BAB II

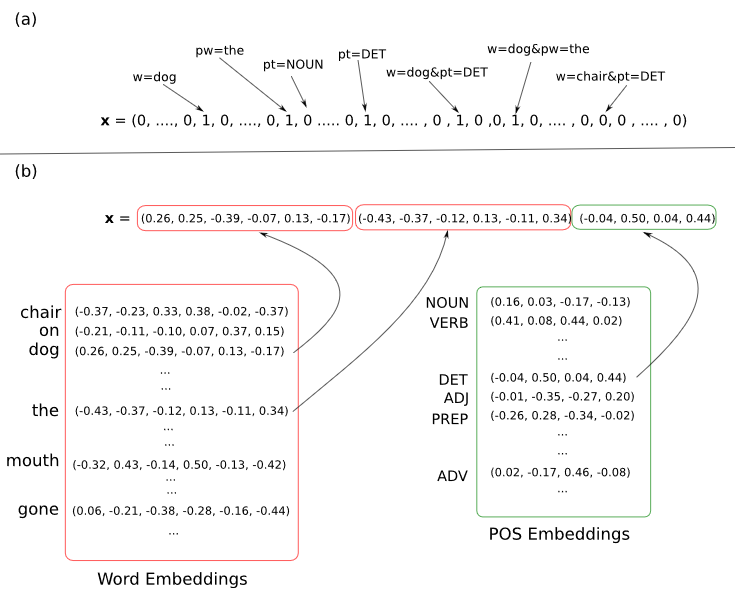
# TEORI PENUNJANG

Bab teori penunjang berisi penjelasan mengenai beberapa teori yang diperlukan sebagai dasar dalam pengerjaan Tugas Akhir ini. Bab ini juga dapat menjadi informasi bantuan bagi pembaca buku tugas akhir ini untuk pemahaman yang lebih baik terhadap hal – hal yang akan disebut atau jelaskan pada bab-bab berikutnya. Teori-teori yang akan disebut dalam bab ini termasuk beberapa teori yang umumnya digunakan dalam penelitian NLP seperti representasi kata (*Word Embedding*), adapun teori *machine learning* seperti Neural Network (NN) dan juga jenis arsitektur NN lainnya yang juga sering digunakan untuk penelitian NER. Beberapa teori lainnya adalah NER, Transformer (Self Attention, Cross Attention), Hungarian Match, BRAT.

## Word Embedding

Representasi kata adalah hal yang penting dalam penelitian NLP karena NLP sendiri mengolah kata agar computer dapat melakukan suatu task. Dan representasi kata memiliki dampak yang signifikan terhadap akurasi hasil dari model yang dibuat. Karena ini, ada beberapa teknik yang diciptakan oleh peneliti untuk menemukan teknik yang dapat membuat representasi kata yang membawa informasi yang penting untuk tiap katanya. Word Embedding, teknik yang pada saat ini menjadi tren dalam NLP karena kemampuannya untuk menyimpan nilai semantik dari satu kata dengan kata lainnya / kata di sekitarnya. [[1]](#footnote-1)

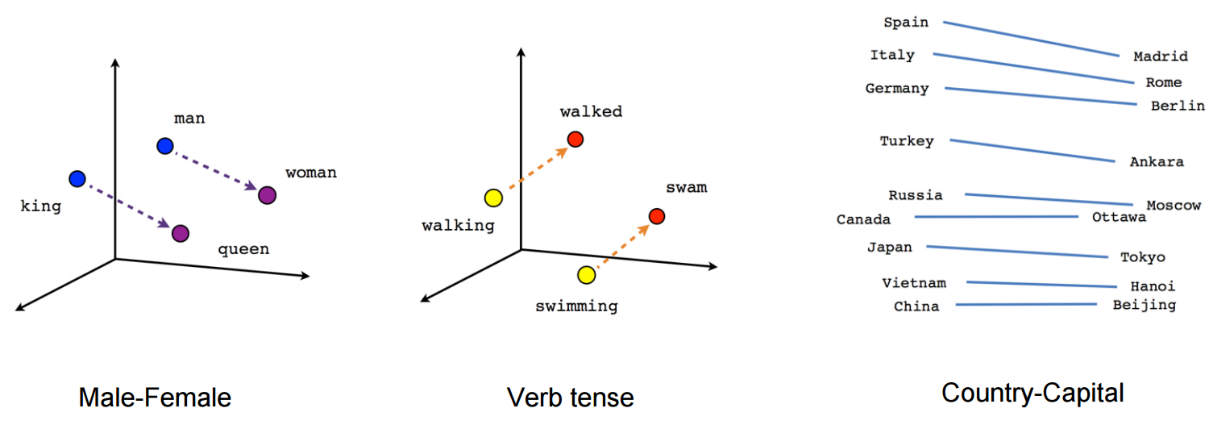
Secara visualisasi, hasil dari word embedding adalah sekumpulan vektor yang berisikan angka riil. Proses konversi kata menjadi vektor. Mengapa word embedding ini menjadi tren dalam penelitian NLP saat ini, salah satunya karena kemampuannya juga untuk memberikan representasi dalam bentuk *dense* (elemen *non-zero*), sedangkan teknik selain word embedding contohnya *Bag of Words*, yang menghasilkan representasi kata *sparse* (elemen yang sebagian besar mengandung nilai nol dan sedikit elemen *non-zero*).



Gambar 2.1  
Contoh Visualisasi (a) Bag of Words dan (b) Word Embedding

Contoh kasus perbandingan representasi kata Bag of Words dengan Word Embedding dapat dilihat pada gambar 2.1. Kata-kata yang perlu di *encode* adalah “the dog”. Jika kita representasikan tiap kata dengan satu dimensi dan memiliki panjang sebanyak kata yang ada dalam satu kosakata dataset tersebut, mengingat kata yang digunakan pasti berjumlah banyak ukuran untuk tiap representasi kata secara keseluruhan akan menjadi besar dan memakan nilai komputasi yang besar. Hal ini yang disebut sebagai representasi sparse. Sedangkan word embedding akan memiliki panjang dimensi yang tidak bergantungan dengan jumlah kata, karena itu tentu akan lebih efisiensi secara komputasi dan memori.

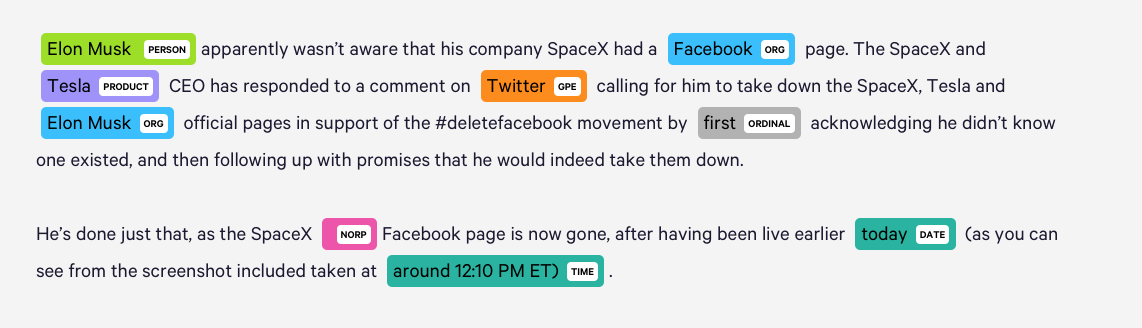
Dengan representasi dense dari word embedding, tiap nilai dari sebuah dimensi memiliki arti. Salah satu yang bisa dicontohkan adalah nilai semantik antar kata lainnya. Dari gambar 2.2 menggambarkan visualisasi nilai semantik yang dapat direpresentasikan. Relasi antar kata yang bisa diberikan seperti lawan kata (Male-Female), kalau dalam bahasa Inggris mampu memberikan nilai relasi bentuk kata kerja yang berbeda (Verb Tense), bahkan juga hubungan secara semantik seperti nama negara dengan nama ibu kotanya (Country-Capital).



Gambar 2.2   
Contoh Nilai Semantik Tersimpan dari Word Embedding

## Named Entity Recognition (NER)

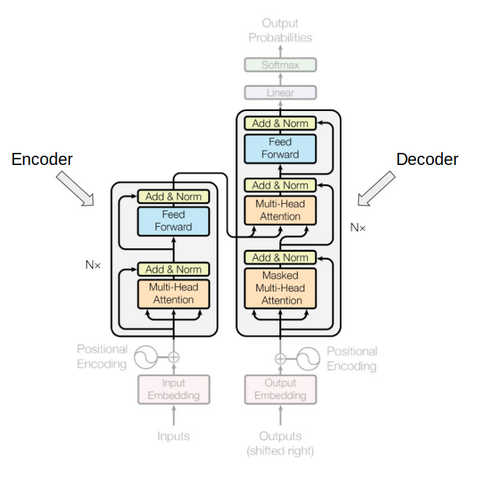
Named Entity Recognition (NER) merupakan task NLP untuk mencari entitas bernama dalam suatu kalimat kemudian menggolongkan entitas tersebut kepada kategori entitas yang ada dari sebuah dokumen/kalimat.[[2]](#footnote-2) Sebuah entitas bernama adalah istilah untuk menyebut sebuah entitas yang ada dalam dokumen teks yang ditulis dengan nama tertentu. Contoh dari kalimat “Perempuan tersebut sedang belajar”, kata perempuan dapat ditandakan sebagai entitas, namun karena entitas tersebut tidak memiliki nama maka tidak termasuk sebagai entitas bernama. Contoh entitas bernama adalah dengan kalimat “Valencia sedang belajar”, maka untuk entitas bernama PERSON (orang) adalah Valencia. Contoh lain entitas bernama adalah lokasi, produk, acara, dan organisasi, waktu dan tanggal. Semua entitas ini dapat memberikan informasi penting dan dapat dimanfaatkan pengguna untuk keperluan analisis lebih lanjut. orang) adalah Valencia. Penggambaran hasil dari task NER dapat dilihat dari gambar dibawah (gambar 2.3).



Gambar 2.3   
Contoh Pengenalan Entitas Bernama[[3]](#footnote-3)

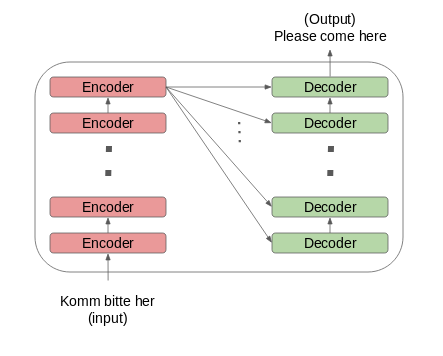
## Transformer

Konsep adanya Transformer muncul dari model *sequence-to-sequence* (*seq2seq*), di mana model tersebut memiliki tujuan untuk mengerti konversi sebuah sequence, contohnya adalah penerjemahan bahasa dari Inggris ke Indonesia. Model ini melakukan tugasnya dengan baik tetapi ada kesulitan dalam nilai ketergantungan yang berjangka lama. Penggunaan Transformer memberi solusi kepada kesulitan seq2seq. Transformer memiliki cara kerja yang berbeda dengan seq2seq, di mana encoder dan decodernya seq2seq menggunakan urutan selaras RNN, Transformer bergantung pada penggunaan *attention* untuk menghitung representasi input dan output nya. Gambar 2.4 adalah Arsitektur Transformers.



Gambar 2.4  
Arsitektur Transformers[[4]](#footnote-4)

Arsitektur dari Transformer berorientasi hanya pada encoder dan decoder yang bisa juga diatur jumlahnya. Isi dari bagian encoder dan decoder Transformer sebenarnya adalah encoder dan decoder yang berjumlah banyak (yang merupakan parameter yang bisa diatur tetapi yang ditetapkan pada paper “All You Need is Attention” adalah enam) dan bertumpukan (dapat dilihat pada Gambar 2.5). Secara detail, 1 bagian Encoder memiliki Multi-Head Attention kemudian diikuti Feed Forward Neural Network. Bagian dari 1 decoder memiliki layer yang sama dengan encoder tetapi ditambahkan dengan Masked Multi-Head Attention. Untuk penjelasan lebih rinci mengenai arsitektur Transformer, dapat dilihat pada subbab berikut mengenai bagian encoder dan decoder sendiri.



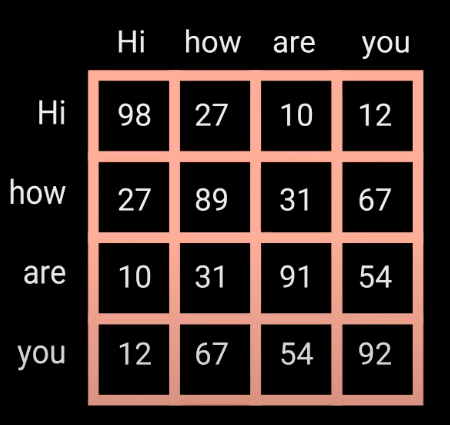
Gambar 2.5  
Representasi Transformer bagian Encoder Decoder

### Bagian Encoder

Bagian encoder memiliki peran untuk merubah urutan input menjadi representasi baru yang memegang informasi dari input tersebut. Struktur bagian ini memiliki 2 bagian besar, *multi-headed attention* dan *feed forward network* (FFN, yang akan dijelaskan pada subbab 2.4) ditambah dengan *residual connection* setelah tiap bagian tersebut. Untuk bagian pertama dari encoder ini adalah multi-headed attention, bagian ini mengaplikasikan ilmu self-attention. Self-attention adalah mekanisme dari ilmu attention dimana *model* dalam machine learning dapat memerhatikan informasi kata-kata sekitarnya dengan jarak yang jauh ke belakang (lebih jauh dibandingkan dengan arsitektur-arsitektur model yang lain seperti RNN, GRU, dan LSTM). [[5]](#footnote-5) Untuk memperjelas, self-attention akan dijelaskan dengan langkah-langkah sebagai berikut:

1. Membuat vektor *Query* (Q), *Key* (K) dan *Value* (V)
2. Membuat matriks *score*
3. Membagi nilai matriks score dengan akar pangkat dimensi vektor key ()
4. Matriks score dilewatkan pada fungsi aktivasi *softmax*
5. Mengalikan matriks score dengan vektor value
6. Menjumlahkan seluruh matriks yang dihitung

Self-attention menggunakan 3 vektor penting yang bernama *Query*, *Key* dan *Value*. Vektor-vektor ini memiliki konsep yang mirip dengan system pengambilan data. Contoh kasus pada website *Youtube*, jika ingin mencari suatu video user akan memasukan sebuah query dan system akan melakukan pencarian berdasarkan query tersebut dengan kumpulan set yang ada (seperti detail video yaitu judul, deskripsi, dsb). Kemudian hasil dari pencarian tersebut akan dikembalikan dalam sebuah nilai (value). Nilai vektor-vektor tersebut diambil dari hasil perkalian word embedding dari input dengan matriks yang dimiliki query, key and value masing-masing yang telah dilewatkan proses training sebelumnya. Gambar 2.6 menunjukkan isi dari matriks score yang memiliki nilai tiap kata dengan kata lain untuk menunjukkan kepentingan relasi antar kata.



Gambar 2.6  
Contoh Isi dari Matriks Score

Selain ketiga vektor itu, ada juga matriks *score* yang dibutuhkan self-attention, matriks ini cukup jelas kegunaannya berdasarkan namanya yaitu untuk memberi skor/nilai terhadap tiap kata. Skor tersebut menandakan seberapa besar fokus pada kata tersebut dengan kata-kata sekitarnya saat ini. Semakin tinggi nilainya, semakin besar fokus yang diberikan. Dan matriks score didapatkan dari perkalian *dot product* antara vektor query dan vektor key.

Matriks score tersebut jika dibiarkan dengan nilai yang relative tinggi, akan menyebabkan efek yang *exploding* kemungkinan besar seperti *exploding gradients* (permasalahan saat training model, dimana model akan memiliki network yang tidak stabil, weight dari model memiliki nilai yang terlalu besar sehingga terjadi overflow bahkan sampai nilai NaN). Karena ini, matriks score perlu diturunkan nilai nya dengan dibagi dengan akar pangkat panjang dimensi vektor key. Nilai matriks score yang baru ini akan membantu untuk menghasilkan nilai gradien yang lebih stabil. Langkah selanjutnya adalah matriks score yang baru akan dilewatkan fungsi aktivasi softmax. Seperti fungsi aktivasi yang lainnya, tujuannya adalah memuncakkan nilai yang tinggi dan merendahkan nilai yang kecil, dengan representasi tiap nilai antara 0 dan 1. Hasil akhir dari softmax ini akan dipanggil *attention weights*. Attention weights tersebut akan dikalikan dengan vektor value yang menjadi output vektor dari bagian encoder ini. Dengan output terakhir ini, nilai tinggi dari attention weights (output softmax) tadi akan memberi dampak tinggi pada kata-kata yang berarti, dan menurunkan nilai/dampak pada kata-kata yang tidak relevan.

Agar dapat diimplementasikan kepada multi-head attention, tiap operasi self-attention ini (yang disebut juga sebagai *head*) akan terjadi sebanyak N kali dan untuk tiap head mendapatkan vektor query, key dan value yang telah dibagi sebanyak N vektor. Output untuk setiap head akan disambung menjadi 1 vector yang sama. Secara teori, dengan tiap head/self-attention melakukan perhitungan yang berbeda, maka informasi yang didapatkan untuk melakukan *decoding* akan menjadi lebih banyak/besar. Dan dapat disimpulkan multi-head attention adalah self-attention yang dilakukan sebanyak N kali (nilai N adalah parameter yang dapat ditentukan sendiri). Dari penjelasan sebelumnya langkah-langkah self-attention didapatkan rumus sebagai berikut:

(2.1)

(2.2)

(2.3)

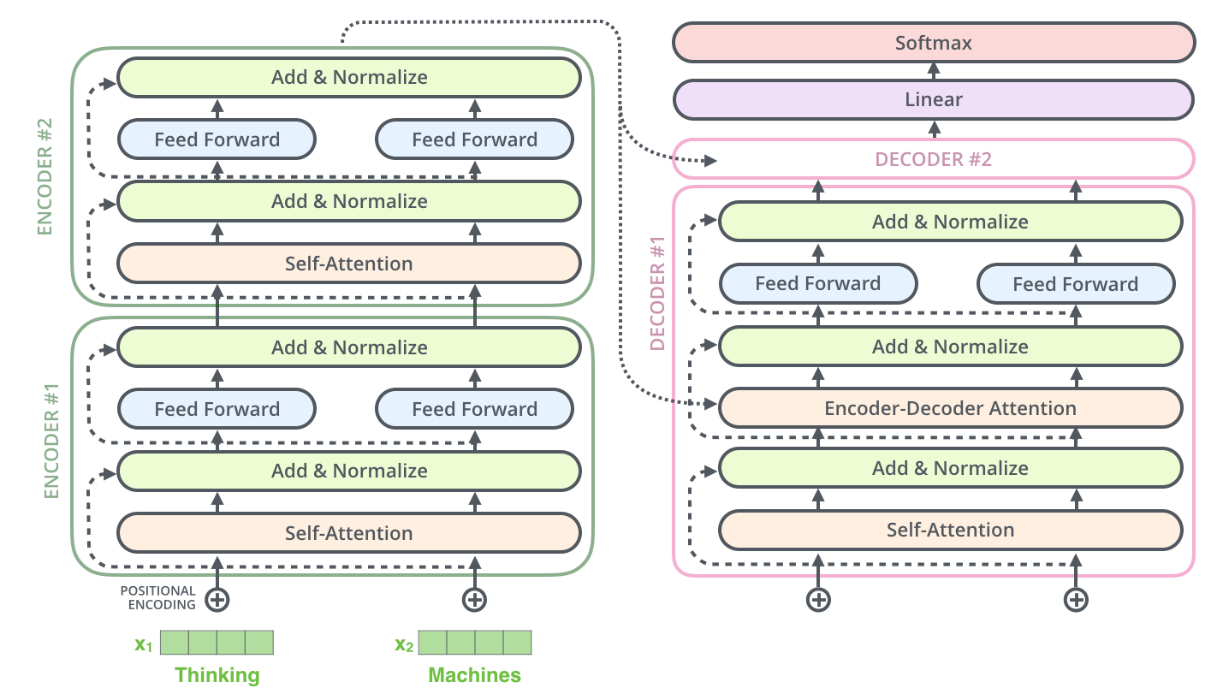
Sebelum lanjut kepada bagian kedua yaitu FFN, perlu diketahui adanya sublayer setelah multi-head attention dan FFN. Sublayer ini adalah *residual connection* dan normalisasi. Residual connection adalah proses menambahkan *positional encoding* dengan input. Selain di dua sublayer tersebut, positional encoding dapat ditemukan saat sebelum word embedding input dimasukkan kepada multi-head attention. Gunanya positional encoding ini adalah memberi informasi posisi dari input embedding nya. Melihat rumus (2.4) dan (2.5), secara mudah disimpulkan untuk indeks yang ganjil akan dibuatkan vector dengan rumus cos, sedangkan untuk yang genap akan dibuat dengan rumus sin. Setelah residual connection, hasil itu dilanjutkan kepada normalisasi atau LayerNorm.

(2.4)

(2.5)

Setelah mengenal residual connection dan LayerNorm, penjelasan akan lanjut kepada FFN. Input dari FFN merupakan output dari residual connection dan LayerNorm, perlu diingat input dari proses residual connection dan LayerNorm tersebut adalah output multi-head attention. Struktur FFN ini adalah dua layer linear dengan fungsi pengaktifan ReLU diantarnya. Dan seperti yang disebut, setelah FFN ini output tersebut dilewatkan proses residual connection dan LayerNorm sekali lagi.

Sebagai penutup subbab ini, encoder merupakan bagian yang menghasilkan representasi kata yang baru dengan informasi dari mekanisme attention. Hal ini membantu decoder untuk fokus kepada kata-kata penting yang menjadi input. Dengan banyaknya encoder yang ditumpuk, tiap encoder dapat mempelajari informasi/representasi yang berbeda tiap hitungan. Multi-head attention berguna untuk mencari informasi dengan pandangan attention, residual attention menjadi faktor pembantu training model karena membantu gradien untuk mengalir lewat network dengan lancar. LayerNorm digunakan untuk memberi kestabilan pada network, dengan network yang stabil dapat mengurangi durasi training, dan FFN melakukan proyeksi/analisa terhadap inputnya sehingga berpotensi untuk memberikan representasi dengan informasi yang banyak. Gambar 2.7 adalah penggambaran isi secara detail mengenai layer encoder dan decoder.



Gambar 2.7  
Representasi Tiap Layer Dari Arsitektur Transformer[[6]](#footnote-6)

### Bagian Decoder

Decoder memiliki tujuan untuk mengambil hasil *encoding* dan generasi *text sequence*. Strukturnya mirip dengan encoder namun diantara multi-head attention dengan FFN terdapat satu multi-head attention lagi tetapi dengan mekanisme masking ditambahkan, juga seperti biasa terdapat sublayer residual connection dan normalisasi setelahnya. Dan pada akhir dari decoder yaitu setelah FFN dan sublayer terakhirnya, diberikan layer linear sebagai *classifier* dan fungsi softmax untuk mendapatkan probabilitas katanya. Dalam Transformer, bagian decoder ini bersifat autoregressive, karena ini dibutuhkan masking dalam salah satu multi-head attentionnya. Tetapi karena pada paper yang dirujuk oleh tugas akhir ini menggunakan decoder yang non-autoregressive, masked multi-head attention tidak akan dibahas.

Bagian decoder dimulai dari input dengan langkah yang sama dengan encoder. Input kata-kata akan dirubah menjadi word embedding, yang kemudian akan ditambahkan dengan positional embedding. Embedding tersebut akan dimasukkan ke layer multi-head pertama tanpa melupakan pembuatan vektor query, key dan value juga matriks score. Kemudian matriks score yang sudah dilewatkan fungsi pengaktifan softmax akan dikalikan denagn query value dan menghasilkan output terakhir multi-head attention yang pertama.

Sebenarnya dalam paper “All You Need is Attention”, seperti yang disebut diawal subbab ini, terdapat masked multi-head attention. Di mana output dari multi-head attention ini terdapat nilai yang masked bertujuan karena sifat autoregressive decodernya, sehingga membutuhkan solusi untuk menghindari informasi yang bocor (*leakage*). Karena paper yang dirujuk pada tugas akhir ini dibuat dengan sifat non-autoregressive decoder, maka permasalahan itu tidak ada, dan tanpa masking decoder ini bisa mendapatkan nilai ketergantungan antar entity nya.

Setelah multi-head pertama, tentu ada sublayer residual connection dan normalisasi. Kemudian dilanjutkan dengan satu multi-head attention tetapi bagian ini menggunakan mekanisme bukan self-attention namun *cross-attention*. Perbedaan dari self-attention dan cross-attention ini hanya berada di pembuatan vektor query, key dan value. Sebelumnya pada self-attention ketiga vektor ini dibuatkan sendiri dengan weight yang sudah ditentukan dan input yang diterima. Tetapi untuk cross-anttention, vektor query dan key didapatkan dari output paling akhir bagian tumpukan encodernya. Dan vektor valuenya diambil dari output multi-head attention decoder yang pertama. Dengan cara ini, cross-attention dapat menyocokkan input encoder dengan input decoder dan decoder mempelajari input encoder bagian mana yang perlu difokuskan. Output dari multi-head attention kedua ini akan dilanjutkan kepada layer FFN yang sebelumnya terdapat sublayer yang sudah disebut sebelum-sebelumnya.

Dan pada tahap paling akhir yaitu layer linear dan layer softmax pada akhir decoder. Kedua layer ini ditempatkan setelah seluruh tumpukan decoder telah dilewatkan. Layer linear seperti yang telah disebut, berperan sebagai *classifier* sebesar *class* yang kita miliki. Untuk visualisasi, diberikan contoh 100 kelas untuk 100 kata. Maka output dari layer linear memiliki ukuran sepanjang 100. Output dari layer ini dilewatkan kepada layer softmax yang menghasilkan nilai probabilitas antara 0 dan 1. Dan nilai yang tertinggi adalah kata yang telah diprediksikan. Tahap decoding tidak berakhir disini. Hasil dari output prediksi tersebut akan ditambahkan kepada daftar input decoder yang paling pertama, sehingga input decoder tidak hanya input encoder tetapi juga *concat* dari output yang telah diprediksikan sebelumnya.

## Neural Network (NN)

Neural Network (NN) merupakan arsitektur paling sering digunakan dalam dunia *machine learning* dan *deep learning*. Struktur dari NN terinspirasi dari struktur otak manusia yang menggunakan neuron untuk menyalurkan informasi dari satu ke yang lainnya. Setiap neuron dalam NN direpresentasikan sebagai *node* yang akan menjalankan penghitungan fungsi matematika linear yang memiliki nilai informasi yang didapatkan dari input nya. Dapat dilihat contoh fungsi linear yang digunakan dalam rumus (2.6), *x* adalah nilai input yang diterima, dan *w* sebagai nilai *weight* yang diterima. Nilai node tersebut kemudian dilewatkan dengan *activation function* (fungsi yang meningkatkan sebuah nilai yang melewati sebuah *threshold*, dan menurunkan nilai yang tidak melewati *threshold* tersebut), contohnya pada rumus (2.7) dengan *a* sebagai activation function.

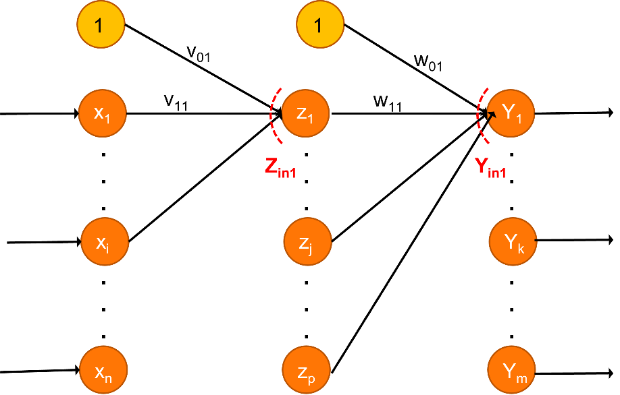
(2.6)

(2.7)

Tiap *layer* (lapisan) dari NN ini akan terdiri dari node yang saling berhubungan, dan tiap node memiliki sifat regresi linear (menghitungkan nilai prediksi berdasarkan nilai variabel yang ada sebelumnya). Untuk struktur dari Neural Network sendiri dapat dibagi menjadi 3 bagian / layer: *input layer*, *hidden layer*, *output layer*. Penggambaran dan penjelasan dari ketiga layer tersebut dapat dilihat pada subbab berikut mengenai Multi-Layer Perceptron (MLP).

### Multi-Layer Perceptron (MLP)

Sebelum membahas MLP perlu mengetahui mengenai *Feed Forward Neural Network* (biasa disingkat menjadi FFN). FFN adalah bentuk neural network yang paling pertama dan paling sederhana dibandingkan dengan NN yang sudah berkembang saat ini.[[7]](#footnote-7) Contoh alur ini dapat dilihat dari gambar 2.8. Di mana koneksi antar node di FFN tidak membentuk siklus, namun informasi atau nilai yang diberikan mengarah ke depan (*forward*). Dari input layer, melewati hidden layer dan berakhir pada output layer. FFN perlu diketahui terlebih dahulu karena MLP memiliki struktur yang mirip dengan FFN, tetapi memiliki perbedaan di mana MLP memiliki tiap layer adalah *fully connected layer* (setiap node berhubungan dengan semua node sebelumnya dan setelahnya) dan dalam beberapa kasus setiap layer memiliki jumlah node yang sama.



Gambar 2.8  
Visualisasi Arsitektur MLP dengan Backpropagation

MLP adalah hasil perkembangan[[8]](#footnote-8) dari algoritma yang dibuat Rosenblatt yaitu *Perceptron*.[[9]](#footnote-9) Alasan adanya MLP adalah untuk menghindari kesulitan Perceptron, yang hanya terdiri dari 1 neuron/node, tidak bisa mengaplikasikan data yang non-linear. MLP mengambil sisi FFN di mana input dikombinasikan dengan *weight* yang awalnya diinisiasi secara *random*. Dan mendapat sisi Perceptron yang mengambil nilai input yang dikalikan dengan weight dan diberikan kepada sebuah activation function.

MLP menggunakan *Backpropagation* sebagai metode trainingnya, terdapat 3 tahap besar[[10]](#footnote-10) yaitu: Forward Pass, Loss Calculate, dan Backward Pass. Forward Pass adalah tahap yang telah dibahas sebelumnya dapat bab Neural Network, di mana MLP akan menghitung nilai output dari rumus (2.6) dari layer pertama sampai akhir. Pada Loss Calculate, sesuai dengan namanya, akan melakukan penghitungan jauhnya perbedaan antara nilai output saat ini dengan output sebenarnya. Banyak jenisnya loss function yang bisa digunakan, secara umum MLP menggunakan rumus Cross-Entropy, tetapi penjelasan di bab ini akan menggunakan *Sigmoid Function* (dapat di lihat pada rumus (2.8)) untuk loss function nya. Dengan variabel adalah nilai loss function, sebagai target output yang diinginkan, nilai node saat itu, sebagai turunan pertama activation function.

(2.8)

(2.9)

Rumus (2.9) adalah rumus penghitungan nilai gradien berdasarkan nilai loss function yang didapatkan. Nilai gradien adalah cara mengevaluasikan nilai loss function, yang kemudian nilai ini akan digunakan dalam proses perubahan weight. sebagai nilai gradien, merupakan nilai *learning rate* dan sebagai nilai node sebelumnya. Tahap yang terakhir adalah tahap backward pass, tahap yang akan melakukan perubahan pada weight sesuai dengan loss function dan gradien yang telah hitung. Untuk penjelasan ini, rumus yang digunakan untuk dapat dilihat pada rumus. Rumus tersebut diambil dari nilai–nilai sebelumnya yang telah dihitung atau ditemukan.

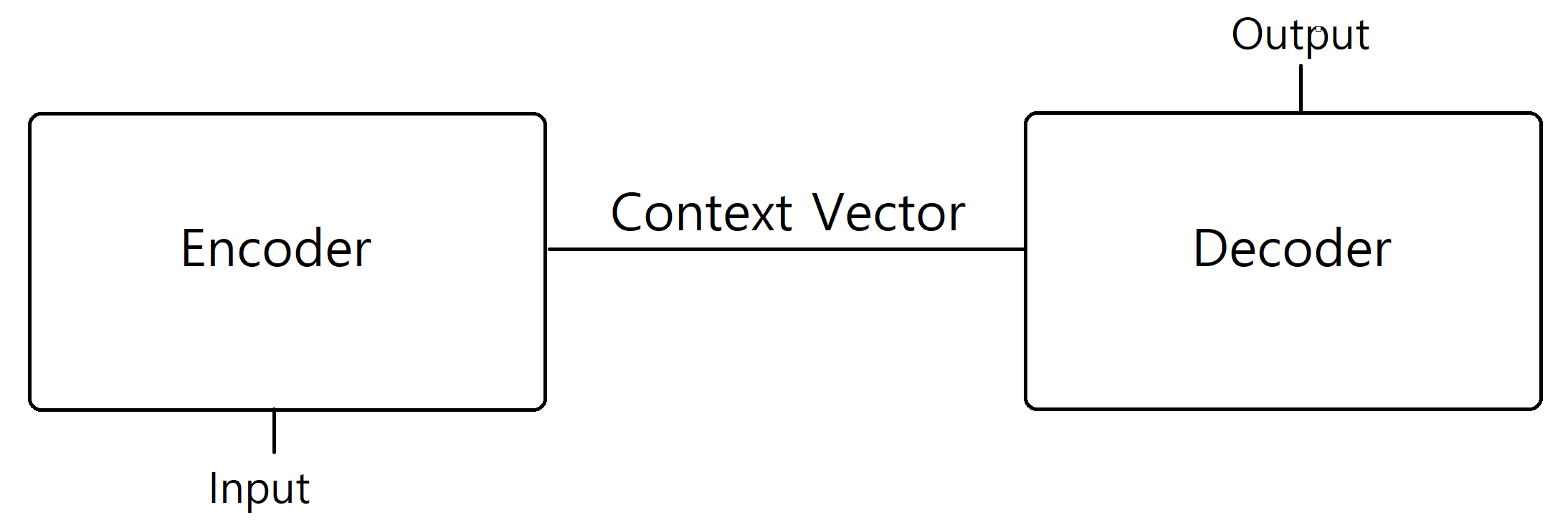
(2.10)

Rumus-rumus yang telah disebut dihitung untuk tiap node yang ada dalam arsitektur MLP, dan tahap-tahap tersebut dilakukan berulang kali sampai iterasi yang telah ditentukan atau saat perubahan nilai gradien bersifat konvergen. MLP sering digunakan dalam dunia machine learning sebagai *classifier* atau penentuan kelas / kategori dari suatu input karena secara hasil MLP terbukti efektif untuk menjadi classifier. Seperti contohnya penggunaan MLP pada bidang pengenalan suara, pengenalan gambar, dan perangkat lunak terjemahan mesin[[11]](#footnote-11) meskipun pada akhirnya *support vector machines* (SVM) lama kemudian lebih dipilih untuk *classifying*. Namun juga saat ini algoritma backpropagation yang digunakan dalam MLP tetap digunakan karena popularitasnya dalam deep learning.

## Sequence to Sequence Models

Adanya subbab seq2seq karena metode sequence-to-set dari tugas akhir ini terinspirasi dari arsitektur seq2seq. Karena itu, perlu adanya pengenalan mengenai metode yang menginspirasikan metode tugas akhir ini. Sequence-to-sequence (seq2seq) model adalah salah satu jenis yang berasal dari arsitektur Recurrent Neural Network (RNN). RNN sendiri ada untuk menyelesaikan permasalahan dari input sequence (kumpulan kata, huruf, timestep, dsb) karena kemampuannya untuk mengingat informasi beberapa timestep/kata/huruf sebelumnya.

Metode seq2seq dikenalkan oleh pihak Google dan kemudian metode tersebut sering digunakan dalam kasus translasi bahasa, dapat diambil contoh dari salah satu penelitian dari Google sendiri mengenai penggunaan metode seq2seq untuk meningkatkan kualitas translasi bahasa.[[12]](#footnote-12) Atau penelitian lainnya dari Google penggunaannya dalam *speech recognition*. Penamaan sequence-to-sequence didapat dari input dan outputnya yaitu *sequence of items*, contoh dari bentuk dari sequence adalah input kalimat dan output kalimat. Sama seperti penggunaan translasi bahasa di mana input kalimat satu bahasa dan output menjadi bahasa lainnya dengan informasi atau konteks yang sama.



Gambar 2.9  
Bentuk Ringkasan Arsitektur Seq2Seq

Tiap bagian yang tercatat di Gambar 2.9 akan dibahas secara ringkas. Bagian encoder akan mengolah tiap token dari kalimat input untuk mendapatkan informasi sebanyak mungkin dan dijadikan sebuah vektor dengan panjang tetap, biasa disebut *context vector*. Context vector adalah vector yang mengandung nilai/inti/informasi dari input saat ini yang akan diberikan kepada decoder, dengan informasi tersebut decoder menerimanya sebagai input dan mulai menentukan prediksi yang akurat sesuai dengan context vector yang diberikan. Untuk decoder, telah disebutkan tugasnya untuk melakukan prediksi sesuai dengan context vector yang diberikan dan memberikan output sequence.

## Hungarian Match

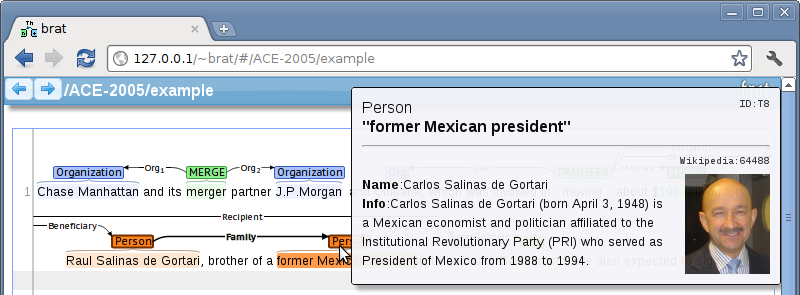
## BRAT

Untuk subbab BRAT ini akan fokus terhadap BRAT sebagai alat anotasi secara keseluruhan baik fitur yang digunakan maupun yang disediakan namun tidak digunakan di tugas akhir ini. BRAT adalah *web-based tool* anotasi untuk dokumen teks. Tujuan adanya BRAT ini untuk memudahkan anotasi dan pemberian catatan/*notes* pada suatu dokumen dengan memberikan juga dokumen anotasi/catatan dengan format yang tetap dan dapat dibaca oleh komputer. Salah satu contoh penggunaan dapat dilihat dari Gambar 2.10. Pada gambar diilustrasikan anotasi untuk anotasi *text-span* (yang digunakan untuk anotasi data Named Entity Recognition) dan anotasi *relation* (yang digunakan untuk anotasi data Relational Extraction). Beberapa jenis fitur untuk anotasi disediakan seperti, fitur *n-ary* associations, BRAT menyatakan dapat menghubungkan sejumlah anotasi lain yang berpartisipasi dalam peran tertentu. Contohnya n-ary associations adalah anotasi untuk jenis anotasi TRANSFER akan dihubungkan secara relation dengan 3 jenis anotasi yang lain dengan relasi yang berbeda-beda (dengan jenis anotasi Money berelasi Money, dengan Person berelasi Beneficary, dengan Org berelasi Recipient). Dan konfigurasi penentuan kategori anotasi, tipenya, bahkan juga peraturan (contohnya peraturan untuk relation Family harus terkoneksi dengan anotasi jenis Person) dapat dilakukan dengan mudah.



Gambar 2.10  
Contoh Anotasi Text-Span pada BRAT

Adapun fitur *normalization annotations* (gambar 2.11), fitur ini mengidentifikasi entitas yang dapat ditemukan dunia nyata yang dirujuk dalam teks yang dianotasikan merupakan bagian penting dalam menganalisis makna teks. Namun untuk fitur ini dibutuhkan persiapan data yang membantu BRAT mengetahui entitas yang ada di dunia nyata. Langkah-langkah tersebut dapat ditemukan dalam halaman https://brat.nlplab.org/normalization.html. Dan fitur terakhir yang menjadi tambahan kecil pada alat anotasi ini adalah penulisan catatan untuk tiap anotasi yang diberikan.



Gambar 2.11  
Contoh Normalization Annotation[[13]](#footnote-13)

Sudah ada banyak penelitian task yang dilakukan dengan bantuan BRAT, dan dalam web BRAT disebutkan beberapa task yang berhasil dan masih berjalan menggunakan BRAT dalam penelitiannya. Beberapa dari contoh penggunaan BRAT dalam task yang ada adalah *entity mention detection*, *event extraction*, *coreference resolution*, *normalization*, *chunking*, *dependency syntax*, *meta-knowledge*. Entity mention detection merupakan anotasi entitas dengan cara text-span yang telah disebut sebelumnya. Event extraction, anotasi yang digunakan untuk mengetahui relasi antar entitas. Coreference resolution adalah task yang dapat mengetahui relasi coreference antar entitas. Normalization telah dijelaskan sebelumnya. Chunking, task membagi teks menjadi anotasi yang tidak tumpang tindih dengan anotasi yang lainnya, anotasi chunking ini sering digunakna untuk mengetahui jenis kata dari suatu teks seperti label NP (Noun Phrase). Dependency syntax merupakan analisis sintaksis, tugas untuk menetapkan *binary relations* antara kata-kata untuk menandai hubungan yang bergantung pada kata utama mereka. Meta-knowledge, tugas mengidentifikasi bagaimana pernyataan faktual harus ditafsirkan, sesuai dengan konteks tekstualnya. Seluruh kegunaan/task yang disebut pada bagian ini dapat ditemukan informasi selengkapnya (contoh visualisasinya, dataset/penelitian yang memberi contoh sesuai) pada halaman website BRAT https://brat.nlplab.org/examples.html#corpus-examples-brat.

Selain fitur utama yang telah disebutkan sebelumnya, fitur yang digunakan secara umum dari BRAT juga banyak dan sangat berguna dalam perannya sebagai alat anotasi. Tugas akhir ini membatas penggunaan fitur sesuai yang dibutuhkan yaitu anotasi text-span saja. Juga dengan penggunaan fitur umumnya seperti visualisasi, alat anotasi BRAT (intuitive editing), zero setup, easy export, always saved, configurable, search. Untuk mempermudah penjelasan dan mempersingkat, penjelasan dapat dibaca sebagai berikut:

* Comprehensive visualization

Konsep “*what you see is what you get*”: semua aspek anotasi yang berdasarkan representasi secara visual dengan cara yang intuitif.

* Intuitive editing

Cara anotasi berbasis penggunaan *mouse* pada umumnya dan menggunakan gerakan intuitif yang familiar dari editor teks, perangkat lunak presentasi, dan banyak alat lainnya. Untuk menandai rentang teks, cukup pilih dengan mouse dengan menekan, menahan, dan menandakan kata-kata yang diinginkan atau dengan mengklik dua kali pada sebuah kata.

* Integration with external resources

Dapat menggunakan data eksternal, bukan berasal dari BRAT, seperti Freebase, Wikipedia, dan Open Biomedical Ontologies.

* Zero setup

BRAT dibuat sepenuhnya dengan teknologi web standar, dan tidak perlu meng-instal perangkat lunak lokal atau plugin browser apa pun untuk menggunakannya.

* Annotation of texts in any language

Memiliki *full Unicode support*, sehingga mendukung hampir 100 skrip yang berbeda.

* Integrated annotation comparison

BRAT mencakup sejumlah fitur untuk membandingkan beberapa set anotasi untuk dokumen yang sama, termasuk perbandingan otomatis untuk mengidentifikasi dan menandai perbedaan dan visualisasi berdampingan.

* An address for each annotation

Setiap anotasi BRAT dapat memiliki URL *address* secara unik. Bersama dengan URL server, memudahkan apabila memerlukan akses langsung terhadap suatu jenis anotasi tertentu (didapatkan pada tombol Link di menu modal apabila menekan atau melakukan anotasi baru).

* Integration with automatic annotation tools

Adanya fitur integrasi dengan metode canggih untuk dukungan anotasi dasar seperti pemisahan kalimat (Inggris dan Jepang) dan tokenization (Jepang).

* High-quality visualization at any scale

Visualisasi anak nakal didasarkan pada Scalable Vector Graphics (SVG), yang dapat di-render dalam detail dan presisi yang diinginkan

* Easy export in multiple formats

Anotasi yang dibuat di brat dapat dieksport dengan mudah dalam format standoff sederhana yang dapat dengan mudah dianalisis, diproses, dan dikonversi ke format lain.

* Always saved, always up to date

BRAT menghilangkan risiko kehilangan anotasi apabila terjadi *crash*, lupa untuk menyimpan pekerjaan, atau bahkan kegagalan total komputer annotator dengan melakukan operasi edit dari annotator ke server brat saat selesai.

* Real-time collaboration

Arsitektur dan desain dari BRAT client-server memungkinkan beberapa annotator untuk bekerja secara bersamaan pada kumpulan dokumen yang sama, atau bahkan pada dokumen yang sama, melihat pengeditan satu sama lain.

* Detailed annotation process measurement

Secara opsional, BRAT dapat dikonfigurasi untuk merekam waktu yang tepat saat annotator membuka dokumen, setiap tindakan edit, dan bahkan waktu yang dihabiskan untuk memilih jenis yang akan ditetapkan ke anotasi setelah memilih tempat untuk menempatkannya.

* Rich set of annotation primitives

BRAT menyediakan serangkaian kategori dasar anotasi yang beragam : anotasi untuk text-span, binary relations, equivalence classes, n-ary associations dan attributes.

* Fully configurable

Semua konfigurasi anotasi menggunakan bahasa konfigurasi sederhana. Setiap kumpulan dokumen memiliki konfigurasinya sendiri, memungkinkan satu server BRAT untuk meng-host banyak proyek dengan target anotasi yang berbeda. Selain itu, sebagian besar visualisasi seperti font, anotasi warna kotak dan busur serta kepala panah dan gaya menggambar busur dapat dikontrol secara detail menggunakan spesifikasi gaya HTML/CSS yang terdokumentasi dengan baik dan dikenal luas.

* Always validated

BRAT memiliki validasi anotasi yang mampu memeriksa semua batasan yang dapat didefinisikan dalam konfigurasi ekspresifnya.

* Search

BRAT mengimplementasikan serangkaian fungsi lengkap untuk mencari dokumen atau koleksi dokumen untuk anotasi jenis apa pun dengan serangkaian batasan yang dapat dikonfigurasi secara terperinci.

* Concordancing

BRAT mendukung key-word-in-context (KWIC) untuk tampilan search berdasarkan kata

1. Arliyanti Nurdin, dkk, Perbandingan Kinerja Word Embedding Word2vec, Glove, Dan Fasttext Pada Klasifikasi Teks, Jurnal TEKNOKOMPAK Vol. 14 No. 2 (2020), hal. 74—79. [↑](#footnote-ref-1)
2. Xiaoya Li, dkk, A Unified MRC Framework for Named Entity Recognition, Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (Juli 2020), Hal 5849 [↑](#footnote-ref-2)
3. Plato, *“Cara Menggunakan Pengenalan Entitas Bernama (NER) Untuk Ekstraksi Informasi”*, (https://zephyrnet.com/id/cara-menggunakan-nama-pengenalan-entitas-untuk-ekstraksi-informasi/) [↑](#footnote-ref-3)
4. Ashish Vaswani, dkk, Attention Is All You Need, 2017. [↑](#footnote-ref-4)
5. Michael Phi, “*Illustrated Guide to Transformers- Step by Step Explanation*”, https://towardsdatascience.com/illustrated-guide-to-transformers-step-by-step-explanation-f74876522bc0 [↑](#footnote-ref-5)
6. Jay Alammar, “*The Illustrated Transformer*”, <http://jalammar.github.io/illustrated-transformer/> diakses, 4 Mei 2022, 12:32 [↑](#footnote-ref-6)
7. Schmidhuber, Jürgen (2015-01-01). "Deep learning in neural networks: An overview". Neural Networks. 61: 85–117 [↑](#footnote-ref-7)
8. Minsky M. L. and Papert S. A. 1969. Perceptrons. Cambridge, MA: MIT Press [↑](#footnote-ref-8)
9. Frank Rosenblatt. The Perceptron, a Perceiving and Recognizing Automaton Project Para. Cornell Aeronautical Laboratory 85, 460–461 (1957) [↑](#footnote-ref-9)
10. Prof. Dr. Ir. Kuswara Setiawan, M.T., Buku Paradigma Sistem Cerdas, (Malang : Bayu Media, 2003) [↑](#footnote-ref-10)
11. Wasserman, P.D., Schwartz, T., Neural networks. II. What are they and why is everybody so interested in them now?, IEEE Expert, 1988, Volume 3, Issue 1, Hal. 10-15 [↑](#footnote-ref-11)
12. Ilya Sutskever, Oriol Vinyals, Quoc V. Le, Sequence to Sequence Learning with Neural Networks, 2014 [↑](#footnote-ref-12)
13. BRAT, mini-introduction to brat, https://brat.nlplab.org/introduction.html [↑](#footnote-ref-13)