BAB V

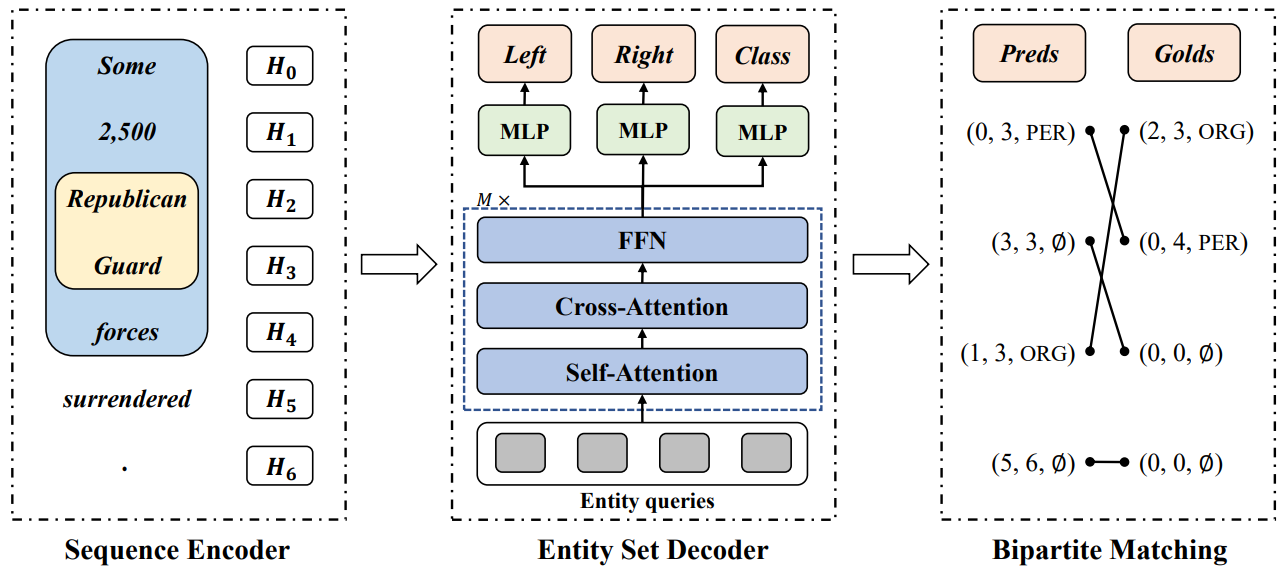
# SEQUENCE TO SET DALAM BAHASA INDONESIA

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 subabb 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* dimana 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.

## Sequence To Set Network

Sesuai pada gambar 5.1, 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. Dimana 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, Char LSTM, BiLSTM, POS Tag). Seluruh hasil embedding dari tiap metode akan kemudian digabung menjadi satu variabel dalam bentuk *vector*. Representasi kata terakhir ini kemudian akan dikenal sebagai *entity queries* dan menjadi output terakhir dari sequence encoder, juga menjadi input untuk bagian berikut yaitu set decoder layer.



Gambar 5.1  
Arsitektur Sistem Sequence-To-Set Network[[1]](#footnote-1)

Entity set decoder layer, mengambil entity queries 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 dibab 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 menghituing pencocokan optimal dari prediksi dan *golden entity*. Nilai optimal dari pencocokan set prediksi dengan golden entity didapatkan dengan bantuan dari Algoritma Hungarian.

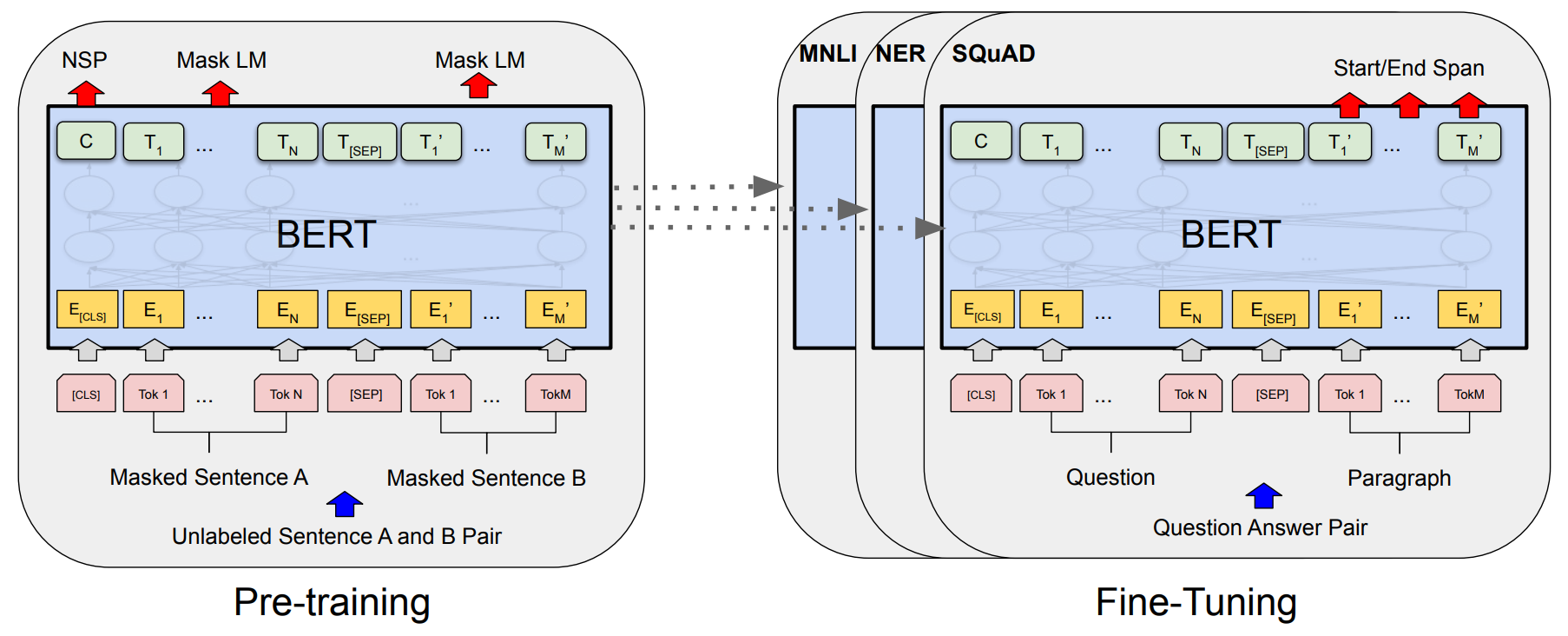
### Sequence Encoder

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 entity queries, bentuk akhir yang dihasilkan dari proses encoder nanti.

#### BERT

Bidirectional Encoder Representations from Transformers (BERT) adalah metode yang telah diteliti dan *publish* oleh Google AI Language[[2]](#footnote-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 sentiment. Hal-hal ini hanya dapat dilakukan dengan BERT memahami informasi mengenai cara bahasa tersebut bekerja.

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 task yang ditujukan. Penjelasan berikut adalah mengenai proses yang dilewatkan untuk BERT.



Gambar 5.2  
Visualisasi Arsitektur BERT QnA

Bagian pertama adalah pretraining (gambar 5.2 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 kedalam 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 paragraph berikut ini.

Bagian pertama (pretraining) 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 dimenggantikan 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 dibagian ini. BERT yang telah pretrained telah mengenal bahasa tersebut, sehingga yang perlu diubah adalah beberapa parameter didalam 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 pretraining adalah input dirubah, kalimat pertama adalah pertanyaan dan kaliamt kedua adalah jawaban (gambar 5.2 kotak kanan/fine tune, bagian Question dan Answer). Output juga dirubah dari awalnya output untuk menebak kata yang dimask, 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]](#footnote-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]](#footnote-4). Pembuatan IndoBERT oleh IndoLEM (Indonesia Language Evaluation Montage) menggunakan konfigurasi yang sama dengan BERT yang telah dijelaskan diatas (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 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]](#footnote-5), Tempo[[6]](#footnote-6), Liputan6[[7]](#footnote-7) (55 juta kata) dan Indonesian Web Corpus (90 juta kata).

#### 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 similaritas 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 diluar 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 ratu.

Word2Vec terinspirasi dari dua jenis arsitektur yaitu Feedforward Neural Net Language Model (NNLM)[[8]](#footnote-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 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 × D + N × D × H + H × V* adalah penilaian untuk komputasi NNLM, perlu diperhatikan bagian *N × D × 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 × H + H × 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 × D + D × 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.

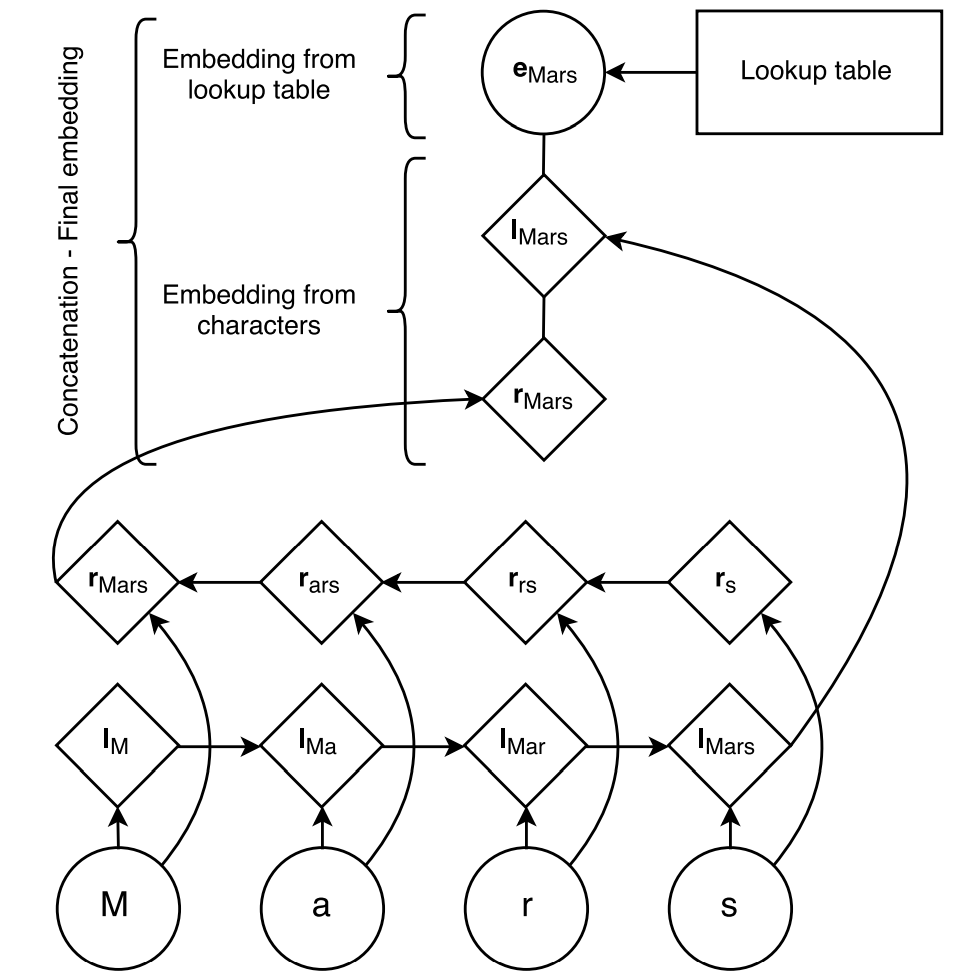
Arsitektur Skip-Gram memiliki kemiripan juga dengan C-BOW, tetapi tujuan mereka berkebalikan. Dimana C-BOW bertujuan untuk menebak suatu kata berdasarkan konteks kata sekitarnya, Skip-Gram berusaha semaksimal mungkin untuk klasifikasi kata berdasarkan suatu kata lain dalam satu kalimat yang sama. Skip-Gram 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 × (D + D × log2(V))*, dimana C adalah jarak maksimal kata-kata sekitarnya.

Belum jelaskan penghitungan skip-gram

#### Char LSTM

Character-level BiLSTM adalah metode embedding yang diambil dari penelitian mengenai Neural Architectures for Named Entity Recognition[[9]](#footnote-9)yang menginginkan input nya memiliki representasi yang sensitive 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.

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 5.3 adalah penggambaran arsitektur dari character-level embedding untuk kata “Mars”. Pada awal pembuatan embedding akan diinisialisasikan 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 5.3  
Arsitektur Character-Level Embedding[[10]](#footnote-10)

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 diakhir 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.

#### BiLSTM

Bidirectional LSTM[[11]](#footnote-11) Sebelum mencoba mengerti apa itu tugas pengenalan named entity (atau NER) secara mendalam, perlu diketahui terlebih dahulu apa yang dapat disebut sebagai sebuah named entity.

#### Entity Queries

Sebelum mencoba mengerti apa itu tugas pengenalan named entity (atau NER) secara mendalam, perlu diketahui terlebih dahulu apa yang dapat disebut sebagai sebuah named entity.

### Entity Set Decoder

Sebelum mencoba mengerti apa itu tugas pengenalan named entity (atau NER) secara mendalam, perlu diketahui terlebih dahulu apa yang dapat disebut sebagai sebuah named entity.

#### Decoder Layer

Sebelum mencoba mengerti apa itu tugas pengenalan named entity (atau NER) secara mendalam, perlu diketahui terlebih dahulu apa yang dapat disebut sebagai sebuah named entity.

##### Self – Attention

Sebelum mencoba mengerti apa itu tugas pengenalan named entity (atau NER) secara mendalam, perlu diketahui terlebih dahulu apa yang dapat disebut sebagai sebuah named entity.

##### Cross – Attention

Sebelum mencoba mengerti apa itu tugas pengenalan named entity (atau NER) secara mendalam, perlu diketahui terlebih dahulu apa yang dapat disebut sebagai sebuah named entity.

##### Feed Forward Network

Sebelum mencoba mengerti apa itu tugas pengenalan named entity (atau NER) secara mendalam, perlu diketahui terlebih dahulu apa yang dapat disebut sebagai sebuah named entity.

#### Classification Layer

Sebelum mencoba mengerti apa itu tugas pengenalan named entity (atau NER) secara mendalam, perlu diketahui terlebih dahulu apa yang dapat disebut sebagai sebuah named entity.

### Bipartite Matching

Sebelum mencoba mengerti apa itu tugas pengenalan named entity (atau NER) secara mendalam, perlu diketahui terlebih dahulu apa yang dapat disebut sebagai sebuah named entity.

## Modifikasi Metode Sequence To Set Network

Sebelum mencoba mengerti apa itu tugas pengenalan named entity (atau NER) secara mendalam, perlu diketahui terlebih dahulu apa yang dapat disebut sebagai sebuah named entity.

## Contoh Kasus Penggunaan Sequence To Set

Sebelum mencoba mengerti apa itu tugas pengenalan named entity (atau NER) secara mendalam, perlu diketahui terlebih dahulu apa yang dapat disebut sebagai sebuah named entity

1. Zeqi Tan, dkk, A Sequence-to-Set Network for Nested Named Entity Recognition, (2021). [↑](#footnote-ref-1)
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. [↑](#footnote-ref-2)
3. Yonghui Wu, dkk, Google’s neural machine translation system: Bridging the gap between human and machine translation, (2016). [↑](#footnote-ref-3)
4. Fajri Koto, dkk, IndoLEM and IndoBERT: A Benchmark Dataset and Pre-trained Language Model for Indonesian NLP, (2020). [↑](#footnote-ref-4)
5. Kompas, https://kompas.com [↑](#footnote-ref-5)
6. Koran TEMPO, https://koran.tempo.co [↑](#footnote-ref-6)
7. Liputan6, https://liputan6.com [↑](#footnote-ref-7)
8. Bengio Y, Ducharme R., Vincent P., A Neural Probabilistic Language Model, Journal of Machine Learning Research, 3: hal.1137-1155, (2003). [↑](#footnote-ref-8)
9. Guillaume Lample, dkk, Neural Architectures for Named Entity Recognition. In Proceedings of NAACL 2016, hal. 260–270, (2016). [↑](#footnote-ref-9)
10. Ibid [↑](#footnote-ref-10)
11. Zhiheng Huang , Wei Xu, Kai Yu, Bidirectional LSTM-CRF Models for Sequence Tagging, (2015). [↑](#footnote-ref-11)