BAB IV
SEQUENCE TO SET DALAM BAHASA INDONESIA

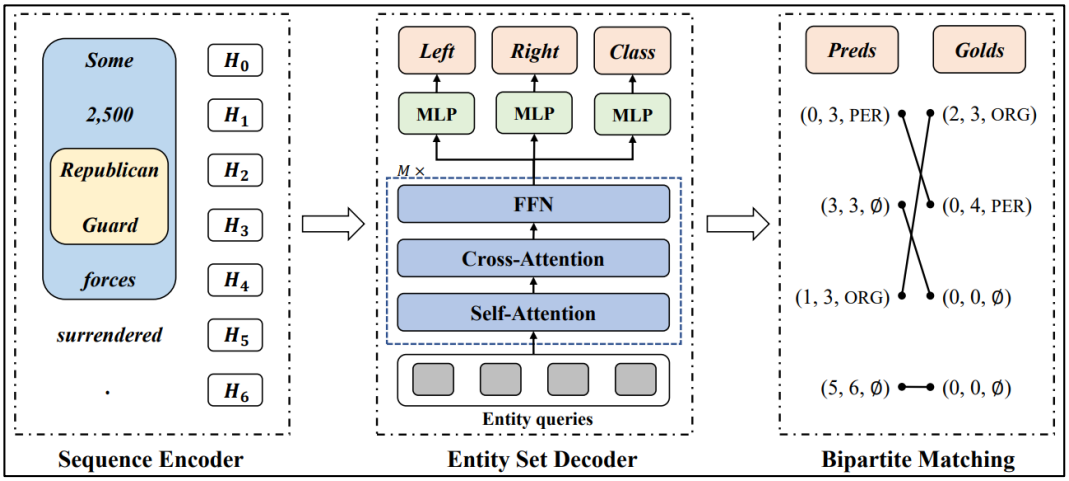
Sebelum memasuki uji coba tugas akhir ini, perlu diketahui seluruh isi dari arsitektur Sequence-To-Set Network secara detail. Sequence-To-Set Network memiliki tiga bagian besar dari arsitekturnya yaitu *encoder layer*, *decoder layer* dan *bipartite matching*. Bab ini akan dibagi menjadi tiga subbab juga, menjelaskan masing-masing tiga bagian besar itu, juga tiap subbab akan dibagi menjadi subbab kecil lagi sesuai dengan komponen/metode/fungsi yang digunakan untuk mencapai struktur akhir dari tiap subbab. Sebagai contoh, untuk encoder layer memiliki lima subbab kecil karena penggunaan empat jenis *word embedding* di mana tiap embedding memiliki cara tersendiri dan diperlukan penjelasan untuk tiap cara embedding tersebut. Namun, sebelum memasuki penjelasan tiap layer akan dijelaskan secara keseluruhan alur dari arsitektur sistem Sequence-To-Set Network.

4.1 Sequence To Set Network

Sesuai pada Gambar 4.1, alur dari Sequence-To-Set Network terdiri dari sequence encoder layer, entity set decoder layer dan bipartite matching dalam urutan yang telah disebut. Alur dari sistem ini adalah pertama kalimat input diterima oleh sequence encoder layer dalam bentuk struktur data yang telah dibahas sebelumnya mengenai pra proses dataset. Di mana output dari encoder ini memiliki tujuan untuk mendapatkan representasi dari kalimat tersebut untuk tiap kata dari kalimat tersebut dengan informasi dan konteks yang didapatkan.

Cara encoder mendapatkan informasi ini adalah dari beberapa metode word embedding yang telah ditentukan. Setiap dari embedding tersebut, seperti yang telah dijelaskan, akan dikenal lebih dalam pada subbab-subbab nya sendiri. Terdapat lima jenis word embedding yang disediakan, namun pada tugas akhir ini hanya digunakan empat (BERT, Word2Vec, Character-level LSTM, BiLSTM, POS Tag). Seluruh hasil embedding dari tiap metode akan kemudian digabung menjadi satu variabel dalam bentuk *vector*. Representasi kata terakhir ini Bersama dengan

sebuah set vektor yang dapat dipelajari (dipanggil sebagai entity queries) dan menjadi output terakhir dari sequence encoder, juga menjadi input untuk bagian berikut yaitu set decoder layer.



Gambar 4.1
Arsitektur Sistem Sequence-To-Set Network[1]

Entity set decoder layer, mengambil entity queries dan representasi token dari sequence encoder yang telah memberikan nilai-nilai konteks dan informasi yang penting untuk tiap kata agar decoder mengerti kata apa yang perlu diperhatikan untuk diprediksikan. Cara untuk memperhatikan dan melakukan proses decoding ini adalah dengan bantuan attention. Pada bab teori penunjang telah menjelaskan sekilas mengenai self-attention dan cross-attention masing-masing. Untuk subbab decoder di bab ini akan menjelaskan lebih rinci terhadap alur, input, output tiap layer mengingat bahwa beberapa dari layer decoder memiliki cara kerja dan peran berbeda.

Dan bagian terakhir ada bagian yang menghitung nilai error dari keseluruhan training yang telah dilakukan encoder dan decoder Sequence-To-Set Network ini. Cara untuk menghitung nilai error dari perbandingan set cukup menyusahkan, karena itu Sequence-To-Set Network memilih bipartite matching sebagai metode penghitungan loss function untuk menghitung pencocokan optimal

[1] Zeqi Tan, dkk, A Sequence-to-Set Network for Nested Named Entity Recognition, (2021).

dari prediksi dan *golden entity*. Nilai optimal dari pencocokan set prediksi dengan golden entity didapatkan dengan bantuan dari Algoritma Hungarian.

4.1.1 Sequence Encoder

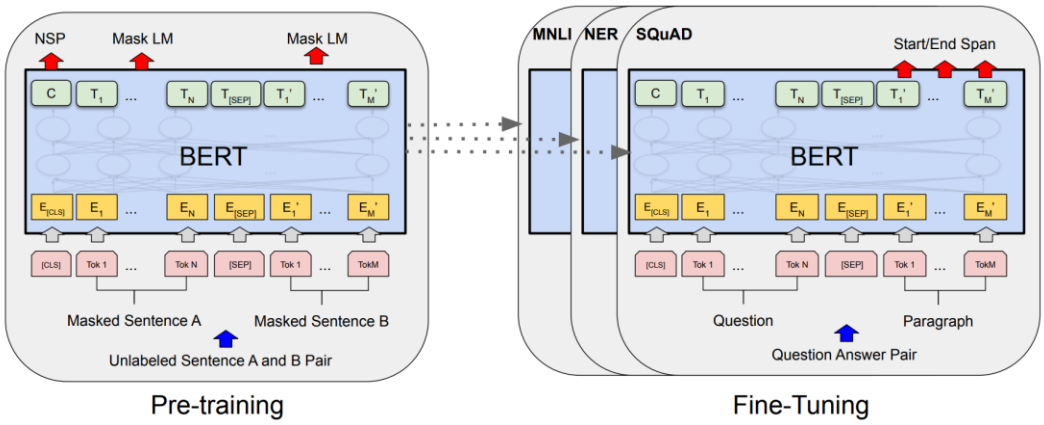
Sequence encoder adalah proses/bagian/layer pertama yang berada di Sequence-To-Set Network. Sama dengan penjelasan sebelumnya, tujuan dari proses ini adalah untuk mendapatkan informasi dan konteks kata-kata dari kalimat input yang didapatkan. Seluruh subbab dari bagian ini adalah semua word embedding yang digunakan pada tugas akhir ini. Penjelasan untuk word embedding yang tidak digunakan (POS tag) tidak akan dijelaskan dalam bab ini. Pada akhir subbab ini akan dijelaskan representasi kata, bentuk akhir yang dihasilkan dari proses encoder nanti.

4.1.1.1 BERT

Bidirectional Encoder Representations from Transformers (BERT) adalah metode yang telah diteliti dan *publish* oleh Google AI Language[2]. Metode BERT ini memiliki tujuan yaitu untuk menunjukkan pentingnya *bidirectional* pre-training dalam sebuah language model, karena pada saat itu metode Long Short Term Memory (LSTM) tidak dapat dinyatakan sebagai bidirectional (meskipun untuk BiLSTM juga tidak disebut mengambil informasi secara bidirectional. Karena ini, BERT dibuat untuk mendemonstrasikan hal tersebut. Secara struktur, BERT mengambil semua bagian dari encoder Transformers tanpa decodernya sekali pun. Memang Transformers mempunyai guna pada task yang paling cocok yaitu translasi bahasa. Namun untuk BERT, atau tumpukkan dari layer encoder dari Transformers, dapat memberikan solusi untuk beberapa pekerjaan seperti translasi, tanya jawaban, juga bisa analisa sentimen. Hal-hal ini hanya dapat dilakukan dengan BERT memahami informasi mengenai cara bahasa tersebut bekerja.

[2] Jacob Devlin, dkk, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, In Proceedings of NAACL 2019, pages 4171–4186, Minneapolis, Minnesota, June 2019.

BERT akan dibagi menjadi dua tahap training, pretrain BERT untuk mengenal bahasa tersebut terlebih dahulu. Kemudian fine tune BERT untuk mengarahkan pemikiran solusi BERT tersebut kepada tugas yang ditujukan. Penjelasan berikut adalah mengenai proses yang dilewatkan untuk BERT.



Gambar 4.2
Visualisasi Arsitektur BERT QnA

Bagian pertama adalah pre-training (Gambar 4.2 kotak kiri), pada gambar akan dijelaskan setiap arti dari kotak yang ada terlebih dahulu. Gambar alur dari pre-training berawal dari bawah menuju ke atas, dengan *unlabeled sentence A and B pair* adalah input dari kalimat A dan B yang kemudian akan diberikan token mask (hal ini akan dibahas dipararaf berikutnya), menghasilkan *Masked Sentence A* dan *Mask Sentence B*. Kalimat tersebut akan dibagi menjadi token yang word-level (kotak merah) yang selanjutnya akan dibuatkan representasi embedding (kotak kuning) untuk tiap token kata tersebut. Embedding akan dimasukkan ke dalam arsitektur BERT dan dari BERT menghasilkan output (kotak hijau). Output akan terdiri dari vektor yang disebut *word vector* dan juga satu output biner (kotak hijau paling kiri label C) untuk mengindikasi apakah kalimat B berhubungan dengan kalimat A. Semua penjelasan mengenai tiap bagian akan dijelaskan pada paragraf berikut ini.

Bagian pertama (pre-training) ini mengajar model untuk mengenal bahasa apa yang dipelajari, apa itu bahasa dan apakah bentuk konteks dalam bahasa tersebut. Cara untuk mencapai pengertian tentang bahasa dan konteks dengan baik adalah dengan dua cara, Masked Model Language (MLM) dan Next Sentence

Prediction (NSP). MLM diaplikasikan pada alur pada input kalimat (Masked Sentence A dan Masked Sentence A) dan NSP pada output nanti (output pada kotak pertama label C).

MLM adalah cara komputer belajar bentuk dari kalimat sebuah bahasa menggunakan token *mask* yang akan digantikan beberapa kata dari suatu kalimat. Dari penutupan kata ini dan juga menebak kata apa yang sesuai, BERT dan memahami apa yang wajar untuk mengisi dengan cara melihat konteks juga dalam satu kalimat. Contoh yang dapat dimengerti untuk metode MLM adalah jika sebuah kalimat awalnya "The [MASK1] brown fox [MASK1] over the lazy dog. Kemudian cara satunya adalah NSP yang dapat dimengerti dari namanya sendiri, yaitu cara belajar komputer untuk mengerti kalimat yang dapat dihubungkan dengan kalimat lain. Hal ini juga dapat mengajar BERT untuk melihat konteks tiap kalimat untuk menyatukan mereka yang berhubungan. Contohnya, kalimat pertama adalah "Kampus ISTTS sangat ramai". Kalimat keduanya, "Gedungnya sangat tinggi". Dan kalau komputer yang sedang memelajari, artinya jawaban dari NSP yang benar seharusnya "benar kalimat pertama berhubungan dengan kalimat kedua".

Setelah mengenal bentuk dari bahasa yang akan dipelajari, BERT perlu di fine tune agar sesuai dengan tujuan dari pembuatan model tersebut. Contoh dari penjelasan ini adalah untuk membuat model yang dapat melakukan tanya jawab dengan baik. Karena tujuan dari model harus menjawab dari sebuah pertanyaan, maka harus diberikan *supervised* dataset yang berisikan pertanyaan dan juga jawaban benar terhadap pertanyaan tersebut. Keunggulan BERT muncul di bagian ini. BERT yang telah pretrained telah mengenal bahasa tersebut, sehingga yang perlu diubah adalah beberapa parameter di dalam BERT tersebut, dan juga jenis output dari BERT harus diubah sesuai dengan yang diinginkan (yaitu output kata-kata). Dengan ini membuktikan BERT adalah metode yang dapat digunakan untuk memahami bahasa dan dengan cepat di fine tune untuk mendapatkan model yang bertujuan untuk suatu task dengan fine tune yang hanya mengubah beberapa dari isi BERT. Cara fine-tune dari arsitektur pre-training adalah input diubah, kalimat pertama adalah pertanyaan dan kalimat kedua adalah jawaban (Gambar 4.2 kotak

kanan/fine-tune, bagian Question dan Answer). Output juga diubah dari awalnya output untuk menebak kata yang di berikan mask, menjadi prediksi yang menjawab pertanyaan dalam bentuk jangkauan kata-kata dari kalimat jawaban (start/end span).

Yang belum dibahas pada bagian ini adalah apakah bentuk embedding dari input kalimat menuju BERT? Embedding terakhir untuk input direpresentasikan dengan gabungan antara tiga jenis embeddings yaitu token embeddings, segment embeddings, position embeddings. Token embeddings adalah token yang sudah dilatih sebelumnya (pretrained) dari metode yang ditentukan paper yaitu (WordPiece[3]). Segment embedding adalah penanda untuk tiap token, bahwa token itu miliki kalimat apa (kalimat pertama atau kedua/ kalimat pertanyaan atau kalimat jawaban). Dan position embeddings diperlukan untuk menandakan token tersebut kalimat ke berapa, mengingat bahwa seluruh proses training akan berlangsung secara bersamaan.

Hal belum dibahas juga adalah konversi word vector dan juga penghitungan loss dari prediksi word vector tersebut. Cara untuk mendapatkan hasil output dalam kata-kata adalah word vector akan dilewatkan kepada sebuah softmax layer dengan 30.000 neuron (jumlah dari neuron bergantung pada jumlah kata dalam kosakata yang digunakan, dalam kasus ini WordPiece memiliki 30.00 kata). Softmax akan memberikan hasil akhir kata yang diprediksikan, kata-kata ini akan dibandingkan dengan kata sebenarnya yang akan direpresentasikan dalam bentuk *one hot encoded* dengan panjang 30.000 (sesuai jumlah kosakata).

Tugas akhir ini tidak akan menggunakan BERT namun BERT versi bahasa Indonesia yaitu IndoBERT[4]. Pembuatan IndoBERT oleh IndoLEM (Indonesia Language Evaluation Montage) menggunakan konfigurasi yang sama dengan BERT yang telah dijelaskan di atas (BERT base uncased). Isi dari IndoBERT adalah 12 hidden layer dengan jumlah 768, attention heads berjumlah 12, and feed-forward layer sejumlah 3.072. Dan untuk word embedding yang digunakan juga

[3]Yonghui Wu, dkk, Google's neural machine translation system: Bridging the gap between human and machine translation, (2016).
[4] Fajri Koto, dkk, IndoLEM and IndoBERT: A Benchmark Dataset and Pre-trained Language Model for Indonesian NLP, (2020).

berasal dari WordPiece tapi versi Indonesia berukuran 31.293. IndoBERT telah dilewatkan berbagai sumber data yaitu Wikipedia Indonesia (74 juta kata), berita dari Kompas[5], Tempo[6], Liputan6[7] (55 juta kata) dan Indonesian Web Corpus (90 juta kata).

4.1.1.2 Word2Vec

Awalnya untuk metode Sequence-to-Set, embedding yang kedua secara tertulis bukan Word2Vec yang digunakan namun GLoVE. Namun, penelitian ini juga menyediakan opsional untuk menggunakan representasi vektor antara GLoVE atau Word2Vec. Tugas akhir ini akan menggunakan Word2Vec yang disediakan oleh Dr. Ir. Joan Santoso dengan data bersumber dari Wikipedia Indonesia.

Tujuan adanya Word2Vec bermula karena pada saat penelitian Word2Vec, belum ada teknik yang cukup efisien untuk pengolahan representasi kata yang dapat menerima data yang berlebihan besarnya (sekitar ratusan juta kata). Word2Vec selain memberikan nilai kesamaan antar kata, teknik ini juga memberikan nilai kesamaan yang lebih dari satu (karena selain kemiripan, suatu kata bergantung pada konteks dapat memiliki arti kata lebih dari satu). Dalam penelitian ini juga ditemukan arti dari sebuah kata dapat berarti di luar cara sintaksis, tetapi juga dapat direpresentasikan dari nilai word vector secara hitungan. Contohnya vektor untuk kata “raja” - vektor untuk kata “laki-laki” + vektor untuk kata “perempuan” dapat mengarah pada word vector untuk kata “ratu”.

Word2Vec terinspirasi dari dua jenis arsitektur yaitu Feedforward Neural Net Language Model (NNLM)[8] dan Recurrent Neural Net Language Model (RNNLM). NNLM terdiri dari layer input, *projection*, hidden and output. Secara alur, input kata-kata akan melewati proses encoding 1-of-V (V adalah panjang kosakata). Kemudian diproyeksikan (perkalian matriks seperti regular/dense/linear layer tetapi tanpa fungsi aktivasi diakhir layer seperti sigmoid/tanh) kepada layer

[5] Kompas, https://kompas.com
[6] Koran TEMPO, https://koran.tempo.co
[7] Liputan6, https://liputan6.com
[8] Bengio Y, Ducharme R., Vincent P., A Neural Probabilistic Language Model, Journal of Machine Learning Research, 3: hal.1137-1155, (2003).

projection (P) dengan dimensi *N x D*. Yang menjadi kekurangan dalam NNLM adalah nilai komputasi yang sangat tinggai akibat nilai dari layer proyeksi dapat memberikan matriks yang *dense* (matriks yang *non-zero*, matriks yang setidaknya satu yang tidak bernilai nol). $Q = N \times D + N \times D \times H + H \times V$ adalah penilaian untuk komputasi NNLM, perlu diperhatikan bagian $N \times D \times H$ adalah yang menitikberatkan nilai komputasi yang besar karena nilai proyeksi (*N x D*) dan nilai layer hidden (*H*) yang besar (layer proyeksi bisa 500 sampai 2000, dan hidden bisa 500 sampai 1000). Untuk meminimalkan nilai komputasi Word2Vec mengusulkan representasi kosakata diubahkan menggunakan *hierarchical softmax* dengan representasi Huffman binary (memberikan kode biner singkat kepada kata yang sering muncul) yang bisa mengurangi jumlah unit output yang perlu dievaluasikan. Terlepas dari cara baru ini bukan percepatan yang penting, Word2Vec akan mengusulkan juga arsitektur tanpa layer hidden yang kemudian akan membuat titik efisiensi berada pada normalisasi softmax.

Perbedaan NNLM dan RNNLM adalah penggunaan RNN yang hanya terdiri dari input, hidden dan output, tidak menggunakan layer proyeksi. RNN dikenal untuk informasi *short term* yang dapat disimpan. Artinya bahwa informasi dalam satu kalimat yang panjang atau kalimat sebelumnya dapat disimpan oleh RNN, membantu memberikan konteks lebih banyak. Juga komputasi yang diberikan oleh RNN adalah $Q = H \times H + H \times V$.

Peneliti Word2Vec menyatakan dari penelitian sebelum-sebelumnya menemukan kesimpulan bahwa NNLM dapat memberikan hasil training yang bagus dengan dua tahap yaitu *continuous* word vector dipelajari dengan model yang sederhana, kemudian dilakukan training N-gram NNLM diatas representasi kata-kata. Kedua tahap itu dapat dilakukan dengan arsitektur Continuous Bag-of-Words Model (C-BOW) dan Continuous Skip-gram Model (Skip-Gram). CBOW memiliki arsitektur yang mirip dengan NNLM namun layer proyeksi diberikan kepada semua kata-kata tidak hanya salah satu (menghasilkan nilai vektor rata-rata). Metode yang digunakan dalam C-BOW juga adalah melihat kata-kata di masa depan (di depan). Menghasilkan nilai komputasi $Q = N \times D + D \times log2(V)$. Berbeda dengan bag-of-words standar, C-BOW menggunakan representasi konteks yang terdistribusi

secara terus menerus. Perlu diketahui juga, matriks weight antara input dan layer poroyeksi juga digunakan untuk semua posisi kata sama dengan cara NNLM.

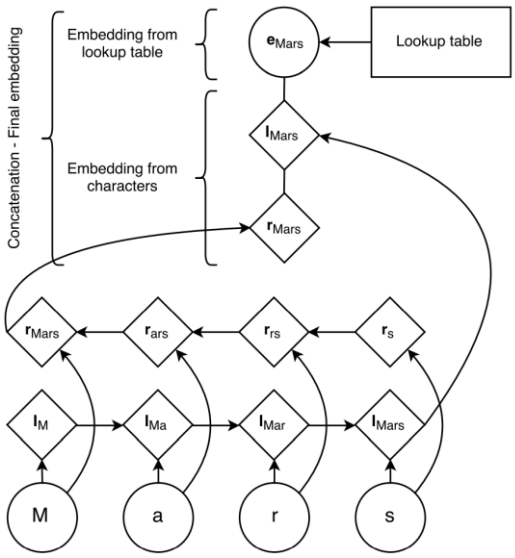
Arsitektur Skip-Gram memiliki kemiripan juga dengan C-BOW, tetapi tujuan mereka berkebalikan. Di mana C-BOW bertujuan untuk menebak suatu kata berdasarkan konteks kata sekitarnya, Skip-Gram berusaha semaksimal mungkin untuk klasifikasi kata berdasarkan suatu kata lain dalam satu kalimat yang sama. Skip-Gram menggunakan setiap *current word* (kata yang difokuskan) sebagai input ke pengklasifikasi log-linear dengan *continuous projection layer*, dan memprediksi kata-kata dalam jangkauan tertentu sebelum dan sesudah center word tersebut. Ditemukan juga ternyata jika jangkauan melihat sebelum dan sesusah kata tersebut dapat menaikkan kualitas hasil dari word vector, perlu diperhatikan juga bahwa hal ini juga dapat menaikkan nilai komputasi. Namun ada juga pernyataan bahwa lebih banyak kata-kata yang jauh dari center word tidak memiliki relasi yang sedikit (secara konteks). Karena ini, nilai/weight untuk kata yang jauh (*distant words*) akan diperkecil. Untuk Skip-Gram akan memiliki nilai komputasi $Q = C \times (D + D \times log2(V))$, di mana C adalah jarak maksimal kata-kata sekitarnya.

4.1.1.3 Character-level BiLSTM

Character-level BiLSTM adalah metode embedding yang diambil dari penelitian mengenai Neural Architectures for Named Entity Recognition[9]yang menginginkan input nya memiliki representasi yang sensitif terhadap penulisan karakter. Input word embedding dari penelitian tersebut tidak hanya memiliki representasi character-level, tetapi juga menggunakan pretrained embeddings dan juga menambahkan layer dropout untuk mendorong model untuk bergantung pada kedua representasi tersebut. Meskipun terdapat tiga proses pembuatan embedding dari penelitian tersebut, Sequence-to-Set Network hanya menggunakan metode character-level embedding, namun kedua tahap berikutnya (pretrained embedding dan layer dropout) akan tetap dibahas.

[9] Guillaume Lample, dkk, Neural Architectures for Named Entity Recognition. In Proceedings of NAACL 2016, hal. 260–270, (2016).

Penggunaan embedding secara character-level memiliki beberapa keuntungan, seperti nilai kegunaan yang tinggi terhadap beberapa tugas NLP seperti bahasa yang secara morfologi cukup beragam, tugas seperti POS Tagging dan language modelling juga untuk dependency parsing yang memiliki kesulitan dengan kata-kata yang tidak dikenal (biasa disebut out-of-vocabulary/OOV). Gambar 4.3 adalah penggambaran arsitektur dari character-level embedding untuk kata "Mars". Pada awal pembuatan embedding akan diinisialisasi dengan nilai random yaitu *lookup table* embedding untuk tiap karakter. Embedding untuk sebuah kata yang telah dipecah menjadi tiap karakter adalah gabungan dari representasi *forward* (depan dari karakter) dan *backward* (arah ke belakang karakter) dari BiLSTM. Representasi kata kemudian akan digabungkan dengan representasi/embedding word-level nya dari sebuah lookup table word-level. Jika kata tersebut tidak memiliki word embedding, maka akan diberikan embedding token kata yang tidak kenal yang biasanya ditandakan UNK (token UNK kan diberikan probabilitas 0,5). Panjang dimensi hidden untuk character-level biLSTM adalah 25 untuk tiap arah, berarti 50 secara total (nilai ini juga menjadi nilai *default* untuk metode Sequence-to-Set Network).



Gambar 4.3
Arsitektur Character-Level Embedding[10]

[10] Ibid

Tahap berikutnya adalah menyediakan lookup table word-level dari sebuah pretrained model. Model ini adalah skip-n-gram, sebuah variasi dari Word2Vec dengan memerhatikan penulisan kata secara urutan. Untuk mendapatkan skip-n-gram tentu dilakukan fine tune kepada embedding saat proses training. Penentuan menggunakan word embedding lookup table yang pretrained adalah karena hasil penelitian mengalami peningkatan yang baik dibandingkan dengan word embedding yang random. Dimensi untuk embedding ini adalah 100 untuk bahasa Inggris dan 64 untuk yang lainnya (embedding ini dicoba dalam bahasa Spanyol). Dan minimum frekuensi kata adalah empat kali dan *window size* (jangkauan melihat berapa kata sebelum dan sesudah focus word) adalah delapan.

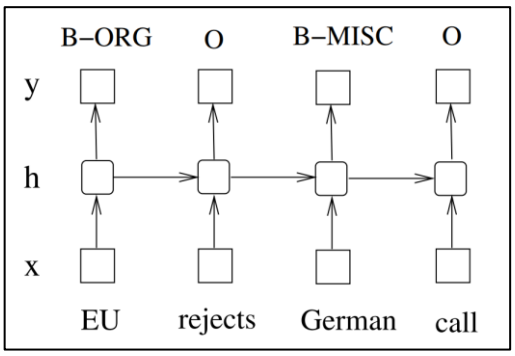
Dan terakhir adalah layer dropout yang ditambahkan di akhir embedding, sebelum input memasuki arsitektur sebenarnya. Dropout diberikan untuk menghindari model lebih bergantung pada salah satu representasi. Dengan dropout dialami peningkatan yang bagus juga dalam penelitian tersebut, karena model fokus terhadap kedua representasi, tidak salah satu saja.

4.1.1.4 BiLSTM

Bidirectional LSTM[11] (BiLSTM) sendiri akan digunakan sebagai representasi akhir setelah ketiga jenis word embedding diatas digabungkan. Representasi akhir secara detail akan dibahas di subbab encoded tokens, untuk subbab ini akan fokus jalan kerjanya BiLSTM sendiri. Arsitektur BiLSTM terinspirasi dari Long Short Term Memory (LSTM) yang juga terinspirasi dari Recurrent Neural Network (RNN). RNN menjadi contoh dari LSTM karena RNN sebelumnya terbukti memberikan performa yang bagus terhadap tugas language model dan *speech recognition* karena cara penyimpinan informasi (bisa juga disebut sebagai fitur dari kata) waktu lampau, dan informasi ini dapat disimpan dalam jangka waktu yang lama. Gambar 4.4 menunjukkan struktur dari isi RNN

[11] Zhiheng Huang , Wei Xu, Kai Yu, Bidirectional LSTM-CRF Models for Sequence Tagging, (2015).

dengan input kata-kata “EU rejects German call” dan tiga layer diatasnya yaitu input (*x*), hidden (*h*), dan output (*y*).



Gambar 4.4
Arsitektur Recurrent Neural Network (RNN)[12]

Contoh dari Gambar 4.4 adalah contoh RNN dengan tugas memberikan tag entitas dalam bentuk IOB. Layer input merepresentasikan fitur kata input dalam bentuk one-hot-encoding, vektor dense, atau vektor sparse (vektor dengan isi sebagian besar bernilai nol). Dan layer input merepresentasikan fitur (informasi) pada saat itu/time (dengan anotasi *t*), untuk layer output akan merepresentasikan probabilitas distribusi tag entitas pada time ke *t*. RNN memberikan koneksi antara hidden state sebelumnya dengan hidden state saat ini, karena ini informasi lama dapat selalu dilihat oleh RNN. Untuk nilai dari tiap layer dapat dilihat pada rumus $\mathrm{h}(t) = f(\mathrm{Ux}(t) + \mathrm{Wh}(t-1))$ (4.1) dan rumus $\mathrm{y}(t) = g(\mathrm{Vh}(t))$ (4.2), dimana *f* dan *g* adalah fungsi aktivasi *sigmoid* (rumus $f(z) = \frac{1}{1+e^{-z}}$ (4.3) dan *softmax* (rumus $g(z_m) = \frac{e^{zm}}{\sum_k e^{z_k}}$ (4.4)

dan nilai U, W, V adalah koneksi weight yang melewatkan training.

$$\mathrm{h}(t) = f(\mathrm{Ux}(t) + \mathrm{Wh}(t-1)) \quad \text{..... (4.1)}$$

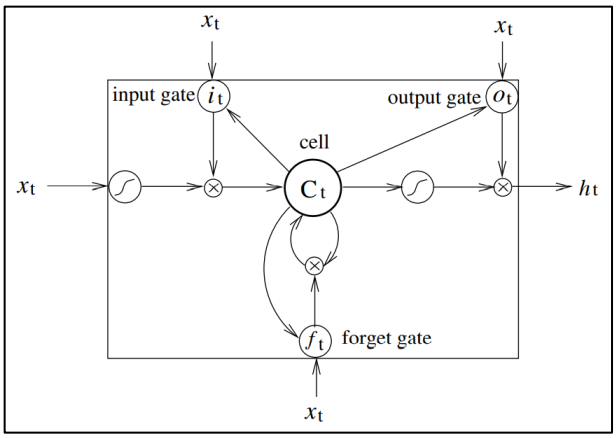
$$\mathrm{y}(t) = g(\mathrm{Vh}(t)) \quad \text{..... (4.2)}$$

$$f(z) = \frac{1}{1+e^{-z}} \quad \text{..... (4.3)}$$

[12] Ibid

$$g(z_m) = \frac{e^{z_m}}{\sum_k e^{z_k}} \quad \text{..... (4.4)}$$

RNN memang memiliki kemampuan menyimpan informasi berjangka lama, namun RNN memiliki permasalah *vanishing gradient*. Selain keunggulan dalam komputasi, LSTM dapat menyelesaikan permasalahan vanishing gradient dari RNN. LSTM memiliki arsitektur yang sama dengan RNN dengan perbedaan hidden layer diubah dengan sebuah sel yang dinamakan *memory cell*. Dengan bantuan memory cell, LSTM mendapat sifat yang lebih bagus dalam mencari ketergantungan jarak jauh dalam data. Isi dari sebuah memory cell dapat dilihat pada Gambar 4.5.



Gambar 4.5
(Satu) Memory Cell LSTM[13]

Implementasi dari memory cell LSTM menggunakan rumus $i_t = \sigma(W_{xi}x_t + W_{hi}h_t - 1 + W_{ci}c_t - 1 + b_i$ .....
(4.5), $f_t = \sigma(W_{xf}x_t + W_{hf}h_t - 1 + W_{cf}c_t - 1 + b_f$.....
(4.6), $c_t = \sigma(W_{xc}x_t + W_{hc}h_t - 1 + W_{hc}c_t - 1 + b_c$ .....
(4.7), dan $h_t = o_t \tanh(c_t)$.....
(4.8). Dimana σ sebagai fungsi logistik sigmoid, kemudian *i*, *f*, *o* dan *c* adalah gate input, gate forget, gate output dan vektor sell. Setiap dari alur gate memiliki weight matriks dan berguna sesuai dengan namanya masing-masing.

[13] Ibid

Contohnya W_{hi} adalah weight matriks untuk hidden-input gate, W_{xo} untuk weight matriks gate input-output.

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_t - 1 + W_{ci}c_t - 1 + b_i \quad \text{.....(4.5)}$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_t - 1 + W_{cf}c_t - 1 + b_f \quad \text{.....(4.6)}$$

$$c_t = \sigma(W_{xc}x_t + W_{hc}h_t - 1 + W_{hc}c_t - 1 + b_c \quad \text{.....(4.7)}$$

$$h_t = o_t \tanh(c_t) \quad \text{.....(4.8)}$$

Penelitian yang dirujuk untuk menggunakan BiLSTM[14], memiliki fokus untuk penelitian berbagaim macam LSTM dalam tugas *sequence tagging*. Dan dalam tugas tersebut memberi akses untuk mengetahui fitur (informasi) lampau dan masa depan dalam waktu tertentu. Karena hal ini, dapat menggunakan BiLSTM untuk menggunakan fitur lampau kepada forward hidden state dan fitur masa depan kepada backward hidden state. BiLSTM menggunakan cara training back-propagation through time (BPTT)[15]. Backpropagation tetap dilakukan seperti biasanya disebuah network, namun setiap nilai hidden states dapat dilihat dari semua timestep (*t*) yang telah dilewatkan. Dan khusus untuk implementasi BiLSTM pada sequence tagging, akan dilakukan forward dan backward pass untuk satu kalimat utuh juga reset hidden state menjadi nol dilakukan disetiap awal tiap kalimat. Dan proses training kalimat akan dilakukan beberapa secara bersamaan karena implementasi training dalam bentuk *batch* (sekumpulan/sekelompok).

4.1.1.5 Encoded Tokens

Serangkaian input yang diberikan akan dilewatkan akan dibuatkan representasi token ke-i (dilambangkan sebagai x_i). Hasil dari x_i adalah penggabungan (*concatenation*) dari embedding kontekstual oleh BERT (dilambangkan sebagai ${x_i}^{bert}$), embedding Word2Vec (dilambangkan sebagai ${x_i}^{w2v}$), embedding character-level (dilambangkan sebagai ${x_i}^{char}$). Perlu diingat

[14] Ibid
[15] Mikael Boden, A Guide To Recurrent Neural Networks And Backpropagation, In the Dallas Project, (2002).

bahwa Sequence-to-Set Network sebenarnya menggunakan embedding GLoVe dan embedding POS Tag, namun penelitian tugas akhir ini memberi batasan POS Tag tidak diberikan dan GLoVE digantikan dengan embedding Word2Vec. Ketiga jenis embedding telah dijelaskan pada subbab sebelumnya dan juga dari penelitian yang telah dirujukkan. Kemudian dari semua representasi kata, akan berlaku sebagai input kepada BiLSTM (subbab 4.1.1.4) untuk mendapatkan serangkaian representasi yang final (dilambangkan $\mathrm{H} \in \mathbb{R}^{l\,x\,d}$). Penulisan matematis untuk menghasilkan representasi token dapat dilihat pada rumus $x_i = x_i{}^{bert} \oplus x_i{}^{w2v} \oplus x_i{}^{char}$ (4.9, $\overrightarrow{\mathrm{H}}_i = LSTM_f(x_i; \overrightarrow{\mathrm{H}}_{i-1}; \theta_f)$ (4.10,$\overleftarrow{\mathrm{H}}_i = LSTM_b(x_i; \overleftarrow{H}_{i+1}; \theta_b)$ (4.11, dan $\mathrm{H}_i = \overrightarrow{\mathrm{H}}_i \oplus \overleftarrow{\mathrm{H}}_i$ (4.12. Dimana *l* adalah panjang dari input sequence, *d* hidden size dari LSTM, $\oplus$ adalah lambang dari proses concatenation/penggabungan. Kemudian θ_f dan θ_b parameter untuk forward dan backward LSTM. $\overrightarrow{\mathrm{H}}_i$ dan $\overleftarrow{\mathrm{H}}_i$ adalah hidden state pada posisi ke *i* dari forward dan backward LSTM.

$$x_i = x_i{}^{bert} \oplus x_i{}^{w2v} \oplus x_i{}^{char} \quad \text{..... (4.9)}$$

$$\overrightarrow{\mathrm{H}}_i = LSTM_f(x_i; \overrightarrow{\mathrm{H}}_{i-1}; \theta_f) \quad \text{..... (4.10)}$$

$$\overleftarrow{\mathrm{H}}_i = LSTM_b(x_i; \overleftarrow{H}_{i+1}; \theta_b) \quad \text{..... (4.11)}$$

$$\mathrm{H}_i = \overrightarrow{\mathrm{H}}_i \oplus \overleftarrow{\mathrm{H}}_i \quad \text{..... (4.12)}$$

4.1.2 Entity Set Decoder

Entity Set Decoder adalah bagian berikutnya dalam Sequence-to-Set Network yang bertugas untuk melakukan prediksi dari kata-kata yang diberikan dari encoder dengan informasi dan kata-kata yang perlu difokuskan oleh decoder. Subbab ini akan dibagi menjadi dua subbab kecil sesuai dengan isinya Entity Set Decoder yaitu layer decoder dan *classification*. Dengan decoder terdiri dari tiga layer, yaitu self-attention, cross-attention, feed forward network. Untuk layer classification akan dijelaskan mengenai tiga Multilayer Perceptron yang digunakan untuk mengkalisifikasi output decoder layer menjadi prediksi akhirnya yaitu

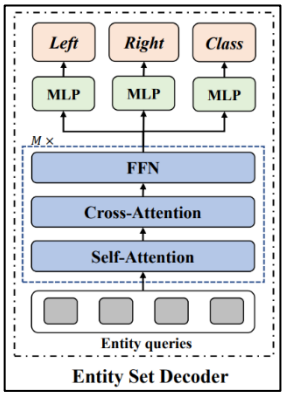
batasan awal dan akhir kata yang akan dilabel dan juga jenis label/entitas yang diberikan.

4.1.2.1 Decoder Layer

Bagian decoder dari Sequence-to-Set Network mengambil inspirasi dari Transformers, sehingga menggunakan self-attention dan cross-attention untuk mengubah sejumlah *N* entity queries. Namun ada perbedaan dengan layer decoder dari Transformers yaitu decoder untuk Sequence-to-Set Network dibentuk dengan sifat *non-autoregressive*. Sifat ini yang membentuk model untuk mendapatkan prediksi sebanyak *N* dalam sekali iterasi.

Sifat autoregressive ini didapatkan dari layer decoder Transformers sendiri dari satu layer yang dimiliki Transformer yaitu Masked Multi-Head Attention. Gunanya layer tersebut adalah untuk memberi mask kepada token berikutnya yang akan diprediksi. Contoh dalam kasus translasi bahasa Indonesia menuju bahasa Inggris dengan kalimat “Saya mencintai anda”. Token bahasa Inggris yang perlu diprediksikan adalah “I”, “love”, “you”. Misalkan pada saat ini didalam decoder Transformers telah memprediksikan sampai token kedua / “love”, artinya pada kasus ini yang diberikan mask adalah token ketiga. Hal ini dilakukan agar layer attention yang berada di decoder Transformers akan menekankan fokusnya kepada kedua token pertama (“I”, “love”).

Karena sifat dari decoder Sequence-to-Set Network adalah non-autoregressive, maka tidak akan menggunakan layer Masked Multi-Head Attention. Alasan juga diberikan sifat tersebut adalah keperluan informasi semantik kontekstual lengkap. Meskipun layer Masked Multi-Head Attention tidak dimasukkan dalam decoder ini, Sequence-to-Set Network tetap menggunakan layer-layer umum dari Transformers yaitu self-attention dan cross-attention bahkan juga Feed Forward Network. Dan layer decoder ini dapat ditentukan berapa jumlah tumpukkan layer decoder yang diinginkan (dilambangkan dengan *M*).



Gambar 4.6
Arsitektur Bagian Entity Set Decoder[16]

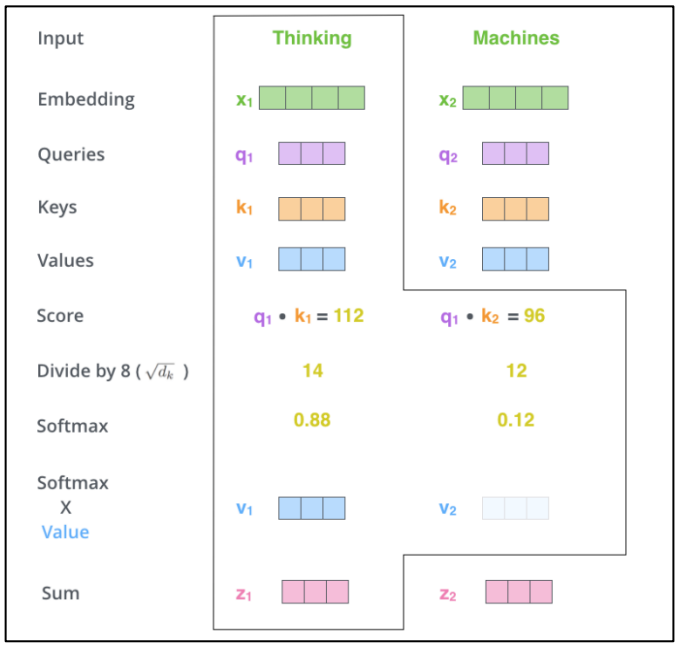
Layer self-attention adalah layer pertama dari bagian encoder. Layer ini berguna untuk memberi fokus antar entitas dan mengambil nilai juga hubungan ketergantungan antar entitas. Dalam Sequence-to-Set Network, layer self-attention akan dibilang sebagai multi-head attention (sama dengan Transformers yang mengenalkan multi-head attention setelah self-attention dijelaskan. Secara singkat, multi-head attention adalah penggabungan dari penghitungan self-attention (dapat dipanggil sebagai attention head). Namun untuk mempermudah penjelasan, akan dijelaskan self-attention terlebih dahulu, kemudian dilanjutkan dengan multi-head attention.

Penggambaran yang mudah untuk ilmu self-attention adalah dari contoh kata "Kucing Andi tidak menyukai permen, ia menyukai susu". Dari kalimat tersebut, model dapat memahami bahwa subjek "ia" adalah "Kucing Andi". Untuk manusia tentu gampang untuk memahaminya, namun untuk komputer akan lebih susah. Cara self-attention mendapat informasi tersebut adalah, saat kata "ia" sedang diproses, self-attention memberi kemampuan model untuk melihat kata-kata disekitar sequence kalimat tersebut untuk mencari informasi tersebut.

Penjelasan penghitungan self-attention akan dijelaskan dalam bentuk vektor (implementasi aslinya menggunakan matriks). Tahap pertama untuk melakukan proses self-attention adalah kebutuhan tiga vektor bernama Query, Key dan Value. Ketiga vektor ini diinisialisasi sama dengan weight pada neural network biasa, dengan nilai random. Vektor-vektor ini akan berguna untuk menghitung *score* tiap

[16] Zeqi Tan, dkk, A Sequence-to-Set Network for Nested Named Entity Recognition, (2021).

kata, score merupakan nilai fokus yang perlu diberikan kepada kata tersebut menurut self-attention. Penghitungan score adalah dengan dot matriks antar vektor Query dengan vektor Key yang sedang dinilai. Contohnya jika sedang menghitung score kata pertama, maka score pertama adalah q_1 dikalikan dengan k_1. Untuk score kedua maka perkalian dilakukan antara q_1 dan k_2. Tiap dari score akan dibagi akar pangkat dimensi vektor didalam penelitian (d_k). Contoh jika dimensi vektor adalah 64 maka tiap score akan dibagi 8, dan nilai baru tersebut dimasukkan kedalam fungsi softmax. Nilai dari softmax akan dikalikan dengan vektor Key dan langkah terakhir adalah menjumlahkan seluruh nilai vektor terbaru, ini menghasilkan output self-attention untuk posisi kata tersebut (dilambangkan vektor *z*). Gambar 4.7 dapat membantu pemahaman alur dari penghitungan self-attention.



Gambar 4.7
Alur Kalkulasi Self-Attention[17]

Dari self-attention, dikembangkan menjadi multi-head attention dimana keunggulannya adalah attention dapat disebarkan lebih luas pada posisi kata berbeda-beda. Tiap "head" akan memiliki matriks query, key dan value sendiri.

[17] Jay Alammar, The Illustrated Transformer, (https://jalammar.github.io/illustrated-transformer/).

Pada Transformers (dan tugas akhir ini) akan menggunakan delapan attention-head sehingga terdapat delapan layer encoder dan decoder juga delapan matriks query, key dan value. Cara penghitungan multi-head attention sama dengan self-attention tetapi dilakukan delapan kali dengan 8 matriks berbedanya menghasilkan delapan matriks *z* yang berbeda-beda. Kedepalan matriks agar dapat diterima oleh layer-layer berikutnya perlu digabung, kemudian hasil dari matriks yang digabung perlu dikalikan dengan matriks W^o (matriks ini telah melewatkan training bersama dengan modelnya). Hasil akhir ini yang akan diberikan kepada layer berikutnya yaitu cross-attention. Pada tugas akhir ini, ditentukan nilai dari query, key dan value akan sama dengan nilai entity queries ($\mathrm{Q} = \mathrm{K} = \mathrm{V} = \mathrm{Q}_{\mathrm{span}}$).

Layer cross-attention merupakan layer berikutnya setelah self-attention, layer ini dapat secara efektif mengambil informasi konstekstual dari kalimatnya/tokennya. Cross-attention juga akan dihitung secara multi-head attention, tetapi cara penghitungan cross-attention tidak ada perbedaanya dengan self-attention, yang berbeda adalah nilai dari matriks Query, Value dan Key. Dalam Transformers, cross-attention akan menerima matriks Query dan Key dari output encoder paling terakhir (berbeda dengan self-attention yang matriksnya akan dibuat bukan diterima dari layer lain). Nilai dari query, key dan value untuk cross-attention adalah query akan sama dengan nilai entity queries ($\mathrm{Q} = \mathrm{Q}_{\mathrm{span}}$). Dan nilai key dan value sama dengan representasi token dari encoder ($\mathrm{K} = \mathrm{V} = \mathrm{H}$). Secara keseluruhan rumus untuk self, cross, dan multi-head dapat ditulis seperti rumus $\mathrm{Attention}(\mathrm{Q}, \mathrm{K}, \mathrm{V}) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)\mathrm{V}$ (4.13), 4.14, dan $R = \mathrm{Concat}(\mathrm{head}_1, \ldots, \mathrm{head}_h)W^O$ (4.15). Dengan $\mathrm{W}_i^Q, \mathrm{W}_i^K, \mathrm{W}_i^V$ dan W^O sebagai nilai proyeksi yang dapat diganti (*trainable*) dan R sebagai nilai akhir oleh cross-attention yang akan menjadi input untuk layer Feed Forward Network (FFN). $\mathrm{head}_i = \mathrm{Attention}\left(\mathrm{QW}_i^Q, \mathrm{KW}_i^K, \mathrm{VW}_i^V\right)$ (4.14)

$$\mathrm{Attention}(\mathrm{Q}, \mathrm{K}, \mathrm{V}) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)\mathrm{V} \quad \ldots\ldots\ldots\ldots\ldots\ldots (4.13)$$

$$\text{head}_i = \text{Attention}\left(\text{QW}_i^Q, \text{KW}_i^K, \text{VW}_i^V\right) \quad \text{..... (4.14)}$$

$$R = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O \quad \text{..... (4.15)}$$

Layer FFN terdiri dari 3 buah layer perceptron dengan fungsi aktivasi ReLU dan layer proyeksi linear. Layer ini akan menghasilkan embedding final (dilambangkan sebagai $U \in \mathbb{R}^{Nxd}$), embedding ini memiliki label tambahan ($\emptyset$) untuk membantu menandakan kata yang memang tidak memiliki entitas karena panjang dari hasil prediksi akan tetap dengan jumlah *N* entitas, dimana nilai *N* akan ditentukan lebih besar dari pada jumlah entitas yang ada dari sequence nya.

4.1.2.2 Classification Layer

Seperti namanya, layer ini bertugas untuk mengubah prediksi dari layer decoder dan diklasifikasikan sesuai dengan prediksinya. *Classifier* yang digunakan adalah Multi Layer Perceptron (MLP) berjumlah tiga, satu classifier untuk tiap prediksi yang perlu klasifikasikan (batasan kiri, batasan kanan dan jenis label/entitas). Tiap MLP akan juga memiliki fungsi aktivasi softmax didalamnya dan hasil dari tiap MLP adalah probabilitas klasifikasi (p^c, p^l, p^r) untuk tiap jenis label/entitas sendirim batasan kiri, dan batasan kanan. Perumusan untuk klasfikasi dapat dilihat pada rumus $p^c = \text{MLP}_c(u)$ (4.16), $\text{H}_{fuse} = dup(u, l) \oplus \text{H}$ (4.17), $p^l = \text{MLP}_l\left(\text{H}_{fuse}\right)$ (4.18), dan $p^r = \text{MLP}_r\left(\text{H}_{fuse}\right)$ (4.19), dengan *u* adalah entity query ($u \in \mathbb{R}^d$) dari U.

$$p^c = \text{MLP}_c(u) \quad \text{..... (4.16)}$$

$$\text{H}_{fuse} = dup(u, l) \oplus \text{H} \quad \text{..... (4.17)}$$

$$p^l = \text{MLP}_l\left(\text{H}_{fuse}\right) \quad \text{..... (4.18)}$$

$$p^r = \text{MLP}_r\left(\text{H}_{fuse}\right) \quad \text{..... (4.19)}$$

4.1.3 Bipartite Matching

Untuk membuat sebuah model mengetahui nilai error nya (melakukan prediksi) saat training memerlukan fungsi menghitung loss. Menjadi kesulitan untuk melakukan penghitungan perbandingan nilai hasil prediksi dengan hasil sebenarnya (disebut juga *golden entities*). Metode *bipartite matching* adalah solusi yang dipilih untuk Sequence-to-Set Network. Sebelum menghitung nilai loss nya perlu dicarikan pasangan optimal diantara set prediksi sejumlah *N* ($\hat{y} = \{\hat{y}_i\}_{i=1}^{N}$) dan set golden (*y*) nya. Mengingat bahwa set prediksi memiliki jumlah tetap sebanyak *N*, maka set golden (*y*) akan diberikan lapisan pengisi (*padding*) dengan label entitas kosong (∅). Menghitung nilai *cost* dari penyocokan antara prediksi dengan golden akan menggunakan cara penghitungan algoritma Hungarian[18] (L_{match}). Rumus untuk algoritma Hungarian dapat ditemukan pada rumus

$$L_m\left(y_i, \hat{y}_{\hat{\beta}_i}\right) = -1_{\{c_i \neq \emptyset\}}\left[p^c_{\hat{\beta}_i}(c_i) + p^l_{\hat{\beta}_i}(l_i) + p^r_{\hat{\beta}_i}(r_i)\right] \quad (4.21).$$

Dengan menemukan nilai cost penyocokan prediksi dengan golden, dapat dilakukan pencarian penyocokan paling optimal dari cost yang terrendah (rumus $\hat{\beta} = \text{argmin} \sum_i^N L_{match}(y_i, \hat{y}_{\hat{\beta}_i})$ (4.20)). Setelah mendapatkan pasangan optimal ($\hat{\beta}(i)$), langsung dapat menghitung loss akhirnya yaitu rumus

$$L(y, \hat{y}) = \sum_{i=1}^{N}\begin{Bmatrix} -\log p^c_{\hat{\beta}_i}(c_i) + 1_{\{c_i \neq \emptyset\}} \\ \left[-\log p^l_{\hat{\beta}_i}(l_i) - \log p^r_{\hat{\beta}_i}(r_i)\right] \end{Bmatrix} \quad (4.22).$$

$$\hat{\beta} = \text{argmin} \sum_i^N L_{match}(y_i, \hat{y}_{\hat{\beta}_i}) \quad \text{..... (4.20)}$$

$$L_m\left(y_i, \hat{y}_{\hat{\beta}_i}\right) = -1_{\{c_i \neq \emptyset\}}\left[p^c_{\hat{\beta}_i}(c_i) + p^l_{\hat{\beta}_i}(l_i) + p^r_{\hat{\beta}_i}(r_i)\right] \quad \text{..... (4.21)}$$

$$L(y, \hat{y}) = \sum_{i=1}^{N}\begin{Bmatrix} -\log p^c_{\hat{\beta}_i}(c_i) + 1_{\{c_i \neq \emptyset\}} \\ \left[-\log p^l_{\hat{\beta}_i}(l_i) - \log p^r_{\hat{\beta}_i}(r_i)\right] \end{Bmatrix} \quad \text{..... (4.22)}$$

[18] Harold W Kuhn, The Hungarian Method for The Assignment Problem. Naval research logistics quarterly, 2(1-2): hal.83–97, (1955).

4.2 Library dan Tools

Pembuatan program ini didukung dengan beberapa library dan tools yang membantu mempercepat beberapa tugas. Bagian dari bab ini akan menjelaskan dan mendeskripsikan library yang telah digunakan. Juga beberapa fungsi yang diaplikasikan ke dalam program Sequence-To-Set Network ini.

4.2.1 PyTorch

PyTorch[19] adalah framework *machine learning* yang bersifat *open source* (kode sumber program tersedia untuk dimodifikasi atau digunakan kembali) yang berdasarkan dari library Torch dan dikembangkan oleh Meta AI. Selain NLP, library ini dapat juga digunakan unutk tugas *computer vision*. Framework ini sudah membantu beberapa program ternama didunia NLP seperti Hugging Face's Transformers[20], PyTorch Lightning[21]. Begitupun juga dengan program selain NLP seperti Catalyst[22] (framework untuk Deep Learning) dan Tesla Autopilot.

Beberapa fitur yang sering digunakan pada pengaplikasian NLP dan juga pada tugas akhir ini adalah penggunaan jenis *class* PyTorch (yang juga berasal dari Torch awalnya) adalah Tensor. Mirip dengan NumPy array, PyTorch Tensor dapat dioperasikan dengan NVIDIA GPU yang menggunakan CUDA (tugas akhir ini juga menggunakan jenis GPU tersebut). Ada banyak jenis Tensor yang disediakan, namun tidak akan dijelaskan. Fitur *modules* yang disediakan juga menjadi fitur yang selalu dipakai dalam pembuatan program Sequence-To-Set Network. Beberapa modules dari PyTorch adalah autograd module (efisien dalam menghitung diferensiasi parameter di forward pass), optim module (modul yang memiliki berbagai algoritma optimasi yang digunakan untuk membangun neural network), dan nn module (memudahkan untuk membuat computational graphs dan mengambil nilai gradien).

---

[19] GitHub - Pytorch (https://github.com/pytorch/pytorch)
[20] Pytorch-Transformers, (https://pytorch.org/hub/huggingface_pytorch-transformers/)
[21] GitHub - PyTorch Lightning, (https://github.com/PyTorchLightning/pytorch-lightning/)
[22] GitHub - Catalyst, (https://github.com/catalyst-team/catalyst)

4.2.2 Hugging Face (Transformers)

Hugging Face[23] adalah library Python yang menyediakan sekian ribu model yang sudah dilewatkan proses training (pretrained) sehingga dapat langsung dipake untuk pengaplikasian tugas NLP. Dan jenis tugas NLP yang disediakan modelnya beragam dari teks, visual dan audio. Tugas akhir ini menggunakan Hugging Face untuk mengambil library Transformers juga mengambil pretrained model IndoBERT. Beberapa arsitektur terbesar yang umum digunakan dalam penelitian disediakan dalam Hugging Face, contohnya BERT, GPT, GPT-2, RoBERTa, dan lain-lainnya. Hugging Face juga merupakan suatu platform yang sangat besar, memiliki fitur-fitur dalam websitenya seperti membuat model baru, mencari dataset, adapun untuk berkomunitas. Beberapa fungsi yang digunakan dari transformers oleh Hugging Face dalam tugas akhir ini seperti AdamW (optimizer), BertConfig, BertTokenizer .

4.2.3 Google Colab

Google Colab[24] adalah hasil dari Google Research, siapa saja dapat menulis dan mengeksekusi kode python melalui browser, dan sangat cocok untuk pembelajar interaktif mengenai machine learning, data analysis, dan lainnya. Colab bisa dikatakan sebagai *notebook* Jupyter yang dihosting. Karena tersedia secara online, fasilitas ini tidak membutuhkan setup apapun hanya memerlukan akun Google dan gratis tidak berbayar. Tetapi karena produk gratis, pasti ada beberapa batasan dari pihak Google yaitu penggunaan GPU tidak akan selalu tersedia, tiap notebook memiliki batas waktu *timeout* jika tidak melakukan apapun (tidak interaktif meskipun ada program yang sedang berjalan didalamnya) apabila melewati waktu itu maka koneksi notebook dengan *runtime* akan diputuskan. Dan apabila notebook tersebut ditutup saat sedang menjalankan suatu program, maka runtime akan terputus dan harus menyambungkan dari awal kembali.

[23] Hugging Face, (https://huggingface.co)/
[24] Google Colab, (https://colab.research.google.com/?utm_source=scs-index).

Dengan banyaknya batasan terdapat beberapa akun *premium* yang disediakan oleh Google Colab namun berbayar. Terdapat dua jenis akun, Colab Pro dan Colab Pro +. Kedua dari akun tersebut akan mendapatkan GPU dan TPU yang lebih cepat, runtime lebih lama dan memory (RAM) lebih banyak. Untuk perbedaannya, Colab Pro + memiliki fitur eksekusi program dibelakang layer, sehingga user tidak perlu melakukan interaksi dan program dapat berjalan tanpa diputuskan oleh Google Colab sendiri. Dan untuk jumlah waktu runtime, jumlah memory dan GPU tentu lebih banyak daripada Colab Pro.

Tugas akhir ini akan menggunakan Google Colab Pro sebagai bantuan untuk melakukan training model. GPU yang diberikan untuk akun Colab Pro adalah NVIDIA Tesla P100 PCIe 16 GB atau NVIDIA Tesla T4 16GB. Adapun pilihan untuk eksekusi notebook dengan pilihan High-RAM. Namun pilihan itu hanya berlaku untuk satu notebook yang sedang online saja.

4.3 Modifikasi Metode Sequence To Set Network

Penelitian metode pada tugas akhir ini melakukan beberapa modifikasi sesuai dengan fasilitas dan sumber daya yang digunakan. Pertama, bagian representasi kata yang dihasilkan oleh layer encoder merupakan gabungan dari beberapa jenis word embedding. Pada tugas akhir ini tidak akan menggunakan word embedding jenis POS Tag, selain karena dataset tidak disediakan POS Tag, tugas akhir ini mencoba untuk melakukan tugas NER tanpa bantuan POS Tag.

Kedua, adalah parameter freeze_transformer akan diberikan nilai True, alias tiap kali model melakukan training Transformers tidak akan melakukan training ulang. Sehingga untuk weight yang digunakan adalah weight yang sudah pernah training sebelumnya dari Transformers. Hal ini dilakukan karena terbatasnya GPU yang ada untuk penelitian ini. Training model dilakukan di dua lingkungan (*environment*) yang berbeda. Yang pertama berada di Google Colab dengan GPU 16GB dan lainnya berada di OS Linux dengan GPU GeForce RTX 3070 dengan 8GB.

Untuk menyesuaikan dengan bahasa dan dataset yang ditujukan, Sequence To Set Network dalam bahasa Indonesia akan menggantikan pretrained model

untuk word embedding. Yang pertama adalah dari BERT menjadi IndoBERT (dikembangkan oleh IndoLEM). Dan GLoVE akan digantikan dengan Word2Vec model yang disediakan dari bapak Dr. Ir. Joan Santoso, S.Kom, M.Kom.

Karena berada di dua lingkungan yang berbeda, ada beberapa modifikasi yang diberikan khusus untuk tiap lingkungan. Program pada Google Colab akan memiliki fitur *checkpoint* untuk tiap epoch yang dijalankan. Fitur ini disediakan karena durasi *runtime* yang ambigu dari Google Colab Pro. Untuk yang berada di Linux tidak akan diberikan checkpoint, namun modifikasi yang diberikan (karena hanya 8GB) adalah ukuran hidden layer dari arsitektur akan dikurang. Yang seharusnya FFN memiliki ukuran hidden 1024 dan modelnya memiliki ukuran hidden 768. Diubah kedua-duanya menjadi 504 agar dapat menjalankan training di dalam GPU Linux. Karena keterbatasan memory GPU, jumlah epoch yang disarankan dari penelitian Sequence-to-Set Network yang seharusnya 100 epoch akan diubah menjadi 30 epoch.

4.4 Contoh Kasus Penggunaan Sequence To Set

Subbab terakhir ini akan memberikan penjabaran penghitungan matematis yang terjadi dalam serangkaian Sequence-To-Set Network. Untuk input kalimat akan diberikan “Jalan Ir. Soekarno” dan penghitungan akan berawal dari encoder sampai penghitungan output layer decoder.

Word2Vec Embedding

Input : Jalan Ir Soekarno

Bobot Word2Vec Embedding

$$W_{w2v} = \begin{bmatrix} 0.4124 & 0.4761 \\ 0.4420 & 0.4288 \\ 0.4444 & 0.4902 \end{bmatrix}$$

Representasi One Hot Encoding Tiap Kata

$$x_0 = [1,0,0] \quad x_1 = [0,1,0]\, x_2 = [0,0,2]$$

Representasi Word2Vec EmbeddingTiap Kata

$$x_0 = x_0 \times W_{w2v} = [1,0,0] \times \begin{bmatrix} 0.4124 & 0.4761 \\ 0.4420 & 0.4288 \\ 0.4444 & 0.4902 \end{bmatrix} = \begin{bmatrix} 0.412 \\ 0.476 \end{bmatrix}$$

$$x_1 = x_1 \times W_{w2v} = [0,1,0] \times \begin{bmatrix} 0.4124 & 0.4761 \\ 0.4420 & 0.4288 \\ 0.4444 & 0.4902 \end{bmatrix} = \begin{bmatrix} 0.442 \\ 0.4288 \end{bmatrix}$$

$$x_2 = x_2 \times W_{w2v} = [0,0,1] \times \begin{bmatrix} 0.4124 & 0.4761 \\ 0.4420 & 0.4288 \\ 0.4444 & 0.4902 \end{bmatrix} = \begin{bmatrix} 0.4444 \\ 0.4288 \end{bmatrix}$$

Output Hasil Embedding

$$x_0 = \begin{bmatrix} 0.412 \\ 0.476 \end{bmatrix}$$

$$x_1 = \begin{bmatrix} 0.442 \\ 0.4288 \end{bmatrix}$$

$$x_2 = \begin{bmatrix} 0.4444 \\ 0.4288 \end{bmatrix}$$

Bobot Forward LSTM

$$w_a^{fwd} = \begin{bmatrix} 4{,}228 & 476 \\ 2{,}042 & 2{,}441 \end{bmatrix} \quad u_a^{fwd} = \begin{bmatrix} 4{,}561 & -746 \\ 195 & 5{,}231 \end{bmatrix} \quad b_a^{fwd} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$w_i^{fwd} = \begin{bmatrix} -4{,}212 & -3{,}436 \\ 5{,}777 & -4{,}112 \end{bmatrix} u_i^{fwd} = \begin{bmatrix} 133 & -3{,}952 \\ 872 & 8{,}111 \end{bmatrix} \quad b_i^{fwd} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$w_f^{fwd} = \begin{bmatrix} 2{,}981 & -3{,}784 \\ -5{,}323 & -427 \end{bmatrix} u_f^{fwd} = \begin{bmatrix} -2{,}032 & 2{,}688 \\ -2{,}092 & 9{,}231 \end{bmatrix} \quad b_f^{fwd} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$w_o^{fwd} = \begin{bmatrix} -659 & 6{,}471 \\ -7{,}133 & -956 \end{bmatrix} \quad u_o^{fwd} = \begin{bmatrix} -5{,}732 & 5{,}612 \\ 0{,}43 & 3{,}333 \end{bmatrix} \quad b_o^{fwd} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Timestep Pertama ($t = 0$)

$$a_0^{fwd} = \tanh(w_a^{fwd} \cdot x_0 + u_a^{fwd} \cdot h_{-1} + b_a^{fwd})$$

$$= \tanh\left(\begin{bmatrix} 4{,}228 & 476 \\ 2{,}042 & 2{,}441 \end{bmatrix} \cdot \begin{bmatrix} 0.412 \\ 0.476 \end{bmatrix} + \begin{bmatrix} 4{,}561 & -746 \\ 195 & 5{,}231 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}\right)$$

$$= \tanh\left(\begin{bmatrix} 197 \\ 2{,}004 \end{bmatrix} + 0 + 0\right)$$

$$= \begin{bmatrix} 0.194 \\ 0.197 \end{bmatrix}$$

$$i_0^{fwd} = \sigma(w_i^{fwd} \cdot x_0 + u_i^{fwd} \cdot h_{-1} + b_i^{fwd})$$

$$= \sigma\left(\begin{bmatrix} -4{,}212 & -3{,}436 \\ 5{,}777 & -4{,}112 \end{bmatrix} \cdot \begin{bmatrix} 0.412 \\ 0.476 \end{bmatrix} + \begin{bmatrix} 133 & -3{,}952 \\ 872 & 8{,}111 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}\right)$$

$$= \sigma\left(\begin{bmatrix} -3{,}373 \\ 425 \end{bmatrix} + 0 + 0\right)$$

$$= \begin{bmatrix} 0.416 \\ 0.510 \end{bmatrix}$$

$$f_0^{fwd} = \sigma(w_f^{fwd} \cdot x_0 + u_f^{fwd} \cdot h_{-1} + b_f^{fwd})$$

$$= \sigma\left(\begin{bmatrix} 2{,}981 & -3{,}784 \\ -5{,}323 & -427 \end{bmatrix} \cdot \begin{bmatrix} 0.412 \\ 0.476 \end{bmatrix} + \begin{bmatrix} -2{,}032 & 2{,}688 \\ -2{,}092 & 9{,}231 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}\right)$$

$$= \sigma\left(\begin{bmatrix} -5{,}722 \\ -42{,}282 \end{bmatrix} + 0 + 0\right)$$

$$= \begin{bmatrix} 0.485 \\ 0.395 \end{bmatrix}$$

$$o_0^{fwd} = \sigma(w_o^{fwd} \cdot x_0 + u_o^{fwd} \cdot h_{-1} + b_o^{fwd})$$

$$= \sigma\left(\begin{bmatrix} -659 & 6{,}471 \\ -7{,}133 & -956 \end{bmatrix} \cdot \begin{bmatrix} 0.412 \\ 0.476 \end{bmatrix} + \begin{bmatrix} -5{,}732 & 5{,}612 \\ 0{,}43 & 3{,}333 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}\right)$$

$$= \sigma\left(\begin{bmatrix} 280{,}907 \\ -33{,}968 \end{bmatrix} + 0 + 0\right)$$

$$= \begin{bmatrix} 0.569 \\ 0.415 \end{bmatrix}$$

$$c_0^{fwd} = c_{-1}^{fwd} * f_0^{fwd} + i_0^{fwd} * a_0^{fwd}$$

$$= \begin{bmatrix} 0 \\ 0 \end{bmatrix} * \begin{bmatrix} 0.485 \\ 0.395 \end{bmatrix} + \begin{bmatrix} 0.416 \\ 0.510 \end{bmatrix} * \begin{bmatrix} 0.194 \\ 0.197 \end{bmatrix}$$

$$= 0 + \begin{bmatrix} 0.173 \\ 0.260 \end{bmatrix}$$

$$= \begin{bmatrix} 0.173 \\ 0.260 \end{bmatrix}$$

$$h_0^{fwd} = o_0^{fwd} * tanh(c_0^{fwd})$$

$$= \begin{bmatrix} 0.569 \\ 0.415 \end{bmatrix} * tanh\left(\begin{bmatrix} 0.173 \\ 0.260 \end{bmatrix}\right)$$

$$= \begin{bmatrix} 0.569 \\ 0.415 \end{bmatrix} * \begin{bmatrix} 0.171 \\ 0.254 \end{bmatrix}$$

$$= \begin{bmatrix} 0.097 \\ 0.106 \end{bmatrix}$$

Bobot Backward LSTM

$$w_a^{bwd} = \begin{bmatrix} 0.602 & 0.202 \\ 0.853 & -5{,}515 \end{bmatrix} \quad u_a^{bwd} = \begin{bmatrix} 0.197 & 0.280 \\ 0.686 & 0.239 \end{bmatrix} \quad b_a^{bwd} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$w_i^{bwd} = \begin{bmatrix} -4{,}938 & 0.153 \\ -2{,}557 & 0.248 \end{bmatrix} \quad u_i^{bwd} = \begin{bmatrix} 0.428 & 0.162 \\ 0.803 & -3{,}118 \end{bmatrix} \quad b_i^{bwd} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$w_f^{bwd} = \begin{bmatrix} -2{,}498 & -4{,}614 \\ -3{,}643 & -7{,}995 \end{bmatrix} u_f^{bwd} = \begin{bmatrix} -3{,}865 & -3{,}649 \\ 0.333 & 0.380 \end{bmatrix} b_f^{bwd} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$w_o^{bwd} = \begin{bmatrix} 0.646 & 0.332 \\ 0.112 & 0.276 \end{bmatrix} \quad u_o^{bwd} = \begin{bmatrix} 0.543 & 0.426 \\ -6{,}966 & -8{,}945 \end{bmatrix} b_o^{bwd} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Timestep Pertama ($t = 0$)

$$a_0^{bwd} = \tanh(w_a^{bwd} \cdot x_0 + u_a^{bwd} \cdot h_{-1} + b_a^{bwd})$$

$$= \tanh\left(\begin{bmatrix} 0.602 & 0.202 \\ 0.853 & -5{,}515 \end{bmatrix} \cdot \begin{bmatrix} 0.412 \\ 0.476 \end{bmatrix} + \begin{bmatrix} 0.197 & 0.280 \\ 0.686 & 0.239 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}\right)$$

$$= \tanh\left(\begin{bmatrix} 0.344 \\ -2.273 \end{bmatrix} + 0 + 0\right)$$

$$= \begin{bmatrix} 0.344 \\ -2.273 \end{bmatrix}$$

$$i_0^{bwd} = \sigma(w_i^{bwd} \cdot x_0 + u_i^{bwd} \cdot h_{-1} + b_i^{bwd})$$

$$= \sigma\left(\begin{bmatrix} -4{,}938 & 0.153 \\ -2{,}557 & 0.248 \end{bmatrix} \cdot \begin{bmatrix} 0.412 \\ 0.476 \end{bmatrix} + \begin{bmatrix} 0.428 & 0.162 \\ 0.803 & -3{,}118 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}\right)$$

$$= \sigma\left(\begin{bmatrix} -1.963 \\ -0.936 \end{bmatrix} + 0 + 0\right)$$

$$= \begin{bmatrix} -1.963 \\ -0.936 \end{bmatrix}$$

$$f_0^{bwd} = \sigma(w_f^{bwd} \cdot x_0 + u_f^{bwd} \cdot h_{-1} + b_f^{bwd})$$

$$= \sigma\left(\begin{bmatrix} -2{,}498 & -4{,}614 \\ -3{,}643 & -7{,}995 \end{bmatrix} \cdot \begin{bmatrix} 0.412 \\ 0.476 \end{bmatrix} + \begin{bmatrix} -3{,}865 & -3{,}649 \\ 0.333 & 0.380 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}\right)$$

$$= \sigma\left(\begin{bmatrix} -3.226 \\ -5.308 \end{bmatrix} + 0 + 0\right)$$

$$= \begin{bmatrix} -3.226 \\ -5.308 \end{bmatrix}$$

$$o_0^{bwd} = \sigma(w_o^{bwd} \cdot x_0 + u_o^{bwd} \cdot h_{-1} + b_o^{bwd})$$

$$= \sigma\left(\begin{bmatrix} 0.646 & 0.332 \\ 0.112 & 0.276 \end{bmatrix} \cdot \begin{bmatrix} 0.412 \\ 0.476 \end{bmatrix} + \begin{bmatrix} 0.543 & 0.421 \\ -6{,}966 & -8{,}945 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}\right)$$

$$= \sigma\left(\begin{bmatrix} 0.425 \\ 0.178 \end{bmatrix} + 0 + 0\right)$$

$$= \begin{bmatrix} 0.425 \\ 0.178 \end{bmatrix}$$

$$c_0^{bwd} = c_{-1}^{bwd} * f_0^{bwd} + i_0^{bwd} * a_0^{bwd}$$

$$= \begin{bmatrix}0\\0\end{bmatrix} * \begin{bmatrix}-3.226\\-5.308\end{bmatrix} + \begin{bmatrix}-1.963\\-0.936\end{bmatrix} * \begin{bmatrix}0.344\\-2.273\end{bmatrix}$$

$$= 0 + \begin{bmatrix}0.344\\-2.273\end{bmatrix}$$

$$= \begin{bmatrix}0.344\\-2.273\end{bmatrix}$$

$$h_0^{bwd} = o_0^{bwd} * tanh(c_0^{bwd})$$

$$= \begin{bmatrix}0.425\\0.178\end{bmatrix} * tanh\left(\begin{bmatrix}0.344\\-2.273\end{bmatrix}\right)$$

$$= \begin{bmatrix}0.425\\0.178\end{bmatrix} * \begin{bmatrix}0.331\\-0.979\end{bmatrix}$$

$$= \begin{bmatrix}0.140\\-0.174\end{bmatrix}$$

Self dan Cross Attention

$$input = \begin{pmatrix}1 & 0 & 1 & 0\\0 & 2 & 0 & 2\\1 & 1 & 1 & 1\end{pmatrix}$$

$$weight_{key} = \begin{pmatrix}0 & 0 & 1\\1 & 1 & 0\\0 & 1 & 0\\1 & 1 & 0\end{pmatrix} \quad weight_{query} = \begin{pmatrix}1 & 0 & 1\\1 & 0 & 0\\0 & 0 & 1\\0 & 1 & 1\end{pmatrix} weight_{value} =$$

$$\begin{pmatrix}0 & 2 & 0\\0 & 3 & 0\\1 & 0 & 3\\1 & 1 & 0\end{pmatrix}$$

$$matriks_{key} = weight_{key} * input$$

$$= \begin{pmatrix}0 & 0 & 1\\1 & 1 & 0\\0 & 1 & 0\\1 & 1 & 0\end{pmatrix} * \begin{pmatrix}1 & 0 & 1 & 0\\0 & 2 & 0 & 2\\1 & 1 & 1 & 1\end{pmatrix}$$

$$= \begin{pmatrix}0 & 1 & 1\\4 & 4 & 0\\2 & 3 & 1\end{pmatrix}$$

$$matriks_{query} = weight_{query} * input$$

$$= \begin{pmatrix}1 & 0 & 1\\1 & 0 & 0\\0 & 0 & 1\\0 & 1 & 1\end{pmatrix} * \begin{pmatrix}1 & 0 & 1 & 0\\0 & 2 & 0 & 2\\1 & 1 & 1 & 1\end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 2 \\ 2 & 2 & 2 \\ 2 & 1 & 3 \end{pmatrix}$$

$$matriks_{value} = weight_{value} * input$$

$$= \begin{pmatrix} 0 & 2 & 0 \\ 0 & 3 & 0 \\ 1 & 0 & 3 \\ 1 & 1 & 0 \end{pmatrix} * \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 2 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 2 & 3 \\ 2 & 8 & 0 \\ 2 & 6 & 3 \end{pmatrix}$$

Attention Scores

$$att = weight_{query} * {weight_{key}}^{T}$$

$$= \begin{pmatrix} 2 & 4 & 4 \\ 4 & 16 & 12 \\ 4 & 12 & 10 \end{pmatrix}$$

$$Updated\ Scores = softmax(att)$$

$$= softmax\left(\begin{pmatrix} 2 & 4 & 4 \\ 4 & 16 & 12 \\ 4 & 12 & 10 \end{pmatrix}\right)$$

$$= \begin{pmatrix} 0{,}0633 & 0{,}4683 & 0{,}4683 \\ 0{,}00000603 & 0{,}9820 & 0{,}0179 \\ 0{,}00029539 & 0{,}88054 & 0{,}1191 \end{pmatrix}$$

untuk tracing yang readable nilai sotmax diubah menjadi seperti dibawah

$$= \begin{pmatrix} 0 & 0{,}5 & 0{,}5 \\ 0 & 1 & 0 \\ 0 & 0{,}9 & 0{,}1 \end{pmatrix}$$

$$Weighted\ Values = scores * matriks_{value}$$

$$= \begin{pmatrix} 0 \\ 0.5 \\ 0.5 \end{pmatrix} * \begin{pmatrix} 1 & 2 & 3 \\ 2 & 8 & 0 \\ 2 & 6 & 3 \end{pmatrix}$$

$$= \begin{pmatrix} 0 & 0 & 0 \\ 1 & 4 & 0 \\ 1 & 3 & 1.5 \end{pmatrix}$$

$$output1 = sum\ weighted\ values$$

$$= (0 \quad 0 \quad 0) + (1 \quad 4 \quad 0) + (1 \quad 3 \quad 1.5)$$

$$= (2 \quad 7 \quad 1.5)$$

Feed Forward Network

$$input = \begin{pmatrix} 1.3238 \\ 3.1021 \\ 2.4202 \\ 1.3737 \\ 2.0869 \end{pmatrix}$$

$$linear1 = Linear(input) = Linear\left(\begin{pmatrix} 1.3238 \\ 3.1021 \\ 2.4202 \\ 1.3737 \\ 2.0869 \end{pmatrix}\right) = \begin{pmatrix} 0.6193 & -0.7424 \\ 1.5766 & -0.454 \\ 1.2095 & -0.5646 \\ 0.6461 & -0.7343 \\ 1.0301 & -0.6187 \end{pmatrix}$$

$$dropout1 = Dropout(linear1, 0.2)$$

$$= Dropout\left(\begin{pmatrix} 0.6193 & -0.7424 \\ 1.5766 & -0.454 \\ 1.2095 & -0.5646 \\ 0.6461 & -0.7343 \\ 1.0301 & -0.6187 \end{pmatrix}, 0.2\right) = \begin{pmatrix} 1.6547 \\ 3.8776 \\ 3.0253 \\ 1.7171 \\ 0 \end{pmatrix}$$

$$linear2 = Linear(dropout1) = Linear\left(\begin{pmatrix} 1.6547 \\ 3.8776 \\ 3.0253 \\ 1.7171 \\ 0 \end{pmatrix}\right) = \begin{pmatrix} 1.6309 & 0 \\ 3.0508 & 0 \\ 2.5064 & 0 \\ 1.6707 & 0 \\ 0.5739 & 0 \end{pmatrix}$$

$$dropout2 = Dropout(linear2, 0.2) = Dropout\left(\begin{pmatrix} 1.6309 & 0 \\ 3.0508 & 0 \\ 2.5064 & 0 \\ 1.6707 & 0 \\ 0.5739 & 0 \end{pmatrix}, 0.2\right) =$$

$$\begin{pmatrix} 1.6309 & 0 \\ 3.0508 & 0 \\ 0 & 0 \\ 1.6707 & 0 \\ 0.5739 & 0 \end{pmatrix}$$

$$tgt = input + dropout2 = \begin{pmatrix} 1.3238 \\ 3.1021 \\ 2.4202 \\ 1.3737 \\ 2.0869 \end{pmatrix} + \begin{pmatrix} 1.6309 \\ 3.0508 \\ 0 \\ 1.6707 \\ 0.5739 \end{pmatrix} = \begin{pmatrix} 2.9547 \\ 6.1529 \\ 2.4202 \\ 3.0444 \\ 2.6608 \end{pmatrix}$$

$$output\_ffn = LayerNorm(tgt) = LayerNorm\left(\begin{pmatrix} 2.9547 \\ 6.1529 \\ 2.4202 \\ 3.0444 \\ 2.6608 \end{pmatrix}\right)$$

$$= \begin{pmatrix} 1.3527\text{e}-05 \\ -4.9784\text{e}-05 \\ -1.8162\text{e}-05 \\ -2.1128\text{e}-05 \\ -5.0033\text{e}-06 \end{pmatrix}$$

MLP Classifier

Classifier Jenis Entitas

$$input = \begin{pmatrix} 1.3527\text{e}-05 & 0.3527 \\ -4.9784\text{e}-05 & -3.9784 \\ -1.8162\text{e}-05 & -0.8162 \\ -2.1128\text{e}-05 & -1.1128 \\ -5.0033\text{e}-06 & -4.0033 \end{pmatrix}$$

$$linear1 = Linear(input) = Linear\left(\begin{pmatrix} 1.3527\text{e}-05 & 0.3527 \\ -4.9784\text{e}-05 & -3.9784 \\ -1.8162\text{e}-05 & -0.8162 \\ -2.1128\text{e}-05 & -1.1128 \\ -5.0033\text{e}-06 & -4.0033 \end{pmatrix}\right) =$$

$$\begin{pmatrix} 0.0501 \\ -2.3495 \\ -0.5975 \\ -0.7618 \\ -2.3633 \end{pmatrix}$$

$$output\_mlp\_class = \begin{pmatrix} 0.0501 \\ -2.3495 \\ -0.5975 \\ -0.7618 \\ -2.3633 \end{pmatrix}$$

Classifier Boundary

$$tgt = \begin{pmatrix} 1.3527\text{e}-05 \\ -4.9784\text{e}-05 \\ -1.8162\text{e}-05 \\ -2.1128\text{e}-05 \\ -5.0033\text{e}-06 \end{pmatrix}$$

$$concat\_bilstm = concat\left(h_0^{bwd}, h_0^{fwd}\right) = \begin{pmatrix} 0.0978 & 0.1409 \\ 0.1060 & -0.1745 \end{pmatrix}$$

$$fuse = concat(concat\_bilstm, output_{ffn})$$

$$= \begin{pmatrix} -4.9784\text{e}-05 & 0.0978 & 0.1409 \\ -1.8162\text{e}-05 & 0.1060 & -0.1745 \\ -2.1128\text{e}-05 & 0 & 0 \\ -5.0033\text{e}-06 & 0 & 0 \end{pmatrix}$$

$$linear_{mlp} = Linear(fuse) = Linear\left(\begin{pmatrix} -4.9784\text{e}-05 & 0.0978 & 0.1409 \\ -1.8162\text{e}-05 & 0.1060 & -0.1745 \\ -2.1128\text{e}-05 & 0 & 0 \\ -5.0033\text{e}-06 & 0 & 0 \end{pmatrix}\right)$$

$$linear_{mlp} = \begin{pmatrix} -0.288 & -0.1416 \\ -0.3115 & -0.1618 \\ -0.1907 & -0.1618 \end{pmatrix}$$

$$linear_{mlp} = softmax\left(linear_{mlp}\right) = softmax\left(\begin{pmatrix} -0.288 & -0.1416 \\ -0.3115 & -0.1618 \\ -0.1907 & -0.1618 \end{pmatrix}\right)$$

$$linear_{mlp} = \begin{pmatrix} 0.3248 & 0.3296 \\ 0.3173 & 0.3474 \\ 0.358 & 0.323 \end{pmatrix}$$