BAB II
TINJAUAN PUSTAKA

2.1 Kajian Terkait

Kajian terkait beberapa penelitian terdahulu yang telah melakukan penelitian tentang mesin penerjemah jaringan saraf tiruan (NMT) diantaranya adalah penelitian yang dilakukan oleh (Bahdanau et al., 2015) yang berjudul "Neural Machine Translation By Jointly Learning To Align And Translate" dengan melakukan penelitian penerjemahan dari bahasa Inggris ke Bahasa Francis, menggunakan korpus gabungan dengan jumlah 348 juta dengan pelatihan yang sama dan dataset yang sama untuk ekperimen dua model yaitu model dengan panjang kalimat sampai 30 kata dan panjang kalimat sampai 50 kata dengan mekanisme *attention* dan tanpa *attention*. Dari dua model ini menghasilkan akurasi tertinggi pada model 50 kata dengan *attention* sebesar 28.45 BLEU poin untuk terjemahan semua kalimat, 36.15 BLEU poin untuk terjemahan tanpa kata yang tidak diketahui dan model 30 kata dengan attention sebesar 21.50 BLEU poin untuk terjemahan semua kalimat, 31.44 BLEU poin untuk terjemahan tanpa kata yang tidak diketahui. Hasilnya penerjemahan jaringan saraf tiruan lebih baik seperti yang diharapkan daripada penerjemahan lain yang ada saat ini.

Penelitian lainnya yang dilakukan oleh (Luong et al., 2015) yang berjudul "*Effective Approaches to Attention-based Neural Machine Translation*" melakukan penerjemahan dua arah dari bahasa Inggris ke bahasa Jerman pada mesin penerjemah jaringan saraf tiruan (NMT) dengan mekanisme *attention* pendekatan *global* yang menggunakan semua masukan kalimat sumber dan *local* yang hanya

menggunakan subset atau sebagian masukan kalimat sumber dalam satu waktu, dengan model ini menghasilkan nilai akurasi sebesar 25,9 BLUE poin.

Menurut penelitian (Abidin et al., 2018) yang berjudul "Translation of Sentence Lampung-Indonesian Languages with Neural Machine Translation Attention Based Approach". Menjelaskan Penerjemahan bahasa Lampung ke bahasa Indonesia dengan pendekatan NMT *attention* menggunakan 3000 kalimat pada korpus paralel dan konfigurasi dimensi yang terbaik diperoleh pada dimensi *word embedding* (penyematan kata) sebesar 620 dan dimensi *hidden layer* (lapisan tersembunyi) pada bagian *encoder* dan *decoder*-nya sebesar 1000 dengan perolehan rata-rata nilai akurasi otomatis BLEU sebesar 51.96 %.

Penelitian lainnya yang dilakukan oleh (Gunawan et al., 2021) yang berjudul "Analisis Perbandingan Nilai Akurasi Mekanisme *Attention* Bahdanau dan Luong pada *Neural Machine Translation* Bahasa Indonesia ke Bahasa Melayu Ketapang dengan Arsitektur *Recurrent Neural Network*". Penelitiannya melakukan penerjemahan satu arah dari bahasa Indonesia ke bahasa Melayu Ketapang dengan korpus paralel sejumlah 5000 baris kalimat, dengan hasil pengujian penggunaan penambahan epoch secara konsisten dengan nilai akurasi skor bleu pada attention Bahdanau sebesar 35,96% pada jumlah epoch 40, sedangkan pada attention Luong dengan nilai akurasi 26,19% pada jumlah epoch 30.

Berdasarkan studi literatur yang telah dirangkumkan dapat disimpulkan bahwa penelitian ini akan mengimplementasikan mesin penerjemah jaringan saraf tiruan (NMT) dengan arsitektur RNN encoder-decoder dan penambahan jumlah epoch secara konsisten pada mekanisme attention Bahdanau dalam penerjemahan bahasa Indonesia ke bahasa Tiochiu Pontianak berdasarkan hasil evaluasi otomatis BLUE (Bilingual Evaluation Understudy) dan pengujian oleh ahli bahasa. Penelitian ini bertujuan untuk mengetahui akurasi mekanisme Attention Bahdanau yang menghasilkan nilai akurasi terhadap mesin penerjemah jaringan saraf tiruan (NMT) pada penerjemahan bahasa Indonesia ke bahasa Tiochiu Pontianak.

Adapun perbandingan attention yang dilakukan oleh peneliti dapat dilihat pada tabel 2.1 berikut.

Tabel 2. 1 Tabel Perbandingan Attention Oleh Peneliti

No.	Penulis	Judul Penelitian	Keterangan
1.	Bahdanau dkk	*Neural Machine Translation By Jointly Learning To Align And Translate*	-Korpus paralel dengan jumlah 348 juta kalimat. -Eksperimen dengan dua model yang memiliki panjang kalimat sampai 30 dan 50 kata. -Model dengan *attention* dan tanpa *attention.* -Menggunakan *function score alignment additive/concat.* -NMT diimplementasikan menggunakan Theano. -Mengukur akurasi skor BLEU dengan dan tanpa dengan UNK token. -Pada WMT'14 korpus Inggris ke Prancis, model akurasi tertinggi dihasilkan mencapai skor 36.15 BLEU poin.
2.	Luong dkk	*Effective Approaches to Attention-based Neural Machine Translation*	-Korpus paralel dengan 4,5 juta kalimat. -Eksperimen penerjemahan dua arah bahasa Inggris ke Jerman. -Pendekatan model *global attention* dan *local attention.* -Menggunakan *function score alignment* dengan *dot product, general, additive/concat,* dan *location-based*. -NMT diimplementasikan menggunakan Matlab. -Pada WMT'15 korpus Inggris ke Jerman, model akurasi tertinggi dihasilkan mencapai skor 25.9 BLEU poin.
3.	Abidin dkk	Penerjemahan Kalimat Bahasa Lampung-Indonesia Dengan Pendekatan *Neural Machine Translation* Berbasis *Attention*	-Korpus paralel dengan jumlah 3000 kalimat bahasa Lampung ke bahasa Indonesia. -Pengujian menggunakan 25 kalimat tunggal tanpa *out of vocabulary* (OOV), 25 kalimat tunggal dengan OOV, 25 kalimat majemuk tanpa OOV dan 25 kalimat majemuk dengan OOV. -NMT diimplementasikan menggunakan Theano.

			-Akurasi yang dihasilkan dengan BLEU adalah 51.96 %.
4.	Gunawan dkk	Analisis Perbandingan Nilai Akurasi Mekanisme *Attention* Bahdanau dan Luong pada *Neural Machine Translation* Bahasa Indonesia ke Bahasa Melayu Ketapang dengan Arsitektur *Recurrent Neural Network*	-Korpus paralel dengan jumlah 5000 kalimat bahasa Indonesia ke bahasa Melayu Ketapang. -Eksperimen penerjemahan satu arah bahasa Indonesia ke bahasa Melayu Ketapang. -Eksperimen membandingkan nilai akurasi *attention* Bahdanau dengan *attention* Luong. -Menggunakan *function score alignment additive/concat* pada *attention* Bahdanau dan *dot product* pada *attention* Luong. -NMT diimplementasikan menggunakan *Framework deep learning Tensorflow*. -Menggunakan hyperparameter penambahan jumlah *epoch* secara konsisten. -Pengujian Menggunakan *K-Fold Cross Validation* pada BLEU dan Ahli Bahasa. -Akurasi yang dihasilkan dengan BLEU pada *attention* Bahdanau sebesar 35,96% pada jumlah epoch 40, sedangkan pada *attention* Luong dengan nilai akurasi 26,19% pada jumlah epoch 30.

2.2 Penerjemahan

Penerjemah Dalam Kamus Besar Bahasa Indonesia (KBBI) kata "terjemah/menerjemahkan" adalah menyalin (memindahkan) suatu bahasa ke bahasa lain atau mengalih bahasakan. Selain itu, (Hariyanto, 1996) mengemukakan bahwa "Terjemahan adalah kesempurnaan dari teks yang terikat bahasa teks yang tidak mungkin terjadi. Penerjemahan yang berfokus pada tujuan penulisan teks sumber adalah yang pasti terjadi". Hal ini dapat dibuktikan dengan banyaknya terjemahan karya sastra menjadi bahasa lainnya. McGuire (1980:2) menyatakan bahwa "*Translation involves the rendering of a source language (SL) text into the target language (TL) so as to ensure that (1) the surface meaning of the two will be*

approximately similar and (2) the structure of the SL will be preserved as closely as possible, but not so closely that the TL structure will be seriously distorted." yang diartikan sebagai penerjemahan melibatkan usaha pengalihan teks dari suatu bahasa sumber (BSu) ke dalam bahasa sasaran/target (BT) dengan mempertimbangkan bahwa (1) makna hubungan dari kedua teks kurang lebih sama dan (2) struktur bahasa dari hubungan bahasa sumber dapat dipertahankan sedekat mungkin, tetapi tidak terlalu dekat untuk menghindari penyimpangan pada struktur bahasa sasaran. Yang dimaksud bahasa sumber (BSu) yang akan diterjemahkan disebut *Source Language*, dan sedangkan *receptor language* adalah bahasa target (BT) hasil terjemahan.

Sebuah proses dalam mengalih bahasakan yang berarti mengalih eja suatu bahasa secara tulisan dari satu bahasa ke bahasa lain tanpa mengubah pesan yang ingin disampaikan merupakan pengertian penerjemahan, baik terjadi perubahan dalam bentuk klausa, frasa, kalimat dan paragraf. Nida dan Taber (1982) Penerjemahan merupakan kegiatan mengungkap kembali pesan dari bahasa sumber dengan padanan alami terdekat dalam bahasa sasaran dengan memperhatikan dari segi makna dan segi gaya bahasa. Definisi ini menegaskan bahwa yang harus dipertahankan pesan dan gaya bahasa, bukan struktur atau bentuk kata.

2.3 Mesin Penerjemah

Mesin penerjemah hadir sebagai salah satu solusi untuk menghubungkan interaksi antar-suku bahasa. Mesin penerjemah merupakan cabang penting dari pemrosesan bahasa alami yang bertujuan untuk menerjemahkan bahasa alami menggunakan komputer. Penerapan mesin penerjemah (MT) di Indonesia sudah dan masih banyak dilakukan dengan berbasis statistik (SMT) khususnya dalam eksperimen penerjemahan bahasa daerah. Dalam beberapa tahun terakhir, mesin penerjemah jaringan saraf tiruan (NMT) telah mencapai kesuksesan yang luar biasa dan menjadi metode pilihan baru dalam praktik mesin penerjemah bahkan sampai sekarang sudah diterapkan di google. Mesin penerjemah merupakan sebuah media yang memproses perpindahan satu bahasa ke bahasa lain. Mesin Penerjemah adalah perangkat media yang memproses mengalih bahasa yang satu dengan bahasa lainnya dalam bentuk teks maupun suara. Pendekatan jaringan saraf tiruan adalah pendekatan yang digunakan untuk menanggulangi kekurangan dari pendekatan

sebelumnya, seperti pendekatan berbasis statistik. Keuntungan dari penerapan pendekatan jaringan saraf tiruan pada mesin penerjemah adalah dapat menangani pengurutan kata yang lebih baik dan pendekatan informasi makna yang mendalam.

Mesin Penerjemah (*Machine Translation*) merupakan alat/perangkat yang dapat melakukan proses penerjemahan otomatis pada sebuah kata, teks atau kalimat dari satu bahasa ke bahasa lainnya. Mesin Penerjemah berfungsi untuk membantu memudahkan manusia dalam berkomunikasi antara satu sama lainnya yang memiliki bahasa yang berbeda.

Sama halnya seperti mesin penerjemah statistik, mesin penerjemah jaringan saraf tiruan parameternya diambil dari hasil analisis korpus teks paralel juga. Kunci perbedaannya ialah keluaran dari terjemahan dihasilkan setelah menyelesaikan masukan bahasa sumber, karena kata pertama dari kalimat terjemahan sangat mungkin memerlukan informasi/pendekatan yang diambil dari urutan input secara lengkap. Korpus paralel adalah pasangan korpus yang berisi kalimat-kalimat dalam suatu bahasa dan terjemahannya. Korpus berisi teks paralel yang merupakan hasil text mining (kumpulan data dalam bentuk teks) yang memperoleh pola berupa pasangan teks dari suatu bahasa ke bahasa lain. Penerjemahan dengan metode mesin penerjemah jaringan saraf tiruan untuk menghasilkan terjemahan yang lebih baik dengan syarat kualitas korpus paralel yang dimasukkan ke dalam sistem mempunyai kualitas yang baik dan cukup banyak jumlahnya.

2.4 Korpus

Korpus merupakan kumpulan teks alami, maupun bahasa lisan ataupun bahasa tulis yang disusun secara sistematis. Dikatakan alami karena teks yang dikumpulkan merupakan teks yang diproduksi dan digunakan secara wajar dan tidak dibuat-buat. Korpus adalah sekumpulan contoh bahasa teks tulis atau lisan dalam bentuk data teks yang dapat dibaca dengan menggunakan seperangkat mesin dan dapat diberi catatan berupa berbagai bentuk informasi linguistik. Informasi linguistik adalah kajian ilmu yang mempelajari tentang bahasa baik dari bentuk bahasa, makna bahasa, dan bahasa dalam konteks pendekatan dan sudut pandang penelitian. (McEnery et al., 2006) juga merangkum kriteria korpus, yang telah menjadi kesepakatan banyak ahli, yakni:

1) Dapat dibaca dengan menggunakan seperangkat mesin.
2) Berupa teks otentik.
3) Digunakan sebagai sampel.
4) Mewakili bahasa atau variasi bahasa tertentu.

Pembuatan korpus memerlukan usaha dan energi yang cukup besar untuk dilakukan. Korpus memiliki bagian dokumen teks dalam jumlah yang cukup banyak, bisa sampai jutaan dokumen. Pengumpulan puluh ribuan dan jutaan dokumen memerlukan waktu yang lama, pemilihan kata dan bahasanya harus tepat, kalimatnya harus efektif dan teks dalam korpus tersusun secara sistematis.

2.5 Bahasa Tiochiu Pontianak

Indonesia bagian Kalimantan Barat terdiri dari 14 kabupaten serta kota, yaitu Kota Pontianak, Kabupaten Kubu Raya, Kabupaten Mempawah, Kota Singkawang, Kabupaten Sambas, Kabupaten Bengkayang, Kabupaten Landak, Kabupaten Sanggau, Kabupaten Sekadau, Kabupaten Sintang, Kabupaten Melawi, Kabupaten Kapuas Hulu, Kabupaten Ketapang dan Kabupaten Kayong Utara. Di Kalimantan Barat juga dihuni banyak suku, diantaranya suku Dayak, Melayu, Bugis, Tionghoa, Madura, dan masih banyak yang lainnya. Begitu juga masalah bahasa, Kalimantan Barat memiliki banyak ragam bahasa dan setiap daerah didominasi bahasa yang berbeda beda. Ada bahasa khek, dayak ahe, melayu, tiochiu dan lain-lain. Bahasa Tiochiu di Kalimantan Barat hampir sama semua dalam aspek istilahnya tetapi yang membedakannya adalah ucapan dialek dengan nada yang berbeda-beda.

Bahasa Tiochiu merupakan salah satu bahasa etnis yang banyak digunakan orang Tionghua Pontianak di Kalimantan Barat Indonesia, walaupun penutur Tiochiu banyak tersebar di berbagai wilayah Indonesia, mayoritas daerah penutur bahasa Tiochiu di Indonesia diantaranya Kota Pontianak, Kabupaten Ketapang Kalimantan Barat, Jambi, Riau, Kepulauan Riau, Sumatra Utara, dan Sumatra Selatan. Bahasa Tiochiu di Indonesia berasal dari berbagai kota di provinsi Guangdong, Republik Rakyat Tiongkok diantaranya kota Jieyang, Chaozhou dan Shantou. Dan daerah asal orang Tiochiu biasanya disebut Chaoshan (gabungan dari kata kota Chaozhou dan Shantou) kedatangannya untuk merantau dan berdagang di

Indonesia hingga menetap dan menjadi salah satu suku/etnis di Indonesia. Sebagian besar penutur bahasa Tiochiu Indonesia banyak di wilayah Pontianak Selatan diantaranya daerah Jl. Gajah Mada, Jl. Purnama, Jl. Perdana, dan lainnya. Berdasarkan pencatatan sensus penduduk tahun 2019, penduduk Kota Pontianak berjumlah 665.017 jiwa, dari penilaian Bahasa Tiochiu pastinya banyak digunakan oleh etnis Tionghua. Penduduk Kota Pontianak didomisili oleh etnis Melayu (32,83%) dan yang kedua oleh etnis Tionghua (18,09%), dan lainnya dibawah. Dalam keseharian di Kota Pontianak, dapat diketahui dari etnis Tionghua yang menggunakan bahasa Tiochiu lebih banyak dibandingkan Hakka dan lainnya, dapat dilihat di perusahaan-perusahaan besar, usaha-usaha kecil, kantor, dan lainnya.

Bahasa Tiochiu Pontianak terkenal dengan dialek yang sulit di ikuti dan bahkan hampir setiap teks sama dengan arti yang berbeda dengan dialek berbeda. Contoh kalimat dalam bahasa Tiochiu Pontianak misalnya “Wa u chi no ciak kau” yang artinya “Saya ada pelihara dua ekor monyet” atau bisa “Saya ada pelihara dua ekor anjing”. Kata “Kau” bisa menjadi dua arti (anjing/monyet) tetapi dialeknya berbeda dalam konteks kalimat tersebut. Jika tidak didalam kalimat itu kata “Kau” memiliki banyak arti seperti: Sembilan, parit/kali, hubungan/kaitan, sampai, anjing, monyet, tebal, dan lainnya. Selain kata “Kau” kata lainnya sangat jarang sekali kata yang sama arti. Bahkan Bahasa Tiochiu masih ada kata kasar dan halus nya seperti: kata “makan” versi halus “ciak” dan versi kasar nya “chi that”. Ada juga tidak ada yang sama arti per kata dalam satu kalimat contohnya “Wa eng mopit khe pasat” yang berarti “Saya pergi ke pasar menggunakan motor”.

2.6 Mesin Penerjemah Jaringan Saraf Tiruan

Mesin penerjemah jaringan saraf tiruan dikenal dengan *neural machine translation* (NMT) adalah pendekatan mesin penerjemah yang menggunakan jaringan saraf tiruan atau biasa disebut *neural network* untuk memprediksi kemungkinan urutan masukan kata dan biasanya melakukan permodelan seluruh kalimat dalam satu model yang terintegrasi. Dengan proses pelatihan data korpus maka mesin penerjemah akan mengenali pola data teks yang dilatih, terjemahan semakin bagus jika ketersediaan data teks yang dilatih semakin banyak.

NMT menerjemahkan seluruh kalimat dalam satu waktu, bukan hanya memenggal atau membagi kata begitu saja. NMT dengan menerjemahkan satu

kalimat, hasil yang diterjemahkan lebih sesuai/cocok dengan makna konteks yang dimaksud. Dengan kata lain, NMT mampu menghasilkan bahasa yang lebih sesuai/cocok dengan bahasa manusia (natural) yang bekerja seperti otak manusia dengan membentuk banyak jaringan saraf pada mesin penerjemah. Jaringan saraf tiruan (NMT) menggunakan konteks yang lebih luas untuk mencari tahu terjemahan yang paling relevan atau berhubungan, yang kemudian menata/disusun kembali dan menyesuaikannya untuk menjadi lebih seperti layaknya berbicara dengan manusia menggunakan tata bahasa yang benar (Zein, 2018).

NMT berasal dari pendekatan frasa berbasis statistik yang digunakan secara terpisah sebagai subkomponen seperti pembuatan pemodelan bahasa (*language model*), penyelarasan kata (*word alignment*), dan pembuatan tabel frasa (*phrase table*). NMT bertujuan untuk memperkirakan *conditional distribution* yang tidak diketahui P=(y | x) memberikan dataset D dimana, y dan x adalah variabel acak yang mewakili output target dan input sumber. NMT pada umumnya didasarkan dengan fungsi utama pada framework *sequence to sequence*(seq2seq).

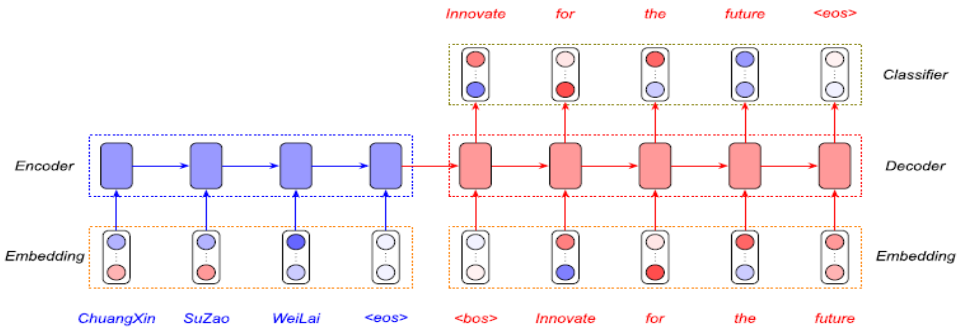
2.6.1 Gambaran Umum Mesin Penerjemah Jaringan Saraf Tiruan

2.6.1.1 *Modeling*

Penerjemahan dapat dimodelkan pada tingkatan yang berbeda, seperti dokumen, paragraf atau kalimat. Pada penelitian ini fokus pada tingkatan kalimat dalam penerjemahan. Selain itu, menganggap kalimat *input* sumber dan *output* target sebagai *sequence* atau urutan. Demikian model NMT dapat dilihat sebagai model *sequence to sequence*. Dengan asumsi kita berikan kalimat sumber $x = \{x1, x2, \dots, xn\}$ dan kalimat target $y = \{y1, y2, \dots, yn\}$ dengan menggunakan aturan *conditional distribution* dapat memfaktorkan dari kiri kekanan (L2R), seperti pada Persamaan 2.1 (Tan et al., 2020).

$$P(\mathbf{y}|\mathbf{x} = \prod_{t=1}^{T} P(y_t|y_0, \dots, y_{t-1}, \mathrm{x}). \qquad (2.1)$$

Hampir semua model mesin penerjemah jaringan saraf tiruan menggunakan *encoder-decoder framework* (Cho et al., 2014). *encoder-decoder framework* ini terdiri dari empat komponen dasar yaitu *embedding layer*, *encoder network, decoder network* dan *classification layer*, Seperti pada Gambar 2.1.



Gambar 2. 1 Contoh *encoder-decoder framework* NMT dasar [Tan et al., 2020]

Embedding layer mewujudkan prinsip *continuous representation* yaitu representasi lanjutan. Hal ini memetakan simbol diskrit x_t menjadi *continuous vector* $x_t \in \mathbb{R}^d$, dimana d menunjukan dimensi vektor. Kemudian dimasukan kedalam lapisan selanjutnya untuk *feature extraction*.

Encoder network memetakan *embedding source* ke *hidden continuous representation*, untuk mempelajari *expressive representation. Encoder* harus dapat memodelkan bahasa sumber dengan *recurrent neural network* (RNN), dengan komputasi seperti pada Persamaan 2.2 (Tan et al., 2020).

$$\mathbf{h}_t = \mathrm{RNN}_{\mathrm{ENC}}(\mathbf{x}_t, \mathbf{h}_{t-1}). \tag{2.2}$$

Dengan menerapkan *state transition function* $\mathrm{RNN}_{\mathrm{ENC}}$ pada urutan *input*, kita dapat menggunakan *final state* h_s sebagai representasi untuk seluruh kalimat sumber, dan kemudian memasukkannya ke *decoder*.

Decoder network dapat dilihat sebagai *language model conditioned* pada h_s. *Decoder network* mengekstrak informasi yang diperlukan dari *encoder output* dan memodelkan dependensi jarak antara kata target.

Diberikan simbol awal y_0 = <sos> dan *inisial state* $s_0 = h_s$, RNN pada *decoder* mengkompres *decoding history* pada $\{y_0, \ldots, y_{t-1}\}$ menjadi *state vector* $s_t \in \mathbb{R}^d$, seperti pada Persamaan 2.3 (Tan et al., 2020).

$$\mathbf{s}_t = \mathrm{RNN}_{\mathrm{DEC}}(\mathbf{y}_{t-1}, \mathbf{s}_{t-1}). \tag{2.3}$$

Classification layer memprediksi token target. Classification layer biasanya merupakan lapisan linier dengan *softmax activation function*. Dengan asumsi kosakata bahasa target adalah v dan $|v|$ adalah ukuran dari kosakata. Diberikan *decoder output* $s_t \in \mathbb{R}^d$, *classification layer* pertama-tama memetakan **h** ke vektor **z** didalam ruang kosakata $\mathbb{R}^{|v|}$ dengan peta linier. Kemudian *softmax*

function digunakan untuk memastikan *output vector* adalah probabilitas yang valid seperti pada Persamaan 2.4 (Tan et al., 2020).

$$\text{softmax}(\mathbf{z}) = \frac{\exp(\mathbf{z})}{\sum_{i=1}^{|V|} \exp(\mathbf{z}_{[i]})}, \tag{2.4}$$

Merujuk pada Persamaan 2.4, dimana kita menggunakan $z_{[i]}$ untuk menyatakan komponen ke-i dalam **z**.

2.6.1.2 Inference

Pada model NMT dan x sebagai kalimat sumber, bagaimana menghasilkan a sebagai terjemahan dari model adalah masalah penting. Idealnya kita akan menemukan y sebagai kalimat target dengan memaksimalkan model prediksi $P = (y|x = x; \theta)$ sebagai terjemahan. Namun, karena ukuran ruang pencariannya yang sangat besar, tidak praktis untuk menemukan terjemahan dengan probabilitas tertinggi. Oleh karena itu, NMT biasanya menggunakan algoritma pencarian seperti *greedy search* atau *beam search* untuk menemukan terjemahan terbaik.

2.6.1.3 Training NMT Model

NMT biasanya menggunakan maximum log-likehood (MLE) sebagai *training object function* yang merupakan metode yang umum digunakan untuk memperkirakan parameter *probability distribution*. Secara formal, memberikan *training set* $D = \{\langle x^{(s)}, y^{(s)} \rangle\} s_{s=1}$, tujuan dari *training* untuk menemukan satu set parameter model yang memaksimalkan log-likehood pada *training set* seperti pada Persamaan 2.5 (Tan et al., 2020).

$$\widehat{\boldsymbol{\theta}}_{\text{MLE}} = \underset{\theta}{\text{argmax}}\{\mathscr{L}(\boldsymbol{\theta})\}, \tag{2.5}$$

Dimana log-likehood didefinisikan sebagai Persamaan 2.6 (Tan et al., 2020).

$$\mathscr{L}(\boldsymbol{\theta}) = \sum_{s=1} \log P(\mathbf{y}^{(s)}|\mathbf{x}^{(s)}; \boldsymbol{\theta}). \tag{2.6}$$

Berdasarkan algoritma *back-propagation* kita dapat secara efisien menghitung gradien ℓ terhadap θ. Pelatihan model NMT biasanya mengadopsi algoritma *stochastic gradient search* (SGD). Dari pada menghitung *gradient* pada semua *training set*, SGD menghitung *lose function* dan *gradient* pada *minibatch*

training set. SGD optimizer memperbarui parameter model NMT dengan aturan seperti pada Persamaan 2.7 (Tan et al., 2020).

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla \mathcal{L}(\boldsymbol{\theta}), \tag{2.7}$$

Dimana α adalah *learning rate*. Dengan *learning rate* yang dipilih dengan baik, parameter NMT dijamin menjadi *local optima*. Dalam praktiknya, selain penggunaan SGD biasa, *adaptive learning rate* seperti Adam yang mengacu pada penelitian (Kingma & Ba, 2015) ditemukan sangat mengurangi waktu *training*.

2.6.2 Arsitektur Mesin Penerjemah Jaringan Saraf Tiruan

2.6.2.1 Evolusi Arsitektur Mesin Penerjemah Jaringan Saraf Tiruan

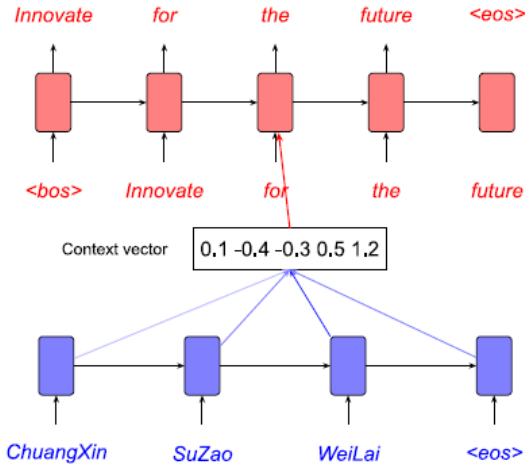
Sejak tahun 2013, ada upaya untuk membangun NMT murni. Diawali dengan arsitektur seperti RCTM oleh (Kalchbrenner & Blunsom, 2013), RNNEncDec oleh (Cho et al., 2014a), dan Seq2Seq oleh (Sutskever et al., 2014) dengan mengadopsi pendekatan panjang tetap atau *fixed-length*, dimana ukuran representasi sumber adalah panjang tetap kalimat sumber. Arsitektur ini biasa menggunakan *recurrent neural network* (RNN) sebagai *decoder network* untuk menghasilkan panjang variabel terjemahan. Namun, menurut (Cho et al., 2014b) ditemukan bahwa kinerja dari pendekatan ini menurun seiring bertambahnya panjang kalimat masukan, representasi dengan panjang tetap atau *fix-length* telah menjadi hambatan selama proses *encoding* untuk kalimat panjang. Sehingga *encoder* dipaksa untuk mengompres seluruh kalimat sumber menjadi satu set vektor dengan panjang tetap atau *fix-length*, karena hal ini beberapa informasi penting mungkin hilang saat diproses.

Karena keterbatasan ini, arsitektur NMT beralih menjadi *variabel-length* representasi kalimat sumber, dimana representasi kalimat sumber tergantung pada panjang kalimat sumber. Arsitektur RNN search oleh (Bahdanau et al., 2015), penelitiannya memperkenalkan mekanisme *attention*, dengan pendekatan penting dalam menerapkan *variable-length* representasi kalimat sumber (berdasarkan panjang kalimat sumber).

Saat ini arsitektur mesin penerjemah jaringan saraf tiruan terdiri dari dua jenis yaitu arsitektur Recurrent Neural Network (RNN) dan arsitektur Transformer. Arsitektur transformer masih proses adopsi praktek industry MT, tetapi saat ini

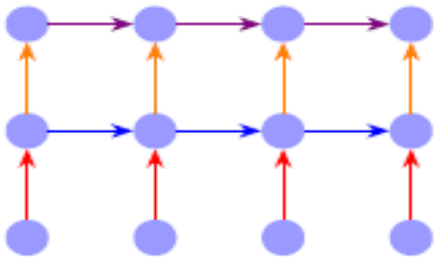
yang banyak diterapkan adalah arsitektur RNN menurut (Mylapore et al., 2020). Pada penelitian kali ini hanya akan membahas dan mengimplementasikan arsitektur RNN.

Pada Gambar 2.2 menunjukkan perbandingan antara pendekatan *fixed-length* dan *variabel-length*. Dengan menggunakan mekanisme *attention*, jalur antara setiap kata sumber dan kata target berada dalam panjang konstan. Akibatnya, mekanisme *attention* telah meringankan kesulitan dengan optimasi. mekanisme *attention* secara dinamis menghasilkan *context vector* berdasarkan representasi sumber yang paling relevan untuk memprediksi kata target berikutnya.



Gambar 2. 2 Contoh proses pada setiap langkah decoding [Tan et al., 2020]

Dengan inovasi dari *deep learning*. NMT dengan *deep neural network* telah menarik banyak minat penelitian. Seperti Seq2Seq oleh (Sutskever et al., 2014) adalah arsitektur pertama yang menunjukan potensi *deep* NMT. Ada banyak cara untuk membangun *deep* RNN, dan yang paling populer adalah dengan menumpuk atau berlapis pada beberapa jaringan RNN. Bentuk proses *deep* RNN dapat dilihat dari gambar 2.3.



Gambar 2. 3 deep RNN

Arsitektur selanjutnya seperti GNMT oleh (Wu et al., 2016), ByteNet oleh (Kalchbrenner et al., 2016), ConvSeq2Seq oleh (Gehring et al., 2017), dan Transformer oleh (Vaswani et al., 2017) semuanya menggunakan *multi-layered neural network.* ByteNet dan ConvSeq2Seq telah mengganti *recurrent neural networks* (RNN) dengan *convolutional neural networks* (CNN) dalam arsitekturnya. Sementara *Transformer* mengandalkan sepenuhnya pada *self-attention networks* (SAN). Keduanya CNN maupun SAN dapat mengurangi *sequential operation* yang terlibat dalam RNN dan mendapat manfaat dari komputasi paralel yang disediakan oleh perangkat modern seperti *graphical processing unit* (GPU) dan *tensor processing unit* TPU.

2.6.2.2 Mekanisme Attention

Mekanisme *attention* diperkenalkan oleh (Bahdanau et al., 2015) yang menjadi awal sejarah dalam penelitian arsitektur NMT. *Attention network* menghitung relevansi setiap *value vector* berdasarkan *query* dan *key.*

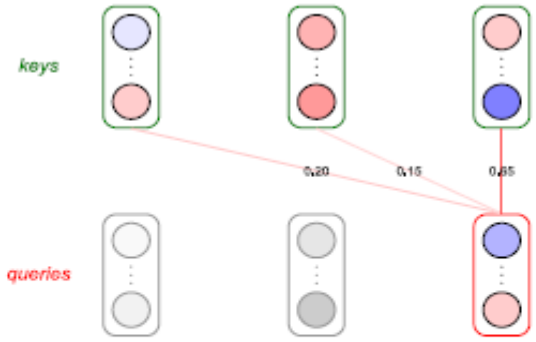
Secara formal, diberikan satu set *m* sebagai *query vectors* $\mathbf{Q} \in \mathbb{R}^{mxd}$, satu set *n* sebagai *key vectors* $\mathbf{K} \in \mathbb{R}^{nxd}$ dan gabungan *value vectors* $\mathbf{V} \in \mathbb{R}^{nxd}$,, perhitungan dari *attention network* melalui dua langkah. Langkah pertama adalah menghitung relevansi antara *keys* dan *values*, secara formal merujuk pada Persamaan 2.8 (Tan et al., 2020).

$$\mathbf{R} = \text{score}(\mathbf{Q}, \mathbf{K}), \tag{2.8}$$

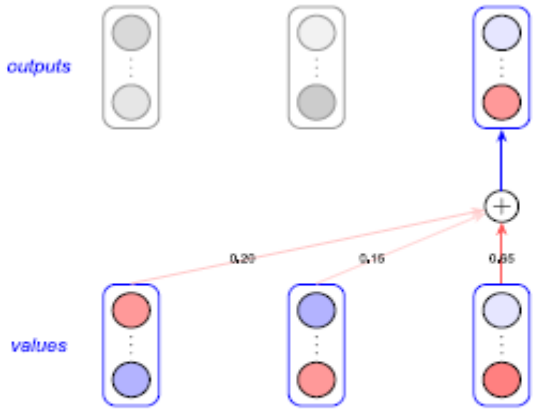
Dimana skornya adalah *score function* yang memiliki beberapa alternatif, $\mathbf{R} \in \mathbb{R}^{mxn}$ adalah matriks yang menyimpan skor relevansi antara setiap *keys* dan *values*. Langkah selanjutnya adalah menghitung *output vector*. Untuk setiap *query vectors*, *output vector* yang sesuai dinyatakan sebagai jumlah bobot dari *value vectors* seperti merujuk pada Persamaan 2.9 (Tan et al., 2020).

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\mathbf{R}) \cdot \mathbf{V}. \tag{2.9}$$

Pada Gambar 2.4 menunjukkan dua langkah yang dilalui dalam perhitungan mekanisme *attention.*



(a) Given a query vector and key vectors, the attention network first computes a weight vector through the scoring function



(b) NMT with variable-length representation and attention mechanism.

Gambar 2. 4 Contoh detail perhitungan dalam mekanisme attention [Tan et al., 2020]

Tan dkk., (2020) mengemukakan dalam mempertimbangkan *score function*, *attention network* dapat diklasifikasi dalam dua kategori yaitu *additive attention* oleh (Bahdanau et al., 2015) dan *dot-product attention* oleh (Luong et al., 2015). Model *additive attention* mencetak skor melalui *feed-forward neural network* seperti pada Persamaan 2.10 (Tan et al., 2020).

$$\mathbf{R}_{[i,j]} = \mathbf{v}^{\top} \tanh\left(\mathbf{W}_s \mathbf{Q}_{[i]} + \mathbf{U}_s \mathbf{K}_{[j]}\right), \tag{2.10}$$

Dimana $\mathbf{W}_s \in \mathbb{R}^{dxd}$, $\mathbf{U}_s \in \mathbb{R}^{dxd}$, dan $\mathbf{v} \in \mathbb{R}^{dx1}$ adalah *learnable parameters*. Pada sisi lain *dot-product attention* menggunakan *dot-product* untuk menghitung skor yang cocok seperti pada Persamaan 2.11 (Tan et al., 2020).

$$\mathbf{R}_{[i,j]} = \mathbf{Q}_{[i]}^{\top} \mathbf{K}_{[j]}. \tag{2.11}$$

Dalam praktiknya, *dot-product attention* jauh lebih cepat dari pada *additive attention.* Namun *dot-product attention* ternyata kurang stabil dibanding *additive attention* ketika *d* ukurannya besar (Vaswani et al., 2017) .

Mekanisme *attention* biasanya digunakan sebagai bagian dari *decoder network.* Jenis lain dari *attention network* yang disebut *self-attention network* banyak digunakan baik di *encoder* dan *decoder* NMT.

Menurut Mylapore dkk., (2020) arsitektur *transformer* masih proses adopsi praktek industry MT, tetapi saat ini yang banyak diterapkan adalah arsitektur RNN.

2.6.2.3 RNN, CNN, dan SAN

Encoder dan *decoder* adalah komponen utama dari arsitektur NMT. Ada banyak metode untuk membangun *encoder* dan *decoder* yang kuat, yang secara umum dapat dibagi menjadi tiga kategori yaitu *recurrent neural network* (RNN), *convolution neural network* (CNN), dan *self-attention network* (SAN). Ada beberapa aspek yang perlu diketahui untuk mempertimbangkan membangun *encoder* dan *decoder* seperti berikut.

1. *Receptive field*, yaitu mengharapkan setiap *output* yang dihasilkan oleh *encoder* dan *decoder* menyandikan informasi semaunya di urutan *input*.
2. *Computational complexity*, yaitu menginginkan untuk menggunakan *network* dengan kompleksitas komputasi yang lebih rendah.
3. *Sequential operations,* yaitu terlalu banyak operasi sekuensial akan menghalangi komputasi paralel dalam urutan.
4. *Position awareness,* jaringan harus membedakan pesanan dalam urutan.

Perhatikan kedua ilustrasi pada Gambar 2.5 dan Gambar 2.6 menampilkan ringkasan komputasi pada aspek RNN, CNN, dan SAN. Pada Gambar 2.5 menunjukkan perbandingan antara *neural network* yang berbeda. Kita menggunakan R.F. untuk menunjukkan *receptive field,* S.O. untuk menunjukkan jumlah operasi yang berurutan dan P.A untuk menunjukkan *position awareness* dalam layer. *t* adalah posisi dalam urutan, *l* adalah nomor layer. pada CNN, *k* adalah lebar filter dan $W^{(i)}$ adalah bobot dari filter *i*-th.

Layer	Computation	R.F.	Complexity	S.O.	P.A.
RNN	$\mathbf{h}_{l,t} = \mathbf{W}\mathbf{h}_{l-1,t} + \mathbf{U}\mathbf{h}_{l,t-1}$	∞	$O(n \cdot d^2)$	$O(n)$	Yes
CNN	$\mathbf{h}_{l,t} = \sum_{i=1}^{k} \mathbf{W}^{(i)}\mathbf{h}_{l-1,t+i-\frac{k+1}{2}}$	k	$O(k \cdot n \cdot d^2)$	$O(1)$	Yes
SAN	$\mathbf{h}_{l,t} = \sum_{i=1}^{n} a_{l,i}\mathbf{h}_{l-1,i}$	∞	$O(n^2 \cdot d)$	$O(1)$	No

Gambar 2. 5 Perbandingan antara *neural network* yang berbeda [Tan et al., 2020]

Pada Gambar 2.6 menampilkan perbandingan arsitektur dasar pada NMT. V.R. menunjukkan apakah arsitektur menggunakan *variable representation*. path_E menunjukkan jalur terpanjang antara token sumber dan target. path_D menunjukkan jalur terpanjang antara dua token target.

Model	Encoder	Decoder	Complexity	V.R.	Path_E	Path_D
RCTM 1 (Kalchbrenner and Blunsom, 2013)	CNN	RNN	$O(S^2 + T)$	No	S	T
RCTM 2 (Kalchbrenner and Blunsom, 2013)	CNN	RNN	$O(S^2 + T)$	Yes	S	T
RNNENCDEC/ SEQ2SEQ (Cho et al., 2014a; Sutskever et al., 2014)	RNN	RNN	$O(S + T)$	No	$S + T$	T
RNNSEARCH (Bahdanau et al., 2015)	RNN	RNN	$O(ST)$	Yes	1	T
BYTENET (Kalchbrenner et al., 2016)	CNN	CNN	$O(S + T)$	Yes	c	c
CONVSEQ2SEQ (Gehring et al., 2017)	CNN	CNN	$O(ST)$	Yes	1	c
TRANSFORMER (Vaswani et al., 2017)	SAN	SAN	$O(S^2 + ST + T^2)$	Yes	1	1

Gambar 2. 6 Perbandingan arsitektur dasar [Tan et al., 2020]

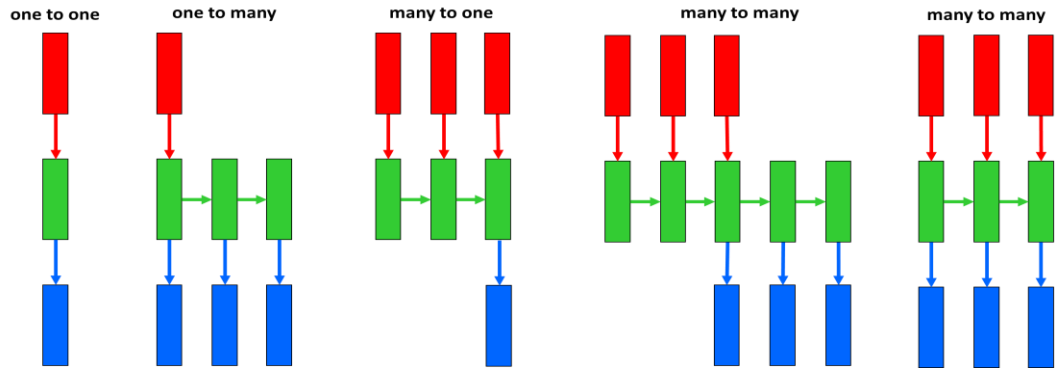
2.7 *Sequence to Sequence Model*

Sequence to sequence adalah sebuah model untuk pembelajaran sekuensial menggunakan jaringan saraf tiruan dengan fungsi dasar mengambil urutan sumber sebagai masukan X={x1,x2,…,xn} mencoba untuk menghasilkan urutan target sebagai keluaran Y={y1,y2,…,yn} yang dimana Xi dan Yi adalah simbol masukan dan keluaran. Sequence to sequence model memiliki dua bagian yaitu encoder

sebagai menerima urutan sumber masukan dan decoder sebagai menerima keluaran dari encoder kemudian menghasilkan keluaran urutan target.

Pada sequence data (urutan data) mempunyai karakteristik di mana sampel diproses dengan suatu urutan (misalnya waktu), dan suatu sampel dalam urutan mempunyai hubungan erat satu dengan yang lain. Contoh data sequence dan aplikasinya misalnya rangkaian kata dalam penerjemahan bahasa, sinyal audio dalam pengenalan suara, dan rangkaian kata-kata yang klarifikasikan sentimen (pandangan/pendapat yang berlebihan terhadap sesuatu).

Berikut adalah jenis dari *sequence to sequence* model seperti terlihat pada Gambar 2.7.



Gambar 2. 7 Jenis-jenis sequence to sequence

Dapat dilihat pada Gambar 2.7 sequence to sequence terdiri dari satu ke satu, satu ke banyak, banyak ke satu dan banyak ke banyak. Setiap persegi panjang adalah vector(perantara), panah adalah fungsi, persegi panjang bewarna merah adalah vektor masukan, persegi panjang bewarna hijau adalah RNN, persegi panjang bewarna biru adalah vektor keluaran.

Penjelasan jenis sequence to sequence model sebagai berikut:

1. Satu ke satu adalah mode pemrosesan vanilla tanpa RNN dari masukan berukuran tetap ke keluaran berukuran tetap misalnya klasifikasi gambar.
2. Satu ke banyak adalah sequence keluaran yang banyak misalnya menghasilkan pembuatan sintesa musik yang dimana masukan hanya satu misalnya jenis musik (jazz, dangdut, dan rock) atau kalimat deskripsi dari sebuah masukan berupa gambar (satu gambar ada banyak objek).

3. Banyak ke satu adalah sequence masukan yang banyak misalnya analisis sentimen dari sebuah kalimat untuk menghasilkan sentimen positif atau negatif.
4. Banyak ke banyak dengan panjang vektor tidak tetap adalah sequence masukan dan keluaran banyak misalnya mesin penerjemah: RNN membaca kalimat masukan dalam bahasa Indonesia kemudian mengeluarkan kalimat dalam bahasa Tiochiu Pontianak.
5. Banyak ke banyak dengan panjang vektor tetap adalah sequence masukan dan keluaran yang disinkronkan misalnya klasifikasi video di mana ingin memberi label pada setiap rangkaian gambar pada video.

Pada sequence to sequence model ini merujuk pada makalah penelitian yang berjudul "Sequence to Sequence Learning with Neural Networks" oleh (Sutskever et al., 2014) yang mengimplementasikan encoder-decoder model untuk neural machine translation (NMT).

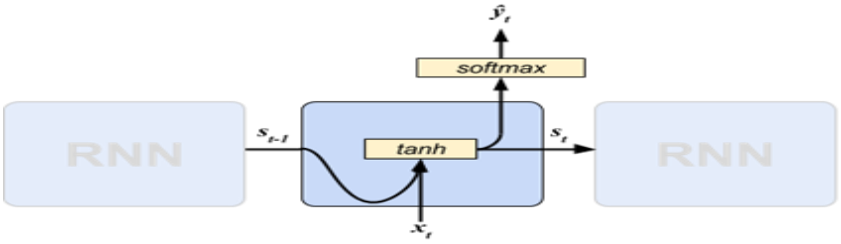
2.8 *Recurrent Neural Network*

Banyak mesin penerjemah jaringan saraf tiruan dibangun dengan *arsitektur Recurrent Neural Network* (RNN) yang seperti penelitian sebelumnya oleh (Sutskever et al., 2014) dan (Cho et al., 2014). Model ini menggunakan jaringan *encoder* dan *decoder* untuk mempelajari urutan informasi dan membuat model prediksi.

Jaringan saraf berulang atau *Recurrent Neural Network* (RNN) adalah jenis arsitektur jaringan saraf tiruan yang pemrosesannya secara bertahap dipanggil berulang-ulang untuk memproses kalimat sumber atau disebut data sekuensial. RNN masuk dalam kategori deep learning karena data diproses melalui banyak lapis (layer). RNN telah mengalami kemajuan yang pesat dan telah merevolusi bidang-bidang seperti pemrosesan bahasa alami (NLP), pengenalan suara, sintesa musik, pemrosesan data finansial seri waktu, analisa deret DNA, analisa video, dan sebagainya.

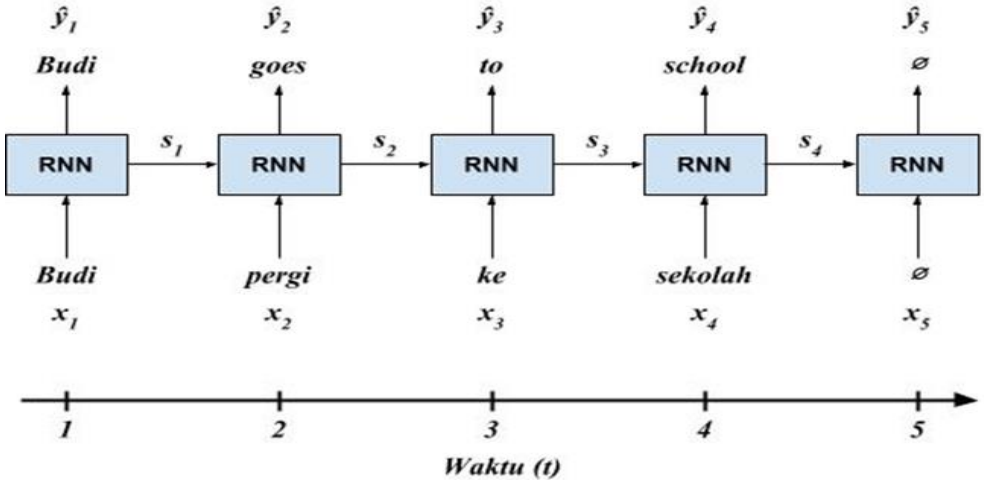
RNN memproses kalimat input, sampel per sampel dari data sekuensial. Dalam tiap pemrosesan, keluaran yang dihasilkan tidak hanya merupakan komputasi dari sampel itu saja, tapi juga berdasarkan *state internal* yang merupakan

hasil dari pemrosesan sampel-sampel sebelumnya bisa disebut hidden layer. RNN juga bisa disebut ekstraksi fitur dan update konten. Pada Gambar 2.8 bisa dilihat bentuk RNN sederhana.



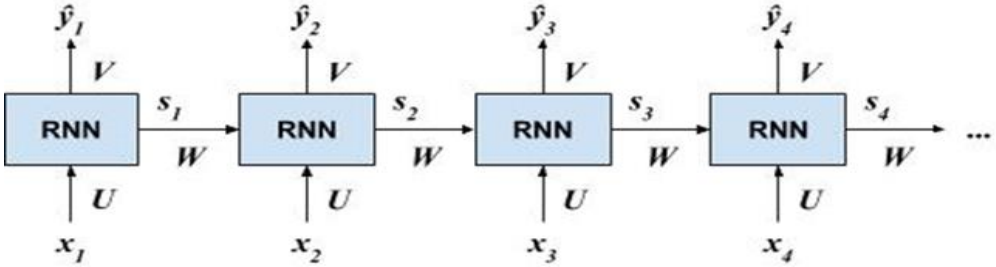
Gambar 2. 8 Gambar RNN sederhana [Prijono, 2018]

Berikut adalah ilustrasi bagaimana RNN bekerja dapat dilihat pada Gambar 2.9. Misalnya membuat RNN untuk menerjemahkan bahasa Indonesia ke bahasa Inggris.



Gambar 2. 9 RNN dengan menerjemahkan bahasa [Prijono, 2018]

Dengan contoh seperti di Gambar 2.9, kita bisa generalisasikan arsitektur RNN seperti pada Gambar 2.10 sebagai berikut.

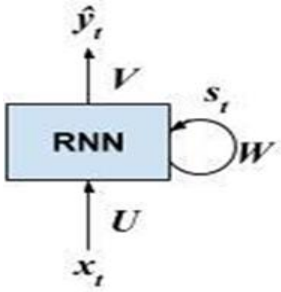


Gambar 2. 10 Representasi RNN [Prijono, 2018]

Tambahan yang tidak terdapat di Gambar 2.9 adalah U, V, dan W, yang merupakan parameter-parameter yang dimiliki RNN. Berikut pembahasan mengenai pemakaian parameter-parameter ini.

Penting untuk dipahami bahwa walaupun ada empat kotak RNN pada gambar 2.10, empat kotak itu mencerminkan satu modul RNN yang sama (satu model instan dengan parameter-parameter U, V, dan W yang sama). Penggambaran di atas hanya agar aspek sekuensialnya lebih tergambar.

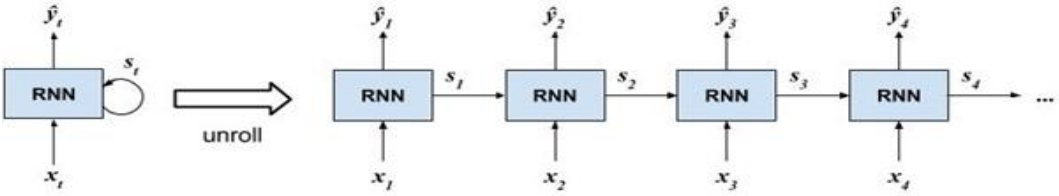
Alternatif representasinya dapat dilihat pada Gambar 2.11, agar lebih jelas bahwa hanya ada satu modul RNN:



Gambar 2. 11 Representasi RNN *loop* [Prijono, 2018]

Inilah sebabnya kenapa arsitektur ini disebut RNN. Kata *recurrent* (berulang) dalam RNN timbul karena RNN melakukan perhitungan yang sama secara berulang-ulang atas input yang diberikan.

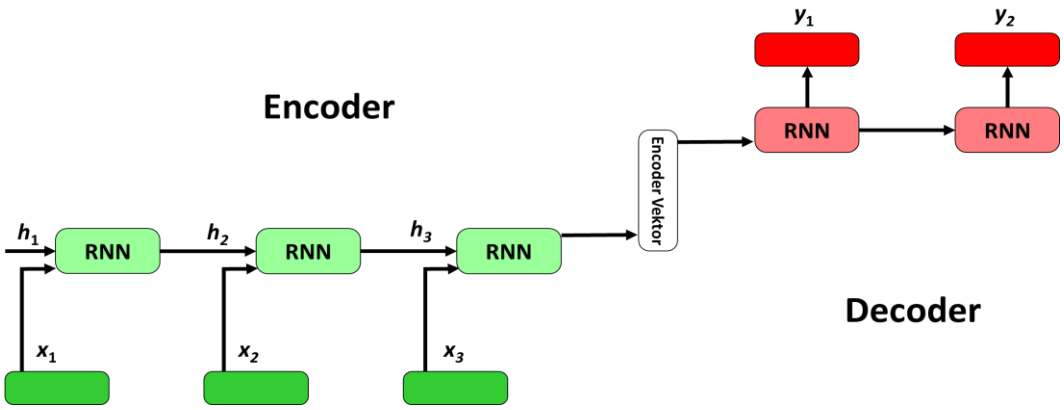
Kedua ilustrasi pada Gambar 2.11 dan Gambar 2.11 digabungkan jadi satu dapat dilihat di gambar 2.12 sebagai berikut:



Gambar 2. 12 Penjabaran RNN [Prijono, 2018]

Sesuai dengan Gambar 2.12, ilustrasi di sebelah kanan adalah penjabaran (*unrolled*) dari versi berulang di sebelah kiri.

Penerapan arsitektur RNN pada mesin penerjemah jaringan saraf tiruan berada antara encoder dan decoder pada *sequence to sequence* bisa dilihat pada Gambar 2.13.



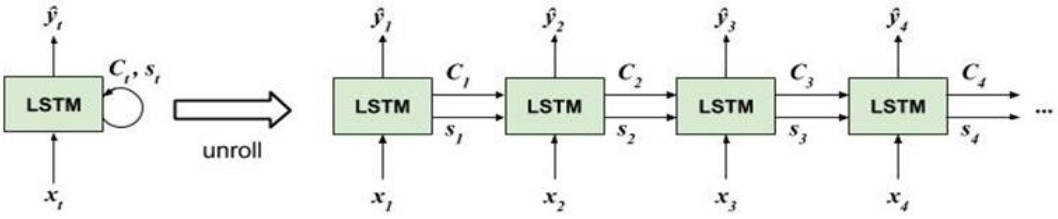
Gambar 2. 13 *Encoder-decoder sequence to sequence model*

2.9 Komputasi Unit atau Jenis RNN

Para peneliti selama beberapa tahun mengembangkan teknik dan tipe-tipe RNN yang lebih canggih untuk mengatasi kekurangan pada model vanilla RNN. Jenis dari sebuah RNN yang digunakan untuk mengatasi masalah *gradient* yang hilang terdiri dari LSTM dan GRU.

2.9.1 LSTM (*Long Short Term Memory*)

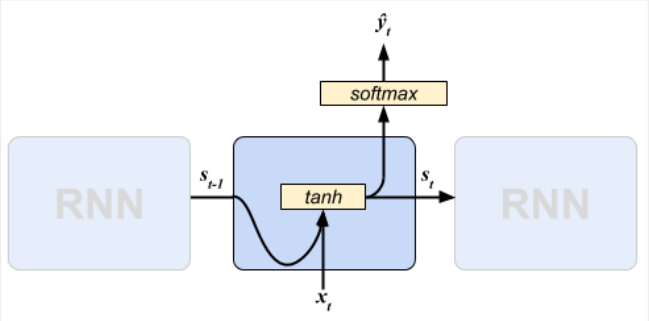
LSTM (*Long Short Term Memory*) adalah jenis modul pemrosesan lain untuk RNN. LSTM diciptakan oleh (Hochreiter dan Schmidhuber, 1997) kemudian dikembangkan dan dipopulerkan oleh banyak periset. Seperti RNN, jaringan LSTM (LSTM *network*) juga terdiri dari modul-modul dengan pemrosesan berulang. Bedanya adalah modul-modul yang membentuk jaringan LSTM adalah modul LSTM dapat dilihat pada Gambar 2.14.



Gambar 2. 14 LSTM [Prijono, 2018a]

Modul LSTM mempunyai pemrosesan yang berbeda dengan modul RNN biasa. Perbedaan lain adalah adanya tambahan sinyal yang diberikan dari satu langkah waktu ke langkah waktu berikutnya, yaitu konteks, direpresentasikan dengan simbol C_t.

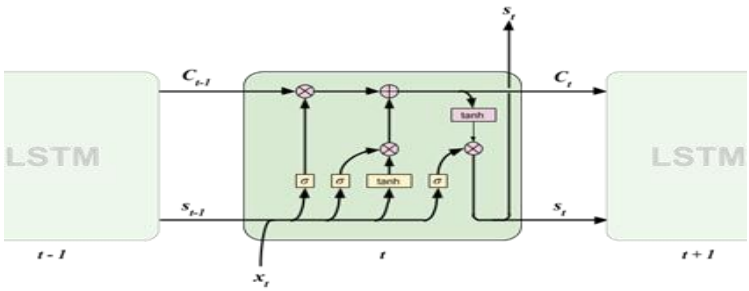
Pemrosesan dalam satu modul RNN cukup sederhana, hanya ada satu lapis *tanh*, dan kalau dikehendaki *output* dari modul itu maka *state internal* akan dilewatkan pada fungsi aktivasi seperti *softmax* untuk mendapatkan *output* y_t seperti yang terlihat pada Gambar 2.15.



$$s_t = tanh(U \cdot x_t + W \cdot s_{t-1})$$
$$\hat{y}_t = softmax(V \cdot s_t)$$

Gambar 2. 15 Modul RNN sederhana [Prijono, 2018a]

Sedangkan pemrosesan dalam modul LSTM mengandung lebih banyak komputasi, seperti pada Gambar 2.16.



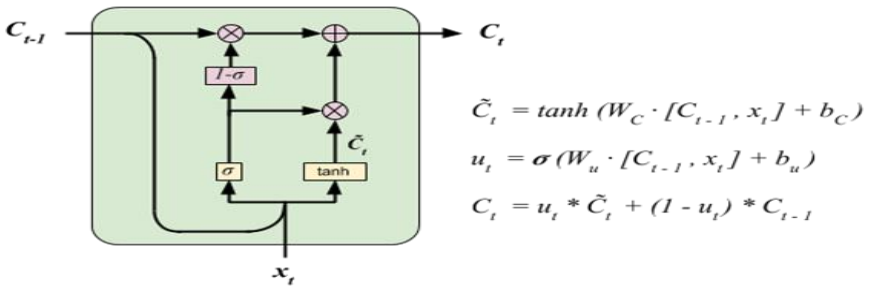
Gambar 2. 16 Pemrosesan LSTM [Prijono, 2018a]

2.9.2 GRU (*Gated Recurrent Unit*)

LSTM mempunyai banyak sekali varian, misalnya LSTM dengan koneksi lubang intip (*peephole connection*), LSTM yang menggabungkan gerbang *input* dengan gerbang *forget*, dan sebagainya. Salah satu varian yang populer adalah *gated recurrent unit* atau disingkat GRU. GRU dimunculkan dalam makalah oleh (Cho et al., 2014).

Keutamaan GRU adalah komputasinya lebih sederhana dari LSTM, namun mempunyai akurasi yang setara dan masih cukup efektif untuk menghindari permasalahan gradien yang menghilang.

Dengan GRU, pertama perlu kalkulasi kandidat konteks $\tilde{C}t$ sebagai *tanh* dari gabungan konteks lama C_{t-1} dan *input* x_t. Lalu sebuah gerbang sigmoid dinamakan gerbang *update* u_t untk menentukan seberapa banyak konteks yang baru C_t berasal dari kandidat $\tilde{C}t$ dan seberapa banyak dari konteks lama C_{t-1} yang dapat dilihat pada Gambar 2.17.

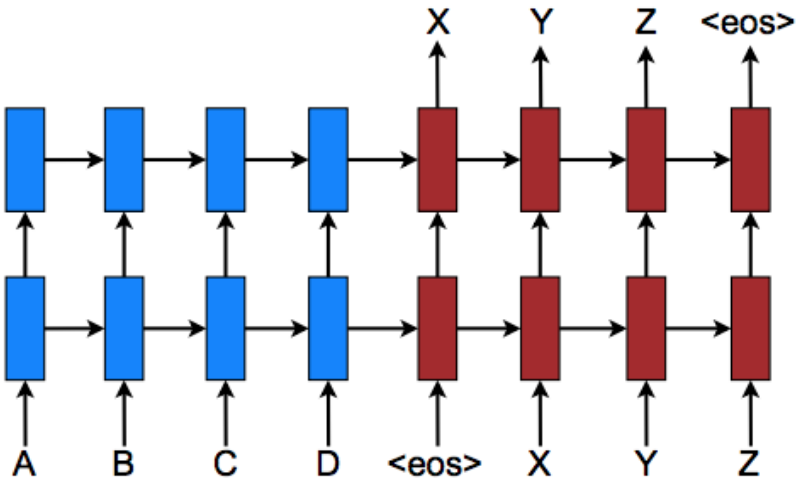


Gambar 2. 17 Pemprosesan GRU [Prijono, 2018a]

Untuk melihat manakah varian RNN yang terbaik, merujuk pada makalah oleh (Greff et al., 2017) yang melakukan perbandingan varian LSTM, menemukan bahwa semuanya hampir sama, tidak ada varian dari LSTM yang dapat ditingkatkan secara signifikan. (Jozefowicz et al., 2015) telah mengevaluasi berbagai arsitektur RNN untuk menemukan arsitektur yang kinerjanya baik dari LSTM. Meskipun ada arsitektur yang mengungguli LSTM pada beberapa masalah, tetapi tidak dapat menemukan arsitektur yang secara konsisten mengalahkan LSTM dan GRU di semua kondisi eksperimental.

2.10 NMT dengan Encoder-Decoder Model

Mesin penerjemah jaringan saraf tiruan adalah model *neural network* yang secara langsung melakukan probabilitas kondisional $p(y|x)$ dari kalimat sumber $x1 \ldots xn$ ke kalimat target $y1 \ldots yn$. NMT dengan model *encoder* dan *decoder* dapat dilihat pada Gambar 2.18.



Gambar 2. 18 NMT dengan model encoder dan decoder [Luong et al., 2015]

Bentuk dasar NMT terdiri dari dua komponen

1. *Encoder*

   Menghitung representasi dari setiap kalimat sumber

2. *Decoder*

   Menghasilkan satu target kalimat pada satu waktu dan menguraikan *conditional probability* seperti pada Persamaan 2.12 (Luong et al., 2015).

$$\log p(y|x) = \sum_{j=1}^{m} \log p\left(y_j|y_{<j}, \boldsymbol{s}\right) \tag{2.12}$$

   Penulis menggunakan GRU untuk *encoder* dan *decoder*. Probabilitas *decoding* dari setiap kata y_j seperti pada Persamaan 2.13 (Luong et al., 2015).

$$p\left(y_j|y_{<j}, \boldsymbol{s}\right) = \text{softmax}\left(g\left(\boldsymbol{h}_j\right)\right) \tag{2.13}$$

   Dengan g adalah fungsi transformasi yang menghasilkan kosakata berukuran vektor. Disini h_j adalah RNN *hidden unit*, secara abstrak dihitung seperti pada Persamaan 2.14 (Luong et al., 2015).

$$\boldsymbol{h}_j = f(\boldsymbol{h}_{j-1}, \boldsymbol{s}), \tag{2.14}$$

Dengan *f* menghitung *hidden state* saat ini dengan *hidden state* sebelumnya dan unit GRU. Representasi sumber *s* menyiratkan satu set *hidden state* sumber dikonsultasikan selama proses penerjemahan.

Loss dihitung sebagai berikut (ini adalah fungsi yang digunakan untuk pelatihan) seperti pada Persamaan 2.15 (Luong et al., 2015).

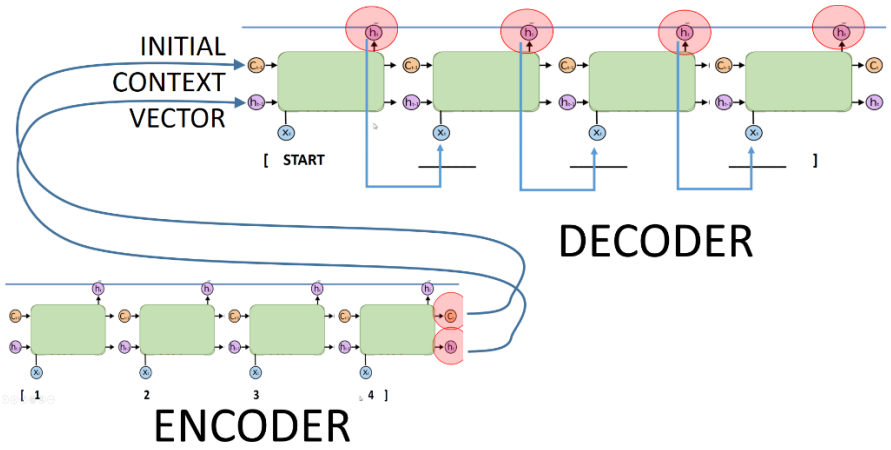
$$J_t = \sum_{(x,y)\in\mathbb{D}} -\log p(y|x) \tag{2.15}$$

Dengan *D* adalah korpus latih. Penulis menggunakan algoritma pelatihan *batch* secara berurutan untuk melatih data korpus.

2.11 *Attention Mechanism*

Mekanisme attention merupakan sebuah pendekatan encoder-decoder dasar mesin penerjemah jaringan saraf tiruan (NMT) yang menggunakan konteks vector yang memiliki panjang tetap. Menurut penemu mekanisme *attention*, Bahdanau dkk (2015) bahwa salah satu motivasi dibalik pendekatan yang diusulkan (*attention mechanism*) adalah penggunaan *context vector* yang memiliki panjang tetap dalam pendekatan *encoder-decoder* dasar.

Untuk memahami bagaimana mekanisme *attention* bekerja, pertama-tama membandingkan model *encoder-decoder* dasar dengan model *encoder-decoder attention.*



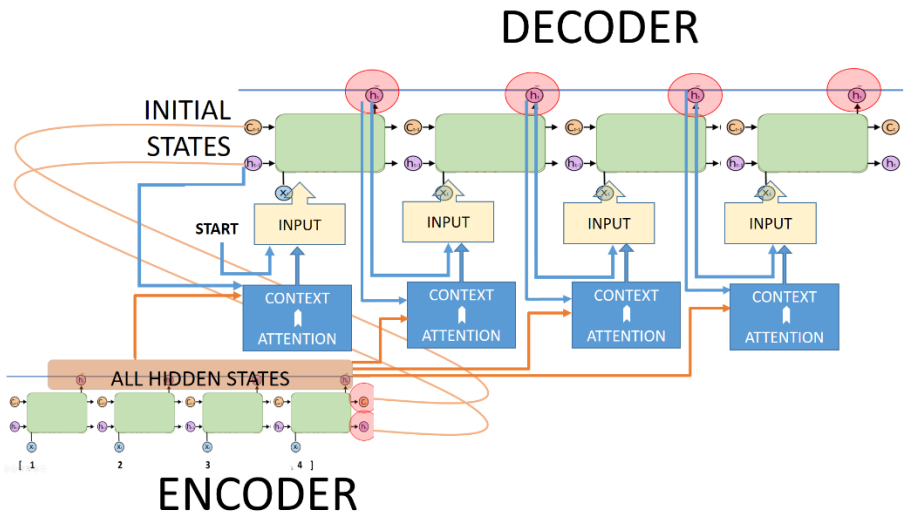
Gambar 2. 19 Model *encoder* dan *decoder* dasar [Karakaya, 2020]

Pada Gambar 2.19 model encoder-decoder dasar dapat diperhatikan bahwa:

1. Model *encoder-decoder* dasar hanya menggunakan *decoder hidden state*

dan *cell state* pertama.

2. Inisialisasi *context vector* pada *decoder state*.

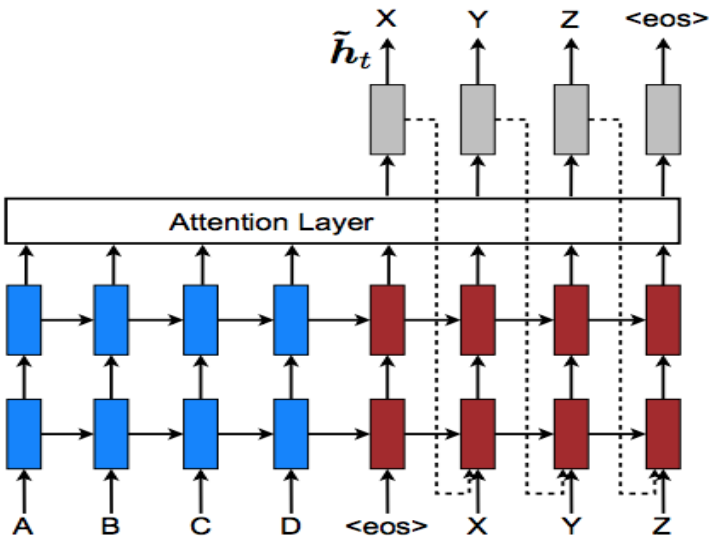


Gambar 2. 20 Model encoder dan decoder attention [Karakaya, 2020]

Pada Gambar 2.20 model encoder decoder dengan attention diperhatikan bahwa:

1. Tidak hanya menggunakan *hidden states* dan *cell state* pertama tetapi juga hasil dari *decoder hidden states* pada semua *time step*.
2. Menggunakan semua *decoder hidden states* di semua *time step* yang berurutan.

Pada dasarnya, pertama-tama pada awalnya menginisialisasi *decoder state* menggunakan *hidden state encoder* pertama. kemudian pada setiap *time step decoding*, dengan menggunakan semua *encoder hidden states* dan *decoder output* sebelumnya untuk menghitung menjadi *context vector* dengan menerapkan mekanisme *attention*. Pada bagian akhir, menggabungkan *context vector* dengan *decoder output* sebelumnya untuk membuat *input* ke *decoder*.



Gambar 2. 21 NMT dengan *attention layer* [Luong et al., 2015]

Pada Gambar 2.21 model *encoder-decoder* dengan *attention*. *Attention* dibagi menjadi dua jenis yaitu.

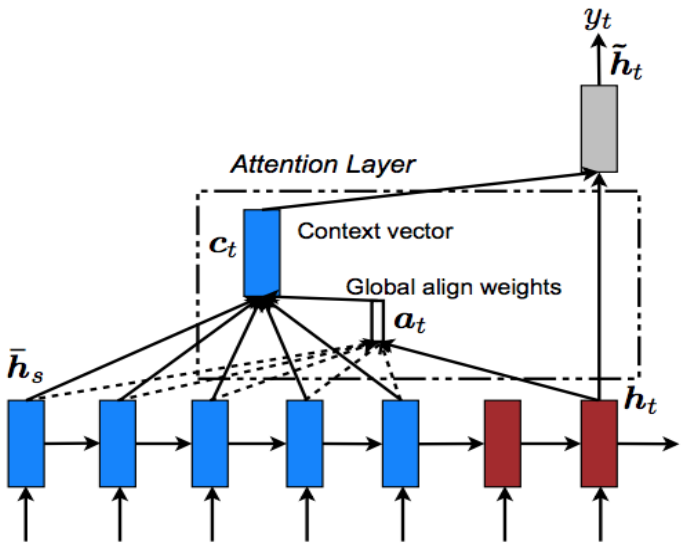
1. *Global Attention*

   Attention dengan memperhatikan semua item urutan sumber.

2. *Local Attention*

   Attention yang hanya memperhatikan bagian tertentu dari item urutan sumber.

Kedua jenis *attention* berbeda dari model *encoder-decoder* normal hanya pada fase decoding. Metode *attention* ini memiliki perbedaan pada cara menghitung *context vector* (C_t).



Gambar 2. 22 NMT dengan global attention model [Luong et al., 2015]

Menurut Luong et al., (2015) bahwa mekanisme *attention* disebut dengan *global attention model,* yang merupakan ide untuk mempertimbangkan semua *hidden states* dari *encoder* h_bar_s ketika diberikan pada *context vektor* c_t. Berdasarkan Gambar 2.22 penjelasannya pada poin berikut.

h_t : *hidden state* target saat ini.

c_t : *context vector*.

y_t : kata target saat ini.

h_bar_t : *attentional hidden state.*

h_bar_s : *hidden state* sumber.

a_t : *alignment vector*.

Berikut langkah menghitung *attention.*

1. Kedua pendekatan pertama-tama mengambil *hidden state* h_t diatas layer RNN yang berwarna coklat/target di *decoder hidden state*.
2. Menjalankan c_t untuk menangkap informasi sisi sumber yang relevan untuk membantu memprediksi y_t. c_t pada dasarnya adalah *context vector* yang telah kita buat untuk setiap kata yang kemudian dihitung sebagai rata-rata pada bobot *alignment* dan *encoder hidden state.*
3. Menghitung h_bar_t dari rangkaian sederhana h_t dan c_t (sel abu-abu diatas) dengan Persamaan 2.16a (Luong et al., 2015).

$$\tilde{\boldsymbol{h}}_t = \tanh(\boldsymbol{W_c}[\boldsymbol{c}_t; \boldsymbol{h}_t]) \quad (2.16a)$$

Berbeda dengan model tanpa menggunakan *attention* yang hanya *output* akhir dari *encoder* diberikan ke *decoder*, h_bar_t memiliki akses ke semua *hidden state* dari *encoder* yang memberikan tampilan informatif dari kalimat sumber.

4. *Attention vector* ditransformasikan menggunakan *softmax layer* untuk menghasilkan distribusi prediktif. Disarankan menggunakan *softmax layer* karena harus menemukan kata yang paling mungkin dari semua kata yang tersedia dalam *vocabulary*, yang ditunjukkan pada Persamaan 2.16b (Luong et al., 2015).

$$p(y_t|y_{<t}, x) = \text{softmax}(\boldsymbol{W_s}\tilde{\boldsymbol{h}}_t) \quad (2.16b)$$

Penjelasan lanjut.

1. a_t bergantung pada h_t dan h_bar_s.
2. c_t bergantung pada a_t dan h_bar_s.
3. h_bar_t bergantung pada c_t dan h_t.

Pada paragraf sebelumnya sudah menjelaskan langkah *attention. Context vector* c_t dihitung secara berbeda pada *attention local* dan *global*. Pada poin selanjutkan hanya akan menjelaskan *attention global*.

1. *Attention Global*

   Attention global mempertimbangkan semua *encoder hidden state* untuk menjalankan *context vector* c_t. Untuk menghitung c_t, maka menghitung a_t yang merupakan panjang *alignment vector*. *Alignment vector* diperoleh dengan menghitung ukuran kesamaan antara h_t dan h_bar_s dimana h_t adalah *hidden state* target sedangkan h_bar_s adalah *hidden state* sumber.

   a) *Score function*

      Score function adalah menghitung skor untuk menghubungkan semua *encoder hidden state* dan *decoder output* sebelumnya. Ada banyak cara penghitungan skor menurut Bahdanau et al., (2015) penghitungan skor dengan *additive/concat* seperti pada Persamaan 2.17 (Marathe, 2020).

$$\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) = \boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W_1 h}_t + \boldsymbol{W_2}\bar{\boldsymbol{h}}_s\right) \quad (2.17)$$

Sedangkan menurut Luong et al., (2015) penghitungan skor dengan *dot product, general dan concat* yang dapat dilihat seperti pada Persamaan 2.18 (Luong et al., 2015).

$$\text{score}(h_t, \bar{h}_s) = \begin{cases} h_t^\top \bar{h}_s & \textit{dot} \\ h_t^\top W_a \bar{h}_s & \textit{general} \\ v_a^\top \tanh\left(W_a[h_t; \bar{h}_s]\right) & \textit{concat} \end{cases} \quad (2.18)$$

b) *Alignment vector* a_t.

A*lignment vector* a_t adalah menghitung bobot *attention* dengan menormalisasikan skor. Sederhananya, dapat menggunakan *softmax()* untuk menghitung *probability distribution*. Dapat didefinisikan seperti pada Persamaan 2.19 (Luong et al., 2015).

$$\begin{aligned} a_t(s) &= \text{align}(h_t, \bar{h}_s) \\ &= \frac{\exp\left(\text{score}(h_t, \bar{h}_s)\right)}{\sum_{s'} \exp\left(\text{score}(h_t, \bar{h}_{s'})\right)} \end{aligned} \quad (2.19)$$

c) Context vector (c(t))

Context vector (ct(t)) dengan menerapkan *attention weights* pada semua *encoder hidden state h_bar*$_s$. Dengan demikian, kita akan memiliki bobot *encoder hidden state* sampai akhir. Dapat didefinisikan seperti pada Persamaan 2.20 (Karakaya, 2020).

$$\boldsymbol{c}_t = \sum_s \alpha_{ts} \bar{\boldsymbol{h}}_s \quad (2.20)$$

Setelah menghitung *context vector* kita dapat mengalikannya dengan *decoder hidden state (output)* sebelumnya untuk menghasilkan *input* untuk proses *decoder* berikutnya.

2.11.1 Bahdanau *Attention*

Bahdanau *Attention* atau biasa yang dikenal dengan *Additive Attention* merupakan salah satu jenis mekanisme *Attention* yang diperkenalkan lebih dulu pada mesin penerjemah jaringan saraf tiruan (NMT) pada makalah penelitiannya dengan judul "*Neural Machine Translation by Jointly Learning to Align and Translate*" oleh (Bahdanau et al., 2015). Penelitiannya menggunakan kata "*align*"

atau penyelarasan dalam judul makalahnya yang berarti menyesuaikan bobot yang secara langsung berhubungan dengan skor saat melatih model. Menurut penemu mekanisme *attention,* Bahdanau dkk (2015) bahwa salah satu motivasi dibalik pendekatan yang diusulkan (*attention mechanism*) adalah penggunaan *context vector* yang memiliki panjang tetap dalam pendekatan *encoder-decoder* dasar. Penilitiannya menduga bahwa keterbatasan ini dapat membuat pendekatan *encoder-decoder* dasar berkinerja buruk pada kalimat yang panjang. Dapat disimpulkan mekanisme attention oleh Bahdanau adalah pendekatan model encoder-decoder dasar dengan nilai skornya berdasarkan panjang kalimat sumber dan kalimat target. Dalam makalahnya pada *encoder* menggunakan Bi (*bidirectional*) dengan jenis RNN GRU (*Gated Recurrent Encoder*) untuk meng-enkode setiap kata sumber dengan lebih baik, *encoder* memiliki dua RNN, maju dan mundur yang membaca masukan dalam arah yang berlawanan. Untuk setiap token, status dari dua RNN digabungkan. Untuk mendapatkan skor *attention*, dengan *multi layer perceptron* (MLP) ke status *encoder* dan *decoder*. Pada *decoder* menggunakan jenis RNN GRU (*Gated Recurrent Unit*).

Hal yang perlu diperhatikan tentang mekanisme *attention* Bahdanau.

1. *Encoder* adalah bidirectional dua arah (maju dan mundur) dengan *gated recurrent unit* (BiGRU)
2. Fungsi skor di *attention layer* adalah *additive/concat*.
3. Input kelangkah *decoder* berikutnya adalah penggabungan antara kata yang dihasilkan dari *time step* sebelumnya dan *context vector* dari *time step* saat ini.
4. Akurasi yang dihasilkan mencapai skor 36.15 BLEU poin pada kumpulan data bahasa Inggris ke Prancis WMT'14.

Attention diterapkan diantara setiap langkah-langkah *decoder***.** Seluruh proses langkah-langkah penerapan *attention* Bahdanau adalah sebagai berikut.

1. Memproduksi *hidden state encoder*.
   encoder memproduksi *hidden state* dari setiap elemen urutan *input*.
2. Menghitung skor *alignment* antara *decoder hidden state* sebelumnya dan menghitung masing-masing *hidden state encoder* (catatan : *hidden state encoder* terakhir dapat digunakan sebagai *hidden state pertama* di dalam

decoder).

3. *softmaxing* skor *alignment.*

   skor *alignment* untuk setiap *hidden state encoder* digabungkan dan direpresentasikan dalam satu vektor.

4. Menghitung *context vector*.

   encoder hidden state dan skor *alignment* masing-masing dikalikan untuk membentuk *context vector*.

5. *Decoding output*.

   Context vector digabungkan dengan *decoder output* sebelumnya dan dimasukkan kedalam RNN *decoder* untuk *time step* tersebut bersama dengan *decoder hidden state* sebelumnya untuk menghasilkan *output* baru.

6. Proses langkah 2 sampai 5 diulang untuk setiap *time step decoder* sampai diproduksi atau *output* melewati panjang maksimum yang ditentukan.

Dalam *attention* Bahdanau, cara untuk menghitung skor *alignment*, yaitu dengan *concat/additive*, seperti pada Persamaan 2.21 (Marathe, 2020).

$$\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) = \boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W_1}\boldsymbol{h}_t + \boldsymbol{W_2}\bar{\boldsymbol{h}}_s\right) \tag{2.21}$$

Cara *alignment score* dihitung di *attention* Bahdanau, dimana *decoder hidden state* dan *encoder hidden state* dikalikan dengan bobot *matrix* terlebih dahulu untuk mendapatkan bobot *matrix* masing-masing kemudian dijumlahkan. fungsi aktivasi *tanh* akan dikalikan dengan bobot *matrix* untuk menghasilkan *alignment score* seperti pada Persamaan 2.22 (Loye, 2019).

$$W_{combined} \cdot tanh(W_{decoder} \cdot H_{decoder} + W_{encoder} \cdot H_{encoder}) \tag{2.22}$$

2.12 *Toolkit* Mesin Penerjemah Jaringan Saraf Tiruan

Untuk membangun dan mengembangkan mesin penerjemah jaringan saraf tiruan dibutuhkan alat dengan sumber terbuka (*open-source*) yang dimana dalam praktiknya siapapun yang berminat dan tertarik pada NMT bisa mengembangkan mesin penerjemah sesuai kebutuhannya.

Dengan kemajuan pesat *deep learning*, sehingga menyediakan banyak *deep learning framework* dengan sumber terbuka seperti *tensorflow* oleh (Abadi et

al., 2016) dan *pytorch* (Paszke et al., 2019).

Perhatikan pada Gambar 2.22 yang menunjukkan beberapa *toolkit open-source* NMT populer pada github yang digunakan dalam membangun dan mengembangkan NMT yang diambil pada Desember 2020.

Name	Language	Framework	Status
Tensor2Tensor	Python	TensorFlow	Deprecated
FairSeq	Python	PyTorch	Active
Nmt	Python	TensorFlow	Deprecated
OpenNMT	Python/C++	PyTorch/TensorFlow	Active
Sockeye	Python	MXNet	Active
Nematus	Python	Tensorflow	Active
Marian	C++	–	Active
THUMT	Python	PyTorch/TensorFlow	Active
NMT-Keras	Python	Keras	Active
Neural Monkey	Python	TensorFlow	Active

Gambar 2. 23 Daftar *toolkit open source* NMT pada github [Tan et al., 2020]

Beberapa daftar *toolkit open source* NMT yang ditampilkan pada Gambar 2.23 akan diuraikan sebagai berikut.

1. *Tensor2Tenso*r

   *Tensor2Tenso*r diperkenalkan oleh (Vaswani et al., 2018) merupakan *library* dari model *deep learning* dan set data berdasarkan *TensorFlow* oleh (Abadi et al., 2016). *Library* ini dikembangkan oleh tim Google Brain. *Tensor2tensor* menyediakan implementasi dari beberapa arsitektur NMT misalnya *transformers* untuk tugas penerjemahan. Pengguna dapat menjalankannya dengan mudah di CPU, GPU, dan TPU, baik secara lokal maupun *cloud*.

2. *FairSeq*

   FairSeq diperkenalkan oleh (Ott et al., 2019) merupakan *toolkit* pemodelan urutan (*sequence*) dikembangkan oleh *Facebook* AI *Research*. *Toolkit* ini berbasis pada *Pytorch* (Paszke et al., 2019) dan memungkinkan pengguna untuk melatih model khusus untuk tugas terjemahan. *FairSeq* mengimplementasikan model berbasis RNN tradisional dan model *Transformer*. Selain itu, juga terjemahan berbasis model CNN misalnya *LightConv* dan *DinamicConv*.

3. NMT

NMT diperkenalkan oleh (Luong et al., 2017) merupakan *toolkit* yang dikembangkan oleh *Google Research*. *Toolkit* ini menjabarkan arsitektur GNMT (Wu et al., 2016). Selain itu proyek NMT juga menyediakan tutorial yang bagus untuk membangun model NMT yang kompetitif. Dengan basis *code* NMT berkualitas tinggi dan ringan, yang ramah bagi pengguna untuk menambahkan disesuaikan dengan model.

4. *OpenNMT*

   OpenNMT adalah *toolkit open-source* yang dikembangkan oleh kerja sama Universitas Harvard dan SYSTRAN. Pada *Toolkit* ini mengimplementasikan dua jenis yaitu *OpenNMT* -py dan *OpenNMT*-tf.

5. *Sockeye*

   Sockeye diperkenalkan oleh (Hieber et al., 2017) merupakan *toolkit* yang didasarkan pada MXNet (Chen et al., 2015). *Sockeye* dikelola oleh Amazon dan layanan mesin penerjemahan seperti Amazon translate. *Toolkit* ini memiliki fitur model mesin penerjemahan canggih dan inferensi CPU cepat yang berguna untuk riset maupun produksi.

6. *Nemantus*

   Nemantus adalah toolkit yang dikembangkan oleh NLP grup di Universitas Edinburgh. Toolkit ini berbasis TensorFlow, mendukung arsitektur RNN dan Transformer. Selain itu, *Nemantus* juga merilis performa tinggi model NMT dengan mencakup 13 arah terjemahan.

7. *Marian*

   Marian diperkenalkan oleh (Junczys-Dowmunt et al., 2015) yang dikembangkan oleh tim penerjemah Microsoft. *Framework* ditulis dengan C++. *Marian* digunakan oleh banyak perusahaan dan organisasi.

8. THUMT

   THUMT diperkenalkan oleh (Zhang et al., 2017) merupakan *toolkit* yang dikembangkan oleh NLP grup di Universitas Tsinghua. Toolkit ini termasuk implementasi *Theano* oleh (The Theano Development Team et al., 2016), *TensorFlow*, *Pytorch*. Mendukung vanilla RNN dan *Transformer* serta memudahkan pengguna untuk membuat model baru.

9. NMT-Keras

NMT-Keras diperkenalkan oleh (Peris & Casacuberta, 2018) merupakan toolkit fleksibel yang dikembangkan oleh *Pattern Recognition* dan *Human Language Technology Research Center* pada Universitas Politeknik di Valencia. *Toolkit* ini berbasis keras dengan menggunakan *Theano* atau *Tensorflow* sebagai *backend*. NMT-Keras menekankan pada pengembangan aplikasi dengan system NMT seperti NMT interaktif dan pembelajaran online serta tugas lainnya seperti *image/video captioning* dan klasifikasi kalimat.

10. *Neural Monkey*

    Neural Monkey diperkenalkan oleh (Helcl & Libovický, 2017) adalah *toolkit open-source* yang dibuat pada *library TensorFlow* dan menyediakan API tingkat tinggi disesuaikan untuk prototyping cepat untuk arsitektur kompleks.

2.13 Python

Python adalah bahasa pemrograman multifungsi yang dibuat oleh Guido van Rossum dan dirilis pada tahun 1991. GvR, nama lainnya disebut di komunitas python, menciptakan python untuk menjadi interpreter yang memiliki kemampuan penanganan kesalahan (*exception handling*) dan mengutamakan sintaksis yang mudah dibaca serta dimengerti (*readability*). Didesain untuk memudahkan dalam prototyping, Python menjadi bahasa yang sangat mudah dipahami dan fleksibel.

Efektivitas python cukup terbukti dengan banyaknya jumlah pengguna bahasa pemrograman ini. Berbagai survei memasukkan python dalam top-3 sebagai bahasa dengan penggunaan terbanyak, bersaing dengan Java dan PHP. Python dapat digunakan dalam mengakomodasi berbagai gaya pemrograman, termasuk structured, prosedural, berorientasi-objek, maupun fungsional. Python juga dapat berjalan pada berbagai sistem operasi yang tersedia. Beberapa pemanfaatan bahasa Python di antaranya:

1. *Web development* (server-side).
2. *Software development*.
3. *Mathematics & data science*.
4. *Machine learning*.
5. *System scripting*.

6. *Internet of Things (IoT) development*.

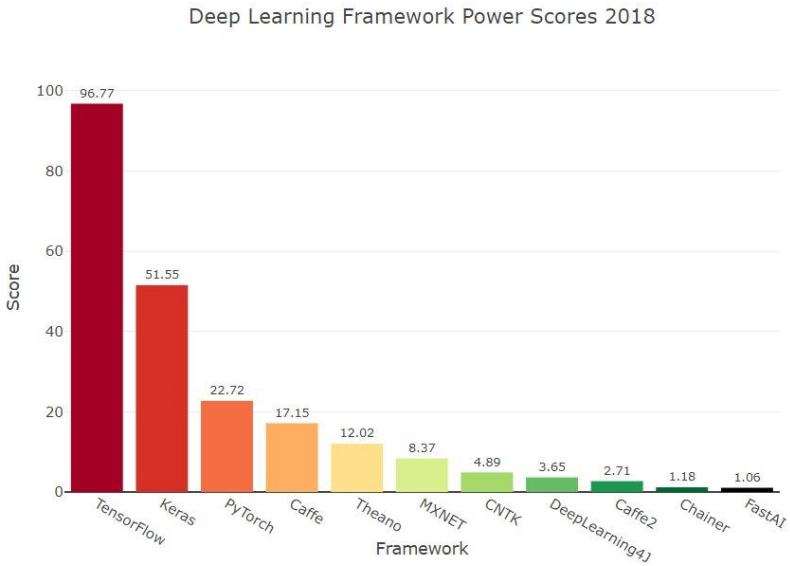
Saat ini, python juga menjadi salah satu bahasa pilihan untuk masuk ke dunia *data science*. Tiga hal utama pada data science yaitu *machine learning*, *data analysis*, dan *data visualization* banyak disediakan berbasis python. Sejumlah pustaka paling banyak digunakan dalam *machine learning* berbasis Python, misalnya: *Scikit-Learn*, *Tensorflow*, dan *PyTorch*.

2.14 TensorFlow

TensorFlow (TF) adalah *end-to-end open-source platform* yang dikembangkan oleh Google Brain dan sangat populer untuk pengembangan *machine learning* berskala besar. *TensorFlow* memiliki ekosistem *tools*, *library*, dan sumber daya komunitas yang komprehensif dan fleksibel, yang memungkinkan para peneliti dan pengembang membangun dan menerapkan aplikasi *machine learning* dengan mudah.

Pada awalnya *tensorflow* digunakan untuk menjalankan komputasi numerik kompleks pada riset AI dan *machine learning* di internal Google. Dalam perkembangannya kemudian, *tensorflow* menjadi *tools* efektif dan powerful untuk menyelesaikan permasalahan *deep learning* di kalangan masyarakat luas.

Seorang pegiat *data science*, Hale (2018) melakukan riset pada akhir tahun 2018 tentang *Deep Learning Framework Power Scores*. Riset tersebut menggunakan 11 sumber data pada 7 kategori yang berbeda untuk mengukur penggunaan dan popularitas *framework* dan ketertarikan pengguna. Hale (2018) menghitung dan melakukan visualisasi data risetnya pada *platform kaggle*, kemudian mempublikasikan hasil risetnya di medium. Dari riset tersebut, *tensorflow* menempati urutan pertama seperti ditunjukkan pada Gambar 2.24 berikut.



Gambar 2. 24 Grafik *framework deep learning* [Hale, 2018]

Tensorflow dirilis pertama kali pada akhir tahun 2015. Versi stabil *tensorflow* mulai dapat digunakan pada tahun 2017, sementara versi 2.x dirilis untuk publik pada bulan September 2019. Saat ini, *tensorflow* digunakan secara luas pada berbagai produk keluaran google yang kita gunakan sehari-hari, antara lain, Google *Cloud Speech*, Google *Photos*, Gmail, dan Google *Search*.

Abadi et al., (2016) dalam makalahnya yang berjudul "*Tensorflow: A System for Large Scale Machine Learning*" menjelaskan bahwa pada *tensorflow*, data dimodelkan sebagai tensor (array berdimensi-n) dengan elemen yang memiliki salah satu dari tipe data int32, float32, atau string. Secara alami, tensor mewakili masukan untuk operasi matematika dalam berbagai algoritma *machine learning*. Sebagai contoh, perkalian matriks membutuhkan dua buah tensor 2-D dan akan menghasilkan tensor 2-D juga.

TensorFlow menggunakan python sebagai front-end API-nya sehingga mudah dan nyaman digunakan, bahkan oleh pemula sekalipun. Perlu dicatat bahwa *tensorflow* ditulis dan dieksekusi dengan bahasa C++ yang berkinerja tinggi. Beberapa keunggulan *tensorflow* antara lain:

- Bisa dijalankan di hampir semua platform: GPU, CPU, dan TPU (*TensorFlow Processing Units*) yang secara khusus dimanfaatkan untuk mengerjakan matematika tensor.
- Memberikan performa terbaik dengan kemampuan melakukan iterasi

dan melatih model secara cepat sehingga mampu menjalankan lebih banyak eksperimen.

- Skalabilitas komputasi yang tinggi pada kumpulan data yang sangat besar.

TensorFlow menyediakan semua *tools* dan *library* yang kita butuhkan untuk proyek *machine learning* dari tahap *training* model hingga tahap produksi. Itulah alasan mengapa tensorflow disebut sebagai *end-to-end platform* untuk *machine learning*.

2.15 Google *Colaboratory*

Google *Colab* atau *Colaboratory* adalah salah satu produk google berbasis *cloud* yang bisa kita gunakan secara gratis dan dibuat khusus untuk tujuan penelitian. Programmer atau peneliti yang mungkin kesulitan untuk mendapatkan akses sumber daya dan komputer dengan spesifikasi tinggi dan mahal akan teratasi dengan menggunakan google *colaboratory*. Google *colaboratory* menggunakan bahasa pemrograman Python memiliki tampilan notebook mirip sekali dengan jupiter *notebook* gratis berbentuk *cloud* yang dijalankan menggunakan *browser*, seperti Mozilla Firefox dan Google Chrome. Seperti halnya Github yang menyediakan sumber kode penggunaan google colab sangat cocok dalam kolaborasi pekerjaan bersama yang memberikan layanan CPU, GPU, TPU dan RAM secara gratis. Tersedianya GPU dan TPU secara gratis memudahkan dalam model pelatihan terutama model deep learning yang membutuhkan waktu berjam-jam di CPU, bisa dihadapi dengan hitungan detik atau menit. Dalam menggunakan google *colaboratory* kita bisa memilih akses gratis atau berbayar, untuk akses gratis tentunya memiliki batasan penggunaan yaitu 12 jam dalam satu waktu proses, dengan gratis penggunaan RAM 12 GB dan Disk 107 GB.

Dapat lihat spesifikasi runtime berbeda yang ditawarkan oleh Google *Colaboratory* pada Gambar 2.25.

CPU	GPU	TPU
Intel Xeon Processor with two cores @ 2.30 GHz and 13 GB RAM	Up to Tesla K80 with 12 GB of GDDR5 VRAM, Intel Xeon Processor with two cores @ 2.20 GHz and 13 GB RAM	Cloud TPU with 180 teraflops of computation, Intel Xeon Processor with two cores @ 2.30 GHz and 13 GB RAM

Gambar 2. 25 Spesifikasi *runtime* google *colaboratory* [Sharma, 2020]

2.16 Google *Drive*

Google *Drive* adalah layanan google untuk media penyimpanan data online (daring) berbasis *cloud* atau Internet yang pertama kali beredar pada tanggal 24 April 2012. Pada dasarnya layanan google *drive* sama seperti *cloud storage* lain seperti *Dropbox* atau *OneDrive*. Layanan google *drive* memfasilitasi penggunanya untuk berkolaborasi, membuat, menyimpan, membagikan dan menyinkronkan file pada seluruh perangkat dengan catatan perangkat komputer atau smartphone harus terhubung dengan jaringan internet. Dengan google *drive* yang dapat menyimpan file-file berupa dokumen, gambar, audio ataupun video dengan kapasitas gratis sebesar 15 *gigabyte*. Jika ingin lebih dari itu maka harus *upgrade account*.

2.17 *Automatic Evaluation*

Suatu mesin penerjemah statistik membutuhkan suatu sistem evaluasi otomatis untuk menentukan kualitas terjemahan. Kualitas mesin penerjemah statistik secara umum dinilai dari hasil terjemahan yang dihasilkan. Penilaian dapat dilakukan secara manual dan otomatis. Penilaian secara manual merupakan cara penilaian yang terbaik karena memberikan nilai akurasi yang lebih tinggi. Namun penilaian secara manual memiliki kekurangan yaitu membutuhkan sumber daya manusia (ahli bahasa) dan tentunya membutuhkan waktu yang lama.

Sistem evaluasi otomatis yang populer saat ini adalah BLEU (*Bilingual Evaluation Understudy*). BLEU adalah sebuah algoritma yang berfungsi untuk mengevaluasi kualitas dari sebuah hasil terjemahan yang telah diterjemahkan oleh mesin dari satu bahasa alami ke bahasa lain. Ide utama dibalik ini adalah “semakin dekat terjemahan sebuah mesin dengan terjemahan manusia, maka akan semakin baik” (Papineni, 2002). BLEU mengukur *modified n-gram precision score* antara hasil terjemahan otomatis dengan terjemahan rujukan dan menggunakan konstanta yang dinamakan *brevity penalty.*

Untuk menghitung nilai *precision score* dapat dilakukan dengan menghitung jumlah kata pada hasil terjemahan (*unigram*) yang sesuai dengan rujukan dan dibagi dengan jumlah kata (*unigram*) yang ada pada hasil terjemahan. Namun sayangnya mesin penerjemah statistik dapat menghasilkan kata-kata dengan berlebihan, sehingga menghasilkan terjemahan yang mustahil tetapi memiliki *precision* yang tinggi. Dalam penelitian yang dilakukan oleh Papineni, dihasilkan beberapa perubahan yang dikenal dengan metode *modified* n-*gram precision.* Untuk menghitungnya, pertama kali hitung berapa kali jumlah maksimal dari kata yang muncul dalam terjemahan rujukan tunggal. Selanjutnya gabungkan jumlah total dari setiap kalimat terjemahan dengan jumlah maksimal rujukan. Contoh perhitungan dengan *modified unigram precision* diperlihatkan pada Tabel 2.2.

Tabel 2. 2 Contoh Perhitungan *Modified Unigram Precision* (Tanuwijaya, 2009)

Hasil Terjemahan	**Rujukan**
the the the the the the the	*The cat is on the mat*

Nilai *precision unigram* adalah 7/7. Sedangkan nilai *modified unigram* 2/7. Jumlah maksimum kata "*the*" pada rujukan adalah 2 dan jumlah *unigram* pada hasil terjemahan adalah 7 (Tanuwijaya, 2009). Nilai BLEU didapat dari hasil perkalian antara *brevity penalty* dengan rata-rata geometri dari *modified precision score*. Semakin tinggi nilai BLEU, maka semakin akurat dengan rujukan. Nilai dari BLEU berada pada rentang 0 sampai 1. Suatu terjemahan akan mencapai nilai 1 jika terjemahan tersebut identik dengan terjemahan rujukan. Oleh karena itu, meskipun dengan penerjemahan oleh manusia tidak mungkin akan menghasilkan nilai 1. Sangat penting untuk diketahui bahwa semakin banyak terjemahan rujukan per kalimatnya, maka akan semakin tinggi nilainya. Untuk menghasilkan nilai BLEU yang tinggi, panjang kalimat hasil terjemahan harus mendekati panjang dari kalimat referensi dan kalimat hasil terjemahan harus memiliki kata dan urutan yang sama dengan kalimat referensi. Rumus BLEU sebagai berikut dapat dilihat pada Persamaan 2.23, 2.24 dan 2.25.

$$BP_{BLEU} = f(x) = \begin{cases} 1, & if\ c > r \\ e^{(1-r/c)}, & if\ c \leq r \end{cases} \tag{2.23}$$

$$P_n = \frac{\sum_{C \epsilon \text{ corpus n-gram} \epsilon C} \sum count_{clip}(\text{n-gram})}{\sum_{C \epsilon \text{ corpus n-gram} \epsilon C} \sum count(\text{n-gram})} \quad (2.24)$$

$$BLEU = BP_{BLEU} . e^{\sum_{n-1}^{N} w_n \log P_n} \quad (2.25)$$

Keterangan:

BP = *brevity penalty*

c = jumlah kata dari hasil terjemahan otomatis

r = jumlah kata rujukan

Pn = *modified precision score*

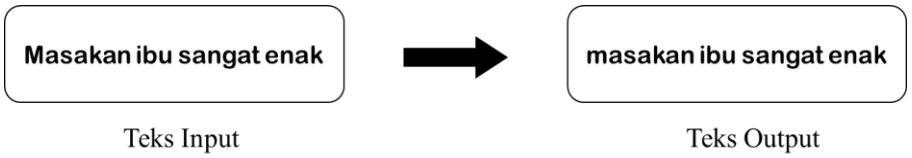
wn = 1/N (standar nilai N untuk BLEU adalah 4)

pn = jumlah *n-gram* hasil terjemahan yang sesuai dengan rujukan dibagi jumlah n-gram hasil terjemahan

2.18 *Case Folding*

Case folding merupakan tahapan mengubah huruf dalam dokumen menjadi huruf kecil. Karakter selain huruf dihilangkan dan dianggap pembatas (*delimiter*) (Marlinda & Rianto, 2013).

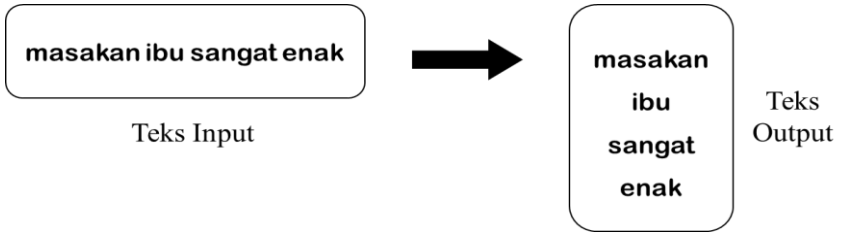
Contoh penggunaan *case folding* yaitu pada Gambar 2.26.



Gambar 2. 26 Contoh *case folding*

2.19 *Tokenizing*

Tahap *tokenizing* adalah tahap pemotongan string *input* berdasarkan tiap kata yang menyusunnya (Marlinda & Rianto, 2013). Contoh dari tahap *tokenizing* seperti terlihat pada Gambar 2.27.

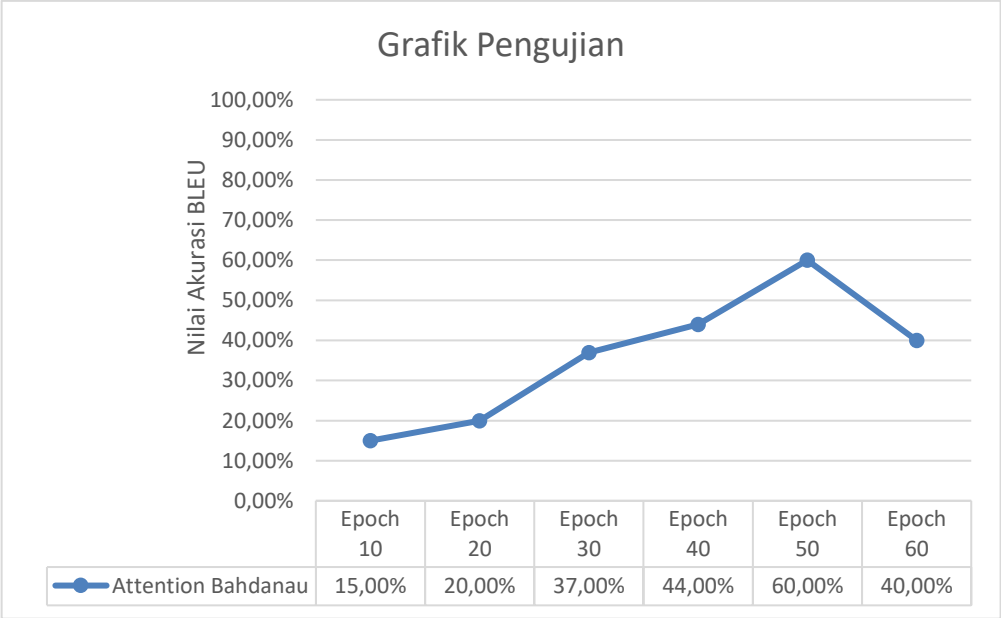


Gambar 2. 27 Contoh *tokenizing*

2.20 Cara Pengujian

Adapun cara pengujian yang akan dilakukan dengan menguji korpus yang sudah ada dengan percobaan kedua mekanisme *attention* yang digunakan untuk melihat grafik yang dihasilkan. Dilakukannya pengujian beberapa kali dengan berdasarkan jumlah *epoch*.

Berikut cara pengujian dapat dilihat pada Gambar 2.28.



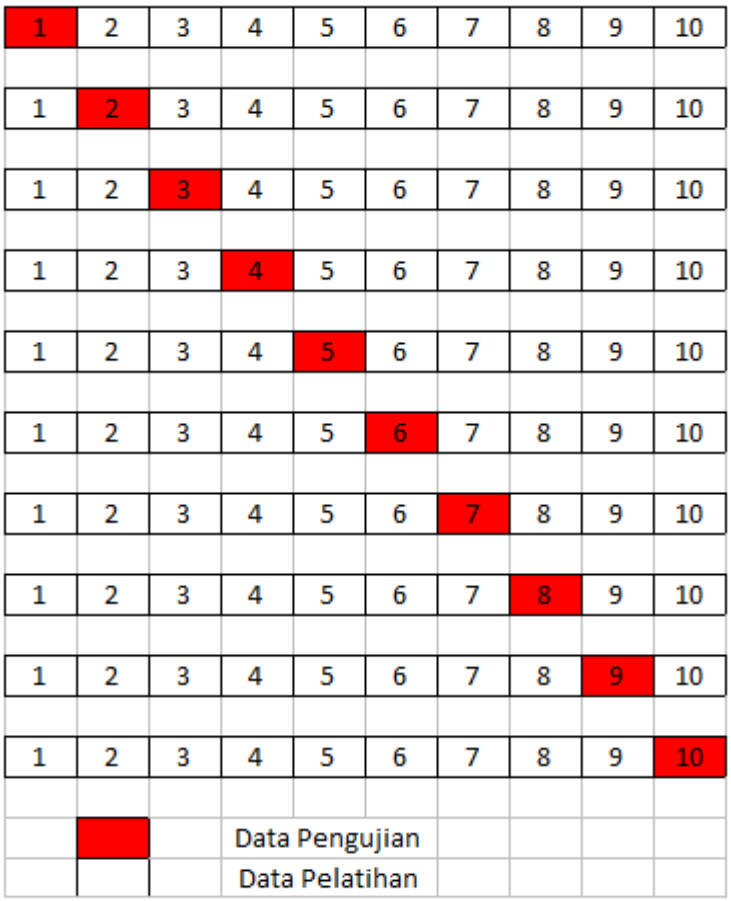
Gambar 2. 28 Contoh grafik pengujian jumlah *epoch*

Adapun untuk penelitian ini akan dilakukan pengujian seperti pada Gambar 2.28 dengan melakukan percobaan dengan penambahan secara konsisten yaitu 10 *epoch* sampai pada jumlah *epoch* yang hasil akurasinya menurun yang dihitung dengan BLEU, pada pengujian ini menggunakan data sampel korpus yang jumlah korpus uji dan korpus latihnya ditetapkan sendiri dengan metode *K-Fold Cross Validation*. Semua data korpus tersebut dilakukan pelatihan dengan jumlah *epoch* yang ditentukan sampai nilai BLEU *score* yang dihasilkan terlihat penurunan akurasi dari sebelumnya dan di rata-ratakan, maka dari situ pengujiannya sudah bisa

disimpulkan dan untuk jumlah *epoch* yang akan dijadikan untuk memilih data sampel yaitu terletak pada nilai BLEU *Score* pada masing-masing sampel data yang nilai akurasinya paling tinggi pada perbandingan diatas. Kemudian setelah mengetahui jumlah *epoch* pada data sampel yang memiliki nilai BLEU score tertinggi tersebut sebagai hasil pengujian otomatis dan pengujian berlanjut dengan manual pada sampel yang didapatkan dilakukan metode *K-Fold Cross Validation*.

2.21 *K-Fold Cross Validation*

Cross-validation (CV) adalah metode statistik yang dapat digunakan untuk mengevaluasi kinerja model atau algoritma dimana data dipisahkan menjadi dua subset yaitu data proses pembelajaran dan data validasi / evaluasi. Model atau algoritma dilatih oleh subset pembelajaran dan divalidasi oleh subset validasi. Selanjutnya pemilihan jenis CV dapat didasarkan pada ukuran dataset. Biasanya CV *K-Fold* digunakan karena dapat mengurangi waktu komputasi dengan tetap menjaga keakuratan estimasi dapat dilihat pada Gambar 2.29.



Gambar 2. 29 Contoh skema 10 *fold cross validation*

Adapun untuk penelitian ini akan digunakan skema 10 *fold cross validation*, sehingga akan dilakukan 10 kali percobaan pengujian akurasi setiap mekanisme *attention*. Jadi dalam pemilihan data untuk korpus uji dari jumlah total korpus kalimat sebanyak 5.000 dibagi menjadi 10 kali percobaan didapatkan 500 kalimat untuk setiap percobaannya. Misalnya pada percobaan 1 korpus ujinya diambil dari kalimat 1 sampai 500 sedangkan untuk korpus yang dilatih diambil dari kalimat 501 sampai 5000, percobaan 2 korpus ujinya diambil dari kalimat 501 sampai 1000 sedangkan korpus yang dilatih diambil dari kalimat 1 sampai 500 dan 1001 sampai 5000 begitu seterusnya sampai percobaan ke 10. Adapun untuk perhitungan rata-rata nilai akurasi dari algoritma yang nilai BLEU *score* nya paling tinggi pada mekanisme *attention* dapat dinyatakan dengan Persamaan 2.26 sebagai berikut.

$$rata - rata\ akurasi = \frac{Nilai\ akurasi_1 + Nilai\ akurasi_2 + .. + Nilai\ akurasi_n}{Jumlah\ Percobaan} \quad (2.26)$$

Salah satu cara utama untuk menggunakan validasi silang daripada menggunakan validasi konvensional (misalnya mempartisi kumpulan data menjadi dua set, yaitu 70% untuk pelatihan dan 30% untuk pengujian) adalah tidak cukup data yang tersedia untuk mempartisinya menjadi pelatihan terpisah dan data pengujian akan kehilangan pemodelan atau kemampuan pengujian yang signifikan. Dalam kasus ini, cara yang adil untuk memprediksi model prediksi dengan tepat adalah dengan menggunakan validasi silang sebagai teknik umum yang kuat (Grossman, 2010).

Singkatnya, validasi silang menggabungkan (rata-rata) ukuran kecocokan (prediksi *error*) dengan melihat nilai rata-rata pada setiap putaran untuk mendapatkan perkiraan kinerja model prediksi yang lebih akurat.