BAB II

TINJAUAN PUSTAKA

2.1 Berita Hoaks

Berita hoaks adalah informasi palsu yang sengaja dibuat serta disebarluaskan untuk menyesatkan atau memanipulasi opini publik, seringkali dengan tujuan tertentu seperti keuntungan politik, ekonomi, atau untuk menimbulkan kekacauan di masyarakat. Di era digital ini, penyebaran berita maupun berita hoaks menjadi lebih cepat tersebar khususnya melalui perkembangan teknologi dan media sosial yang memungkinkan informasi menyebar tanpa batasan geografis atau waktu (Bae & Hasan, 2024). Dampak dari adanya berita hoaks dapat merugikan banyak orang, termasuk memicu konflik sosial, serta dapat merusak kepercayaan publik terhadap media dan institusi penting lainnya (Au *et al*., 2022). Penelitian lain telah melakukan deteksi berita hoaks, yaitu Reis *et al*., (2019) menerapkan *KNN, Naive Bayes, Random Forest, SVM, dan XGBoost* dalam bahasa Inggris. Penelitian tersebut hanya mengukur nilai akurasi dengan rata-rata 78%. Yesugade *et al*., (2021) menerapkan LSTM untuk mendeteksi berita hoaks, namun hanya mengukur akurasi sebesar 91,05%. Luo, (2021) menggunakan arsitektur ALBERT yang berbasis *transformer* untuk deteksi berita hoaks dalam bahasa Spanyol. Penelitian tersebut hanya mengukur nilai akurasi dan *f1-score* dengan rata-rata 82%.

2.2 *Text Preprocessing*

Text preprocessing adalah proses mengolah teks mentah menjadi bentuk yang lebih terstruktur dan siap untuk dianalisis lebih lanjut. Tujuan dari *text*

preprocessing adalah untuk membersihkan teks dari elemen-elemen yang tidak relevan dan mengubahnya menjadi format yang lebih mudah diproses oleh model. Tahapan dalam *text preprocessing* meliputi *case folding*, *punctuation removal*, serta *stopword removal* (Imrona *et al*., 2020).

1. *Case Folding*

Teknik mengubah setiap huruf dalam teks menjadi huruf kecil (*lowercase*) dikenal dengan proses *Case folding*. Tujuan dari *case folding* adalah untuk mengurangi variasi kata yang disebabkan oleh perbedaan penggunaan huruf besar dan kecil (Ramadhani *et al*., 2024). C*ase folding* diperlukan dalam *text preprocessing* untuk memastikan bahwa analisis teks tidak terganggu oleh perbedaan kapitalisasi yang tidak relevan.

2. *Punctuation Removal*

Proses menghilangkan tanda baca dari teks termasuk menghilangkan titik, koma, tanda tanya, tanda seru, tanda kutip, dan simbol serupa lainnya disebut *punctuation removal*. Tujuan dari *punctuation removal* adalah untuk membersihkan teks dari elemen-elemen yang tidak memiliki kontribusi signifikan terhadap pemahaman atau analisis teks. Penghapusan tanda baca pada teks dapat membuat lebih fokus pada kata-kata relevan yang memudahkan proses analisis oleh model dalam tahap berikutnya (Garg & Sharma, 2022).

3. *Stopword Removal*

Stopword removal adalah proses menghapus kata-kata umum yang sering muncul dalam teks, tetapi memiliki nilai informasi yang rendah, seperti "dan," "atau," "adalah," "dari," dan sebagainya. *Stopwords* biasanya tidak memiliki makna

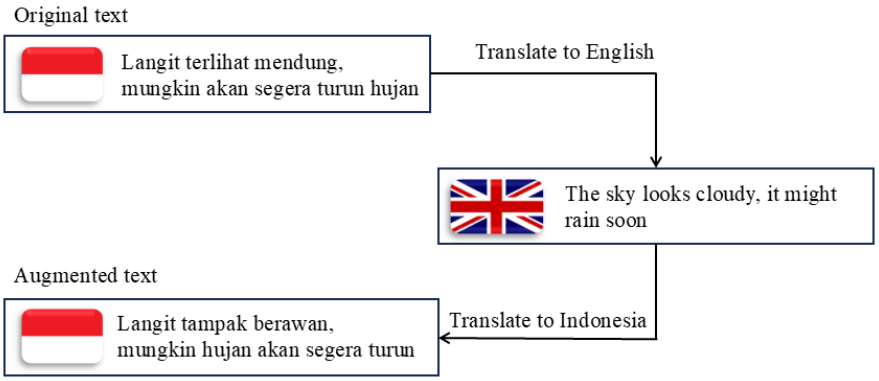
signifikan dalam konteks analisis teks, sehingga penghapusannya membantu fokus pada kata-kata yang lebih penting dan bermakna dalam teks. Menghilangkan *stopwords* dapat membuat model lebih efektif dalam menganalisis konten yang relevan (Fayaza & Farhath, 2021).

2.3 Augmentasi Data

Augmentasi data adalah teknik yang digunakan untuk menambah jumlah dan variasi data dalam suatu *dataset* dengan memodifikasi data yang sudah ada tanpa mengubah informasi utamanya (Bayer *et al*., 2022). Teknik ini sering digunakan dalam untuk memperluas *dataset* yang terbatas dan meningkatkan kemampuan generalisasi model. Hasilnya model akan menjadi lebih tahan terhadap variasi data baru yang belum pernah ditemui.

1. *Back Translation*

Metode *back translation* merupakan teknik augmentasi data dalam pemrosesan bahasa alami (*Natural Language Processing*) di mana sebuah teks diterjemahkan ke bahasa lain, lalu diterjemahkan kembali ke bahasa asli (Beddiar *et al*., 2021). Teknik ini digunakan untuk menghasilkan variasi teks yang tetap mempertahankan makna aslinya. Ilustrasi pada *back translation* dapat dilihat pada Gambar 2.1.



Gambar 2.1 Ilustrasi *Back Translation* (Sumber: Beddiar *et al*., 2021)

Berdasarkan Gambar 2.1, proses augmentasi *back translation* dilakukan dengan menerjemahkan suatu bahasa ke bahasa lain lalu menerjemahkan lagi ke bahasa asal. Pada penelitian ini teks bahasa Indonesia diterjemahkan ke bahasa Inggris, lalu diterjemahkan lagi ke bahasa Indonesia.

2. *TextAttack*

Metode *TextAttack* adalah *framework* berbasis *Python* yang dikembangkan untuk melakukan *adversarial attacks, adversarial training*, dan *data augmentation* pada tugas NLP (*Natural Language Processing*). *TextAttack* menggunakan berbagai metode transformasi teks seperti *synonym replacement, random insertion, random swap, random deletion,* dan *paraphrasing*. *TextAttack* dapat menghasilkan data yang lebih bervariatif (Morris *et al*., 2020).

2.4 Tokenisasi

Tokenisasi merupakan proses dalam NLP yang membagi teks menjadi unit-unit lebih kecil yang disebut token. Token ini bisa berupa kata, frasa, atau simbol yang berguna untuk analisis teks lebih lanjut (Desiani *et al*., 2023). Tokenisasi memungkinkan teks mentah dipecah menjadi bagian-bagian yang lebih mudah dikelola dan dianalisis, seperti dalam model klasifikasi teks.

2.5 *Padding* dan *Sequence*

Padding adalah teknik yang digunakan untuk menambahkan elemen tambahan, biasanya nol atau token khusus pada data agar semua *input* memiliki panjang yang konsisten. Hal ini penting dalam pemrosesan teks untuk memastikan bahwa setiap urutan data, seperti kalimat, memiliki dimensi yang sama ketika diproses oleh model. *Sequence* merujuk pada urutan elemen yang saling terkait, seperti kata dalam kalimat atau langkah dalam proses yang penting untuk memahami struktur dan konteks data (Yesugade *et al*., 2021).

2.6 *Embedding Layer*

Embedding layer adalah tahap yang mengubah kata-kata menjadi representasi vektor. Tujuan utama dari *embedding layer* adalah untuk menangkap makna semantik dari kata-kata dan hubungan antar kata dalam bentuk vektor yang dapat digunakan dalam proses pelatihan model (Desiani *et al.*, 2023). Pada NLP, *embedding layer* mengubah kata-kata menjadi vektor dengan dimensi tetap, memungkinkan model untuk mempelajari dan memahami konteks serta pola dalam data teks (Pilehvar *et al.*, 2020). Rumus perhitungan pada *embedding layer* ditunjukkan pada Persamaan (2.1) (Nathani *et al*., 2020).

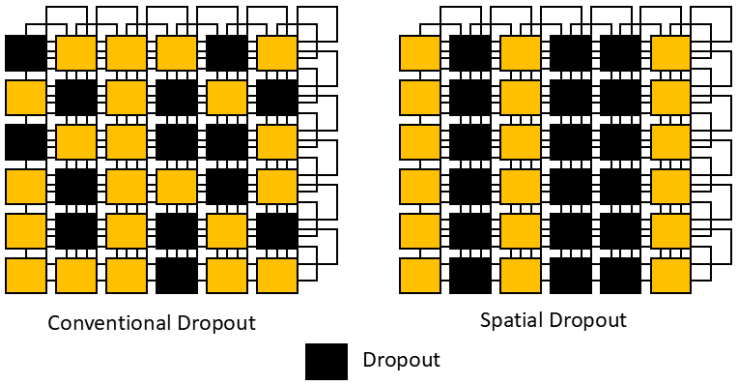
$$X = [T_1W; T_2W; T_3W; \dots;\ T_nW] \tag{2.1}$$

Di mana T_i adalah *one hot encoding* dari token ke-i dalam bentuk matriks dengan $i = 1,2, \dots, n$ dan W adalah matriks *embedding* yang dibangun secara acak.

2.7 *Spatial Dropout Layer*

Jenis khusus dari lapisan *dropout* yang disebut *spatial dropout* digunakan untuk mempercepat pelatihan model dan mencegah terjadinya *underfitting* maupun

overfitting (Shunk, 2022). Lapisan *dropout* memodifikasi data yang tersisa dalam jaringan dan menghilangkan nilai-nilai acak dari matriks (Uddin *et al.*, 2021). *Spatial dropout* secara eksplisit menghilangkan nilai-nilai acak pada dimensi tertentu dari matriks dengan menetapkan nilai-nilai tersebut menjadi nol (Zhang *et al*., 2022). Ilustrasi untuk *spatial dropout* dapat dilihat pada Gambar 2.2.



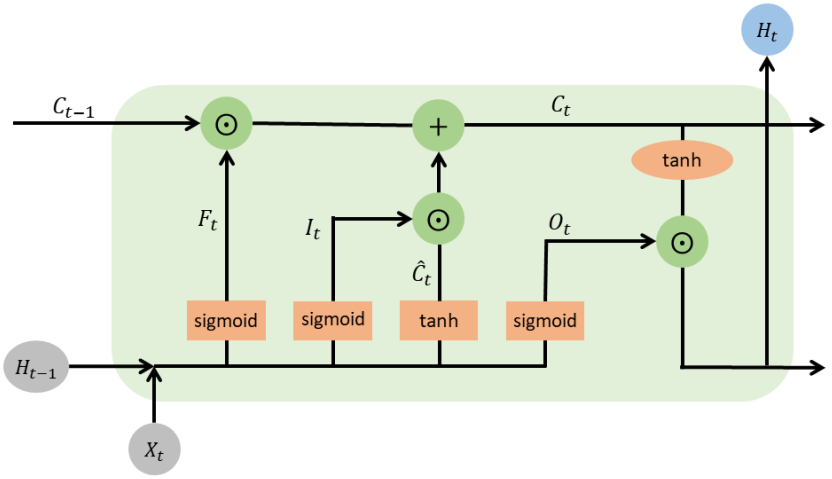
Gambar 2.2 Ilustrasi *Spatial Dropout* (Sumber: Zhang *et al*., 2022)

Berdasarkan Gambar 2.2, warna hitam menunjukkan entri matriks yang dilakukan *dropout* dengan mengubah entri matriks tersebut menjadi nilai nol, sedangkan warna jingga menunjukkan entri matriks yang tidak dilakukan *dropout*. *Conventional dropout* menghilangkan elemen-elemen matriks secara acak. Pada *spatial dropout*, elemen-elemen matriks dihapus berdasarkan dimensi tertentu. Elemen yang dihilangkan pada *spatial dropout* digantikan dengan nilai nol, menghasilkan deretan nilai nol pada dimensi tertentu.

2.8 *Bidirectional* LSTM

Bidirectional LSTM (BiLSTM) adalah jenis arsitektur LSTM yang memproses data dari dua arah, yaitu *forward* dan *backward*. BiLSTM dapat menangkap informasi konteks dari kedua sisi urutan *input*. BiLSTM menggunakan dua LSTM dalam strukturnya (Adil *et al*., 2021). *Input gate, forget gate*, dan *output*

gate adalah tiga gerbang utama yang membentuk arsitektur LSTM. Data yang ada dapat dibaca, disimpan, dan diperbarui melalui ketiga gerbang tersebut (Elsaraiti & Merabet, 2021). Gambar 2.3 merupakan ilustrasi dari arsitektur LSTM.



Gambar 2.3 Ilustrasi Arsitektur LSTM (Sumber: Dong *et al*., 2020)

Berdasarkan Gambar 2.3, dengan menggunakan *input* X_t hasil dari proses embedding dan H_{t-1}, algoritma LSTM pertama-tama memutuskan informasi mana yang harus dihilangkan. Fungsi aktivasi sigmoid digunakan pada *forget gate*. Proses selanjutnya adalah *input gate* yang menggunakan fungsi aktivasi sigmoid untuk memutuskan informasi baru yang akan ditambahkan. Proses berlanjut ke *output gate*, di mana fungsi aktivasi *tanh* diterapkan untuk memproses data dari *input gate* untuk menciptakan informasi baru, sehingga menghasilkan *output*.

LSTM terdiri dari dua jenis, yaitu LSTM *forward* dan LSTM *backward*. LSTM *forward* memproses data secara sekuensial dari waktu $t = 1$ ke $t = T$, sementara itu LSTM *backward* memproses ulang data dari waktu $t = T$ kembali ke $t = 1$. Persamaan yang digunakan untuk perhitungan LSTM *forward* dapat dilihat pada Persamaan (2.2) hingga Persamaan (2.7) (Wadawadagi & Pagi, 2020).

$$F_t = sigmoid(W_{XF}X_t + W_{HF}H_{t-1} + B_F) \quad (2.2)$$

$$I_t = sigmoid(W_{XI}X_t + W_{HI}H_{t-1} + B_I) \quad (2.3)$$

$$\hat{C}_t = \tanh\,(W_{X\hat{C}}X_t + W_{H\hat{C}}H_{t-1} + B_{\hat{C}}) \quad (2.4)$$

$$C_t = F_t \odot C_{t-1} + I_t \odot \hat{C}_t \quad (2.5)$$

$$O_t = sigmoid(W_{XO}X_t + W_{HO}H_{t-1} + B_O) \quad (2.6)$$

$$\vec{H}_t = H_t = O_t \odot tanh\,(C_t) \quad (2.7)$$

Di mana $\odot$ adalah operasi *element-wise product* merupakan operasi perkalian antar entri matriks yang bersesuaian, F_t adalah *forget gate,* I_t adalah *input gate*, $\hat{C}_t$ adalah *modulation gate*, C_t adalah *cell state,* O_t adalah *output gate,* H_t adalah *hidden state,* dan X_t nilai dari *input* pada waktu ke-t. W dan B mengacu pada matriks bobot dan bias pada setiap *gate* dan *state*. Persamaan LSTM *backward* ditunjukkan dalam Persamaan (2.8) hingga Persamaan (2.15)

$$\delta H_t = \triangle_t + \triangle H_t \quad (2.8)$$

$$\delta C_t = \delta H_t \odot O_t \odot (1 - tanh^2(C_t)) + \delta C_{t+1} \odot F_{t+1} \quad (2.9)$$

$$\delta \hat{C}_t = \delta C_t \odot I_t \odot (1 - \hat{C}_t^2) \quad (2.10)$$

$$\delta I_t = \delta C_t \odot \hat{C}_t \odot I_t \odot (1 - I_t) \quad (2.11)$$

$$\delta \mathrm{F}_t = \delta C_t \odot C_{t-1} \odot F_t \odot (1 - F_t) \quad (2.12)$$

$$\delta \mathrm{O}_t = \delta H_t \odot \tanh\,(C_t) \odot O_t \odot (1 - O_t) \quad (2.13)$$

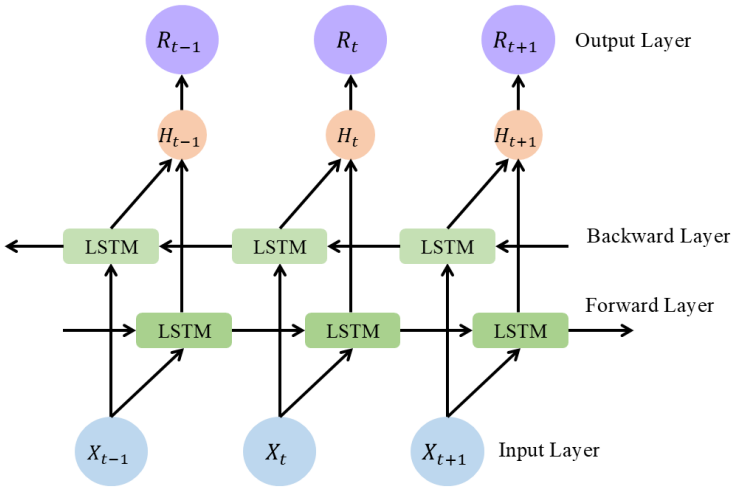
$$\delta X_t = W_X^T \delta gates_t = W_{XF}^T \delta F_t + W_{XI}^T \delta I_t + W_{X\hat{C}}^T \delta \hat{C}_t + W_{XO}^T \delta O_t \quad (2.14)$$

$$\overleftarrow{H}_t = W_H^T \delta gates_t = W_{HF}^T \delta F_t + W_{HI}^T \delta I_t + W_{H\hat{C}}^T \delta \hat{C}_t + W_{HO}^T \delta O_t \quad (2.15)$$

Di mana $\triangle_t$ adalah *error* antara hasil prediksi dengan label asli, sedangkan δI_t merepresentasikan nilai *input gate,* δF_t merupakan *forget gate,* δO_t merupakan

output gate, $\delta\hat{C}_t$ merupakan *modulation gate,* $\overleftarrow{H}_t$ merupakan *hidden state,* dan δX_t merupakan *input*.

Berbeda dengan LSTM yang hanya menggunakan satu lapisan LSTM, *Bidirectional LSTM* menggunakan dua lapisan LSTM, yaitu *LSTM forward* dan *LSTM backward* dalam arsitekturnya (Wadawadagi & Pagi, 2020). Ilustrasi arsitektur BiLSTM dapat dilihat pada Gambar 2.4.



Gambar 2.4 Ilustrasi Arsitektur *Bidirectional* LSTM (Sumber: Bahad *et al*., 2019)

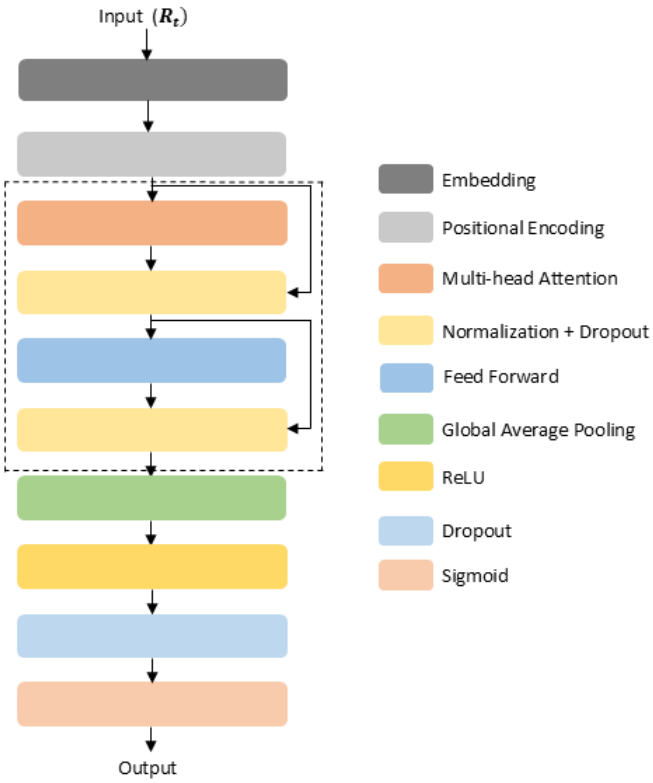
Menurut ilustrasi pada Gambar 2.4, X_t merupakan matriks input hasil *embedding*, H_t merupakan matriks hasil dari LSTM dan R_t merupakan matriks output hasil dari BiLSTM. Arsitektur BiLSTM memiliki dua lapisan LSTM. Proses operasi pada BiLSTM mengikuti langkah-langkah yang sama seperti LSTM, namun perbedaannya adalah BiLSTM melakukan operasi LSTM dua kali, yaitu pada lapisan *forward* dan lapisan *backward*. Perhitungan dalam arsitektur BiLSTM dijelaskan pada Persamaan (2.16).

$$R_t = W_{\vec{H}R}\vec{H}_t + W_{\overleftarrow{H}R}\overleftarrow{H}_t + B_R \tag{2.16}$$

Di mana $\vec{H}_t = H_t$ adalah *hidden state* pada lapisan *forward* dengan t mulai dari 1 hingga T, $\overleftarrow{H}_t$ adalah *hidden state* pada lapisan *backward* dengan t mulai dari T hingga 1, dan R_t adalah *output.*

2.9 *Transformer*

Transformer adalah arsitektur jaringan syaraf tiruan yang digunakan dalam pemrosesan bahasa alami yang menggabungkan pendekatan *sequence-to-sequence* dan memanfaatkan lapisan *self-attention* untuk menghasilkan representasi matriks dari data *input* (Kumari *et al*., 2023). Pada arsitektur *Transformer*, urutan data seperti kata-kata dalam kalimat disematkan dan diteruskan melalui *positional encoding* yang menetapkan matriks berdasarkan posisi kata dalam kalimat. Ilustrasi arsitektur *Transformer* dapat dilihat pada Gambar 2.5.



Gambar 2.5 Ilustrasi Arsitektur *Transformer* (Sumber: Desiani *et al*., 2023)

Berdasarkan ilustrasi *Transformer* pada Gambar 2.5, blok *encoder* terdiri dari *multi-head attention* dan jaringan *feed-forward* yang menerima penyematan ini dan menghitung matriks *attention* untuk memahami hubungan antara kata-kata dalam kalimat. Matriks *attention* ini kemudian diteruskan melalui jaringan *feed-forward* ke blok *decoder* satu per satu. Paralelisasi dalam lapisan *multi-head attention* memungkinkan model untuk secara efisien menangkap dan memproses hubungan antar kata dalam data (Acheampong *et al*., 2021).

2.9.1 *Positional Encoding*

Positional encoding digunakan untuk memberikan informasi mengenai posisi token dalam sebuah urutan, serta menyajikan posisi relatif dan absolut setiap token dalam urutan tersebut. Dimensi dari *positional encoding* sesuai dengan *embedding* model. Fungsi sinus dan cosinus pada berbagai frekuensi dapat digunakan untuk menentukan posisi token dalam urutan (Liu *et al*., 2020). *Positional encoding* diterapkan dengan fungsi sinus dan cosinus pada frekuensi yang berbeda, seperti yang ditunjukkan dalam Persamaan (2.17) dan Persamaan (2.18).

$$Pe(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d}}\right) \tag{2.17}$$

$$Pe(pos, 2i+1) = \cos\left(\frac{pos}{10000^{2i/d}}\right) \tag{2.18}$$

Di mana pos merupakan posisi, d adalah dimensi matriks, dan i adalah indeks yang memetakan kolom dengan nilai $0 \leq i < d/2$. Persamaan (2.19) digunakan untuk hasil matriks *positional encoding*. Persamaan (2.20) digunakan untuk menambahkan hasil *positional encoding* ke matriks hasil BiLSTM.

$$E_{pos} = [Pe(pos, 0), Pe(pos, 1), Pe(pos, 2), \dots, Pe(pos, d-1)] \quad (2.19)$$

$$Z = R_t + E_{pos} \quad (2.20)$$

E_{pos} merupakan hasil dari *positional encoding* yang berbentuk matriks posisi, sedangkan R_t adalah matriks hasil dari proses BiLSTM dan Z adalah matriks hasil penjumlahan dari matriks BiLSTM dengan *positional encoding*.

2.9.2 *Self Attention*

Self-attention digunakan untuk memungkinkan setiap kata berinteraksi dengan kata lainnya (*self*) dan menentukan kata mana yang akan menerima perhatian terbesar (*attention*). Matriks *key, value*, dan *query* dihasilkan dari setiap *input* matriks dari *embedding layer*. Untuk setiap kata, dibuat matriks *query*, *key*, dan *value*. Matriks-matriks ini dihasilkan dari perkalian antara ketiga matriks yang terbentuk selama proses pelatihan, kemudian fungsi *softmax* diterapkan untuk mendapatkan bobot. Matriks *query, key, value*, dan *attention* dapat dihitung menggunakan Persamaan (2.21) hingga Persamaan (2.24).

$$Q = W_Q Z \quad (2.21)$$

$$K = W_K Z \quad (2.22)$$

$$V = W_V Z \quad (2.23)$$

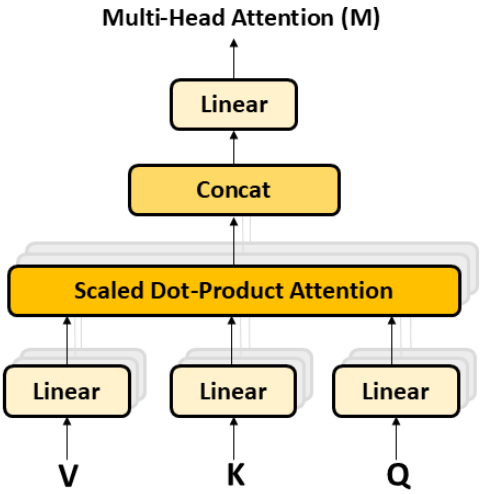
$$A = A(Q, K, V) = softmax \frac{(QK^t)}{\sqrt{d_k}} V \quad (2.24)$$

Di mana Q adalah *query*, K adalah *key*, V adalah *value*, d_k adalah dimensi matriks, Z adalah matriks kata hasil dari tahap *positional encoding*, dan t adalah panjang urutan *input*. Matriks pada tahap ini akan menggunakan fungsi aktivasi

softmax dan matriks *value* untuk menghasilkan *output* akhir dari lapisan *self-attention* (Mohiuddin *et al*., 2023).

2.9.3 *Multi-Head Attention*

Multi-head attention merupakan varian dari *self-attention* dalam arsitektur *transformer* yang memungkinkan model untuk lebih efektif menangkap hubungan dan ketergantungan dalam urutan *input*. Pada *multi-head attention*, *self-attention* dihitung beberapa kali dengan bobot yang berbeda yang disebut sebagai *head*. Setiap *head* memiliki kemampuan untuk memahami berbagai aspek dari urutan tersebut (Mohiuddin *et al*., 2023). Ilustrasi *multi-head attention* dapat dilihat pada Gambar 2.6.



Gambar 2.6 Ilustrasi *Multi-head attention* (Sumber: Mohiuddin *et al*., 2023)

Perhitungan pada *multi-head attention* ini dapat dilihat pada Persamaan (2.25) hingga Persamaan (2.26).

$$Head_i = A(QW_i^Q, KW_i^K, VW_i^V) \tag{2.25}$$

$$M = Concat(Head_1, Head_2, \dots, Head_n)W_0 \tag{2.26}$$

Indeks i menunjukkan posisi ke-i, sedangkan W_O adalah matriks bobot tambahan yang dihasilkan oleh komputer untuk menggabungkan *concat attention head* sebelum diteruskan ke lapisan selanjutnya.

2.9.4 *Normalization Layer*

Normalization layer adalah komponen penting dalam *Transformer*. Lapisan ini berfungsi untuk menjaga agar rentang nilai dalam suatu lapisan tidak berfluktuasi secara berlebihan, sehingga membantu model mencapai konvergensi lebih cepat (Xiong *et al*., 2020). *Normalization Layer* dihitung menggunakan Persamaan (2.27) hingga Persamaan (2.29).

$$\mu_j = \frac{1}{b}\sum_{i=1}^{b} m_{ij} \tag{2.27}$$

$$\sigma_j^2 = \frac{1}{b}\sum_{i=1}^{b} (m_{ij} - \mu_j)^2 \tag{2.28}$$

$$n_{ij} = \frac{m_{ij} - \mu_j}{\sqrt{\sigma_j^2}} \tag{2.29}$$

Di mana b adalah jumlah baris matriks, μ merupakan rata-rata, σ adalah variansi, m_{ij} merupakan entri matriks M pada baris ke-i dan kolom ke-j (Xiong *et al*., 2020).

2.9.5 *Feed Forward Layer*

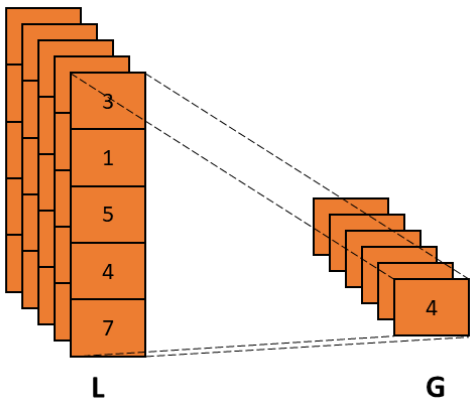
Feed forward layer merupakan lapisan *encoder* dari *transformer* terdiri dari yang diterapkan pada setiap entri-entri matriks. Jaringan *feed forward* memiliki tiga komponen yakni *input layer*, *hidden layer*, dan *output layer.* Dalam menghitung *feed forward layer* dapat menggunakan Persamaan (2.30).

$$l_{ij} = f\left(\sum_{i=1}^{n}(n_{ij}\, w_{ij}) + b_j\right) \tag{2.30}$$

Di mana l_{ij} adalah hasil dari *feed forward layer*, f adalah fungsi aktivasi yang digunakan, yakni fungsi aktivasi ReLU, n_{ij} adalah nilai input, w_{ij} adalah bobot input, b_j adalah bias.

2.9.6 *Global Average Pooling*

Global average pooling merupakan metode yang digunakan untuk mengatasi masalah jumlah parameter yang berlebihan pada jaringan *fully connected*. Teknik ini menghitung rata-rata dari setiap *feature map*, kemudian hasil rata-rata tersebut dimasukkan ke dalam lapisan *output* (Desiani *et al*., 2023). Ilustrasi *global average pooling* dapat dilihat pada Gambar 2.7.



Gambar 2.7 Ilustrasi *Global Average Pooling* (Sumber: Hsiao *et al*., 2019)

2.10 *Fully Connected Layer*

Fully Connected Layer (FCL), atau yang sering disebut sebagai *Dense Layer* adalah salah satu komponen inti dalam jaringan saraf tiruan (*Artificial Neural Network*). *Layer* ini menghubungkan setiap *neuron* pada *layer* sebelumnya dengan setiap *neuron* pada *layer* berikutnya, sehingga semua neuron *saling* terhubung

secara penuh. Untuk menghitung *fully connected layer* atau *dense layer* dapat menggunakan Persamaan (2.31).

$$D = sigmoid(W_D L + B) \tag{2.31}$$

Di mana D merupakan matriks hasil *fully connected layer* dengan menggunakan fungsi aktivasi sigmoid, L merupakan matriks *input* hasil dari *feed forward layer,* W_D merupakan matriks bobot, dan B merupakan matriks bias.

2.11 Fungsi Aktivasi

Sebuah fungsi yang mentransformasikan suatu *input* menjadi *output* tertentu dikenal sebagai fungsi aktivasi (Ahmadi *et al*., 2021). Fungsi aktivasi *Rectified Linear Units* (ReLU) adalah fungsi *non linier* yang mengubah semua nilai negatif menjadi 0 (Bai, 2022). Persamaan untuk fungsi aktivasi ReLU dapat dilihat pada Persamaan (2.32).

$$f_{ij} = ReLU(d_{ij}) = max\left(0, d_{ij}\right) \tag{2.32}$$

Fungsi aktivasi sigmoid adalah fungsi yang mengkonversi nilai f_{ij} menjadi rentang antara 0 hingga 1. Persamaan (2.33) merupakan rumus perhitungan fungsi aktivasi sigmoid (Rajput *et al*., 2021).

$$s_{ij} = sigmoid(f_{ij}) = \frac{1}{1 + e^{-(f_{ij})}} \tag{2.33}$$

Di mana f_{ij} anggota bilangan riil dan nilai s_{ij} adalah hasil dari fungsi aktivasi sigmoid.

2.12 *Loss Function*: *Binary Cross-entropy*

Loss function adalah ukuran kesalahan dalam proses pelatihan model yang digunakan untuk mengevaluasi perbedaan antara nilai yang diprediksi dengan nilai

aktual. *Binary cross entropy* merupakan salah satu *loss function* yang mengukur perbedaan antara dua distribusi probabilitas ketika diberikan kejadian acak (Jadon, 2020). Perhitungan *binary cross entropy* dijelaskan melalui Persamaan (2.34).

$$p = -\frac{1}{n}\sum_{i=1}^{n}\left(\left(y_{ij} log\left(s_{ij}\right)\right) + \left(1 - y_{ij}\right) log\left(1 - \left(s_{ij}\right)\right)\right) \quad (2.34)$$

Di mana s_{ij} merupakan entri matriks hasil prediksi dan y_{ij} merupakan entri matriks label sebenarnya.

2.12 *Optiminization Function*: *Adaptive Moment Estimation* (Adam)

Adam adalah *optimizer* yang dikenal karena menggunakan sedikit memori dan melakukan perhitungan secara efisien (Yin *et al*., 2021). Menurut (Lihua, 2022), Persamaan (2.35) hingga Persamaan (2.37) digunakan untuk menghitung nilai momen pertama pada *epoch* ke-t (m_t) dan momen kedua pada *epoch* ke-t (v_t).

$$G_t = (S - Y)G^T \quad (2.35)$$

$$M_t = \beta_1 M_{t-1} + (1 - \beta_1)G_t \quad (2.36)$$

$$V_t = \beta_2 V_{t-1} + (1 - \beta_2)(G_t)^2 \quad (2.37)$$

Di mana S merupakan matriks hasil prediksi, Y merupakan matriks label sebenarnya, G^T merupakan matriks transpose hasil dari *global average pooling*, β_1 merupakan *decay rate* untuk momen pertama, β_2 merupakan *decay rate* untuk momen kedua, dan G_t adalah gradien yang dihitung pada *epoch* ke-t.

Persamaan (2.38) hingga Persamaan (2.39) digunakan untuk menghitung *estimator* momen pertama ($\widehat{M}_t$) dan momen kedua ($\hat{V}_t$) setelah nilai momen pertama dan kedua ditentukan (Lihua, 2022).

$$\widehat{M}_t = \frac{M_t}{1-(\beta_1)^t} \tag{2.38}$$

$$\widehat{V}_t = \frac{V_t}{1-(\beta_2)^t} \tag{2.39}$$

Tahapan selanjutnya adalah perbarui bobot pada *epoch* ke-t dengan *learning rate* sebesar η dapat dilakukan menggunakan Persamaan (2.40) (Lihua, 2022).

$$W_{baru} = W_{lama} - \eta \frac{\widehat{M}_t}{\sqrt{\widehat{V}_t} + \varepsilon} \tag{2.40}$$

Di mana ε adalah bilangan yang sangat kecil (misalkan 10^{-5}) untuk menghindari pembagian dengan 0.

2.13 *Confusion* Matriks

Confusion matriks adalah alat evaluasi yang menggambarkan kinerja model klasifikasi. Performa model dapat diukur menggunakan *confusion* matriks, dengan akurasi, presisi, *recall, dan* f1-*score* dari model (Desiani *et al*., 2022). *Confusion matrix* pada deteksi berita hoaks dibedakan menjadi dua label, yakni label 1 untuk berita hoaks dan label 0 untuk berita valid. *Confusion matrix* pada klasifikasi berita hoaks dapat dilihat pada Tabel 2.1.

Tabel 2.1 *Confusion Matrix* pada Klasifikasi Berita Hoaks

Aktual	Hasil Prediksi	
	Hoaks	Valid
Hoaks	*True Positive* (TP)	*False Negative* (FN)
Valid	*False Positive* (FP)	*True Negative* (TN)

(Sumber: Bahad *et al*., 2019)

Beberapa evaluasi kinerja yang dapat digunakan untuk klasifikasi nilai-nilai pada *confusion matrix* antara lain adalah akurasi, presisi, *recall*, dan *f1-score* (Chicco & Jurman, 2020). Menurut (Islam *et al*., 2020), akurasi adalah ukuran proporsi dari seluruh prediksi yang benar dibandingkan dengan keseluruhan data.

Presisi adalah rasio antara jumlah prediksi positif yang benar dan total prediksi positif. *Recall* adalah rasio antara jumlah prediksi positif yang benar dan total data positif aktual. F1-*score* adalah rata-rata harmonis dari presisi dan *recall* yang memberikan keseimbangan antara keduanya, terutama ketika terdapat ketidakseimbangan antara kelas positif dan negatif (Nguyen *et al*., 2021). Persamaan (2.41) hingga Persamaan (2.44) menunjukkan rumus untuk menghitung akurasi, presisi, *recall*, dan *f1-score* (Alotaibi *et al*., 2021).

$$akurasi = \frac{TP + TN}{TP + FN + TN + FP} \times 100\% \quad (2.41)$$

$$presisi = \frac{TP}{TP + FP} \times 100\% \quad (2.42)$$

$$recall = \frac{TP}{TP + FN} \times 100\% \quad (2.43)$$

$$f1 - score = 2 \times \frac{recall \times presisi}{recall + presisi} \times 100\% \quad (2.44)$$

Nilai akurasi, presisi, *recall*, dan *f1-score* dibagi menjadi beberapa kelompok berdasarkan (Carneiro *et al*., 2021), seperti yang ditunjukkan pada Tabel 2.2.

Tabel 2.2 Kategori Nilai Evaluasi Kinerja Model

Nilai kinerja (NK) (%)	Kategori
NK > 90%	Sangat Baik
80 < NK ≤ 90	Baik
70 < NK ≤ 80	Cukup Baik
60 < NK ≤ 70	Kurang Baik
NK ≤ 60	Gagal

(Sumber: Carneiro *et al*., 2021)

Tabel 2.2 menunjukkan bahwa kategori nilai evaluasi kinerja model berada dalam rentang tertentu. Nilai kinerja masuk ke dalam kategori sangat baik jika lebih besar dari 90%. Nilai kinerja yang masuk dalam kategori baik adalah nilai yang

berkisar antara 80% hingga 90%. Nilai kinerja dianggap cukup jika berada di kisaran 70% hingga 80%. Nilai kinerja yang termasuk dalam kisaran buruk adalah nilai yang berada di antara 60% dan 70%. Jika nilai kinerja 60% atau lebih rendah, maka dianggap gagal.