

# Studi Komparasi Performa BERT dan DistilBERT Pada Kasus Analisis Sentimen Berbahasa Indonesia

Hilmy Rahmadani<sup>a,1,\*</sup>

<sup>a</sup>Program Studi Statistika, Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Indonesia

<sup>1</sup>rahmadanihilmy@gmail.com

---

## ARTICLE INFO



NPM 2206810490

## ABSTRACT

Penelitian ini mengevaluasi performa model BERT dan DistilBERT dalam tugas analisis sentimen pada cuitan terkait kebijakan Pemberlakuan Pembatasan Kegiatan Masyarakat (PPKM) di Indonesia. Model-model ini dievaluasi berdasarkan metrik accuracy, precision, recall, dan f1-score, serta aspek efisiensi waktu pelatihan dan jumlah parameter yang dapat dilatih. Hasil eksperimen menunjukkan bahwa BERT memberikan performa evaluasi yang unggul, sementara DistilBERT menawarkan efisiensi waktu pelatihan yang lebih baik. Oleh karena itu, DistilBERT dapat menjadi pilihan yang lebih efisien dalam skenario yang memerlukan pelatihan cepat tanpa penurunan performa yang signifikan.

Copyright © 2024.  
All rights reserved.

---

*Keywords:* BERT; DistilBERT; analisis sentimen; PPKM; efisiensi model.

---

## 1 Pendahuluan

Kebijakan Pemberlakuan Pembatasan Kegiatan Masyarakat (PPKM) yang diterapkan di Indonesia sejak masa pandemi COVID-19 merupakan upaya pemerintah untuk menekan laju penyebaran virus. Kebijakan ini berdampak langsung pada aktivitas masyarakat di berbagai sektor, termasuk sosial, ekonomi, dan kesehatan. Sebagai langkah strategis, kebijakan PPKM menjadi salah satu topik yang banyak dibicarakan oleh masyarakat, terutama melalui platform media sosial seperti Twitter. Cuitan masyarakat mengenai kebijakan ini menunjukkan berbagai sudut pandang dan sentimen, mulai dari dukungan hingga kritik, yang dapat memberikan wawasan penting bagi pemerintah terkait respons masyarakat (Adel et al., 2022).

Salah satu pendekatan yang dapat digunakan untuk menangkap dan menganalisis opini masyarakat terhadap suatu kebijakan adalah analisis sentimen. Analisis sentimen merupakan cabang dari pemrosesan bahasa alami (*Natural Language Processing*, NLP) yang bertujuan untuk mengidentifikasi dan mengkategorikan opini yang dinyatakan dalam bentuk teks, khususnya untuk menentukan apakah sentimen yang terkandung dalam suatu opini itu bersifat positif, negatif, atau netral. Dengan semakin meluasnya penggunaan media sosial, analisis sentimen menjadi alat yang sangat penting untuk memahami persepsi masyarakat dalam jumlah besar secara efisien. Melalui teknik ini, pemerintah maupun

peneliti dapat mengevaluasi dampak sosial dari kebijakan yang diterapkan serta mengidentifikasi isu-isu yang menjadi perhatian utama masyarakat. Analisis sentimen dapat dilakukan dengan berbagai macam metode, salah satunya adalah metode tradisional *machine learning* seperti *K-Nearest Neighbor* (KNN), dan *Support Vector Machine* (SVM) (Setiawan, 2024).

Meskipun demikian, metode tradisional untuk analisis sentimen memiliki keterbatasan dalam memahami aspek bahasa dan konteks secara mendalam. Misalnya, sebuah ulasan negatif bisa disebabkan oleh kesalahpahaman, sementara kata-kata yang terdengar seperti puji dapat mengandung unsur ironi. Seiring dengan pesatnya perkembangan teknologi *Natural Language Processing* (NLP), bermunculan berbagai model berbasis Large Language Model (LLM) yang mampu meningkatkan akurasi analisis sentimen, termasuk untuk teks dalam bahasa non-Inggris seperti Bahasa Indonesia. LLM adalah model berbasis jaringan saraf dengan miliaran parameter yang dilatih melalui pembelajaran berbasis *self-supervised learning*, yang memungkinkan model memahami struktur bahasa secara lebih fleksibel (Setiawan, 2024).

Model-model seperti BERT dan DistilBERT memainkan peran penting dalam berbagai aplikasi NLP modern karena kemampuannya dalam menangkap konteks kata berdasarkan posisi dan relasinya dalam kalimat. BERT, salah satu model LLM yang dikembangkan oleh tim Google AI, telah menunjukkan performa state-of-the-art (SOTA) dalam berbagai tugas NLP, dengan pendekatan berbasis representasi kontekstual dua arah yang efektif untuk memahami arti kata dalam berbagai konteks (Devlin et al., 2018). Kemampuan representasi kontekstual ini membuat BERT efektif dalam mengungkap opini publik yang tersembunyi di balik sutatu komentar atau ulasan. Selain itu, pengembangan lebih lanjut seperti BERT-NN, yaitu penggabungan BERT dengan fully-connected neural network, telah berhasil mengungguli metode klasik seperti Logistic Regression, SVM, KNN, CNN, GRU, dan LSTM dalam berbagai studi klasifikasi sentimen (Setiawan, 2024).

Dalam konteks analisis sentimen berbahasa Indonesia, model BERT dan variannya seperti DistilBERT menawarkan solusi yang efektif dalam memahami teks secara lebih efisien. BERT, dengan mekanisme perhatian dua arah (*bidirectional attention*), mampu menangkap hubungan antar kata dalam satu kalimat secara menyeluruh. Studi yang dilakukan oleh Setiawan (2024) berfokus pada implementasi model BERT dalam tugas analisis sentimen terhadap ulasan hotel berbahasa Indonesia. Dalam penelitian tersebut, BERT digunakan sebagai model representasi teks yang kemudian dihubungkan dengan jaringan saraf sederhana (BERT-NN) untuk melakukan klasifikasi sentimen. Hasil penelitian menunjukkan bahwa BERT mampu memahami konteks bahasa Indonesia secara efektif dan memberikan performa yang kompetitif dalam mengklasifikasikan opini publik dari data ulasan.

Meskipun demikian, model BERT memiliki keterbatasan, yaitu membutuhkan waktu pelatihan yang lama dan sumber daya komputasi yang besar. Untuk mengatasi keterbatasan tersebut, dikembangkanlah model DistilBERT, yaitu model yang dikembangkan dari hasil distilasi dari model BERT yang bertujuan untuk menghasilkan model yang lebih ringan dengan jumlah parameter yang lebih kecil daripada model BERT biasa. Meskipun varian BERT ini lebih ringan, varian ini tetap mempertahankan sebagian besar performa BERT namun dengan kelebihan waktu pelatihan yang lebih cepat. Penelitian yang dilakukan oleh Putri (2024) menunjukkan efisiensi dan akurasi model DistilBERT yang lebih unggul daripada BERT pada kasus analisis sentimen pemilihan presiden.

Penelitian ini bertujuan untuk membandingkan performa model BERT dan DistilBERT pada tugas analisis sentimen terkait kebijakan PPKM dari cuitan pengguna media sosial twitter atau X, serta mengevaluasi efek kuantisasi pada efisiensi model. Diharapkan hasil penelitian ini dapat menjadi acuan dalam memilih model NLP yang efisien dan tetap memiliki performa yang baik untuk analisis sentimen dalam konteks kebijakan publik.

## 1.1 Masalah Penelitian

Permasalahan utama yang ingin dijawab dalam penelitian ini adalah sebagai berikut:

1. Bagaimana pengimplementasian model BERT dan DistilBERT dalam melakukan analisis sentimen cuitan pengguna media sosial X?
2. Bagaimana kinerja model BERT dan DistilBERT dalam melakukan analisis sentimen cuitan pengguna media sosial X, berdasarkan akurasi, presisi, *recall*, dan skor F1?

## 1.2 Tujuan Penelitian

Tujuan utama dari penelitian ini adalah sebagai berikut:

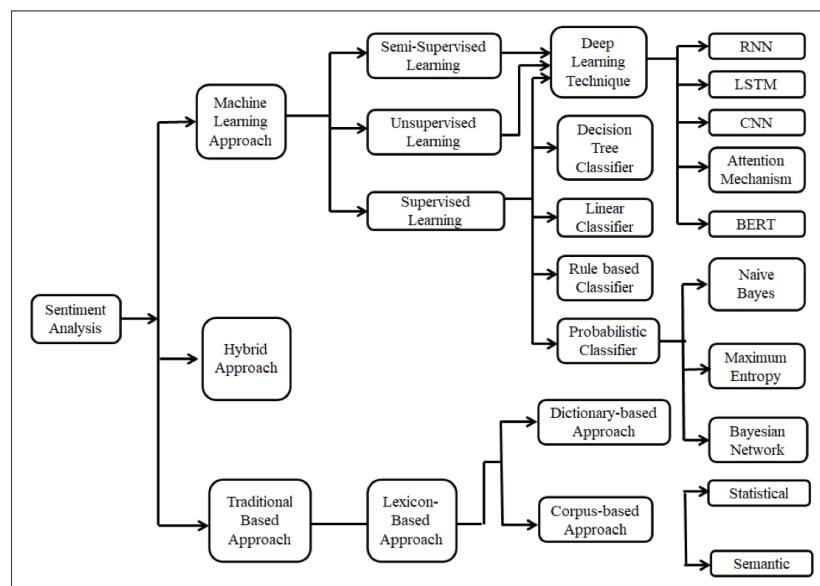
1. Menerapkan model BERT dan DistilBERT dalam melakukan analisis sentimen cuitan pengguna media sosial X.
2. Menganalisis kinerja model BERT dan DistilBERT dalam melakukan analisis sentimen cuitan pengguna media sosial X, berdasarkan akurasi, presisi, *recall*, dan skor F1.

## 2 Tinjauan Pustaka

### 2.1 Analisis Sentimen

Web 2.0 telah mendorong kemunculan blog, forum, dan jejaring sosial online yang memungkinkan pengguna untuk mendiskusikan berbagai topik dan membagikan opini mereka. Mereka, misalnya, dapat mengeluhkan produk yang telah mereka beli, memperdebatkan isu-isu terkini, atau menyampaikan pandangan politik mereka. Memanfaatkan informasi seperti ini tentang pengguna menjadi kunci bagi pengoperasian banyak aplikasi (seperti sistem rekomendasi), dalam analisis survei yang dilakukan oleh berbagai organisasi, maupun dalam perencanaan kampanye politik. Selain itu, analisis opini publik juga sangat penting bagi pemerintah karena dapat menjelaskan aktivitas dan perilaku manusia serta bagaimana opini orang lain mempengaruhi mereka. Dalam bidang sistem rekomendasi dan personalisasi, pengenalan sentimen pengguna sangat berguna untuk mengatasi kurangnya umpan balik eksplisit dari pengguna terhadap layanan yang diberikan.

Analisis Sentimen kemudian berkembang sebagai respons terhadap meningkatnya pertukaran informasi di Internet. Pada dasarnya, SA merupakan cabang dari (NLP) yang berfokus pada pemrosesan dan analisis opini manusia terhadap produk, layanan, atau peristiwa. Analisis sentimen dapat membantu dalam mengklasifikasikan ide dan opini menjadi kategori positif, negatif, atau netral, dan karena itu sering juga disebut sebagai opinion mining. Teknik ini banyak diterapkan pada ulasan produk, survei online, dan platform media sosial untuk menangani respons konsumen, sehingga perusahaan dapat mengevaluasi penerimaan atau penolakan terhadap produk mereka. Dengan memahami opini publik, perusahaan dapat mengoptimalkan strategi bisnis, sedangkan pemerintah dan organisasi lainnya dapat mengambil keputusan berdasarkan analisis perilaku sosial.



Gambar 1: Tiga pendekatan utama analisis sentimen

(Kumar et al., 2023)

Analisis sentimen biasanya menggunakan kombinasi pendekatan berbasis statistik, NLP, dan *machine learning*. Tiga pendekatan utama yang umum digunakan dalam analisis sentimen adalah, lihat Gambar 1: (1) pendekatan berbasis leksikon, yang menggunakan kamus kata-kata sentimen seperti *WordNet* dan *SentiWordNet*; (2) pendekatan berbasis *machine learning* tradisional, yang melibatkan algoritma seperti *Naïve Bayes*, *Support Vector Machines* (SVM), dan *Maximum Entropy*; serta (3) pendekatan hibrida, yang menggabungkan teknik leksikon dengan model *machine learning* untuk meningkatkan akurasi klasifikasi.

## 2.2 Machine Learning

Teknologi *machine learning* adalah bagian dari kecerdasan buatan (Black et al., 2023) yang memungkinkan komputer untuk mempelajari pola dalam data dan melakukan tugas kompleks seperti regresi dan klasifikasi secara otomatis (Sidey-Gibbons & Sidey-Gibbons, 2019). ML berbeda dari pendekatan statistik tradisional yang berfokus pada inferensi, ML lebih menekankan pada kemampuan untuk memberikan prediksi yang akurat terhadap data baru (Goodfellow et al., 2016).

### 2.2.1 Supervised Learning

*Supervised learning* adalah salah satu pendekatan *machine learning* di mana model dilatih dengan data berlabel, yaitu setiap *input* memiliki targetnya yang diketahui. Pendekatan ini mampu mempelajari hubungan antara *input* dan *output* untuk kemudian digunakan dalam memprediksi *output* baru yang belum diketahui (Black et al., 2023). Contoh penerapannya dalam bidang NLP adalah klasifikasi sentimen berdasarkan data ulasan atau cuitan media sosial (Setiawan, 2024).

### 2.2.2 Unsupervised Learning

*Unsupervised learning* adalah pendekatan lain *machine learning* di mana model dilatih menggunakan data yang tidak berlabel. Pendekatan ini mampu mengidentifikasi pola tersembunyi dalam data, seperti klaster alami, asosiasi antar variabel, atau dimensi representasi yang lebih sederhana (Goodfellow et al., 2016). Salah satu contoh pengaplikasian *unsupervised learning* dalam NLP adalah pendekripsi topik secara otomatis yang terdiri dari komentar pengguna yang secara semantik serupa. (Stanik et al., 2021).

## 2.3 Deep Learning

*Deep learning* adalah cabang dari *machine learning*, yang memanfaatkan jaringan neuron yang berlapis, mencontoh konsep jaringan saraf biologis pada manusia, untuk memproses data yang lebih kompleks. (Roberts et al., 2021). Salah satu keunggulan DL dibanding ML adalah, pada kasus data kompleks, dan besar, DL mampu menangkap pola-pola tersembunyi secara otomatis (Li et al., 2023). Jaringan neuron yang lebih berlapis mampu menangkap pola atau fitur yang lebih kompleks dan tersembunyi pada data. Kemampuan ini menjadikan DL lebih unggul, dibanding ML, untuk kasus-kasus seperti *computer vision*, pemrosesan bahasa alami (natural language processing), dan lain sebagainya (Hammad, 2024).

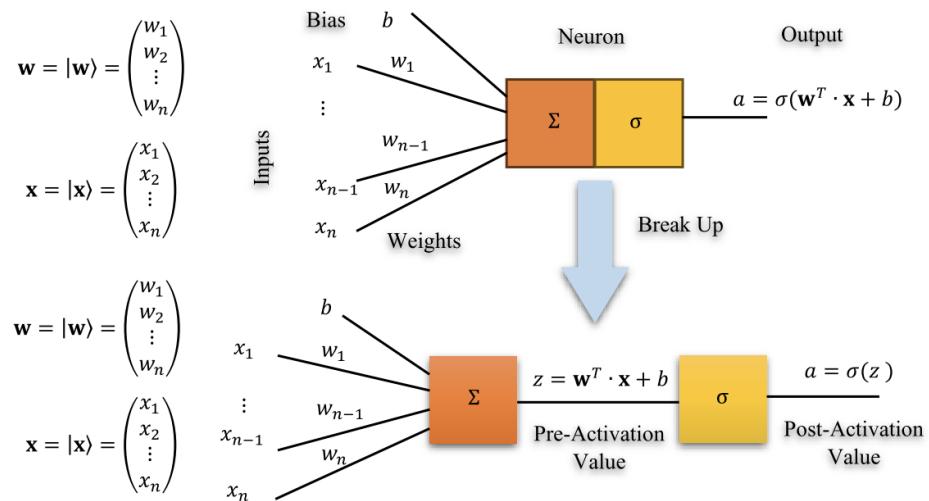
## 2.4 Metode Artificial Neural Network

Sistem pemodelan data nonlinear, yang strukturnya didasarkan pada jaringan saraf biologis manusia, di mana model dibentuk dalam hubungan yang kompleks antara *input* dan *output* disebut *artificial neural network* (ANN) (Qamar & Zardari, 2023; Li et al., 2023).

Setiap neuron, atau node, dalam ANN menerima satu atau lebih *input* (lihat Gambar 2), dan melakukan operasi matematika terhadap *input* tersebut, dan menghasilkan suatu *output*. setiap koneksi antar neuron mempunyai bobot dan bias yang menentukan kekuatan koneksi tersebut. Bobot dan bias dapat berubah selama proses pelatihan untuk meminimalkan galat antara *output* yang diprediksi, dan *output* yang sebenarnya (Hammad, 2024).

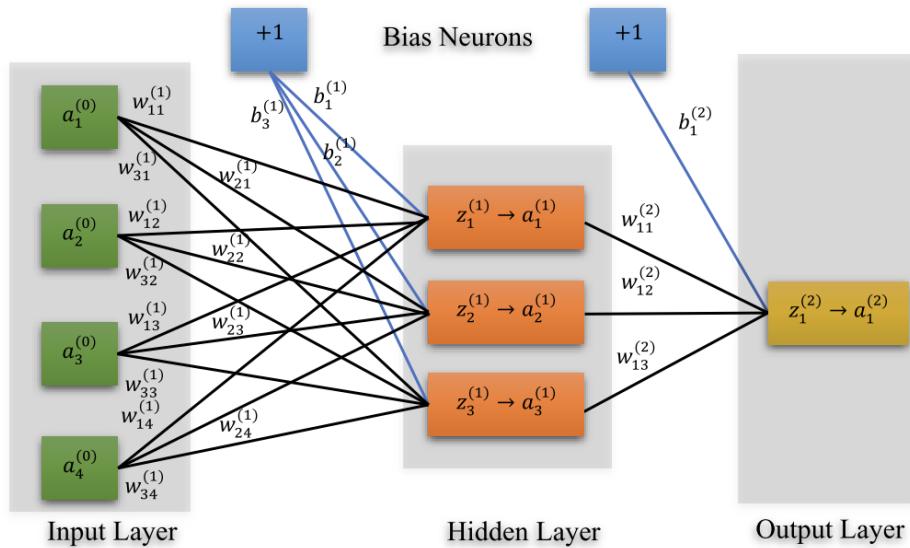
### 2.4.1 Arsitektur Dasar Artificial Neural Network

*feed forward neural network* (FFNN) merupakan arsitektur dasar ANN yang memiliki 3 lapisan utama (lihat Gambar 3), yaitu *input layer*, *hidden layer*, *output layer*. FFNN sering juga disebut sebagai *fully*

Gambar 2: Neuron pada *Artificial Neural Network*.

(Hammad, 2024)

*connected layer* (lapisan FC) atau *dense layer*

Gambar 3: Arsitektur *Artificial Neural Network*.

(Hammad, 2024)

### 1. *Input Layer*

Lapisan masukan atau *input layer* adalah lapisan awal pada arsitektur ANN yang menerima data *input* berupa vektor (lihat Gambar 2)  $\mathbf{x} \in \mathbb{R}^n$ ,  $\mathbf{x} = (x_1, x_2, \dots, x_n)^T$ . Setiap neuron pada lapisan ini merepresentasikan sebuah fitur dari data *input*. Lapisan ini tidak memiliki bobot atau bias (Hammad, 2024).

### 2. *Hidden Layer*

Lapisan tersembunyi atau *hidden layer* adalah lapisan yang berada di antara *input* dan *output*. Lapisan ini berperan untuk melakukan transformasi non-linear pada data dan jumlah lapisan ini bisa lebih dari 1. setiap neuron pada lapisan ini menghitung nilai pra-aktivasi ( $z$ ), yaitu jumlah *input* ( $x_i$ ) dikali bobot ( $w_i$ ) ditambah bias ( $b$ ) yang kemudian diterapkan fungsi aktivasi ( $\sigma$ ).

$$z = \sum_{i=1}^n w_i x_i + b = \mathbf{w}^T \cdot \mathbf{x} + b = \langle \mathbf{w} | \mathbf{x} \rangle + b, \quad (1)$$

$$a = \sigma(z). \quad (2)$$

Ada berbagai macam fungsi aktivasi yang dapat diterapkan pada ANN tergantung kasusnya, salah satu yang paling umum adalah fungsi sigmoid ( $\sigma(z) = \frac{1}{1+e^{-z}}$ ). Fungsi aktivasi berguna untuk menangkap hubungan kompleks fitur-fitur pada data (Hammad, 2024).

### 3. Output Layer

Lapisan output atau *output layer* adalah lapisan paling akhir dari ANN. Lapisan ini menghasilkan output akhir dari komputasi lapisan-lapisan sebelumnya. Banyak neuron yang dihasilkan tergantung tugas yang dilakukan (Hammad, 2024).

#### 2.4.2 Fungsi Aktivasi

Fungsi aktivasi berperan dalam ANN karena kemampuannya untuk menambahkan transformasi non-linear ke dalam model yang berguna untuk menangkap hubungan kompleks dalam data. Berikut beberapa macam fungsi aktivasi ANN yang akan digunakan dalam penelitian ini, berdasarkan definisi yang dikemukakan oleh Hammad (2024).

- Fungsi Sigmoid

Fungsi aktivasi Sigmoid merupakan fungsi non-linear yang sangat populer dan tradisional. Secara umum, fungsi sigmoid berbentuk kurva halus menyerupai huruf “S”. *output* dari fungsi ini berada dalam rentang [0,1]

$$\sigma_{\text{Sigmoid}}(x) = \frac{1}{1+e^{-x}} \quad (3)$$

Fungsi ini sering digunakan pada kasus klasifikasi biner. Salah satu keunggulan fungsi ini yaitu, dapat diturunkan untuk semua nilai  $x$ , yang berguna pada kasus pengaplikasian algoritma optimisasi berbasis gradien, yang membutuhkan derivatif fungsi aktivasi.

- Fungsi *Rectified Linear Unit* (ReLU)

Fungsi ReLU merupakan gabungan dari dua fungsi linear yang berbeda, yang menghasilkan *output*  $x$  untuk  $x$  positif, dan *output* 0 untuk  $x$  negatif. Rentang dari fungsi ini adalah  $[0, \infty)$ .

$$\sigma_{\text{ReLU}}(x) = \max(z, 0) \quad (4)$$

Salah satu keunggulan fungsi ini yaitu, mengalami konvergensi jauh lebih cepat dibandingkan dengan fungsi Sigmoid.

- Fungsi GELU (Gaussian Error Linear Unit)

Fungsi GELU merupakan fungsi aktivasi yang digunakan secara default pada model BERT. Tidak seperti ReLU yang hanya memotong nilai negatif menjadi nol, GELU mempertimbangkan distribusi probabilitas dari *input*. Secara umum, fungsi ini menggabungkan sifat linear dan non-linear dengan cara yang halus dan kontinu.

Bentuk pendekatan dari fungsi GELU diberikan oleh persamaan berikut:

$$\text{GELU}(x) \approx 0.5x \left( 1 + \tanh \left[ \sqrt{\frac{2}{\pi}} (x + 0.044715x^3) \right] \right) \quad (5)$$

Fungsi ini memberikan performa yang lebih baik dibandingkan ReLU atau Sigmoid pada model besar seperti BERT karena dapat menangani distribusi nilai *input* secara lebih alami (Devlin et al., 2018).

- Fungsi Softmax

Fungsi Softmax digunakan dalam mekanisme self-attention pada arsitektur Transformer, termasuk BERT. Fungsi ini mengubah skor atensi menjadi distribusi probabilitas yang dapat dijumlahkan menjadi satu. Diberikan *input* vektor  $z = (z_1, z_2, \dots, z_n)$ , *output* fungsi softmax didefinisikan sebagai:

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^n e^{z_j}} \quad (6)$$

Dalam konteks BERT, fungsi ini digunakan untuk menghitung bobot perhatian (attention weights) antara token-token dalam sebuah urutan teks, yang memungkinkan model memahami konteks global dari kata dalam kalimat (Vaswani et al., 2017).

#### 2.4.3 Forward Propagation

Istilah "forward" menunjukkan aliran *input* data melalui jaringan model, dari *input layer* hingga *output layer*.

1. Proses dimulai dari *input layer* yang menerima data *input* mentah.
2. Dihitung Jumlah *input* dikali bobot ditambah bias pada setiap neuron di *hidden layer* pertama.
3. Nilai ini kemudian disematkan ke fungsi aktivasi (AF), pada setiap neuron.
4. Proses ini diulang pada setiap *hidden layer* di ANN, yaitu nilai *output* dari lapisan sebelumnya digunakan sebagai nilai *input* untuk lapisan berikutnya yang kemudian disematkan ke fungsi aktivasi.
5. *output* dari *hidden layer* terakhir disematkan ke *output layer* untuk menghasilkan *output* akhir dari ANN. Jenis fungsi aktivasi yang digunakan pada *output layer* bergantung pada kasus yang dihadapi.
6. Dilakukan perhitungan galat, yaitu perbedaan nilai *output* prediksi dengan nilai target sebenarnya, menggunakan fungsi kerugian-yang akan dibahas pada subbab selanjutnya.

Selama pelatihan, bobot dan bias akan terus diperbarui menggunakan algoritma optimasi-yang akan dibahas pada subbab selanjutnya-untuk meminimalkan galat, memungkinkan ANN untuk belajar dari data (Hammad, 2024).

#### 2.4.4 Loss Function

Selama proses pelatihan ANN, diperlukan suatu fungsi yang mengukur seberapa baik performa model. Pengukuran tersebut diperoleh melalui perhitungan fungsi kerugian atau *loss function*, yang menghitung perbedaan nilai prediksi model dengan nilai sebenarnya (*ground truth*). Seperti halnya fungsi aktivasi, fungsi loss memiliki berbagai macam bentuk, yang kegunaannya bergantung pada kasus yang dihadapi. Pada penelitian ini, kasus yang dihadapi adalah kasus klasifikasi biner, sehingga, *loss function* yang dipakai adalah *Binary-Cross Entropy* (BCE), yang sering juga disebut *logistic loss function* (Hammad, 2024).

*Binary-Cross Entropy* (BCE) umum digunakan untuk permasalahan klasifikasi biner, yaitu ketika hanya ada dua kelas yang ingin diprediksi, seperti 0 atau 1, negatif atau positif, ya atau tidak. Persamaan BCE adalah sebagai berikut (Hammad, 2024).

$$\mathcal{L}_{\text{BCE}} = -\frac{1}{n} \sum_{j=1}^n [y_j \log(a_j) + (1 - y_j) \log(1 - a_j)], \quad (7)$$

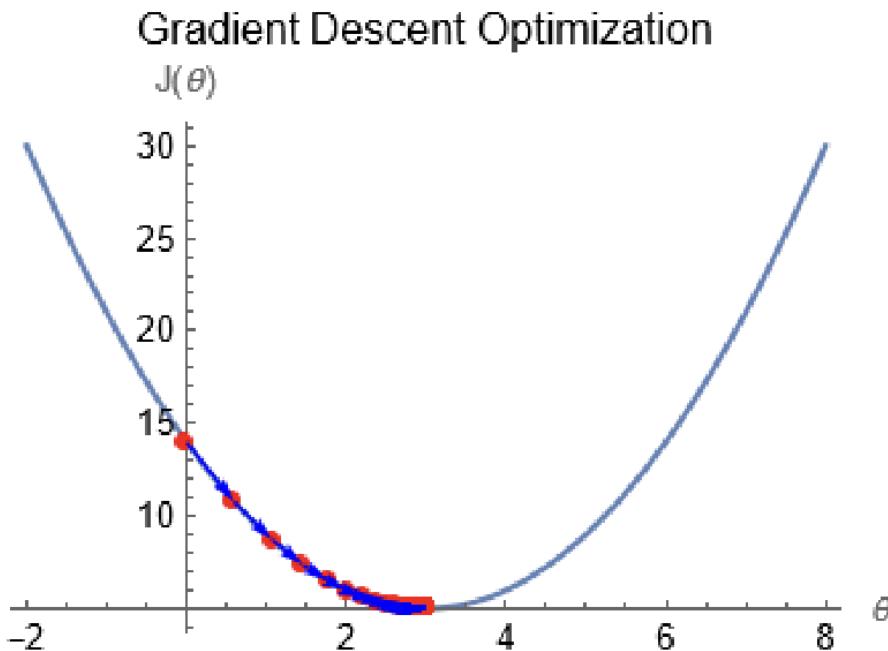
di mana:

- $n$  = jumlah entri (data) dalam dataset,

- $y_j$  = label (target) sebenarnya untuk entri ke- $j$  (bernilai **0** atau **1**),
- $a_j$  = probabilitas prediksi bahwa entri ke- $j$  termasuk dalam kelas **1**.

#### 2.4.5 Gradient Descent

*Gradient Descent* (GD) adalah algoritma optimisasi orde pertama yang digunakan untuk mengoptimalkan parameter model DL dengan meminimalkan *loss function* (lihat Gambar 4).



Gambar 4: Ilustrasi optimasi *gradient descent* yang diaplikasikan terhadap *quadratic loss function*,  $J(\theta) = (\theta - 3)^2 + 5$ .

(Hammad, 2024)

Objektif dari GD adalah mencari nilai  $\boldsymbol{\theta}$  untuk meminimalkan  $J(\boldsymbol{\theta})$ , dengan  $\boldsymbol{\theta}$  adalah nilai parameter model, dan  $J(\boldsymbol{\theta})$  adalah *loss function*. Berikut cara kerja GD (Hammad, 2024).

1. inisialisasi nilai awal  $\boldsymbol{\theta}$  dari fungsi  $J(\boldsymbol{\theta})$  yang akan dipotimasi.
2. Menghitung gradien fungsi  $J(\boldsymbol{\theta})$  terhadap  $\boldsymbol{\theta}$ , yang dinotasikan sebagai  $\nabla J(\boldsymbol{\theta})$ , yang berbentuk vektor turunan parsial  $J(\boldsymbol{\theta})$  terhadap setiap parameternya  $\theta_i$ .

$$\nabla J(\boldsymbol{\theta}) = \left( \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_1}, \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_2}, \dots, \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_n} \right). \quad (8)$$

3. memperbarui nilai parameter  $\theta_i$

$$\theta_i = \theta_i - \alpha \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_i}. \quad (9)$$

Dengan  $\alpha$  adalah *learning rate*. Langkah ini berguna untuk meminimalkan fungsi  $J(\boldsymbol{\theta})$ .

4. Langkah 2 dan 3 diulangi secara iteratif hingga mencapai iterasi maksimum, atau hingga konvergen, yaitu tidak ada perubahan signifikan pada nilai  $J(\boldsymbol{\theta})$ .

Ada berbagai macam varian algoritma GD yang lebih tinggi, salah satunya adalah optimasi *adaptive moment estimation* (Adam), yang seterusnya akan digunakan dalam penelitian ini.

## 2.4.6 Optimasi *Adaptive Moment Estimation* (Adam)

---

### **Algorithm 5.8:** Adam

$\mathbf{g}_t^2$  indicates the elementwise square  $\mathbf{g}_t \odot \mathbf{g}_t$ . Good default settings for the tested machine learning problems are  $\alpha = 0.001$ ,  $\beta_1 = 0.9$ ,  $\beta_2 = 0.999$  and  $\epsilon = 10^{-8}$ . All operations on vectors are element-wise. With  $\beta_1^t$  and  $\beta_2^t$  means  $\beta_1$  and  $\beta_2$  to the power  $t$ .

Require:  $\alpha$ : Stepsize.  $\beta_1, \beta_2 \in [0, 1]$ : Exponential decay rates for the moment estimates.  $\mathcal{L}$ : Stochastic objective function with parameters  $\mathbf{w}$ .

Initialization:  $\mathbf{w}_0$ : Initial parameter vector.  $\mathbf{m}_0 \leftarrow \mathbf{0}$  (Initialize 1st moment vector).  $\mathbf{v}_0 \leftarrow \mathbf{0}$  (Initialize 2nd moment vector).  $t \leftarrow 0$  (Initialize timestep).

For each iteration of the optimization process:

1. Sample a minibatch of  $m$  examples from the training set  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  with corresponding targets  $\mathbf{y}^{(i)}$ .
2. Compute the gradient of the loss function with respect to the parameters:

$$\mathbf{g}_t \leftarrow \nabla_{\mathbf{w}} \left( \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}; \mathbf{w}_{t-1}) \right).$$

3. Update biased first moment estimate:

$$\mathbf{m}_t \leftarrow \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \mathbf{g}_t.$$

4. Update the biased second raw moment estimate:

$$\mathbf{v}_t \leftarrow \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2) \mathbf{g}_t^2.$$

5. Compute bias-corrected first moment estimate:

$$\hat{\mathbf{m}}_t \leftarrow \frac{\mathbf{m}_t}{1 - \beta_1^t}.$$

6. Compute bias-corrected second moment estimate:

$$\hat{\mathbf{v}}_t \leftarrow \frac{\mathbf{v}_t}{1 - \beta_2^t}.$$

7. Compute the update for each parameter:

$$\Delta \mathbf{w}_t \leftarrow - \frac{\alpha}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon} \hat{\mathbf{m}}_t.$$

8. Update the parameters:

$$\mathbf{w}_t \leftarrow \mathbf{w}_{t-1} + \Delta \mathbf{w}_t.$$

9. Repeat the iterative update process for a predetermined number of iterations or until convergence criteria are met.
- 

Gambar 5: Ilustrasi algoritma Adam (Hammad, 2024)

*Adaptive moment estimation* (Adam) adalah algoritma optimasi yang pertama dikenalkan oleh Kingma & Ba (2017). Adam populer digunakan dalam DL karena konvergensi yang cepat dan mudah digunakan. Adam bekerja dengan mempertahankan dua rata-rata bergerak, yaitu rata-rata eksponensial menurun dari gradien masa lalu (momen pertama) dan rata-rata eksponensial menurun dari kuadrat gradien masa lalu (momen kedua). Kedua rata-rata ini diinisialisasi sebagai vektor nol. Pada setiap iterasi, Adam menghitung gradien terkini dari *loss function* terhadap parameter. Selanjutnya, Adam memperbarui rata-rata bergerak dari gradien dan kuadrat gradien tersebut (Hammad, 2024). Gambar 5 menunjukkan algoritma optimasi Adam.

Tergantung pada kasus yang dilakukan, Adam mempunyai beberapa *hyperparameter* yang dapat disesuaikan, yaitu *learning rate*  $\alpha$ , laju *exponential decay* untuk momen pertama  $\beta_1$ , laju *exponential decay* untuk momen kedua  $\beta_2$ , dan konstanta kecil  $\epsilon$  untuk mencegah pembagian dengan nol.

## 2.4.7 Back Propagation

Setelah melalui proses *forward propagation*, seperti yang dijelaskan pada subbab 2.4.3, bobot dan bias pada ANN akan diperbarui sehingga mengurangi *loss*, fase ini disebut sebagai *back propagation*. Karena

yang akan diperbarui adalah bobot, yang merupakan parameter model, maka aturan umum untuk memperbarui bobot  $w_{ij}$  diberikan oleh.

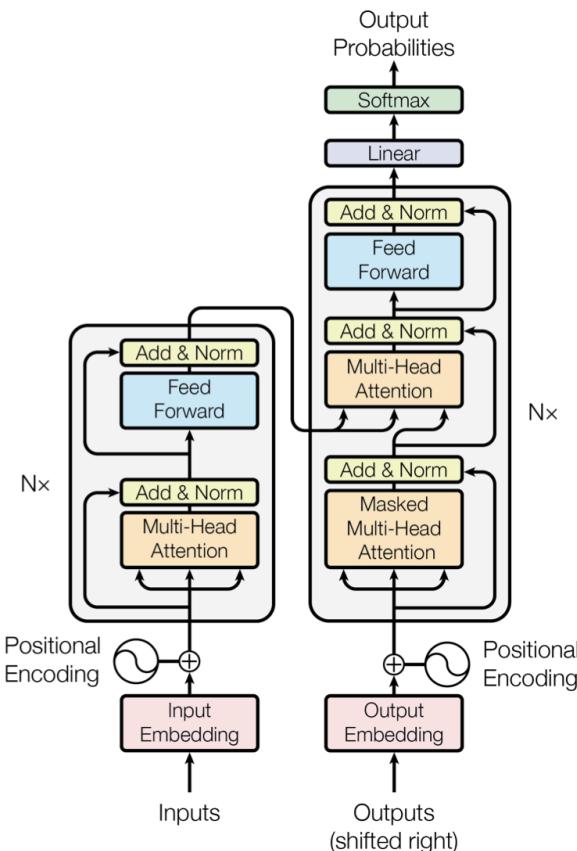
$$w_{ij}^{\text{baru}} = w_{ij}^{\text{lama}} - \alpha \frac{\partial \mathcal{L}}{\partial w_{ij}}, \quad \Delta w_{ij} = -\alpha \frac{\partial \mathcal{L}}{\partial w_{ij}} \quad (10)$$

Seperti pada persamaan 9, tetapi dengan perubahan notasi  $\partial J(\boldsymbol{\theta}) = \partial \mathcal{L}$ , dan  $\theta_{ij} = w_{ij}$ . Tujuan dari *back propagation* adalah untuk menghitung turunan parsial  $\partial \mathcal{L}/\partial w$  dan  $\partial \mathcal{L}/\partial b$  dari fungsi kerugian  $\mathcal{L}$  terhadap bobot  $w$  atau bias  $b$  mana pun dalam jaringan ANN. Ini adalah esensi dari *supervised learning* dalam pelatihan ANN.

## 2.5 Transformer

Transformer merupakan arsitektur DL yang dikembangkan oleh Vaswani et al. (2017) dan dimanfaatkan dalam arsitektur model BERT. Transformer secara revolusioner meningkatkan performa model pemrosesan bahasa alami (NLP), khususnya dalam tugas-tugas seperti *machine translation*, *text summarization*, dan analisis sentimen. Berbeda dengan pendekatan sebelumnya seperti RNN (*Recurrent Neural Networks*) atau LSTM (*Long Short-Term Memory*), Transformer tidak bergantung pada urutan data secara eksplisit, sehingga mampu memproses seluruh urutan *input* secara paralel. Hal ini membuat pelatihan model menjadi jauh lebih cepat dan efisien (Setiawan, 2024).

Komponen utama dari Transformer adalah mekanisme Attention, khususnya Self-Attention. Mekanisme ini memungkinkan model untuk fokus pada bagian-bagian penting dari *input* saat menghasilkan *output*, tanpa kehilangan konteks. Misalnya, dalam kalimat panjang, Transformer mampu memahami hubungan antara kata pertama dan terakhir tanpa harus membaca kata per kata secara berurutan.



Gambar 6: Arsitektur *Transformer*  
(Vaswani et al., 2017)

Arsitektur Transformer terdiri dari dua komponen utama: *encoder* dan *decoder*. *Encoder* yang bertugas untuk memproses *input* dan membentuk representasi kontekstual dari setiap token (kata), berisi

$N$  tumpukan (pada umumnya  $N = 6$ ), yang pada setiap tumpukan mengandung dua sub-lapisan, yaitu *Multi-Head Self-Attention Layer*, dan *Position-wise Feed-Forward Neural Network* yang masing-masing sub-lapisan dikelilingi dengan koneksi residual dan lapisan normalisasi. Sedangkan *Decoder* yang, menggunakan representasi dari *Encoder* untuk menghasilkan *output* secara bertahap seperti kalimat hasil terjemahan, (pada umumnya  $N = 6$ ), yang pada setiap tumpukan mengandung dua sub-lapisan, yaitu *Masked Multi-Head Self-Attention Layer*, *Encoder-Decoder Attention*, dan (*Position-wise Feed-Forward Neural Network* yang masing-masing sub-lapisan dikelilingi dengan koneksi residual dan lapisan normalisasi.

Arsitektur Transformer terdiri dari dua komponen utama: *encoder* dan *decoder*. *Encoder* yang bertugas untuk memproses *input* dan membentuk representasi kontekstual dari setiap token (kata), berisi  $N$  tumpukan (pada umumnya  $N = 6$ ), yang pada setiap tumpukan mengandung dua sub-lapisan, yaitu *Multi-Head Self-Attention Layer* dan *Position-wise Feed-Forward Neural Network*, yang masing-masing sub-lapisan dikelilingi dengan koneksi residual dan lapisan normalisasi. Sedangkan *Decoder*, yang menggunakan representasi dari *Encoder* untuk menghasilkan *output* secara bertahap seperti kalimat hasil terjemahan, juga terdiri dari  $N$  tumpukan (pada umumnya  $N = 6$ ), yang pada setiap tumpukan mengandung tiga sub-lapisan, yaitu *Masked Multi-Head Self-Attention Layer*, *Encoder-Decoder Attention*, dan *Position-wise Feed-Forward Neural Network*, yang masing-masing sub-lapisan dikelilingi dengan koneksi residual dan lapisan normalisasi.

- **Multi-Head Self-Attention Layer** memungkinkan setiap token dalam sekuens masukan untuk memperhatikan seluruh token lain, membangun representasi kontekstual yang kaya. Mekanisme *attention* yang digunakan adalah *Scaled Dot-Product Attention*, yang dirumuskan sebagai:

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

di mana  $Q$  adalah matriks query,  $K$  adalah matriks key,  $V$  adalah matriks value, dan  $d_k$  adalah dimensi dari key. Operasi ini menghasilkan distribusi perhatian, yang menentukan kontribusi setiap token terhadap representasi token lainnya.

- **Multi-Head Attention** memperluas *single attention mechanism* menjadi beberapa *head* yang berjalan secara paralel:

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

dengan

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

di mana  $W_i^Q$ ,  $W_i^K$ , dan  $W_i^V$  adalah matriks bobot untuk *query*, *key*, dan *value* pada head ke- $i$ , dan  $W^O$  adalah matriks bobot *output*.

- **Masked Multi-Head Self-Attention Layer** pada *Decoder* serupa dengan *self-attention*, namun menggunakan masking untuk mencegah posisi tertentu memperhatikan posisi setelahnya, memastikan sifat auto-regressive pada generasi sequence.
- **Encoder-Decoder Attention** pada *Decoder* menghubungkan informasi representasi input yang diproses oleh *encoder* ke dalam proses decoding. Query berasal dari decoder, sedangkan *key* dan *value* berasal dari output encoder.
- **Position-wise Feed-Forward Neural Network** diterapkan secara independen di setiap posisi token dan dirumuskan sebagai:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

di mana  $W_1$  dan  $W_2$  adalah matriks bobot, dan  $b_1$ ,  $b_2$  adalah bias pada masing-masing layer.

- **Residual Connection dan Layer Normalization** digunakan di setiap sub-lapisan untuk membantu

mempertahankan informasi awal dan mempercepat konvergensi pelatihan:

$$\text{Output} = \text{LayerNorm}(x + \text{Sublayer}(x))$$

di mana  $x$  adalah input ke sub-lapisan dan  $\text{Sublayer}(x)$  adalah output dari sub-lapisan tersebut.

- **Positional Encoding** ditambahkan ke *embedding input* untuk menyisipkan informasi posisi ke dalam *token embedding*, karena *Transformer* sendiri tidak memiliki pengurutan eksplisit. *Positional Encoding* dirumuskan sebagai:

$$\text{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

$$\text{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

di mana  $pos$  adalah posisi token dalam sequence dan  $i$  adalah indeks dimensi *embedding*.

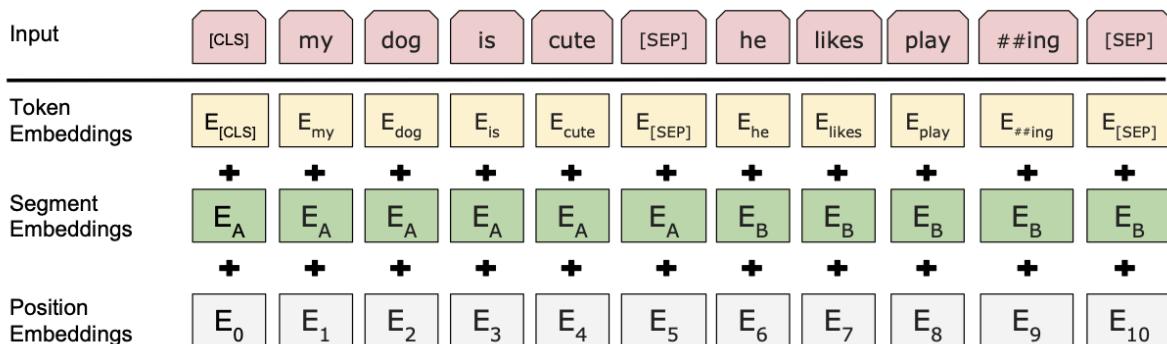
## 2.6 Bidirectional Encoder Representations from Transformers

*Model Bidirectional Encoder Representations from Transformers* (BERT) adalah model transformasi yang dirancang oleh peneliti Google untuk memahami konteks bidirectional dari teks, artinya model ini membaca teks secara menyeluruh dari kiri ke kanan dan dari kanan ke kiri Devlin et al. (2018).

Arsitektur model BERT hanya menggunakan bagian *encoder* dari Transformer, tanpa komponen *decoder*. BERT terdiri dari tumpukan *encoder* yang masing-masing berisi dua komponen utama, yaitu *multi-head self-attention mechanism* dan *position-wise feed-forward networks*. Untuk menjaga stabilitas training dan mempercepat konvergensi, setiap sub-lapisan dilengkapi dengan *residual connection* dan *layer normalization*.

Pada BERT, setiap token *input* dikombinasikan dengan tiga jenis embedding, yaitu token embeddings, segment embeddings, dan positional embeddings, lihat Gambar 7. Kombinasi ini dirumuskan sebagai:

$$\text{Input Embedding} = \text{Token Embedding} + \text{Segment Embedding} + \text{Positional Embedding}$$



Gambar 7: Representasi *input* BERT

BERT dilatih menggunakan dua tugas *pre-training* utama, yaitu:

### 1. Masked Language Modeling (MLM)

Dalam MLM, sebagian token input di-mask dan model diminta untuk memprediksi token yang hilang. Ini mendorong model untuk memahami hubungan antar kata dalam konteks dua arah. Formulasi matematis untuk prediksi token yang di-mask adalah:

$$\hat{x}_i = \arg \max_{w \in V} P(w | x_{/i})$$

dengan  $V$  adalah seluruh kosakata dan  $x_{/i}$  merupakan semua token kecuali token ke- $i$ .

## 2. Next Sentence Prediction (NSP)

Dalam NSP, model diberikan sepasang kalimat dan ditugaskan untuk memprediksi apakah kalimat kedua mengikuti kalimat pertama dalam dokumen asli. Fungsi kerugian NSP dirumuskan sebagai:

$$\mathcal{L}_{\text{NSP}} = -(y \log(p) + (1-y) \log(1-p))$$

di mana  $y$  adalah label (1 untuk kalimat berurutan, 0 jika tidak) dan  $p$  adalah probabilitas prediksi.

BERT melakukan *encoding* terhadap sekuens masukan secara simultan dalam dua arah. Ini berbeda dengan pendekatan sebelumnya seperti GPT yang hanya melakukan pemrosesan sekuensial satu arah (kiri ke kanan). Pendekatan bidirectional ini memungkinkan BERT untuk membangun representasi kontekstual yang lebih dalam dan akurat, baik untuk kata-kata yang ambigu maupun untuk pemahaman kalimat kompleks.

Arsitektur standar BERT-Base memiliki 12 lapisan *encoder*, 12 *attention heads*, dengan dimensi *hidden layer* sebesar 768, dan total parameter sekitar 110 juta. Sementara itu, varian BERT-Large meningkatkan jumlah layer menjadi 24, dengan 16 attention heads dan dimensi hidden layer sebesar 1024.

## 2.7 Linear Scheduler with Warmup

Salah satu pilihan paling penting dalam optimasi berbasis gradien adalah *learning rate* (ukuran langkah)  $\eta$ . Jika  $\eta$  terlalu kecil, pembelajaran mungkin berjalan terlalu lambat atau model mungkin terjebak dalam area yang tidak menguntungkan pada lanskap *loss*. Jika  $\eta$  terlalu besar, pelatihan biasanya akan mengalami divergensi. Dalam praktiknya, umum untuk memilih jadwal *learning rate* dinamis  $\eta_t$ . Jadwal *learning rate* modern untuk pembelajaran mendalam biasanya terdiri dari periode *warmup* di mana  $\eta_t$  dinaikkan secara linear dari nol ke nilai target  $\eta_{\text{trgt}}$  selama waktu *warmup*  $T_{\text{wrn}}$ . Setelah periode *warmup*, biasanya *learning rate* akan mengalami penurunan [9].

**Linear Warmup:** Ini didefinisikan oleh jadwal

$$\eta_t = \eta_{\text{init}} + (\eta_{\text{trgt}} - \eta_{\text{init}}) \left( \frac{t}{T_{\text{wrn}}} \right).$$

Laju *warmup* adalah

$$\alpha := \frac{\eta_{\text{trgt}} - \eta_{\text{init}}}{T_{\text{wrn}}}.$$

$T_{\text{wrn}} = 1$  berhubungan dengan *constant learning rate*. Kecuali jika ditentukan lain, ditetapkan  $\eta_{\text{init}} = 0$  saat merujuk ke *linear warmup*.

## 2.8 Metrik Evaluasi

Metrik evaluasi diperlukan untuk menilai apakah model EfficientNet yang dibangun memiliki performa yang baik dalam mengklasifikasikan gangguan spektrum autisme (ASD) pada pasien. Dalam tugas pengklasifikasian ASD ini, metrik evaluasi yang digunakan adalah akurasi, presisi, recall, dan F1-score.

### 2.8.1 Confusion Matrix

*Confusion matrix* adalah matriks 2x2 (atau lebih besar) yang digunakan sebagai metrik pengukuran performa untuk masalah klasifikasi. Terdapat empat istilah utama dalam *confusion matrix* yang merepresentasikan hasil proses klasifikasi, yaitu *true positive* (TP), *true negative* (TN), *false positive* (FP), dan *false negative* (FN). Matriks ini menjadi dasar untuk menghitung berbagai metrik evaluasi lainnya.

- **True Positive (TP):** Kasus di mana model memprediksi kelas positif dan prediksinya benar. Misalnya, model memprediksi seseorang mengidap autisme, dan kenyataannya orang tersebut memang mengidap autisme.
- **True Negative (TN):** Kasus di mana model memprediksi kelas negatif dan prediksinya benar. Misalnya, model memprediksi seseorang tidak mengidap autisme, dan orang tersebut memang tidak mengidap autisme.

Class designation		Actual class	
		True (1)	False (0)
Predicted class	Positive (1)	TP	FP
	Negative (0)	FN	TN

Gambar 8: Ilustrasi *Confusion Matrix*.  
(?)

- **False Positive (FP):** Kasus di mana model memprediksi kelas positif, padahal nilai aslinya negatif. Ini disebut juga *Type I Error*, contohnya model menyatakan seseorang mengidap autisme padahal tidak.
- **False Negative (FN):** Kasus di mana model memprediksi kelas negatif, padahal nilai aslinya positif. Ini dikenal juga sebagai *Type II Error*, contohnya model gagal mendeteksi individu yang sebenarnya mengidap autisme.

Berdasarkan keempat komponen dalam *confusion matrix*, metrik evaluasi dapat dihitung sebagai berikut:

#### 1. Akurasi

$$\text{Akurasi} = \frac{TP + TN}{TP + TN + FP + FN} \quad (11)$$

Metrik ini menunjukkan proporsi prediksi yang benar terhadap seluruh prediksi yang dilakukan.

#### 2. Presisi (Precision)

$$\text{Presisi} = \frac{TP}{TP + FP} \quad (12)$$

Metrik ini mengukur berapa banyak prediksi positif yang benar-benar positif. Presisi penting untuk menghindari banyaknya prediksi positif yang salah.

#### 3. Recall (Sensitivitas)

$$\text{Recall} = \frac{TP}{TP + FN} \quad (13)$$

Metrik ini mengukur kemampuan model dalam menemukan semua kasus positif yang ada.

#### 4. F1-Score

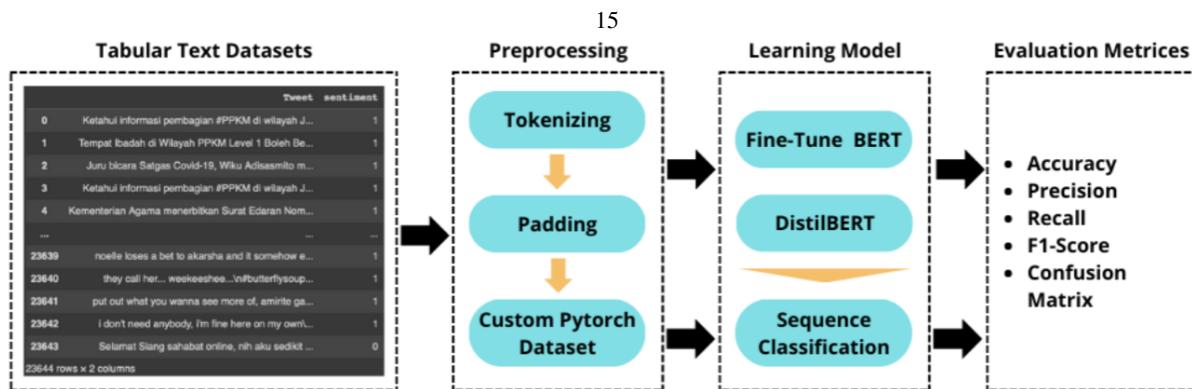
$$\text{F1-Score} = 2 \times \frac{\text{Presisi} \times \text{Recall}}{\text{Presisi} + \text{Recall}} \quad (14)$$

F1-Score adalah harmonic mean dari presisi dan recall, dan berguna ketika terdapat ketidakseimbangan antara kelas positif dan negatif.

### 3 Data dan Metode

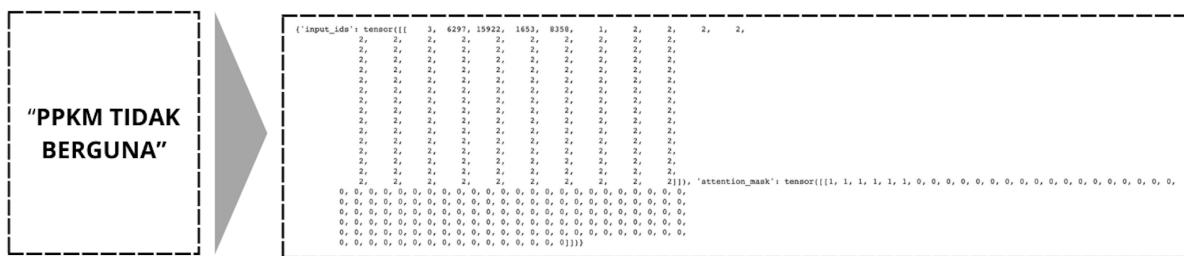
Dataset kumpulan cuitan di media sosial X, mengenai opini pembatasan PPKM yang diterapkan di Indonesia pada masa pandemi Covid-19. Dataset ini berasal dari situs kaggle yang mencakup 23644 baris dengan 4 kolom. Fitur yang digunakan untuk variabel target adalah “sentimen” karena kita ingin memprediksi sentimen cuitan-cuitan di media sosial X mengenai PPKM. Fitur yang digunakan untuk variabel prediktor adalah “Tweet”.

Pada tahap awal eksperimen, langkah preprocessing dilakukan dengan nilai yang hilang guna memastikan semua data memiliki nilai atau *input* sehingga bisa dilakukan pelatihan model, selanjutnya dilakukan tokenisasi data teks, yaitu memecah teks menjadi komponen yang lebih kecil (token) dan dikonversikan menjadi token id yang merepresentasikan kata atau sub-kata tersebut. Selanjutnya dilakukan padding untuk menyamakan panjang semua kalimat agar memiliki panjang yang sama dengan diubah



Gambar 9: Alur eksperimen

menyesuaikan panjang yang telah ditentukan. Gambar 5 menunjukkan hasil preprocessing data teks dari *input* awal hingga penerapan padding.



Gambar 10: Proses Preprocessing: (kiri) *input* teks, (kanan) hasil tokenisasi dan padding

Dalam penelitian ini, eksperimen dilakukan dengan 2 kasus, yaitu kasus pertama dengan model BERT menggunakan 1000 data dengan 80% data pelatihan dengan 10% data pengujian dan 10% data validasi; kasus kedua dengan model BERT dan DistilBERT menggunakan 5000 data dengan 80% data pelatihan dengan 10% data pengujian dan 10% data validasi.

Pembelajaran dengan model BERT dan DistilBERT disiapkan seperti pada Gambar 11 dan Gambar 12. model dilatih untuk total 20 epoch pada kedua kasus. Diterapkan *optimizer* Adam dengan *learning rate*  $2e^{-5}$ , diterapkan juga *learning rate linear schedule with warmup* dan *loss function cross entropy loss*, metode ini dipilih karena akan dilakukan klasifikasi multi-kelas. *Batch size* yaitu jumlah sampel yang diproses sebelum model diperbarui dalam pelatihan jaringan neural, diatur menjadi 32.

```

SentimentClassifier(
    (bert): BertForSequenceClassification(
        (bertModel): BertModel(
            (embeddings): BertEmbeddings(
                (word_embeddings): Embedding(32000, 768, padding_idx=0)
                (position_embeddings): Embedding(512, 768)
                (token_type_embeddings): Embedding(2, 768)
                (layerNorm): LayerNorm(768,), eps=1e-12, elementwise_affine=True
            )
        )
        (encoder): BertEncoder(
            (layer): ModuleList(
                (0-11): 12 x BertLayer(
                    (attention): BertAttention(
                        (self): BertSdpSelfAttention(
                            (query): Linear(in_features=768, out_features=768, bias=True)
                            (key): Linear(in_features=768, out_features=768, bias=True)
                            (value): Linear(in_features=768, out_features=768, bias=True)
                            (dropout): Dropout(p=0.1, inplace=False)
                        )
                    )
                    (output): BertSelfOutput(
                        (dense): Linear(in_features=768, out_features=768, bias=True)
                        (LayerNorm): LayerNorm(768,), eps=1e-12, elementwise_affine=True
                        (dropout): Dropout(p=0.1, inplace=False)
                    )
                )
                (intermediate): BertIntermediate(
                    (dense): Linear(in_features=768, out_features=3072, bias=True)
                    (intermediate_act_fn): GELUActivation()
                )
                (output): BertOutput(
                    (dense): Linear(in_features=3072, out_features=768, bias=True)
                    (LayerNorm): LayerNorm(768,), eps=1e-12, elementwise_affine=True
                    (dropout): Dropout(p=0.1, inplace=False)
                )
            )
        )
        (pooler): BertPooler(
            (dense): Linear(in_features=768, out_features=768, bias=True)
            (activation): Tanh()
        )
        (dropout): Dropout(p=0.1, inplace=False)
        (classifier): Linear(in_features=768, out_features=3, bias=True)
    )
    (drop): Dropout(p=0.3, inplace=False)
)
)

```

Layer (type:depth-idx)	Param #	Kernel Shape
BertForSequenceClassification: 1-1	--	--
BertModel: 2-1	--	--
Embeddings: 3-1	--	--
Embedding: 4-1	24,576,000	[768, 32000]
Embedding: 4-2	393,216	[768, 512]
Embedding: 4-3	1,536	[768, 2]
LayerNorm: 4-4	1,536	[768]
Dropout: 4-5	--	--
BertEncoder: 3-2	--	--
ModuleList: 4-6	85,054,464	--
BertPooler: 3-3	--	--
Linear: 4-7	590,592	[768, 768]
Tanh: 4-8	--	--
Dropout: 2-2	--	--
Linearity: 2-3	2,307	[768, 3]
Dropout: 1-2	--	--
Total params: 110,619,651		
Trainable params: 110,619,651		
Non-trainable params: 0		

Gambar 11: Desain arsitektur BERTforSequenceClassification

```

DistilSentimentClassifier(
    (bert): DistilBertForSequenceClassification(
        (distilbertModel): DistilBertModel(
            (embeddings): Embeddings(
                (word_embeddings): Embedding(119547, 768, padding_idx=0)
                (position_embeddings): Embedding(512, 768)
                (LayerNorm): LayerNorm(768,), eps=1e-12, elementwise_affine=True
                (dropout): Dropout(p=0.1, inplace=False)
            )
        )
        (transformer): Transformer(
            (layer): ModuleList(
                (0-5): 6 x TransformerBlock(
                    (attention): MultiHeadSelfAttention(
                        (dropout): Dropout(p=0.1, inplace=False)
                        (q_lin): Linear(in_features=768, out_features=768, bias=True)
                        (k_lin): Linear(in_features=768, out_features=768, bias=True)
                        (v_lin): Linear(in_features=768, out_features=768, bias=True)
                        (out_lin): Linear(in_features=768, out_features=768, bias=True)
                    )
                    (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
                    (ffn): FFN(
                        (dropout): Dropout(p=0.1, inplace=False)
                        (lin1): Linear(in_features=768, out_features=3072, bias=True)
                        (lin2): Linear(in_features=3072, out_features=768, bias=True)
                        (activation): GELUActivation()
                    )
                    (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
                )
            )
        )
        (pre_classifier): Linear(in_features=768, out_features=768, bias=True)
        (classifier): Linear(in_features=768, out_features=3, bias=True)
        (dropout): Dropout(p=0.2, inplace=False)
    )
    (drop): Dropout(p=0.3, inplace=False)
)
)

```

Layer (type:depth-idx)	Param #	Kernel Shape
DistilBertForSequenceClassification: 1-1	--	--
DistilBertModel: 2-1	--	--
Embeddings: 3-1	--	--
Embedding: 4-1	91,812,096	[768, 119547]
Embedding: 4-2	393,216	[768, 512]
LayerNorm: 4-3	1,536	[768]
Dropout: 4-4	--	--
Transformer: 3-2	--	--
ModuleList: 4-5	42,527,232	--
Linear: 2-2	590,592	[768, 768]
Linear: 2-3	2,307	[768, 3]
Dropout: 2-4	--	--
Dropout: 1-2	--	--
Total params: 135,326,979		
Trainable params: 135,326,979		
Non-trainable params: 0		

Gambar 12: Desain arsitektur DistilBERTforSequenceClassification

#### 4 Hasil dan Diskusi

Berdasarkan hasil eksperimen, hasil evaluasi setiap kasus dipaparkan dalam Tabel 1 dan Tabel 2. Tabel 1 menampilkan kinerja model arsitektur BERT pada dataset dengan 1000 baris dengan pembagian 800 data pelatihan, 100 data tes, 100 data validasi. Tabel Tabel 2 menampilkan kinerja model arsitektur BERT dan DistilBERT pada dataset dengan 5000 baris dengan pembagian 4000 data pelatihan, 500 data tes, 500 data validasi

Tabel 1: Kasus 1

Metric	Value
Accuracy	95%
Precision	94.53%
Recall	95%
F1-Score	94.71%
Running Time	7 min 15 s
Trainable Params	110619651

Tabel 2: Kasus 2, Perbandingan model BERT dan DistilBERT

Model	Accuracy	Precision	Recall	F1-Score	Running Time	Trainable Params
BERT	95.4%	95.16%	95.4%	95.04%	35 min 40 s	110619651
DistilBERT	93.8%	93.21%	93.8%	93.24%	18 min 45 s	135326979

Hasil evaluasi dari kasus pertama menunjukkan bahwa model BERT mencapai akurasi sebesar 95%, yang mengindikasikan bahwa 95% dari seluruh prediksi model terhadap data uji adalah benar. Ini berarti model dapat mengklasifikasikan cuitan dengan tingkat kesalahan yang sangat rendah. Selain itu, nilai presisi sebesar 94.53% menunjukkan bahwa dari seluruh prediksi positif yang dihasilkan oleh model, 94.53% di antaranya benar-benar merupakan kelas positif. Nilai *recall* sebesar 95% menunjukkan kemampuan model untuk menangkap hampir semua data yang seharusnya diprediksi sebagai positif, yang berarti model jarang melewatkannya. Skor F1 yang tercatat sebesar 94.71% merupakan rata-rata harmonis antara presisi dan *recall*, yang menunjukkan keseimbangan performa model dalam mendeteksi semua kasus positif dan ketepatan prediksi kasus positif tersebut. Waktu pelatihan model pada kasus 1 tercatat selama 7 menit 15 detik, dengan jumlah parameter latih sebesar 110,619,651 yang menunjukkan bahwa meskipun kompleksitas parameter cukup tinggi, pelatihan pada data berukuran kecil relatif cepat.

Hasil evaluasi dari kasus pertama menunjukkan bahwa BERT menghasilkan akurasi sebesar 95.4%, sedangkan DistilBERT menghasilkan akurasi sebesar 93.8%. Hal ini menunjukkan bahwa BERT tetap unggul dalam tingkat ketepatan keseluruhan prediksi, meskipun DistilBERT sedikit lebih rendah, akurasinya tetap tergolong sangat baik untuk model yang lebih ringan. Dalam hal nilai presisi, BERT mencatatkan nilai sebesar 95.16%, sementara DistilBERT sebesar 93.21%. Ini berarti bahwa BERT sedikit lebih unggul dalam mengurangi kesalahan prediksi positif. *Recall* dari BERT mencapai 95.4%, sedangkan DistilBERT sebesar 93.8%, yang menunjukkan bahwa BERT lebih baik dalam mendeteksi semua cuitan yang seharusnya diklasifikasikan sebagai positif, menandakan performa sensitivitas yang lebih tinggi. Skor F1 BERT sebesar 95.04% lebih tinggi dibandingkan dengan DistilBERT yang mencapai 93.24%, menunjukkan bahwa BERT lebih seimbang dalam hal presisi dan *recall*. Dari sisi efisiensi, DistilBERT memerlukan waktu pelatihan hanya 18 menit 45 detik, hampir setengah dari waktu pelatihan BERT yang membutuhkan 35 menit 40 detik. Meskipun jumlah parameter pada DistilBERT (135,326,979) sedikit lebih besar dibandingkan BERT (110,619,651), DistilBERT menawarkan kecepatan pelatihan yang jauh lebih baik, menjadikannya pilihan menarik untuk implementasi *real-time* atau sumber daya terbatas.

Berdasarkan hasil eksperimen, dapat disimpulkan bahwa BERT menghasilkan performa klasifikasi yang lebih tinggi dalam hal semua metrik evaluasi dibandingkan DistilBERT. Namun, DistilBERT menawarkan keuntungan signifikan dalam hal kecepatan pelatihan, membuatnya lebih cocok digunakan dalam situasi yang membutuhkan efisiensi waktu dan sumber daya komputasi. Dalam kasus-kasus di mana ketepatan klasifikasi (akurasi dan skor F1) menjadi prioritas utama, seperti dalam kasus medis atau aplikasi *high-risk*, BERT lebih direkomendasikan. Namun, untuk aplikasi yang lebih ringan dan skalabilitas cepat, seperti analisis media sosial *real-time*, DistilBERT menjadi alternatif yang layak.

## Daftar Pustaka

- Adel, H., Dahou, A., Mabrouk, A., Elaziz, M. A., Kayed, M., El-Henawy, I. M., Alshathri, S., & Ali, A. A. (2022). Improving crisis events detection using distilbert with hunger games search algorithm. *Mathematics*, 10. <https://doi.org/10.3390/math10030447>
- Black, J. E., Kueper, J. K., & Williamson, T. S. (2023). An introduction to machine learning for classification and prediction. *Family Practice*, 40, 200–204. <https://doi.org/10.1093/fampra/cmac104>
- Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805. <http://arxiv.org/abs/1810.04805>
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. <http://www.deeplearningbook.org>.
- Hammad, M. M. (2024). *Artificial neural network and deep learning: Fundamentals and theory*. <https://arxiv.org/abs/2408.16002>
- Kingma, D. P. & Ba, J. (2017). *Adam: A method for stochastic optimization*. <https://arxiv.org/abs/1412.6980>
- Kumar, S., Roy, P. P., Dogra, D. P., & Kim, B.-G. (2023). A comprehensive review on sentiment analysis: Tasks, approaches and applications. <http://arxiv.org/abs/2311.11250>
- Li, M., Jiang, Y., Zhang, Y., & Zhu, H. (2023). Medical image analysis using deep learning algorithms. *Frontiers in Public Health*, 11. <https://doi.org/10.3389/fpubh.2023.1273253>
- Putri, M. (2024). Studi empiris model bert dan distilbert: Analisis sentimen pada pemilihan presiden indonesia skripsi mahira putri 11200940000055 program studi matematika. Technical report. <https://repository.uinjkt.ac.id/dspace/bitstream/123456789/77084/1/MAHIRA%20PUTRI-FST.pdf>
- Qamar, R. & Zardari, B. (2023). Artificial neural networks: An overview. *Mesopotamian Journal of Computer Science*, 2023, 130–139. <https://doi.org/10.58496/MJCSC/2023/015>
- Roberts, D. A., Yaida, S., & Hanin, B. (2021). The principles of deep learning theory. Technical report.
- Setiawan, T. (2024). *Studi komparasi kinerja analisis sentimen bahasa indonesia berbasis large language model bert dan gpt = comparative study of sentiment analysis performance of indonesian language based on large language model bert and gpt*.
- Sidey-Gibbons, J. A. & Sidey-Gibbons, C. J. (2019). Machine learning in medicine: a practical introduction. *BMC Medical Research Methodology*, 19. <https://doi.org/10.1186/s12874-019-0681-4>
- Stanik, C., Pietz, T., & Maalej, W. (2021). Unsupervised topic discovery in user comments. <http://arxiv.org/abs/2108.08543>
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. *CoRR*, abs/1706.03762. <http://arxiv.org/abs/1706.03762>

Lampiran 1: *Source code*

<https://colab.research.google.com/drive/19neiyz2PxXGWSoHfdTlgkFgeDDOcEuVp?usp=sharing>