* dari *University of California, Irvine, School of Information and Computer Sciences* yang menghasilkan akurasi terbaik 95.04% (SVM) dan akurasi yang terburuk yaitu 88.72%(kNN) (Mulahuwaish *et al.*, 2020). Pada penelitian oleh Mahajan dan Ingle melakukan klasifikasi artikel menggunakan naïve bayes dan TF-IDF yang menghasilkan hasil *f1-score* tertinggi 68% dengan naïve bayes (Mahajan, 2021).