AUTOMATIC TEXT SUMMARIZATION BERITA BAHASA INDONESIA MENGGUNAKAN METODE *ATTENTIONAL ENCODER DECODER*

Disusun Oleh:

N a m a : Gilang Akbar

NIM : 14523091

PROGRAM STUDI INFORMATIKA – PROGRAM SARJANA

FAKULTAS TEKNOLOGI INDUSTRI

UNIVERSITAS ISLAM INDONESIA

2021

HALAMAN PENGESAHAN DOSEN PEMBIMBING

AUTOMATIC TEXT SUMMARIZATION BERITA BAHASA INDONESIA MENGGUNAKAN METODE *ATTENTIONAL ENCODER DECODER*

TUGAS AKHIR

Disusun Oleh:

N a m a : Gilang Akbar

NIM : 14523091

Yogyakarta, 13 Desember 2021

Pembimbing,

(Ahmad Fathan Hidayatullah, S.T., M.Cs.)

HALAMAN PENGESAHAN DOSEN PENGUJI

AUTOMATIC TEXT SUMMARIZER BERITABAHASA INDONESIA MENGGUNAKAN METODE *ATTENTIONAL ENCODER DECODER*

TUGAS AKHIR

Telah dipertahankan di depan sidang penguji sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer dari Program Studi Informatika – Program Sarjana di Fakultas Teknologi Industri Universitas Islam Indonesia

Yogyakarta, 13 Desember 2021

Tim Penguji

Ahmad Fathan Hidayatullah, S.T., M.Cs. _____

Anggota 1
Chanifah Indah Ratnasari, S.Kom., M.Kom _____

Anggota 2
Chandra Kusuma Dewa, S.Kom., M.Cs., Ph.D. _____

Mengetahui,
Ketua Program Studi Informatika – Program Sarjana
Fakultas Teknologi Industri
Universitas Islam Indonesia

(Dr. Raden Teduh Dirgahayu, S.T., M.Sc.)

HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR

Yang bertanda tangan di bawah ini:

Nama : Gilang Akbar

NIM : 14523091

Tugas akhir dengan judul:

AUTOMATIC TEXT SUMMARIZER BERITA BAHASA INDONESIA MENGGUNAKAN METODE *ATTENTIONAL ENCODER DECODER*

Menyatakan bahwa seluruh komponen dan isi dalam tugas akhir ini adalah hasil karya saya sendiri. Apabila di kemudian hari terbukti ada beberapa bagian dari karya ini adalah bukan hasil karya sendiri, tugas akhir yang diajukan sebagai hasil karya sendiri ini siap ditarik kembali dan siap menanggung risiko dan konsekuensi apapun.

Demikian surat pernyataan ini dibuat, semoga dapat dipergunakan sebagaimana mestinya.

Yogyakarta, 1 Desember 2021

(Gilang Akbar)

HALAMAN PERSEMBAHAN

Tugas ini penyusun persembahkan untuk orang tua, adik, dosen-dosen pengajar, teman-teman seperjuangan dan Universitas Islam Indonesia tempat saya mencari ilmu.

HALAMAN MOTO

Every success and failure, up and down, high and low, laugh and tear, Everything, is a part of the journey to gain and learn from.

KATA PENGANTAR

Assalamualaikum Warahmatullahi Wabarakaatuh

Puji syukur penulis panjatkan ke hadirat Allah SWT, yang atas rahmat-Nya maka penulis dapat selesai menyusun laporan tugas akhir dengan judul "*AUTOMATIC TEXT SUMMARIZER* BERITA BAHASA INDONESIA MENGGUNAKAN *ATTENTIONAL ENCODER DECODER*".

Penyusunan laporan tugas akhir ini merupakan tahap akhir dari masa studi di Teknik Informatika, Fakultas Teknik Industri, Universitas Islam Indonesia.

Selama penyusunan, penulis menerima banyak bantuan dan dukungan dari berbagai pihak. Maka dari itu penulis ingin menyampaikan rasa terima kasih kepada:

1. Orang tua penulis, Bapak Thamrin dan Ibu Mardiah yang telah membesarkan penulis.
2. Adik kandung perempuan, Fadhilatul Hilya yang terus memberikan dukungan moril selama penyusunan laporan.
3. Bapak Fathul Wahid, S.T., M.Sc., Ph.D., sebagai Ketua Rektor Universitas Islam Indonesia beserta.
4. Bapak Prof. Dr. Ir. Hari Purnomo, M.T, selaku Dekan Fakultas Teknologi Industri Universitas Islam Indonesia.
5. Bapak Hendrik, ST., M.Eng. selaku Ketua Jurusan Teknik Informatika Universitas Islam Indonesia.
6. Bapak Dr. Raden Teduh Dirgahayu, S.T., M.Sc. selaku Ketua Program Studi Teknik Informatika Universitas Islam Indonesia.
7. Bapak Ahmad Fathan Hidayatullah, S.T., M.Cs. selaku dosen pembimbing yang terus memberikan pengarahan, dukungan dan kepercayaan.
8. Seluruh jajaran karyawan Universitas Islam Indonesia yang selalu berkomitmen dalam menjaga kelancaran berjalannya kegiatan belajar mengajar.
9. Teman-Teman seperjuangan yang saling berbagi rasa suka dan duka selama masa studi di Universitas Islam Indonesia.
10. Kepada seluruh pihak yang telah berjasa membantu penyusunan laporan skripsi ini, tetapi tidak bisa saya sebutkan satu per satu.

Penulis menyadari bahwa laporan tugas akhir ini masih jauh dari sempurna. Oleh karena itu, penulis menerima semua kritik dan saran atas hasil dari tugas akhir ini. Semoga apa yang telah penulis tuangkan dalam laporan ini dapat berguna bagi rekan-rekan akademik sebagai

referensi untuk segala bentuk kegiatan ke depannya. Akhir kata penulis ucapkan semoga Allah SWT selalu melimpahkan rahmat dan hidayah-Nya kepada kita semua. *Aamiin aamiin ya robbal'alamin.*

Wassalamu'alaikum Warahmatullahi Wabarakatuh

Yogyakarta, 13 Desember 2021

(Gilang Akbar)

SARI

Automatic text summarization (ATS) adalah salah satu bidang pada kecerdasan buatan untuk otomatisasi tugas peringkasan data teks. ATS semakin banyak dibutuhkan seiring bertambah cepatnya peredaran informasi. peringkasan abstraktif menjadi tujuan utama dalam pengembangam program model ATS. model *deep learning attentional encoder decoder* umum digunakan untuk tugas ATS abstraktif. Beberapa penelitian berhasil meningkatkan performa model dengan penerapan metode terbaru seperti, *pointer-generator*, *bidirectional*, dan *word embedding*. Penelitian pada Bahasa Indonesia sudah ada dilakukan, namun belum menerapkan metode-metode terbaru yang disebutkan sebelumnya. Untuk itu akan dibangun model *attentional encoder decoder* ATS berita bahasa indonesia, yang menerapkan metode yang disebutkan sebelumnya. Hasil penelitian menemukan bahwa penggunaan metode *pointer-generator* mempercepat pengurangan *loss* pada proses pelatihan. Analisis nilai ROUGE mendapatkan bahwa model 2 yang hanya menerapkan *bidirectional* memiliki hasil terbaik, dengan nilai ROUGE-1 0.154, ROUGE-2 0.018 dan ROUGE-L 0.148. Hal tersebut membuktikan bahwa performa *loss* tidak berbanding lurus dengan nilai ROUGE. Pada analisa manual hasil ringkasan didapatkan bahwa sesuai dengan evaluasi ROUGE, model 2 berhasil mengambil inti dari dokumen *input*. Repetisi kalimat masih menjadi masalah yang mengurangi kohesifnya hasil ringkasan. Masih diperlukan eksplorasi *hyperparameter* lebih lanjut untuk menemukan performa paling maksimal dari varian model. Penggunaan metode lain seperti *coverage* juga disarankan untuk mengatasi repetisi kata.

Kata kunci: *automatic text summarization*, LSTM, *encoder decoder*, *pointer-generator*, FastText, *bidirectional*, ROUGE, *early stopping*, *teacher forcing*, AdaGrad

GLOSARIUM

Batch	jumlah data yang diolah sebelum dihitung nilai loss.
Deep learning	salah satu bidang di ranah kecerdasan buatan yang mencakup pengembangan program jaringan saraf tiruan dengan parameter pembelajaran yang besar.
Epoch	satu langkah ketika semua data latihan telah selesai diproses
Input	samasukan untuk proses pelatihan model
Iteration	satu langkah ketika satu *batch* selesai di proses.
Konvergen	Proses model mencapai performa optimal.
Layer	lapisan pada model *deep learning*, yang berupa salah satu topologi jaringan.
Learning rate	parameter yang mengontrol seberapa besar perubahan variabel pembelajaran setiap langkah.
Local minima	posisi *loss* yang minimum pada posisi gradien tertentu namun bukan yang paling minimum secara keseluruhan daerah.
Loss	matrik pengukur nilai *error* dari kinerja model.
Optimizer	metode/algoritma yang membantu meningkatkan efektifitas dan efisiensi pelatihan, spesifik ke pengolahan gradien.
Overfitting	keadaan saat performa model terlalu baik dengan data latihan namun buruk dengan data baru/asing.
State	representasi vektor pada RNN untuk tiap sekuens.
Vektor	matriks/tensor dimensi 1.
Vocabulary	kumpulan kata beserta indeksnya masing-masing, yang menjadi acuan model dalam mengubah data teks menjadi numerik atau sebaliknya.

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB I
PENDAHULUAN

1.1 Latar Belakang

Automatic text summarization (ATS) adalah salah satu bidang dari pengolahan bahasa alami kecerdasan buatan yang berfokus pada otomatisasi tugas peringkasan data teks oleh mesin. Semakin bertambahnya informasi yang di sebarkan di dunia maya, tugas peringkasan teks semakin menjadi relevan. Berkat alasan tersebut ATS telah banyak diterapkan. Salah satu penerapan dari ATS adalah pada peringkasan dokumen berita yang dapat mengoptimalkan kegiatan konsumsi berita oleh pembaca (Dehru et al., 2021). Ringkasan berita dapat dijadikan sebagai pratinjau untuk pembaca mengetahui gambaran isi dari sebuah artikel berita sebelum membaca. fitur itu membantu pembaca menimbang keputusan untuk membaca suatu dokumen berita atau tidak, sehingga menambah efisiensi dan efektifitas kegiatan selancar berita. Program ATS juga mengurangi waktu yang diperlukan dalam peringkasan berita terutama jika dibandingkan dengan peringkasan manual oleh manusia, sehingga dapat menambah efisiensi kinerja (Dehru et al., 2021).

Berdasarkan sifat keluarannya, ATS dibedakan menjadi dua, yaitu ekstraktif dan abstraktif. ATS ekstraktif menghasilkan ringkasan dengan cara memilih lalu menggabungkan kalimat-kalimat yang dianggap terbaik merepresentasikan intisari dokumen. Sedangkan ATS abstraktif menghasilkan ringkasan dengan membangun terlebih dahulu representasi semantik internal yang lalu digunakan kembali sebagai sumber untuk menghasilkan ringkasan, dengan harapan mendekati standar ringkasan buatan manusia (Al-Sabahi et al., 2018). ATS abstraktif secara relatif lebih kompleks dibanding ekstraktif, dikarenakan kebutuhan akan pengolahan bahasa lebih mendalam untuk parafrase (Widyassari et al., 2020).

Attentional encoder decoder merupakan salah satu arsitektur deep learning yang biasa digunakan untuk tugas ATS abstraktif. Arsitektur ini terbagi menjadi tiga bagian. *encoder* berfungsi untuk mempelajari dokumen *input* lalu mengolahnya menjadi representasi sekuens untuk tiap kata. *Decoder* berfungsi untuk mengambil representasi data *input* yang dihasilkan *encoder* dan mengolahnya menjadi hasil ringkasan. berdasarkan masukan vektor representasi yang diberikan. *Attentional mechanism* diaplikasikan untuk mengatasi masalah kehilangan informasi pada model *encoder decoder* dengan cara menghasilkan vektor representasi untuk memberi informasi bagi *decoder* mengenai bagian mana dari keluaran *encoder* yang lebih perlu

diperhatikan (Hu, 2018). *encoder* dan *decoder* dapat diaplikasikan menggunakan berbagai macam tipe jaringan *deep learning* seperti CNN, RNN LSTM, RNN GRU hingga deretan *self attention* (*transformer*).

Beberapa penelitian lanjutan berhasil memberikan solusi bagi kekurangan yang masih dimiliki model *attentional encoder decoder* terutama untuk tugas ATS. Penggunaan *bidirectional* LSTM menambah kemampuan *encoder* dan model secara keseluruhan untuk lebih memahami konteks dari dokumen *input* (Al-Sabahi et al., 2018). Penelitian oleh See et al. (2017) menerapkan penggunaan *pointer-generator* dan *coverage*. *Pointer-generator* mengurangi ketergantungan model terhadap *vocabulary* dengan memberikan kemampuan untuk meniru langsung kata dari dokumen masukan. Metode *coverage* berfungsi untuk mengurangi repetisi kata pada hasil ringkasan. Penelitian lebih lanjut oleh Dang & Nguyen (2019) menerapkan *pretrained word embedding* word2vec dan FastText untuk menambah performa dari *attentional encoder decoder*. Penelitan-penelitian tersebut menggunakan data berita Bahasa Inggris CNN/daily mail. Penelitian ATS untuk Bahasa Indonesia menggunakan metode *attentional encoder decoder* juga telah dilakukan. Yoko et al. (2018) menggunakan model *attentional encoder decoder* dipadukan dengan *word embedding* word2vec untuk mengembangakan model peringkas berita bahasa indonesia. selanjutnya ada juga penelitian dengan data jurnal indonesia yang menerapankan *Bidirectional* GRU pada *encoder* (Adelia et al., 2019). Ke dua penelitian berfokus pada bahas indonesia ini menginspirasi penulis untuk mengembangkan penelitian ini.

Model ATS bahasa indonesia pada penelitian Yoko et al. (2018) dan Adelia et al. (2019) belum memanfaatkan metode *pointer-generator* untuk mengambil kata dari dokumen *input*, *bidirectional* LSTM untuk kemampuan pemahaman konteks, dan *word embedding* FastText untuk representasi kata yang lebih baik dari word2vec. Berdasarkan hal tersebut penulis akan menenerapan metode-metode di atas pada *attentional encoder decoder* ATS berita Bahasa Indonesia untuk mengetahui efektifitasnya. Pada penelitian ini akan dikembangkan beberapa model dengan variasi penerapan metode *Bidirectional* LSTM, FastText, dan *pointer-generator*. Evaluasi dilihat dari 3 aspek, yaitu analisis kecepatan grafik *loss* pada saat training, analisis metrik ROUGE (Lin, 2004) dan analisis manual langsung kualitas hasil ringkasan. Data yang digunakan akan diambil dari situs berita online Bahasa Indonesia, liputan6.com (Koto et al., 2020).

1.2 Rumusan Masalah

Berdasarkan dari latar belakang di atas, maka rumusan masalah penelitian dapat di jabarkan sebagai berikut:

a. Apa alur dari pembangunan model *Attentional encoder decoder* untuk pengolahan berita Bahasa Indonesia dengan data dari situs liputan 6?
b. Apa hasil dari penerapan metode bidirectional LSTM, FastText, dan *pointer-generator* terhadap pergerakan grafik *loss*, evaluasi metrik ROUGE dan analisis manual hasil keluaran model *attentional encoder decoder* untuk *automatic text summarization* berita Bahasa Indonesia?

1.3 Tujuan Penelitian

Penelitian ini dikembangkan dengan tujuan sebagai berikut:

a. Membangun model *attentional encoder decoder automatic text summarization* berita Bahasa Indonesia.
b. Mendapatkan informasi mengenai efektifitas dari penggunaan *bidirectional* LSTM, FastText, *pointer-generator* terhadap model *attentional encoder decoder automatic text summarization* berita Bahasa Indonesia.

1.4 Batasan Masalah

Di dalam pengembangan penelitian ini ditetapkan beberapa batasan masalah dengan maksud agar terhindar dari penyimpangan maupun pelebaran pokok masalah. Batasan masalah sebagai berikut:

a. Data yang digunakan diambil hanya dari satu situs berita online bahasa Indonesia, yaitu Liputan6.com.
b. Semua varian model memiliki *hyperparameter* yang sama selain untuk penerapan *layer*.
c. Maksimal panjang data berita *input* dan keluaran ringkasan ditetapkan berdasarkan analisis data.

1.5 Manfaat Penelitian

Dari hasil pengembangan penelitian ini penulis mengharapkan dua manfaat yang dapat dihasilkan, yaitu:

a. Mendapatkan model *automatic text summarization* yang berfungsi dengan baik.

b. Memberi referensi akan efektifitas dari penggunaan bidirectional LSTM, FastText dan *pointer-generator* terhadap model *automatic text summarization* berita Bahasa Indonesia, terutama bagi penelitian lanjutan ke depannya.

1.6 Sistematika Penulisan

Laporan tugas akhir ini ditulis secara sistematis, bersinergi dengan laju pengembangan penelitian. Isi laporan dibagi menjadi lima bagian sebagai berikut:

BAB I Pendahuluan, menjabarkan latar belakang dari keputusan melakukan penelitian dengan judul "*AUTOMATIC TEXT SUMMARIZATION* BERITA BAHASA INDONESA MENGGUNAKAN *ATTENTIONAL ENCODER DECODER*", rumusan masalah, tujuan penelitian, batasan masalah dan manfaat penelitian.

BAB II Landasan Teori, memaparkan beberapa penelitian terkait dan teori-teori penelitian sebagai acuan pendukung atau pembanding dalam pengembangan penelitian ini.

BAB III Metodologi Penelitian, menjabarkan semua tahap dari rencana pengembangan penelitian ini.

BAB IV Hasil & Pembahasan, menjabarkan semua hasil dari penelitian yang telah dilakukan. Di dalamnya berupa pembahasan program dan evaluasi tiap varian obyek penelitian.

BAB V Kesimpulan & Saran, memuat kesimpulan yang didapat penulis dari hasil penelitian dan saran penulis terhadap pengembangan penelitian berikutnya yang masih terkait.

BAB II
LANDASAN TEORI

2.1 Penelitian Terkait

Dalam mengembangkan penelitian ini penulis menggunakan beberapa penelitian sebelumnya sebagai referensi acuan. Ringkasan penelitian-penelitian referensi tersebut dapat dilihat pada Tabel 2.1.

Tabel 2.1 Penelitian Referensi

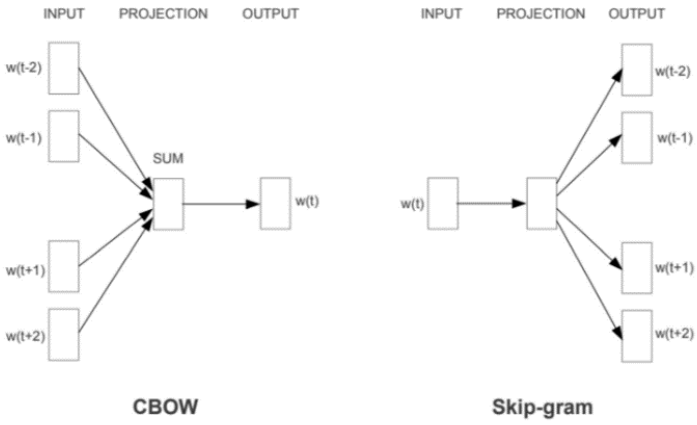
Referensi	Metode	Rangkuman
Yoko et al. (2018)	*Encoder decoder*, LSTM, *attentional mechanism*, *w*ord2vec, ROUGE	Penelitian membandingkan beberapa model dengan variasi *epoch* dan *hidden state*. Data yang digunakan dari dua situs berita detik.com dan Kompas.com. varian model terbaik memiliki akurasi dengan *loss* 0.0006654 dan *accuracy* 99.8810%. namun masih mengalami banyak kendala dengan kata tidak dikenal seperti subjek atau lokasi.
Adelia et al. (2019)	*Encoder decoder*, *bidirectional* GRU, *attentional mechanism*, *w*ord2vec, ROUGE	Menggunakan data jurnal Bahasa Indonesia. Varian model *bidirectional* GRU ada dua dengan perbedaan hidden units 128 dan 64. Hasil penelitian diukur dengan metrik ROGUE-1 dan ROGUE-2. Didapatkan bahwa model dengan *hidden units* 128 memiliki hasil lebih baik dengan nilai ROUGE-1 (*unigram*) sejumlah 0.11975 dan ROUGE-2 (*bigram*) sejumlah 0.011975. Namun, masih jauh dari nilai ideal pada model Bahasa inggris. Inspeksi manual keluaran juga menunjukkan ringkasan yang memiliki struktur yang tidak mumpuni secara semantik.
Al-Sabahi et al. (2018)	*Encoder decoder*, *bidirectional* LSTM, *pointer-generator*, *coverage*, *attentional mechanism*, ROUGE	Model *Bidirectional* LSTM hasil penelitian memiliki hasil yang lebih baik dibanding LSTM tradisional. Dengan tambahan penerapan *pointer-generator* dan *coverage*, penggunaan LSTM *bidirectional* menghasilkan nilai metrik ROUGE tertinggi dengan R-1 42.6% dan R-2 18.8%. model dapat lebih efektif dalam memahami konteks antar kata dalam kalimat.

See et al. (2017)	*Encoder decoder*, b*idirectional* LSTM, *pointer-generator*, *coverage*	Model yang menerapkan metode *pointer-generator* dan *coverage* memiliki nilai skor ROUGE yang paling tinggi dengan ROUGE-1 39.53, ROUGE-2 17.28 dan ROUGE-L 36.38. Model tersebut dapat mengambil kata-kata baru dari data *input* dan juga meminimalisir repetisi kata, menjadikan hasil keluaran yang mendekati buatan manusia.
Dang & Nguyen (2019)	*Encoder decoder, bidirectional* LSTM, *attentional* m*echanism*, *pointer-generator*, *coverage*, word2vec, FastText, ROUGE	Varian model dikembangkan dengan fokus untuk membandingkan performa *word embedding* FastText dan word2vec. Model dasar adalah *base*, *pointer-generator* dan *pointer-generator coverage*. Hasil akhir menunjukkan penggunaan FastText lebih unggul dibanding word2vec dengan nilai ROUGE-1 39.06, ROUGE-2 17.05, dan ROUGE-L 35.85.

Penelitian referensi pada Tabel 2.1 yang menggunakan bahasa indonesia belum memanfaatkan beberapa metode terbaru. Sebaliknya pada penelitian penggunaan *bidirectional* LSTM, *pointer-generator*, dan *word embedding* FastText hanya menggunakan data berita bahasa inggris. akan dilakukan pengembangan model ATS *attentional encoder decoder* berita bahasa indonesia dengan menerapkan metode-metode yang diterapkan oleh penelitian referesnsi yang dapat meningkatkan kualitas hasil ringkasan dan ROUGE. Beberapa metode tersebut adalah *pre-trained word embedding* FastText, *bidirectional*, dan *pointer-generator*. Pemilihan *hyperparameter* seperti *epoch*, *batch*, *iteration*, dan unit akan mengacu pada penelitian referensi juga.

2.2 FastText

FastText merupakan salah satu tipe model dalam *word embedding* dan pengembangan dari model word2vec. *Word embedding* adalah bidang dalam pengolahan bahasa alami yang bertugas menghasilkan representasi kata ke dalam bentuk vektor. Layaknya word2vec, cara FastText bekerja adalah dengan memproses dokumen masukan menggunakan aturan *skip-gram* atau *continuous bag of words* (CBOW) (Mikolov et al., 2013).



Gambar 2.1 Bagan alur CBOW dan *Skip-gram*.
Sumber: Chaudhary (2020)

Fitur tambahan dari FastText adalah kemampuan untuk menghitung representasi dari macam suku kata yang dimiliki sebuah kata (Bojanowski et al., 2016).

Gambar 2.2 Pembagian pada level suku kata.
Sumber: Kulshrestha (2019)

Dapat dilihat pada Gambar 2.2, memilah kata pada level suku kata dapat meningkatkan kemampuan untuk mempelajari fitur kata seperti mengenali imbuhan pada kata Bahasa Indonesia.

2.3 *Encoder Decoder*

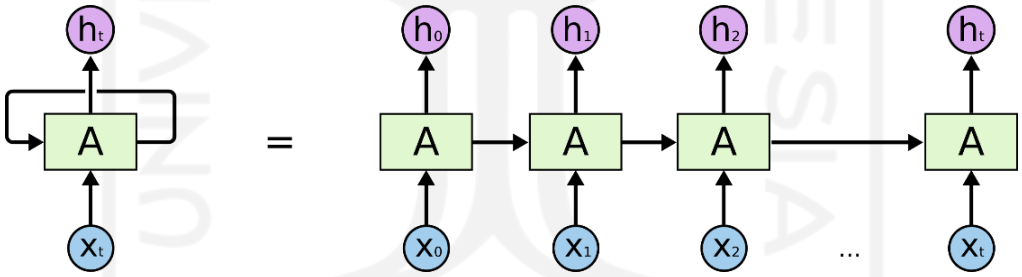
Encoder decoder adalah salah satu arsitektur dalam *deep learning* yang umumnya digunakan untuk pembelajaran *unsupervised*, yaitu data yang tidak memiliki label (Kramer, 1991). Arsitektur ini akan dibagi menjadi dua bagian utama. Bagian pertama adalah *encoder*. bagian ini berfungsi untuk mengolah data masukan menjadi sebuah *state* atau vektor representasi yang mewakili informasi-informasi dari data masukan tersebut. Bagian kedua

adalah *decoder*. bagian ini bertugas untuk mengambil *state* atau vektor representasi yang dihasilkan *encoder*, lalu mengolahnya menjadi keluaran yang diinginkan.

Pada bagian encoder dan decoder dapat terdiri dari beberapa *sub-layer*. Tergantung kebutuhan layer neural networks yang biasa digunakan adalah LSTM, GRU hingga CNN. Arsitektur *encoder decoder* telah banyak dimanfaatkan untuk beberapa tugas, diantaranya adalah untuk kompresi gambar (Cheng et al., 2018), *denoising* gambar (Lee et al., 2021), deteksi anomali (Jin et al., 2019), *dimensionality reduction* (Wang et al., 2015), *machine translation* (Cho et al., 2014) hingga *automatic text summarization* (Shi et al., 2021).

2.4 LSTM (*Long Short-Term Memory*) dan *Bidirectional* LSTM

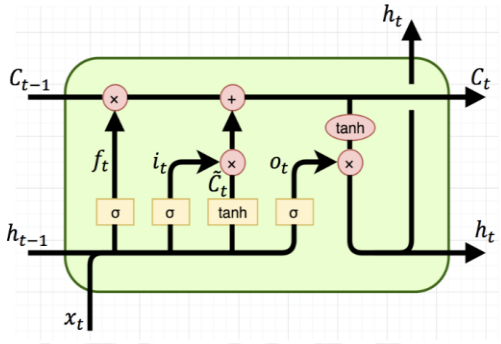
LSTM (*long short-term memory*) merupakan salah satu varian arsitektur dari *recurrent neural network* (RNN). Arsitektur RNN secara umum dirancang khusus untuk mengolah data sekuensial dengan pembelajaran *internal state* melalui hubungan antara layer per urutan sekuens data. Hal tersebut menyebabkan RNN dapat mempelajari informasi sebuah data masukan menggunakan informasi dari data masukan sekuens sebelumnya.



Gambar 2.3 Alur RNN dasar.

Sumber: Olah (2015)

RNN masih memiliki kelemahan pada kemampuan mengingat. Apabila dihadapkan dengan data yang memiliki sekuens sangat Panjang, RNN akan cenderung kehilangan informasi data masukan sekuens awal. Penyebab dari hal tersebut karena terjadinya *vanishing gradient* atau *blow up gradient* di mana pada saat dilakukan backpropagation nilai weight sebuah akan menjadi sangat besar atau sangat kecil sehingga menghambat laju pembelajaran (Hochreiter & Schmidhuber, 1997). Diperlukan *state* tambahan yang khusus bertugas untuk mengingat, maka dari itu dikembangkanlah varian RNN baru Bernama *long short-term memory* atau biasa disingkat LSTM. Alur kerja LSTM dapat dilihat pada Gambar 2.4



Gambar 2.4 Alur LSTM.

Sumber: Olah (2015)

LSTM menjawab masalah pada RNN dasar dengan menyediakan *cell state* sebagai sandingan dari *hidden state* dalam memproses memori per-sekuens data. Kombinasi dua *state* ini dapat menambah kapabilitas mengingat. Pada langkah t, *Cell state* C_t akan berinteraksi dengan data dan *hidden state* melalui tahapan empat gerbang. Gerbang- gerbang tersebut adalah:

a. Gerbang *forget* f_t menentukan informasi mana yang tidak lagi diperlukan di *cell state* yang lalu akan direduksi atau dibuang. Variabel pembelajaran W_f akan lakukan kalkulasi dengan *hidden state* h_{t-1} yang di gabungkan dengan data *input* x_t. b_f ditambahkan sebagai variable pembelajaran bias. Hasil akhir perhitungan diproses dengan fungsi sigmoid σ. Perhitungan dapat dilihat pada persamaan (2.1).

$$f_t = \sigma\left(W_f \,.\, [h_{t-1}, x_t] + b_f\right) \quad (2.1)$$

b. Gerbang *input* i_t menentukan informasi baru yang akan disimpan ke dalam *cell state* dengan perhitungan yang dapat dilihat pada persamaan (2.2) dan (2.3). W_i, b_i, W_C, dan b_C sebagai variabel pembelajaran.

$$i_t = \sigma(W_i \,.\, [h_{t-1}, x_t] + b_i) \quad (2.2)$$

$$\tilde{C}_t = (W_C \,.\, [h_{t-1}, x_t] + b_C) \quad (2.3)$$

c. Gerbang *update* menyimpanan informasi yang telah dipersiapkan sebelumnya ke dalam *cell state*. Perhitungan pada persamaan (2.4).

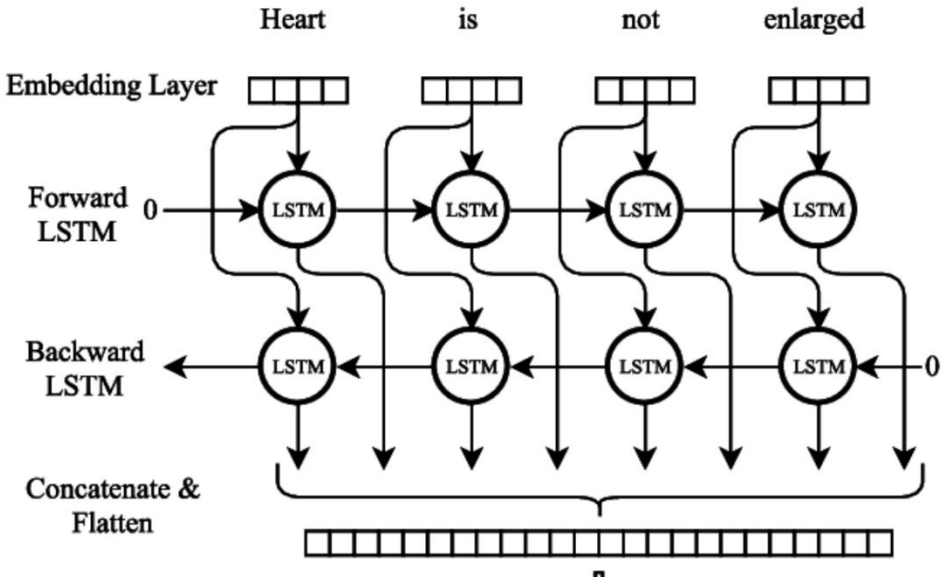
$$C_t = f_t \,.\, C_{t-1} + i_t \,.\, \tilde{C}_t \qquad (2.4)$$

d. Gerbang *output* o_t akan dimasukan bersama data *cell state* terbaru untuk dijadikan *hidden state* terbaru yang akan digunakan pada sekuens berikutnya. Perhitungan gerbang ini dapat dilihat pada persamaan (2.5) dan (2.6). W_o dan b_o adalah variabel pembelajaran.

$$o_t = \sigma(W_o \,.\, [h_{t-1}, x_t] + b_o) \qquad (2.5)$$

$$h_t = o_t \,.\, tanh(C_t) \qquad (2.6)$$

LSTM dasar masih memiliki kekurangan dalam kemampuan memahami konteks kalimat atau dokumen. diperlukan adanya penerapan metode agar dapat mengolah informasi sekuens secara maju dan mundur, *bidirectional* adalah salah satu metode yang dapat melakukannya. Secara sederhana, metode ini terdiri dari dua *layer* LSTM seperti terlihat pada Gambar 2.5.



Gambar 2.5 Alur *bidirectional* LSTM.

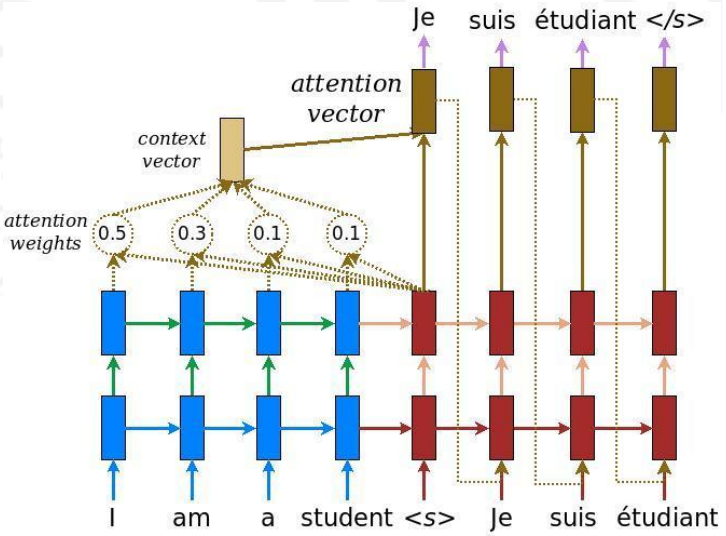
Sumber: Cornegruta et al. (2016)

Masing-masing layer akan diberikan tugas untuk membaca sekuens dari arah yang berbeda. *Layer* pertama akan mengolah *input* sekuens dari awal ke akhir dan *layer* kedua mengolah dari akhir ke awal. Pada setiap sekuens t, *hidden states* akan dihasilkan dengan menggabungkan *hidden states* LSTM maju h_t^f, dan *hidden states* LSTM mundur h_t^b, seperti pada persamaan (2.7).

$$h_t = [h_t^f, h_t^b]\,. \qquad (2.7)$$

2.5 *Attention Mechanism*

Attention mechanism merupakan metode untuk mempelajari tingkat atensi yang diperlukan per-bagian suatu data ke dalam bentuk vector representasi lalu selanjutnya diolah Kembali menjadi *context vector*. Metode ini pertama kali digunakan pada pengembangan *encoder decoder* untuk tugas *machine translation*. *attention mechanism* menjawab permasalahan *bottleneck* yang terjadi dari arsitektur *encoder decoder* (Bahdanau et al., 2014). Vektor representasi yang murni dihasilkan oleh *encoder* masih belum maksimal menyimpan keseluruhan fitur data sehingga *decoder* tidak dapat merekonstruksi keluaran dengan baik. *attention mechanism* memungkinkan model untuk memilih bagian mana dari data masukan yang diperlukan untuk tiap sekuens keluaran. Sehingga informasi yang diproses semakin tepat sasaran dan meningkatkan performa *decoder*. Alur kinerja seperti pada Gambar 2.6.



Gambar 2.6 Alur *attention mechanism.*

Sumber: Singh (2019)

Tahap dari *attention mechanism* dimulai dari *encoder* menghasilkan *hidden state* (*h*) untuk representasi setiap sekuens *input*, setelah itu, untuk memproses *hidden state* decoder pada langkah t maka diperlukan keluaran *decoder* satu langkah sebelumnya (s_t). Kalkulasi akan dilakukan relatif terhadap tiap langkah keluaran *decoder*. Perhitungan dimulai dengan kalkulasi nilai *attention weights* a^t, seperti terlihat pada persamaan (2.8).

$$a^t = softmax\left(v^T \tanh(W_h h_i + W_s s_t + b_{attn})\right) \quad (2.8)$$

W_h adalah variabel pembelajaran yang dikalkulasikan dengan *hidden states* encoder. W_s adalah variabel pembelajaran yang dikalkulasikan dengan *hidden states* decoder, b_{attn} adalah variabel bias pembelajaran jaringan. Berikutnya *context vector* (h_t^*) akan dihasilkan dengan kalkulasi, seperti terlihat pada persamaan (2.9).

$$h_t^* = \sum_i a_i^t h_i \quad (2.9)$$

Context vector yang telah dihasilkan akan digabungkan dengan *decoder hidden state* s_t dan diproses melalui dua *linear layer* untuk menghasilkan distribusi probabilitas *vocabulary* p_{vocab}, seperti terlihat pada persamaan (2.10).

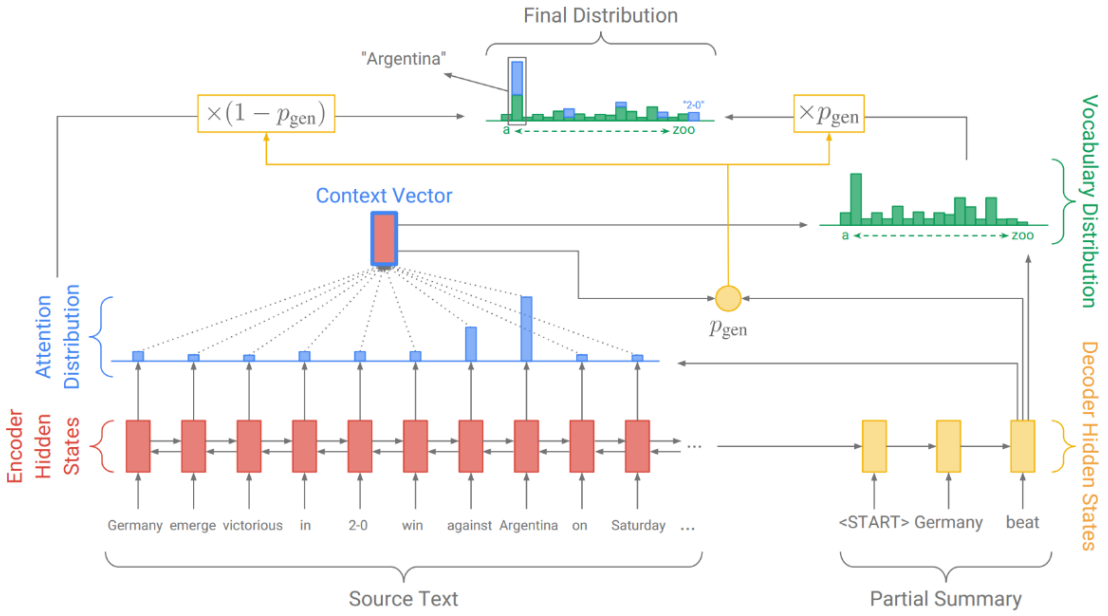
$$p_{vocab} = softmax(V'(V[s_t, h_t^*] + b) + b') \quad (2.10)$$

Nilai V', V, b, dan b' adalah variabel pembelajaran. p_{vocab} adalah probabilitas distribusi keluaran berdasarkan seluruh kata yang tersedia di *vocabulary*, yang akhirnya akan digunakan untuk menentukan distribusi akhir untuk kata yang akan diprediksi.

2.6 *Pointer-Generator*

Pointer-generator adalah strategi yang memberikan kemampuan terhadap model *attentional encoder decoder* untuk melakukan prediksi kata dengan memilih antara menggunakan distribusi probabilitas dari *attention weight input* atau dari *vocabulary* (See et al., 2017). Kemampuan ini mengatasi apabila terdapat kata OOV di dokumen *input*. Karakter atau kata seperti nama dan nominal akan sangat bervariasi dan sulit untuk semuanya dimasukkan ke dalam *vocabulary*. Seperti pada kalimat “klub sepak bola ABC menang 3-1

melawan klub DEF", nama klub sepakbola dan skor 3-1 adalah variabel yang sangat bervariasi namun vital untuk menghasilkan informasi yang akurat. Memanfaatkan kata atau karakter yang hanya ada di *vocabulary* akan dapat mengurangi bahkan merusak informasi-informasi vital tersebut. Gambaran jelas alur proses *pointer-generator* dapat dilihat di Gambar 2.7.



Gambar 2.7 Alur *pointer-generator* pada *attentional encoder decoder*.
Sumber: See et al. (2017)

Berdasarkan gambar di atas. Pada setiap iterasi keluaran, metode *pointer-generator* akan menghasilkan nilai probabilitas p_{gen} yang bertugas untuk menentukan apakah kalimat keluaran pada tahap tersebut mengambil dari distribusi probabilitas *vocabulary* atau *attention weights*. Pada tiap langkah t, p_{gen} dikalkulasi dengan menggunakan nilai *context vector* h_t^*, *decoder state* s_t, dan *decoder input* x_t seperti pada persamaan (2.11).

$$p_{gen} = \sigma\left(w_{h^*}^T h_t^* + w_s^T s_t + w_x^T x_t + b_{ptr}\right) \quad (2.11)$$

Nilai $w_{h^*}^T$, w_s^T, w_x^T, dan skalar b_{ptr} adalah variabel pembelajaran, yang diakhir akan diolah dalam fungsi sigmoid σ. p_{gen} akan berfungsi sebagai *soft switch* penentu antar menghasilkan prediksi menggunakan kata dari distribusi *vocabulary* p_{vocab}, atau memilih kata dari dokumen *input* berdasarkan distribusi probabilitas *attention weights* a_t. Pada setiap

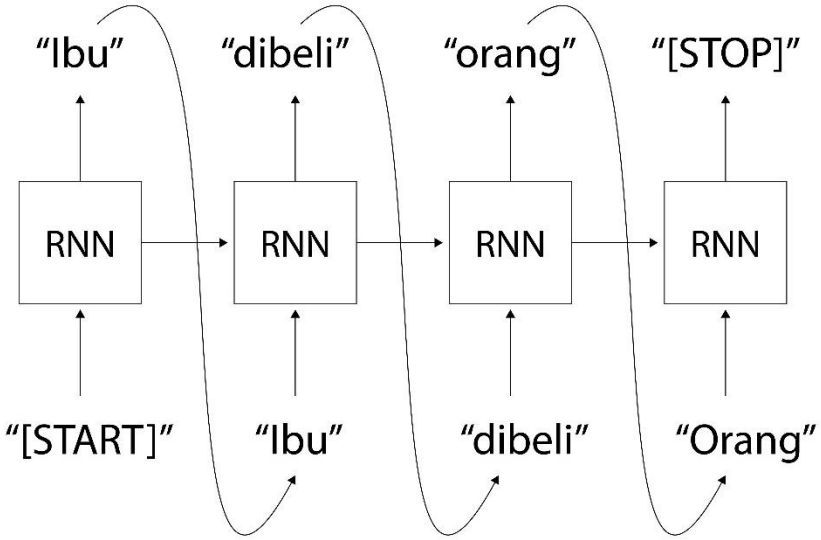
langkah t, kalimat OOV yang terdeteksi akan digabungkan sementara ke *vocabulary*, Lalu menggunakan p_{gen} yang telah dikalkulasi sebelumnya, distribusi final $p(w)$ akan didapatkan dengan perhitungan seperti pada persamaan (2.12).

$$p(w) = p_{gen}p_{vocab}(w) + (1 - p_{gen})\sum_{i:w_i=w} a_i^t \quad (2.12)$$

Dapat dilihat bahwa, apabila kata *w* adalah kata OOV, maka dapat dipastikan $p_{vocab}(w)$ adalah nol, sebaliknya jika w adalah dokumen *input*, maka $\sum_{i:w_i=w} a_i^t$akan bernilai nol. Kemampuan untuk menghasilkan kata OOV adalah kelebihan utama dari penggunaan metode *pointer-generator*.

2.7 *Teacher Forcing*

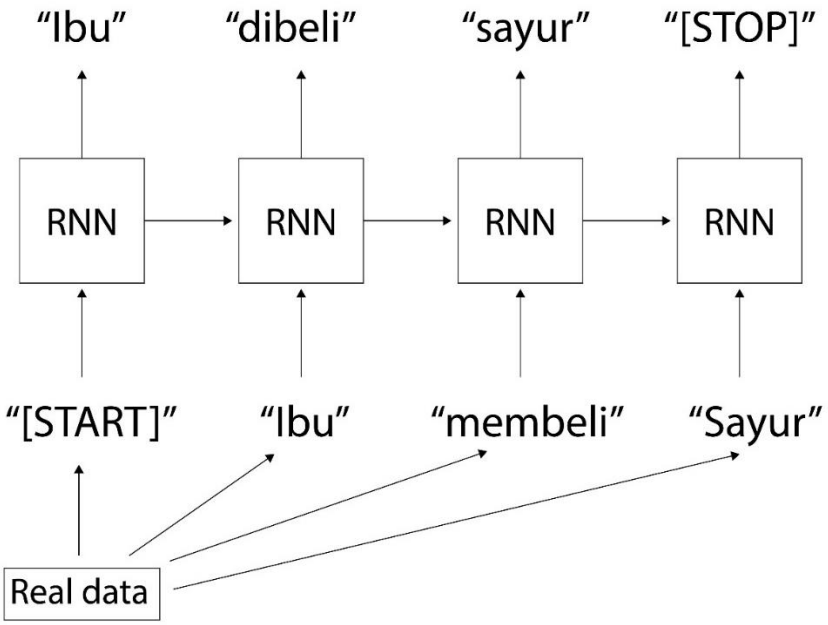
Teacher forcing merupakan strategi dalam pelatihan RNN yang menggunakan data asli sebagai *input*, dengan tujuan untuk mempercepat pembelajaran (Lamb et al., 2016). Alur *teacher forcing* dapat dilihat di Gambar 2.8.



Gambar 2.8 RNN tanpa *teacher forcing.*

Pada alur kerja RNN dasar input pada langkah t akan mengambil keluaran dari langkah sebelumnya. Kelemahan dari alur ini adalah apabila pada suatu langkah terdapat prediksi yang tidak sesuai, maka hal tersebut akan mempengaruhi prediksi-prediksi berikutnya. Fenomena

tersebut akan mengurangi akurasi keluaran RNN sekaligus memperlama waktu pelatihan. Solusi *teacher forcing* dapat dilihat pada Gambar 2.9.



Gambar 2.9 RNN yang menerapkan *teacher forcing*.

Dengan menggunakan data asli, proses pelatihan RNN tidak akan terpengaruhi oleh kesalahan prediksi RNN itu sendiri. Akibat dari hal tersebut durasi proses pelatihan akan dapat dikurangi.

2.8 AdaGrad

Adagrad (*Adaptive gradient algorithm*) merupakan salah satu metode *optimizer* pada *deep learning* yang dapat merubah nilai *learning rate* secara otomatis menyesuaikan kondisi saat pelatihan. Metode ini mengeliminasi kebutuhan untuk menentukan nilai *learning rate* secara manual. Kalkulasi dari metode ini dapat dilihat di persamaan (2.13).

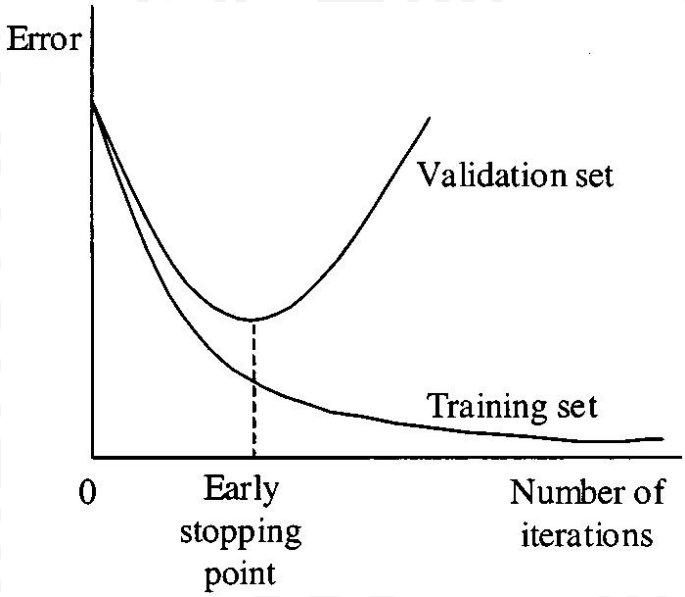
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\varepsilon \mathrm{I} + diag(G_t)}} \cdot g_t \qquad (2.13)$$

Di mana teta (θ) adalah variabel pembelajaran yang akan dicari nilai barunya, eta (η) adalah *learning rate* yang telah ditetapkan sebelum proses pelatihan, epsilon (ε) adalah nilai

kecil yang digunakan untuk menghindari pembagian dengan nilai nol, iota (I) adalah *identity matrix*, g_t adalah estimasi gradient pada langkah t.

2.9 *Early Stopping*

Early stopping merupakan metode yang memberi kemampuan model untuk menghentikan proses pelatihan secara otomatis. *Early stopping* berguna dalam mencegah kemungkinan terjadinya *overfitting* pada pelatihan model (Gençay & Qi, 2001).



Gambar 2.10 Penerapan *early stopping* pada grafik *loss*/*error*.

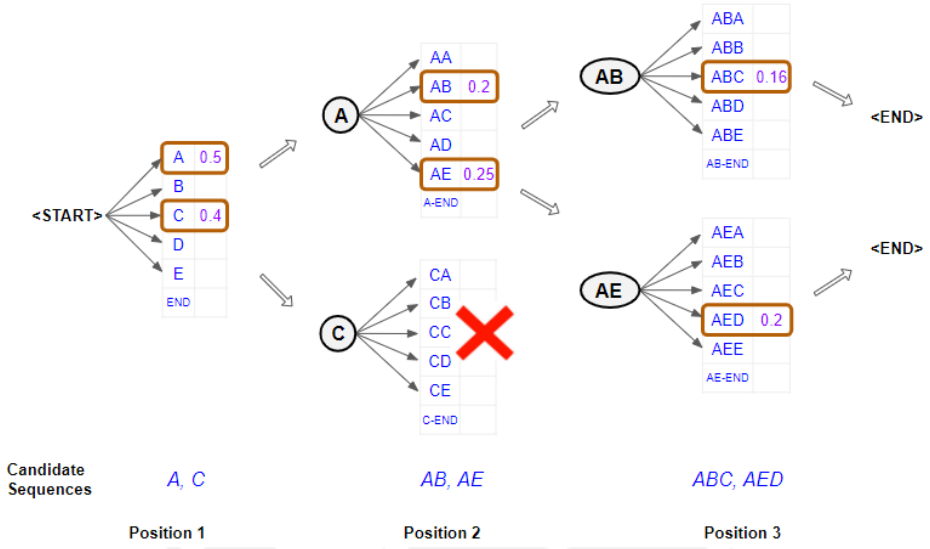
Sumber: Gençay & Qi (2001)

Gambar 2.10 memperlihatkan bahwa dengan menerapkan early stopping, model akan berhenti di titik di mana nilai matrik dari proses validasi paling minimal. Parameter utama dari metode *early stopping* yaitu, matrik monitor, *min_delta*, dan *patience*. metrik yang dimonitor dapat berupa *loss*, *validation_loss*, *accuraccy*, dan sebagainya. nilai *min_delta* adalah penentuan standar minimal nilai matrik yang dianggap memberi perkembangan berarti. *patience* adalah jumlah maksimal langkah pelatihan yang diteruskan walaupan tidak memenuhi standar minimal nilai *min_delta*.

2.10 *Beam Search*

Beam Search merupakan strategi yang berguna dalam mencari kombinasi beberapa hasil keluaran model yang memiliki nilai probabilitas paling besar (Freitag & Al-Onaizan, 2017).

Proses strategi ini adalah dengan menyimpan beberapa kandidat dengan nilai probabilitas tertinggi ditiap langkah prediksi. Gambaran alur *beam search* dapat dilihat pada Gambar 2.11.



Gambar 2.11 Alur kerja *beam search*.

Sumber: Doshi (2021)

Kriteria penghentian *search* diberikan seperti jumlah maksimal kalimat prediksi, atau keluarnya token khusus penanda berhenti, contohnya seperti token "[STOP]". Pada akhir proses prediksi setiap kalimat kombinasi akan dibandingkan total probabilitasnya untuk diambil yang tertinggi. Penggunaan metode ini memberi model ruang untuk dapat mengeksplorasi keluaran paling optimal dengan tidak terus menerus memilih satu kata dengan probabilitas tertinggi pada tiap keluaran probabilitas prediksi kata.

2.11 ROUGE

ROUGE merupakan metrik evaluasi yang membandingkan hasil keluaran teks oleh mesin dengan teks referensi buatan manusia. metrik evaluasi ini umum digunakan dalam pengembangan ATS. ROUGE memiliki banyak variasi namun dalam konteks penelitian ini hanya menggunakan dua macam yaitu ROUGE-N.

ROUGE-N menghitung jumlah *n-gram* yang serupa antara teks keluaran model dengan teks referensi. Yang dimaksud dengan *n-gram* adalah pengelompokan dengan panjang berdasarkan nilai yang ditetapkan. N pada ROUGE-N akan menjadi nilai dari panjang *gram* yang ditentukan. Tabel 2.2 memperlihatkan contoh ROUGE-N dengan menggunakan kalimat "budi pergi ke pasar bersama ibu".

Tabel 2.2 Contoh ROUGE-N

Macam ROUGE-N	Pembagian *gram*
Unigram (ROUGE-1)	[“budi”, “pergi”, “ke”, “pasar”, “bersama”, “ibu”]
Bigram (ROUGE-2)	[“budi pergi”, “pergi ke”, “ke pasar”, “pasar bersama”, “bersama ibu”]
Trigram (ROUGE-3)	[“budi pergi ke”, “pergi ke pasar”, “ke pasar bersama”, “pasar bersama ibu”]

Setelah nilai *n-gram* ditentukan, perhitungan nilai *recall* dapat dilakukan seperti pada persamaan (2.14).

$$recall = \frac{\text{jumlah } ngrams \text{ serupa antara model dan referensi}}{\text{jumlah total } ngrams \text{ pada teks referensi}} \quad (2.14)$$

Nilai *recall* berguna untuk mengetahui seberapa banyak informasi model ATS menyerap informasi dari dokumen *input*, tetapi disisi lain tidak memastikan apakah model tidak hanya mendorong banyak kata saja, tanpa memperhatikan relevansi kata tersebut. Untuk itu dibutuhkan nilai kedua yaitu *precision*, dengan perhitungan seperti pada persamaan (2.15).

$$precision = \frac{\text{jumlah } ngrams \text{ serupa antara model dan referensi}}{\text{jumlah total } ngrams \text{ pada model}} \quad (2.15)$$

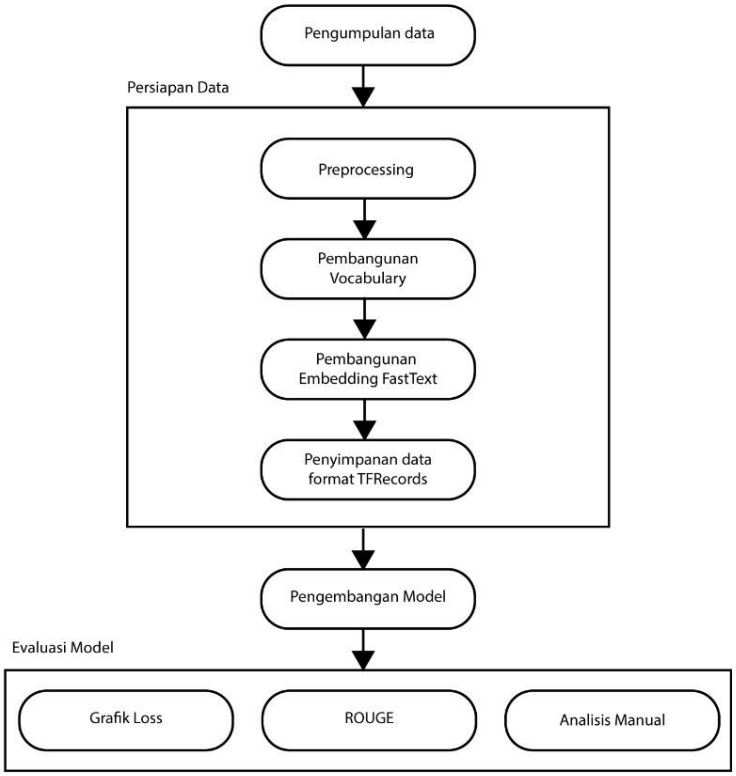
Nilai *precision* dan *recall* akan digunakan untuk menghasilkan nilai *F1-Score* dengan perhitungan seperti persamaan (2.16).

$$F1 - Score = \frac{\text{jumlah } ngrams \text{ serupa antara model dan referensi}}{\text{jumlah total } ngrams \text{ pada model}} \quad (2.16)$$

BAB III
METODOLOGI PENELITIAN

3.1 Langkah Penyelesaian

Bagian ini memperlihatkan gambaran secara umum tahap-tahap pengerjaan penelitian dari awal hingga akhir. Tahap-tahap keseluruhan dalam bentuk flowchart dapat dilihat pada Gambar 3.1.



Gambar 3.1 Bagan langkah penyelesaian.

3.2 Uraian Metodologi

Pada bagian ini, akan dijabarkan tiap tahapan dari langkah penyelesaian sesuai pada Gambar 3.1.

3.2.1 Pengumpulan Data

Data diambil dari situs liputan6 menggunakan metode *web scraping*. Investigasi struktur html dilakukan untuk mengembangkan program penarikan data. Periode data yang diambil

adalah dari awal tahun 2000 dan seterusnya. Pemilihan periode tersebut dikarenakan kualitas data ringkasan yang lebih baik (Koto et al., 2020).

3.2.2 Persiapan Data

Data yang telah dikumpulkan akan melalui beberapa proses persiapan untuk kepentingan pelatihan dan evaluasi model.

A. *Preprocessing*

Proses ini bertujuan untuk membersihkan data dari berbagai komponen atau fitur yang tidak diinginkan sehingga memaksimalkan pembelajaran mesin. Berdasarkan target keluaran alur *preprocessing* disesuaikan oleh peneliti, berikut beberapa tahap *preprocessing* yang akan dilakukan pada penelitian ini.

Tahap pertama adalah eliminasi kata atau karakter, beberapa macam karakter atau kata dianggap tidak memberikan sumbangsih yang berarti untuk penarikan informasi. Pada penelitian ini ditentukan bahwa karakter atau kata yang dieliminiasi adalah *tag* html, spasi berlebih, *‘‘__’’*, dan "--“. Contoh dapat dilihat pada Tabel 3.1

Tabel 3.1 Contoh hasil eliminasi

Contoh sebelum	Contoh sesudah
Kafe ABC merupakan salah satu tempat tongkrongan anak muda di kota C yang sedang <i>happenning.<i> 	Kafe ABC merupakan salah satu tempat tongkrongan anak muda di kota C yang sedang happenning.

Berikutnya adalah penggantian kata atau karakter, beberapa karakter akan diganti agar lebih seragam. Karakter yang diganti adalah “\n” menjadi spasi, '–' menjadi '-' dan '"' menjadi "". Contoh dari proses ini dapat dilihat pada Tabel 3.2

Tabel 3.2 Contoh hasil penggantian

Contoh sebelum	Contoh sesudah
"saya optimis dengan industri batik tahun ini" kata ibu wiyono batik semakin digemari oleh penduduk mancanegara, seperti:\n-inggris, \n-australia, \njepang.	saya optimis dengan industri batik tahun ini kata ibu wiyono karena semakin digemari oleh penduduk mancanegara, seperti: inggris, australia, jepang.

Langkah selanjutnya adalah pemisahan tanda baca. proses akan memungkinkan mesin untuk menjadikan tanda baca untuk diolah menjadi fitur yang bermakna oleh model. Contoh dari proses ini dapat dilihat pada Tabel 3.3.

Tabel 3.3 Contoh hasil pemisahan tanda baca

Contoh sebelum	Contoh sesudah
Sejak pandemi jam buka berbagai tempat perbelanjaan seperti: Amart, Bmart, dan Cmart tutup lebih cepat.	Sejak pandemi jam buka berbagai tempat perbelanjaan seperti : Amart , Bmart , dan Cmart tutup lebih cepat .

Proses berikutnya adalah *case folding*. proses ini bertujuan untuk mengolah semua huruf besar di teks menjadi huruf kecil. Tujuan dari proses adalah untuk mengurangi variasi kata sehingga meringankan kinerja pelatihan model. Contoh proses ini dapat dilihat pada Tabel 3.4.

Tabel 3.4 Contoh hasil proses *case folding*

Contoh sebelum	Contoh sesudah
Presiden Joko melakukan kunjungan ke Kota Palembang pada bulan Desember ini.	presiden joko melakukan kunjungan ke kota palembang pada bulan desember ini.

Langkah terakhir adalah penambahan token pada teks yang akan digunakan. Beberapa token tersebut akan berguna dalam untuk mengkomunikasikan sifat-sifat tertentu terhadap model. Macam token yang digunakan adalah, “[UNK]” untuk menandakan kata yang tidak terdapat di dalam *vocabulary*, token “[PAD]” sebagai penanda bagian teks yang merupakan *padding*, “[START]” untuk penanda dimulainya ringkasan pada *decoder input*, dan “[STOP]” sebagai penanda berhentinya ringkasan pada *decoder target*. Contoh dari proses penambahan token dapat dilihat pada Tabel 3.5.

Tabel 3.5 Contoh penambahan token

Contoh sebelum	Contoh sesudah
Presiden Joko melakukan kunjungan ke Kota Palembang pada bulan Desember ini.	[START] presiden [UNK] melakukan kunjungan ke kota palembang pada bulan desember ini . [STOP] [PAD] [PAD] [PAD]

B. Pembangunan *Vocabulary*

Vocabulary adalah penyimpanan kumpulan kata dengan indeks masing-masing yang akan digunakan sebagai panduan untuk mengubah kata pada data masukan menjadi angka indeks masing-masing. Hal ini dilakukan karena model komputer membutuhkan data teks untuk dirubah ke dalam suatu bentuk numerik sebelum dapat diproses.

C. Pembangunan *Embedding* FastText

Pembangunan *word embedding* FastText akan dilakukan dengan memanfaatkan data teks yang telah dilatih. Setelah pelatihan ini, tiap kata dalam vocabulary akan memiliki vektor representasinya masing-masing, yang lalu dapat digunakan untuk masukan ke pembelajaran model.

D. Penyimpanan Data ke Dalam Format TFRecords

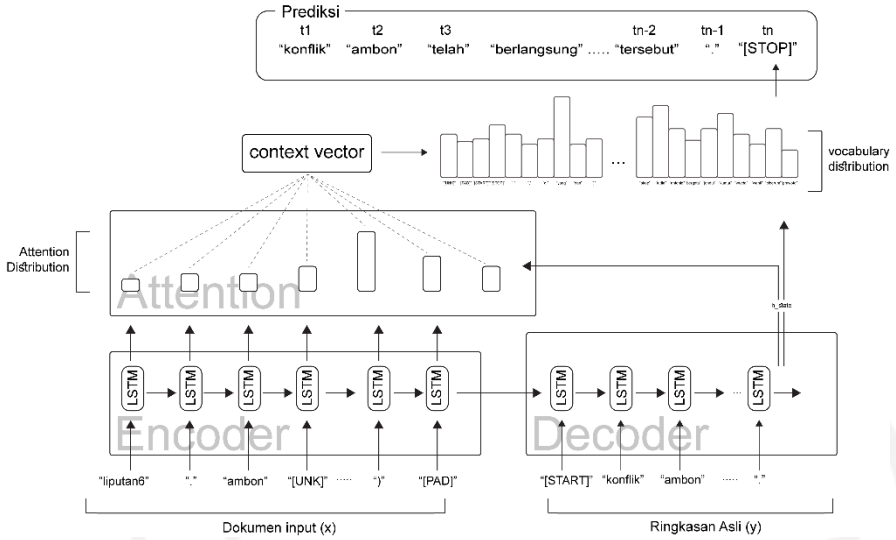
Tensorflow sebagai *library* utama yang dipakai penulis menyediakan format data khusus yang dapat mempercepat proses transfer data ke dalam model tensorflow. Dengan format ini Tensorflow dapat melakukan proses *cache* dan paralelisasi proses transfer data. Kelebihan lain dari penggunaan format ini adalah kompresi data yang cukup besar sehingga data menjadi lebih portable, cocok untuk semisalnya pelatihan di *cloud*.

3.2.3 Pengembangan Model

Pengembangan model dilakukan dengan pertama-tama menentukan arsitektur dari varian model yang akan dipakai. Untuk penelitian ini telah ditentukan empat varian model yang akan dikembangkan dan dievaluasi. Kesamaan dari ke empat model ini adalah penggunaan *teacher forcing* pada RNN LSTM di bagian *decoder*. Ke empat varian model tersebut adalah.

a. Model 1

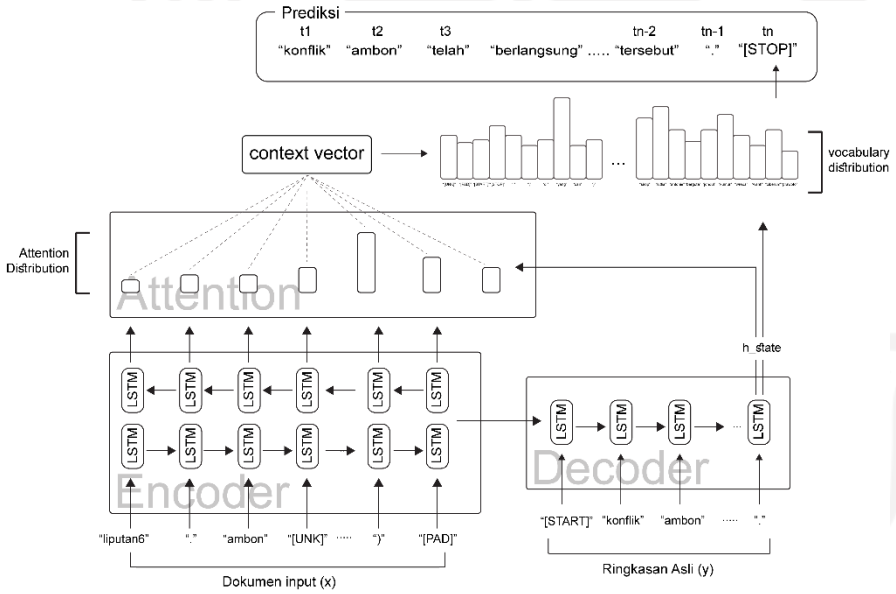
   Model ini adalah model dasar yang hanya terdiri dari arsitektur *attentional encoder decoder* tanpa adanya tambahan metode lebih. Bagan alur model 1 seperti yang terlihat pada Gambar 3.2.



Gambar 3.2 Bagan alur model 1.

b. Model 2 (*Bidirectional* LSTM)

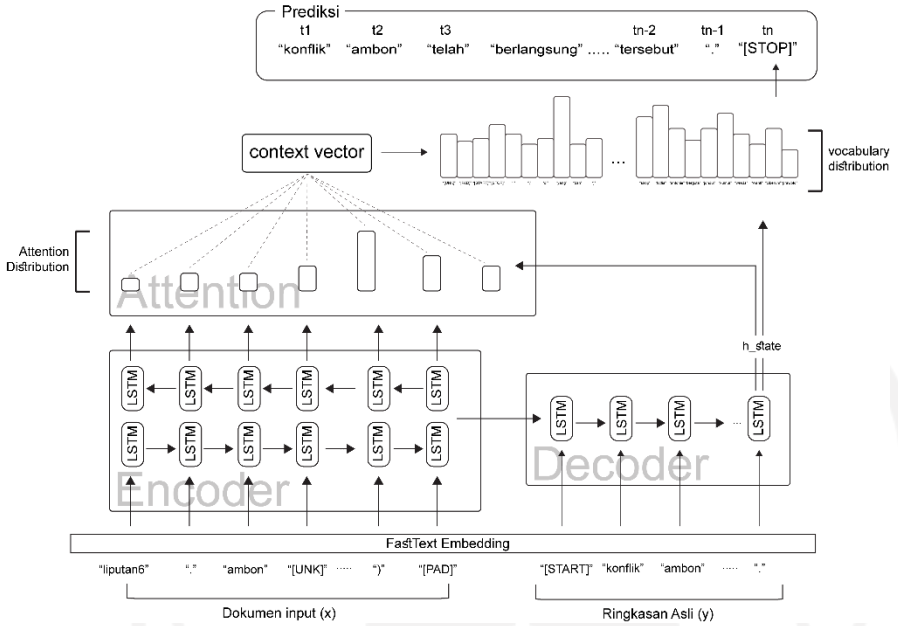
Model ini merupakan perkembangan dari model dasar dengan tambahan penggunaan *bidirectional* LSTM pada *encoder*. Bagan alur model 2 pada Gambar 3.3.



Gambar 3.3 Bagan alur model 2.

c. Model 3 (FastText + *Bidirectional* LSTM)

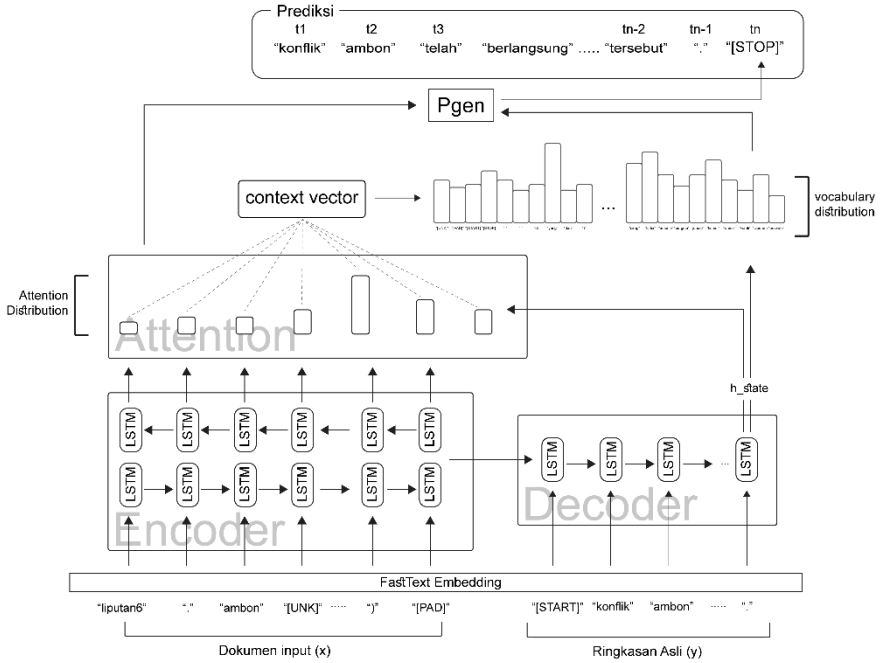
Pada model ini *pre-trained* model FastText akan ditambahkan pada basis model 2. Bagan alur model 3 pada Gambar 3.4.



Gambar 3.4 Bagan alur model 3.

d. Model 4 (FastText + *Bidirectional* LSTM + *Pointer-Generator*)

Pada model ini metode *pointer-generator* digunakan. Komponen lainnya menyamai model 3. Bagan alur model 4 pada Gambar 3.5.



Gambar 3.5 Bagan alur model 4.

Setelah varian model ditetapkan maka akan dimulai pemograman setiap model. Untuk komponen pemrograman dalam pengembangan ini adalah Bahasa python versi 3.5 dan *library* Tensorflow. Kartu grafis yang digunakan untuk melatih model adalah 1080Ti dan V100. *Hyperparameter* setiap model berupa jumlah unit LSTM 256 untuk *unidirectional* dan 128 untuk *bidirectional* (sehingga total *layer* maju dan mundur adalah 256). Jumlah *attention* unit diputuskan sebanyak 256. Nilai *batch size* berupa 16. Jenis optimizer AdaGrad dengan *learning rate* awal 0.15, *initial accumulator* 0.1, dan *maximum gradient clipping* 2. Fungsi perhitungan *Loss* menggunakan *sparse categorical entrophy*. Strategi *early stopping* juga diterapkan dengan nilai *min delta* nol dan *patience* 10. Penerapan *early stopping* tersebut akan memberi kesempatan model untuk terus melakukan proses latihan sebanyak 10 kali secara beruntun tidak menghasilkan progres *loss* melewati batas nilai nol.

3.2.4 Evaluasi Model

Evaluasi model akan dilakukan berdasarkan tiga aspek, yaitu.

A. Aktifitas Grafik *Loss*

pada aspek ini akan diperhatikan laju penurunan *loss* setiap *epoch* selama pelatihan berlangsung. Macam *loss* yang akan diperhatikan adalah *loss* dari sesi training dan *loss* dari sesi validasi yang disebut sebagai *validation loss*. Informasi yang dapat ditarik dari evaluasi aspek ini adalah seberapa cepat proses pengurangan loss tiap model. monitor *validation loss* berguna untuk melihat status model berhubungan dengan *overfitting*.

B. ROUGE

Pada evaluasi ini akan digunakan metrik khusus untuk hasil ringkasan ROUGE. Macam metrik ROUGE yang akan digunakan adalah ROUGE-1, ROUGE-2 dan ROUGE-L. dengan evaluasi ini akan didapatkan informasi mengenai perbandingan kemiripan antara ringkasan keluaran model dan ringkasan buatan manual.

C. Inspeksi Hasil keluaran

Evaluasi ini berfokus pada analisis kualitas keluaran varian model yang tidak dapat diukur secara kualitatif. penulis akan membandingkan hasil keluaran teks ringkasan untuk setiap varian model hasil dari pemprosesan data masukan yang sama.

BAB IV
HASIL DAN PEMBAHASAN

4.1 Hasil Pengumpulan Data

Data diambil dari situs liputan6.com menggunakan metode *web scraper*. *Library* yang digunakan untuk pengambilan data adalah beautiful-soup, library khusus pengambilan informasi dari data html suatu situs. Program yang digunakan dapat dilihat pada Gambar 4.1.

```
def get_id(url):
     return url.split('/')[-2]

def get_summary(text):
     target = ''
    for line in text.split('\n'):
        if 'window.kmklabs.channel =' in line:
            target = line
            break
    temp=target.split('window.kmklabs.article = ')[1]
    temp=temp.split(';')[0]
    data = json.loads(temp)
    return data['shortDescription']

def extract_data(text):
    soup = BeautifulSoup(text)
    title = soup.findAll('title')[0].getText().replace(' - News Liputan6.com', '')
    date  =  soup.findAll('time',  {'class':  'read-page--header--author__datetime
updated'})[0].getText()
    article = []
    contents = soup.findAll('div', {'class': 'article-content-body__item-content'})
    for content in contents:
        article.append(content.getText())
    summary = get_summary(text)
    return title, date, article, summary

def write_file(id, url, title, date, content, summary, target_path):
    json_dict = {}
    json_dict['id']=id
    json_dict['url']=url
    json_dict['title']=title
    json_dict['date']=date
    json_dict['content']='\n'.join(content)
    json_dict['summary']=summary

    with open(f"{target_path}/{id}.json", 'w') as json_file:
        json.dump(json_dict, json_file)

def proceed_one(url, path):
    response = requests.get(url)
    url = response.url
    id = get_id(url)
```

```
    title, date, article, summary = extract_data(response.text)
    write_file(id, url, title, date, article, summary, path)
def proceed(urls, path):
    for url in urls:
        try:
            proceed_one(url, path)
        except:
            print('Failed to proceed ', url, '. Potentially the news has been deleted
 from Liputan6.')
def thread_func(urls, path, num_thread=1):
    os.makedirs(path,exist_ok=True)
    threads = []
    for i in range(num_thread):
        cur_idx = int(i*len(urls)/num_thread)
        cur_urls = urls[cur_idx:cur_idx+int(len(urls)/num_thread)]
        t = threading.Thread(target=proceed, args=(cur_urls, path,))
        threads.append(t)
        t.start()
```

Gambar 4.1 Kode program *web scrapping* situs Liputan6.com.

Dari hasil *web scraping*, didapatkan 193.879 artikel berita untuk data pelatihan, 10.000 masing-masing untuk data tes dan validasi. Data mentah disimpan dalam format JSON dengan satu artikel ditempatkan di *file* masing-masing, hal itu dilakukan agar memberi kemudahan inspeksi data ke depannya, karena akses data tertentu lebih cepat. Tampilan salah satu *file* JSON data mentah pada Gambar 4.2.


```
{"id": "100002", "url": "https://www.liputan6.com/news/read/100002/jepang-minta-
 maaf", "title": "Jepang Minta Maaf", "date": "22 Apr 2005, 18:43 WIB", "content":
 "Liputan6.com, Jakarta: Perdana Menteri Jepang Junichiro Koizumi meminta maaf atas
 kekejaman tentara Jepang pada masa Perang Dunia II di Asia. Permohonan maaf secara
 formal itu Koizumi utarakan di depan pemimpin negara-negara Asia dan Afrika dalam
 Konferensi Tingkat Tinggi Asia-Afrika di Jakarta Convention Center, Jumat
 (22/4).\r\nKoizumi mengatakan, pada masa silam Jepang ambisius untuk berkuasa.
 Ternyata, ambisi itu justru menimbulkan kerusakan dan penderitaan luar biasa bagi
 penduduk di sejumlah negara, terutama di Asia. \"Untuk itu Jepang minta maaf,\"
 kata Koizumi.\r\nSelain meminta maaf, Koizumi berharap KAA dapat membantu
 mempererat persahabatan antara Jepang dan negara di Asia serta Afrika. Jepang,
 tambah Koizumi, siap memberikan sumbangan dan bantuan bagi negara-negara Asia dan
 Afrika yang membutuhkan.\r\nPernyataan Koizumi membuat heran para pengamat. Menurut
 mereka, ini kejadian langka. Mereka menduga, pernyataan Koizumi terkait dengan
 makin panasnya hubungan Jepang dan Cina akhir-akhir ini. Jepang ingin meredakan
 ketegangan melalui negara-negara peserta KAA.\r\nHubungan Cina-Jepang memang terus
 memburuk sejak Jepang menyetujui penerbitan buku pelajaran sejarah nasional Jepang,
 awal April 2005. Apalagi buku itu sama sekali tak menceritakan kekejaman tentara
 Jepang selama masa PD II di Asia dan Pasifik. Istilah Negeri Samurai saat itu
 adalah Dai Toa Senso atau Perang Asia Timur Raya [baca: Cina Menolak Meminta Maaf
 kepada Jepang].(ICH/Tim Liputan 6 SCTV)", "summary": "Pada masa silam Jepang terlalu
 ambisius untuk berkuasa. Ternyata, ambisi itu justru menimbulkan kerusakan dan
 penderitaan luar biasa bagi penduduk di sejumlah negara, terutama di Asia."}
```


Gambar 4.2 Data mentah hasil *web scraping*.

4.2 Persiapan

Persiapan data berperan sangat vital dalam menjamin agar model dapat berlatih dengan baik. berikut penjabaran hasil persiapan data.

4.2.1 *Preprocessing*

Berdasarkan keperluan pengolahan maka dibuat program untuk pengerjaannya, sebagai berikut.

A. Eliminasi Karakter atau Kata

Memanfaatkan *regex* (*regular expression*) yang merupakan deretan karakter spesial untuk mendefinisikan pola dalam pencarian teks, dihasilkan program yang dapat mendeteksi fitur teks yang ingin dihilangkan. Kata yang terdeteksi akan dihapus. Kode program dapat dilihat pada Gambar 4.3.

```
Def cleanhtml(raw_html):
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, '', raw_html)
    return cleantext

def eliminate(word):
    word = word.replace('__', '')
    word = word.replace('--', '')
    return word
```

Gambar 4.3 Kode program eliminasi karakter atau kata.

B. Penggantian karakter atau kata

Masih menggunakan *regex*, program akan mendeteksi karakter atau kata sesuai kriteria dan menggantinya dengan keluaran yang telah ditentukan. Kode program dapat dilihat pada Gambar 4.4.

```
def replace(word):
    word = word.replace('\n', ' ')
    word = word.replace('-', '-')
    word = word.replace('&quot', '"')
    return word
```

Gambar 4.4 Kode program penggantian karakter atau kata.

C. Pemisahan tanda baca

Program ini membaca keseluruhan kata dan menghasilkan kumpulan tanda baca yang dideteksinya. Kode program dapat dilihat pada Gambar 4.5.

```
tokens=re.findall(r"[\w'\%\&\\/\=\+\*$£]+|[\[\]().,!?\:;\"\"\”]", word)
```

Gambar 4.5 Kode pemisahan tanda baca.

D. *Case Folding*

Program ini mendeteksi huruf kapital yang ada baik pada karakter atau kata, setelah itu karakter atau kata akan di reproduksi tanpa mengandung huruf kapital. Kode sederhana dapat dilihat pada Gambar 4.6.

```
word = word.lower()
```

Gambar 4.6 Kode program proses *case folding*.

E. Penambahan Token

Token yang digunakan terdiri dari 4 macam yaitu “[UNK]”, “[PAD]”, “[START]”, dan “[STOP]”. Setiap token akan diberikan nilai indeks masing-masing di dalam *vocabulary*. Token “[UNK]” akan ditambahkan sebagai pengganti karakter atau kata yang tidak termasuk di dalam *vocabulary* atau OOV. Program ini tidak dilaksanakan untuk pemrosesan target *decoder* model *pointer-generator* karena pada model tersebut kata OOV masih akan digunakan. program penambahan token “[UNK]” seperti pada Gambar 4.7.

```
def word_to_id(self, word):
    if word not in self.word2id:
      return self.word2id[Vocab.UNKNOWN_TOKEN]
    return self.word2id[word]
```

Gambar 4.7 Kode program proses penambahan token ”[UNK]”.

Demi kebutuhan pelatihan dengan metode *teacher forcing*. Token “[START]” akan ditambahkan pada setiap awalan *decoder input* dan token “[STOP]” untuk tiap *decoder target*. Program seperti pada Gambar 4.8

```
def abstract_to_sents(abstract):
    cur = 0
    sents = []
    while True:
```

```
      try:
        start_p = abstract.index(Vocab.SENTENCE_START, cur)
        end_p = abstract.index(Vocab.SENTENCE_END, start_p + 1)
        cur = end_p + len(Vocab.SENTENCE_END)
        sents.append(abstract[start_p+len(Vocab.SENTENCE_START):end_p])
      except ValueError as e: # no more sentences
        return sents
```

Gambar 4.8 Kode program proses penambahan token ”[START]” dan ”[STOP]”.

Token “[PAD]” dengan indeks 1 akan ditambahkan terhadap data *encoder input* untuk menyamakan panjang teks tiap *batch* pada data *encoder input*. Berikut program yang melakukan *padding* pada setiap data *input*, dapat dilihat pada “enc_input”, “dec_input” dan “target” diberikan *padding* bernilai 1 yang merupakan indeks “[PAD]”.

```
dataset.padded_batch(batch_size, padded_shapes=({"enc_len":[],
                                  "enc_input" : [None],
                                 "enc_input_extend_vocab"  : [None],
                                 "article_oovs" : [None],
                                 "dec_input" : [max_dec_len],
                                 "target" : [max_dec_len],
                                 "dec_len" : [],
                                 "article" : [],
                                 "abstract" : []}),
                               padding_values={"enc_len":-1,
                                               "enc_input" : 1,
                                               "enc_input_extend_vocab"  : 1,
                                               "article_oovs" : b'',
                                               "dec_input" : 1,
                                               "target" : 1,
                                               "dec_len" : -1,
                                               "article" : b"",
                                               "abstract" : b"",},
                                                drop_remainder=True)
```

Gambar 4.9 Kode program proses *padding*.

Setelah beberapa tahap pemrosesan teks di atas dilalui maka akan dihasilkan keluaran untuk *encoder input* pada Tabel 4.1.

Tabel 4.1 Contoh hasil akhir pemrosesan data *encoder input*

Data mentah	Data terolah
"Liputan6.com, Denpasar: Berbeda dengan sebagian besar warga di dunia, masyarakat Bali mempunyai tradisi unik untuk menyambut pergantian tahun. Tradisi tersebut adalah	[["liputan6", ".", "com", ",", "denpasar", ":", "berbeda", "dengan", "sebagian", "besar", "warga", "di", "dunia", ",", "masyarakat", "bali", "mempunyai", "tradisi", "unik", "untuk",

menggelar upacara adat melepas matahari terakhir 2001 dengan pesta rakyat. Selanjutnya, mereka menyambut terbitnya matahari awal 2002. Dalam perayaan kali ini, sejak Senin (31/12) petang, ribuan masyarakat dari berbagai daerah di Bali berkumpul di Lapangan Puputan Badung, Denpasar, untuk menutup 2001 dan menyambut tahun 2002.\r\nBerdasarkan pemantauan SCTV, festival musik bleganjur atau gamelan yang khusus digunakan untuk prosesi keagamaan mengawali upacara tersebut. Dalam festival musik ini, 21 sekehe atau kelompok turut memeriahkannya. Untuk melepas matahari terakhir 2001 dan menyambut matahari terbit pertama 2002 ini, setiap sekehe melakukan prosesi mengelilingi tugu atau patung catur muka atau empat muka yang menghadap ke segenap penjuru mata angin, yaitu timur, barat, utara, dan selatan. Bagi mereka, catur muka adalah lambang keseimbangan. Menurut seorang peserta, prosesi tersebut dimaksudkan untuk memohon keseimbangan dari Sang Hyang Widi Wasa agar bumi dan jagad raya ini selalu seimbang dan damai.\r\nAcara menyambut 2002 ini berlangsung tertib dan dilanjutkan doa bersama dari seluruh umat beragama, yaitu Hindu, Budha, Protestan, Katolik, dan Islam. Doa bersama itu dilangsungkan tepat pukul 00.00 WITA. Sementara itu, malam menjelang Tahun Baru di Bali seperti tahun-tahun sebelumnya secara alami terpusat di Pantai Kuta, Denpasar. Akibatnya, sejak petang, ruas jalan menuju Pantai Kuta macet total.(ANS/Yudah Prakoso dan Iwan Gunawan)”	"menyambut", "pergantian", "tahun", ".", "tradisi", "tersebut", "adalah", "menggelar", "upacara", "adat", "melepas", "matahari", "terakhir", "[UNK]", "dengan", "pesta", "rakyat", ".", "selanjutnya", ",", "mereka", "menyambut", "terbitnya", "matahari", "awal", "[UNK]", ".", "dalam", "perayaan", "kali", "ini", ",", "sejak", "senin", "(", "[UNK]", ")", "petang", ",", "ribuan", "masyarakat", "dari", "berbagai", "daerah", "di", "bali", "berkumpul", "di", "lapangan", "puputan", "badung", ",", "denpasar", ",", "untuk", "menutup", "[UNK]", "dan", "menyambut", "tahun", "[UNK]", ".", "berdasarkan", "pemantauan", "sctv", ",", "festival", "musik", "bleganjur", "atau", "gamelan", "yang", "khusus", "digunakan", "untuk", "prosesi", "keagamaan", "mengawali", "upacara", "tersebut", ".", "dalam", "festival", "musik", "ini", ",", "[UNK]", "sekehe", "atau", "kelompok", "turut", "memeriahkannya", ".", "untuk", "melepas", "matahari", "terakhir", "[UNK]", "dan", "menyambut", "matahari", "terbit", "pertama", "[UNK]", "ini", ",", "setiap", "sekehe", "melakukan", "prosesi", "mengelilingi", "tugu", "atau", "patung", "catur", "muka", "atau", "empat", "muka", "yang", "menghadap", "ke", "segenap", "penjuru", "mata", "angin", ",", "yaitu", "timur", ",", "barat", ",", "utara", ",", "dan", "selatan", ".", "bagi", "mereka", ",", "catur", "muka", "adalah", "lambang", "keseimbangan", ".", "menurut", "seorang", "peserta", ",", "prosesi", "tersebut", "dimaksudkan", "untuk", "memohon", "keseimbangan", "dari", "sang", "[UNK]", "widi", "wasa", "agar", "bumi", "dan", "jagad", "raya", "ini", "selalu", "seimbang", "dan",

	"damai", ".", "acara", "menyambut", "[UNK]", "ini", "berlangsung", "tertib", "dan", "dilanjutkan", "doa", "bersama", "dari", "seluruh", "umat", "beragama", ",", "yaitu", "hindu", ",", "budha", ",", "protestan", ",", "katolik", ",", "dan", "islam", ".", "doa", "bersama", "itu", "dilangsungkan", "tepat", "pukul", "[UNK]", ".", "[UNK]", "wita", ".", "sementara", "itu", ",", "malam", "menjelang", "tahun", "baru", "di", "bali", "seperti", "tahun-tahun", "sebelumnya", "secara", "alami", "terpusat", "di", "pantai", "kuta", ",", "denpasar", ".", "akibatnya", ",", "sejak", "petang", ",", "ruas", "jalan", "menuju", "pantai", "kuta", "macet", "total", ".", "(", "[UNK]", "prakoso", "dan", "iwan", "gunawan", ")", "."]]

Berikut adalah hasil akhir pemrosesan data untuk *decoder input* dan *decoder target* pada Tabel 4.2.

Tabel 4.2 Contoh hasil akhir pemrosesan data *decoder input* dan *decoder target*

Data mentah	Data terolah	
	Decoder input	*Decoder target*
"Masyarakat Bali merayakan Tahun Baru dengan tradisi unik dan pesta rakyat, yaitu melepas matahari 2001 serta menyambut matahari 2002. Pantai Kuta disesaki ribuan wisatawan."	[["[START]", "masyarakat", "bali", "merayakan", "tahun", "baru", "dengan", "tradisi", "unik", "dan", "pesta", "rakyat", ",", "yaitu", "melepas", "matahari", "[UNK]", "serta", "menyambut", "matahari", "[UNK]", ".", "pantai", "kuta", "disesaki", "ribuan", "wisatawan", "."]]	[["masyarakat", "bali", "merayakan", "tahun", "baru", "dengan", "tradisi", "unik", "dan", "pesta", "rakyat", ",", "yaitu", "melepas", "matahari", "2001", "serta", "menyambut", "matahari", "2002", ".", "pantai", "kuta", "disesaki", "ribuan", "wisatawan", ".", "[STOP]",]]

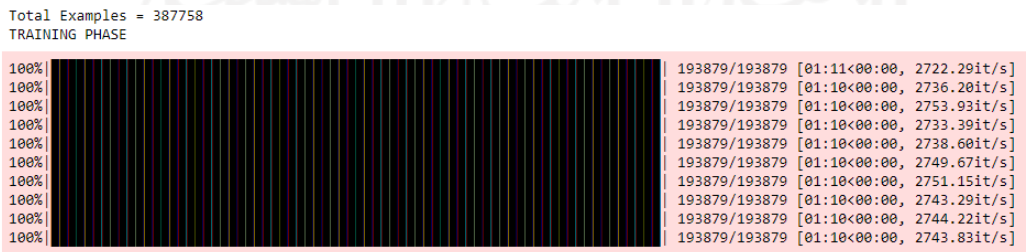
Perlu dicatat pada *decoder target* khusus untuk penggunaan *pointer-generator* tidak mengeliminasi kata OOV, namun pada model tanpa *pointer*-generator, OOV akan tetap diganti dengan token “[UNK]”.

4.2.2 Pembangunan *Vocabulary*

Vocabulary akan menjadi acuan untuk perubahan kata menjadi indeks saat pelatihan dan sebaliknya saat melakukan inferensi. Khusus untuk tahap ini data akan disaring Kembali agar didapatkan hanya kata-kata yang dianggap penulis lebih bermakna. Kata-kata atau karakter yang akan disaring adalah kata dengan variable yang terlalu besar. Umumnya berhubungan dengan angka seperti “Rp600”, “80%”, “Ke-2” dan sejenisnya. Pemrosesan *vocabulary* akan menggunakan bantuan library *text vectorization* dari Keras. Frekuensi kata minimum ditentukan sebanyak 10 kemunculan. *Vocabulary* akan diurutkan mulai dari kata yang paling sering muncul. Batasan jumlah kata dalam *vocabulary* penelitian ini adalah sebanyak 50.000. keputusan jumlah *vocabulary* yang tidak lebih diambil guna memperjelas perbandingan dengan penggunaan metode *pointer-generator*.

4.2.3 Pembangunan FastText

Model *pre-trained* FastText Bahasa indonesia didapatkan dari situs fasttext.cc (Grave et al., 2018). Inspeksi dari *vocabulary* bawaan model *pre-trained* memperlihatkan bahwa masih ada duplikasi kata akibat tidak dilakukannya *case folding* saat pelatihan. Selain itu banyak juga kata yang tidak ditemukan di dalam *vocabulary* bawaan. Untuk itu diputuskan untuk melatih lebih lanjut model *pre-trained* tersebut dengan menggunakan data yang dimiliki.



Gambar 4.10 Progres pelatihan lebih lanjut *pre-trained* FastText.

Dapat dilihat pada Gambar 4.10 Progres pelatihan lebih lanjut *pre-trained* FastText pelatihan dilakukan sebanyak 10 iterasi untuk 193879 dokumen artikel dan ringkasan yang

merupakan data pelatihan. Data validasi dan tes tidak digunakan dalam proses ini, untuk memaksimalkan simulasi data asing saat model ATS melakukan proses validasi dan tes.

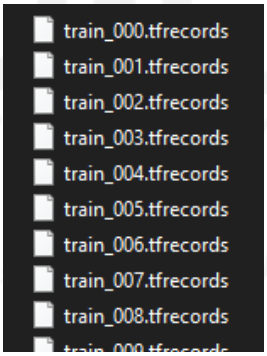
4.2.4 Penyimpanan Data ke Format TFRecords

Data dibagi menjadi 1000 artikel dan ringkasan tiap berkas. Didapatkan data latihan sebanyak 193000 *file*, data validasi 10000 *file* dan data tes 10000 *file*. Gambar Gambar 4.11 Kode program pemroses data ke dalam format TFRecords.memperlihatkan kode program proses pengolah format dan pembagian data sesuai *chunk size*.

```
def chunk_file(set_name, chunk_size):
    in_file = data_path+'%s.tfrecords' % set_name
    print(in_file)
    dataset = tf.data.TFRecordDataset(in_file, compression_type = 'GZIP')
    dataset = dataset.batch(chunk_size, drop_remainder = True)
    chunk_idx = 0
    for batch in dataset:
        batch_ds = tf.data.Dataset.from_tensor_slices(batch)
        chunk_fname  =  os.path.join(chunks_dir,  set_name+'/%s_%03d.tfrecords'  %
(set_name, chunk_idx))
        writer = tf.data.experimental.TFRecordWriter(chunk_fname, compression_type
= "GZIP")
        start_time = time.time()
        writer.write(batch_ds)
        print("Time needed: ", time.time() - start_time, "s", "\t", chunk_fname)

        chunk_idx += 1
```

Gambar 4.11 Kode program pemroses data ke dalam format TFRecords.



Gambar 4.12 Data dalam format TFRecords.

4.2.5 Analisis Data

Analisis data diperlukan untu menentukan jumlah maksimal *input* pada *encoder* dan jumlah tetap ringkasan hasil dari *decoder*. Tahap analisis dilakukan dengan mencari nilai persentil panjang dokumen, dengan hasil penjabaran persentil seperti pada Gambar 4.13.

```
persentil 100, artikel
0 percentile value is 35
10 percentile value is 119
20 percentile value is 140
30 percentile value is 158
40 percentile value is 176
50 percentile value is 197
60 percentile value is 222
70 percentile value is 255
80 percentile value is 307
90 percentile value is 391
100 percentile value is 7379

persentil 100, ringkasan
0 percentile value is 12
10 percentile value is 23
20 percentile value is 26
30 percentile value is 28
40 percentile value is 29
50 percentile value is 30
60 percentile value is 32
70 percentile value is 33
80 percentile value is 35
90 percentile value is 39
100 percentile value is 102

persentil 90, artikel
90 percentile value is 391
91 percentile value is 403
92 percentile value is 415
93 percentile value is 429
94 percentile value is 446
95 percentile value is 466
96 percentile value is 494
97 percentile value is 534
98 percentile value is 599
99 percentile value is 744
100 percentile value is 7379

persentil 90, ringkasan
90 percentile value is 39
91 percentile value is 39
92 percentile value is 40
93 percentile value is 41
94 percentile value is 41
95 percentile value is 43
96 percentile value is 44
97 percentile value is 47
98 percentile value is 51
99 percentile value is 54
100 percentile value is 102
```

Gambar 4.13 Hasil keluaran persentil data.

4.3 Pengembangan Model

Pada bagian ini, akan diperlihatkan kode program yang dikembangkan untuk model ATS serta hasil evaluasinnya.

4.3.1 Program Model

Model akan deprogram terdiri dari beberapa bagian yang memiliki fleksibilitas untuk menyesuaikan keperluan tiap varian model. Eksekusi program akan berfokus pada satu modul yang menampung pilihan tiap *hyperparameter* dan metode yang diinginkan. Keseluruhan model akan otomatis menyesuaikan. program ini akan memanfaatkan metode *subclasssing* tensorflow dalam pengembangan program. Berikut tiap kelas untuk *layer* pada model.

a. *Layer Encoder*

Program kelas pada Gambar 4.14 berfungsi sebagai eksekusi bagian *encoder*.

```
class Encoder(tf.keras.layers.Layer):
  def    __init__(self,    vocab_size,    embedding_dim,    enc_units,    batch_sz,
bidirect=False, embedding_path=None):
    super(Encoder, self).__init__()
    self.batch_sz = batch_sz
    self.enc_units = enc_units

    if embedding_path is not None:
      print("\nInitialize Preembedding in encoder")
      with open(embedding_path, 'rb') as f:
        embedding_initial = tf.keras.initializers.Constant(np.load(f))
      self.embedding   =   tf.keras.layers.Embedding(vocab_size+4,   embedding_dim,
embeddings_initializer=embedding_initial,                          trainable=False,
name='encoder_fasttext_embedding')
    else:
      print("\nInitialize Embedding in encoder")
      self.embedding    =    tf.keras.layers.Embedding(vocab_size,    embedding_dim,
name='encoder_embedding')

    self.LSTM = tf.keras.layers.LSTM(self.enc_units,
                                     return_sequences=True,
                                     return_state=True,
                                     recurrent_initializer='glorot_uniform')
    self.bidirect = bidirect
    if bidirect:
      print("\ninitialize Bidirectional LSTM on encoder")
      self.LSTM                =                tf.keras.layers.Bidirectional(self.LSTM,
name='encoder_bidirectional_LSTM')
    else :
      print("\nusing unidirectional on encoder")

  def initialize_hidden_state(self):
    if self.bidirect:
      return [tf.zeros((self.batch_sz, self.enc_units)) for i in range(4)]
    else:
      return [tf.zeros((self.batch_sz, self.enc_units)) for i in range(2)]

  def call(self, x):
    x = self.embedding(x)
    output= self.LSTM(x, initial_state = self.initialize_hidden_state())
```

```
    return output
```

Gambar 4.14 Kode program *encoder layer*.

a. Layer Decoder

Program kelas pada Gambar 4.15 berfungsi sebagai eksekusi bagian *decoder*. Kelas ini dapat dikostumisasi sesuai pemakaian FastText dan *bidirectional*.

```
class Decoder(tf.keras.layers.Layer):
  def __init__(self, vocab_size, embedding_dim, dec_units, batch_sz, bidirect,
embedding_path=None):
    super(Decoder, self).__init__()
    self.batch_sz = batch_sz
    self.dec_units = dec_units

    if embedding_path is not None:
      print("\nInitialize Preembedding in decoder")
      with open(embedding_path, 'rb') as f:
        embedding_initial = tf.keras.initializers.Constant(np.load(f))
      self.embedding = tf.keras.layers.Embedding(vocab_size+4, embedding_dim,
embeddings_initializer=embedding_initial, trainable=False,
name='decoder_fasttext_embedding')
    else:
      print("\nInitialize embedding in decoder")
      self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim,
name='decoder_embedding')

    self.LSTM = tf.keras.layers.LSTM(self.dec_units, return_sequences=True,
return_state=True,
                                     recurrent_initializer='glorot_uniform',
name='decoder_LSTM')
    self.fc = tf.keras.layers.Dense(vocab_size,
activation=tf.keras.activations.softmax, name='fully_connected_softmax')

def call(self, x, hidden, enc_output, context_vector):
    x = self.embedding(x)
    x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)

    output, state_h, state_c = self.LSTM(x, initial_state=hidden)
    return x, out, state_h, state_c
```

Gambar 4.15 Kode program *decoder layer*.

a. Layer Attention

Program kelas pada Gambar 4.16 ini berfungsi sebagai eksekusi dari metode *attention*.

```
class BahdanauAttention(tf.keras.layers.Layer):
  def __init__(self, units):
    super(BahdanauAttention, self).__init__()
    self.W1 = tf.keras.layers.Dense(units, name='W1')
    self.W2 = tf.keras.layers.Dense(units, name='W2')
```

```
    self.V = tf.keras.layers.Dense(1, name='V')

  def call(self, query, values, attention_weights_sum):

    hidden_with_time_axis = tf.expand_dims(query, 1)

    score = self.V(tf.nn.tanh(self.W1(values) + self.W2(hidden_with_time_axis)))
    attention_weights = tf.nn.softmax(score, axis=1)
    context_vector = attention_weights * values
    context_vector = tf.reduce_sum(context_vector, axis=1)

    return context_vector, attention_weights
```

Gambar 4.16 Kode program *attention layer*.

a. Layer Pointer

Program kelas pada Gambar 4.17 berfungsi sebagai eksekusi metode *Pointer*.

```
class Pointer(tf.keras.layers.Layer):

  def __init__(self):
    super(Pointer, self).__init__()
    self.w_s_reduce = tf.keras.layers.Dense(1, name='w_s_reduce')
    self.w_i_reduce = tf.keras.layers.Dense(1, name='w_i_reduce')
    self.w_c_reduce = tf.keras.layers.Dense(1, name='w_c_reduce')

  def call(self, context_vector, state, dec_inp):
    return
tf.nn.sigmoid(self.w_s_reduce(state)+self.w_c_reduce(context_vector)+self.w_i_red
uce(dec_inp))
```

Gambar 4.17 Kode program *pointer layer*.

4.4 Evaluasi Model

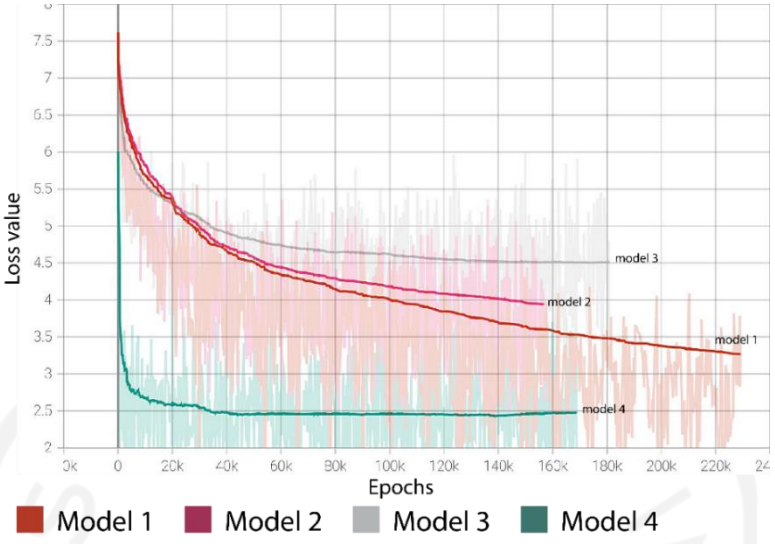
Pada evaluasi model grafik *loss* yang telah dibuat pada tahap pelatihan, sedangkan untuk ROUGE dan analisis manual akan dikembangkan program untuk eksekusinya.

4.4.1 Aktifitas Grafik Loss

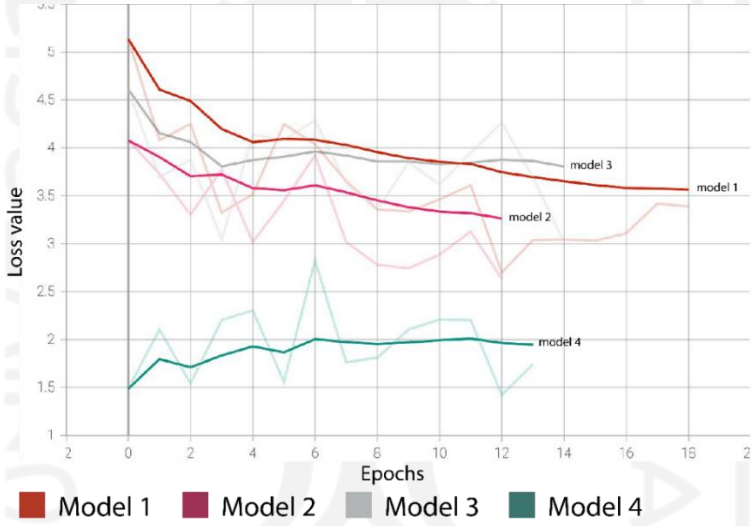
Dari proses pengembangan model didapatkan grafik *loss* selama pelatihan. Terdapat dua macam *loss*, yaitu loss pada tahap pelatihan (*loss*) dan pada saat validasi (*validation loss*). Grafik dihasilkan oleh *library* tensorboard.

a. *Batch loss* dan *epoch loss* seluruh model

Gambar 4.18 dan Gambar 4.19 merupakan data nilai *loss* ke empat model selama pelatihan dilihat dari dua lingkup iterasi yaitu setiap *batch* dan setiap *epoch*.



Gambar 4.18 Grafik *batch loss* semua model.

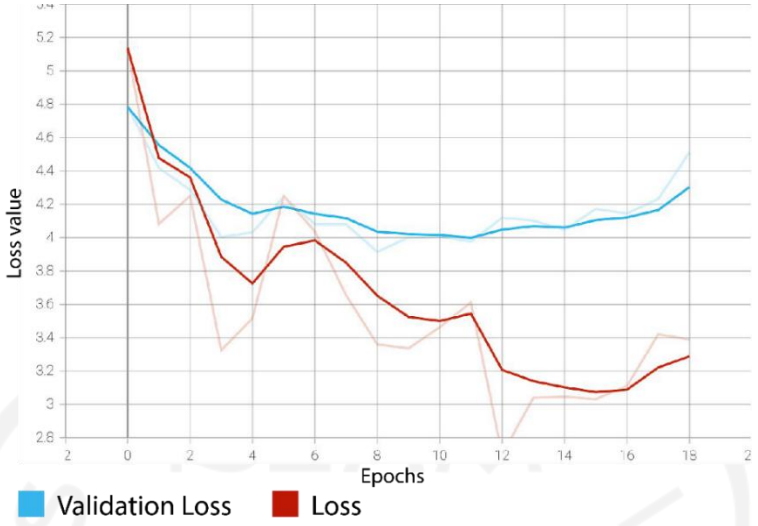


Gambar 4.19 Grafik *epoch loss* semua model.

Pada Gambar 4.18 menunjukkan bahwa konvergen tercepat terjadi pada model 4, namun setelah iterasi ke-400.000 model 4 langsung mengalami stagnansi di sekitar nilai *loss* 2,5. Keseluruhan model belum dapat mencapai nilai loss di bawah 1 hingga akhir pelatihan masing-masing.

b. Model 1

Gambar 4.20 adalah grafik *loss* dari proses pelatihan model 1.

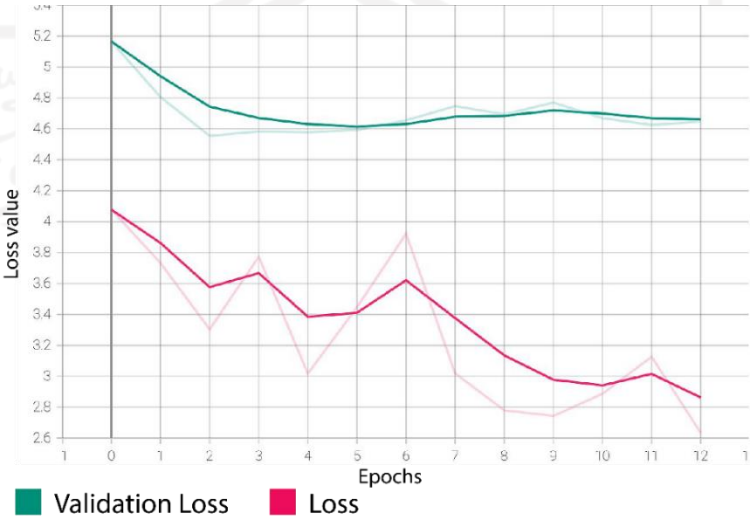


Gambar 4.20 Grafik *epoch loss* model 1.

Grafik *loss* ini menunjukkan proses latihan dihentikan oleh *early stopping* pada epoch ke-18, dengan nilai *patience* sebesar 10 menandakan bahwa nilai *validation loss* terbaik ada pada *epoch* kedelapan. Pergerakan nilai *loss* dimulai sekitar 5 sampai 5,2 yang cukup stabil hingga terjadi lonjakan pada *epoch* keempat, namun pada *epoch* keenam nilai *loss* kembali turun. Nilai *loss* mengalami sedikit kenaikan di antara loss 10 dan 12, juga 16 dan 18.

c. Model 2 (*bidirectional*)

Gambar 4.21 adalah grafik *loss* dan *val loss* dari proses pelatihan model 2.

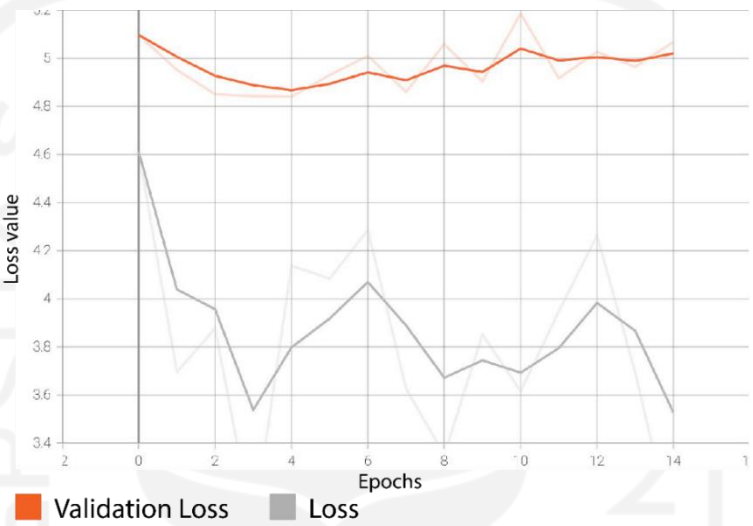


Gambar 4.21 Grafik *epoch loss* model 2.

Model 2 berhenti melakukan pelatihan pada *epoch* ke-12, yang menandakan perkembangan nilai *val loss* terbaik ada di *epoch* 2 sangat awal. Pelatihan 10 *epoch* berikutnya belum bisa memberi progres yang sesuai standar.

d. Model 3 (*bidirectional*, FastText)

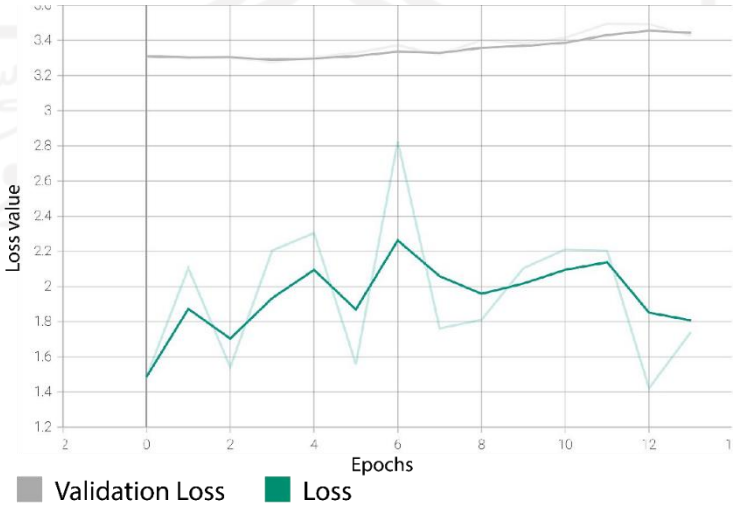
Gambar 4.22 adalah grafik *loss* dan *val loss* dari proses pelatihan model 3.



Gambar 4.22 Grafik *epoch loss* model 3.

e. Model 4 (*bidirectional*, FastText, *pointer-generator*)

Gambar 4.23 adalah *loss* dan *val loss* dari proses pelatihan model 4.



Gambar 4.23 Grafik *epoch loss* model 4.

Gambar 4.23 memperlihatkan bahwa pergerakan *loss* dan *validation loss* mengalami stagnansi. Laju proses pelatihan dihentikan oleh *early stopping* pada *epoch* ke-13 yang menandakan bahwa titik optimal berada *val loss* berada di *epoch* ketiga, sehingga variabel pembelajaran akhir diambil pada *epoch* tersebut.

Secara keseluruhan dapat dilihat bahwa tiap model mengalami stagnansi nilai *loss* di titik tertentu sehingga tidak dapat mencapai *loss* yang lebih optimal. Kemungkinan penyebabnya dapat berasal dari penerapan *optimizer* yang belum optimal sehingga model terjebak di *local minima*.

4.4.2 Evaluasi ROUGE

Perhitungan nilairouge akan dilakukan menggunakan *library* python rouge 1.0.1. data yang digunakan adalah 600 data untuk tiap model. Hasil perhitungan *f-precision* 3 varian ROUGE untuk tiap model dapat dilihat pada Tabel 4.3.

Tabel 4.3 Hasil evaluasi ROUGE terhadap ke empat model

Model	Rouge-1	Rouge-2	Rouge-L
Model 1	0.0982760824	0.0076891341	0.0907636793
Model 2: bidirectional	**0.1546519774**	**0.0181428786**	**0.148606571**
Model 3: bidirectional + FastText	0.1140184411	0.0088405416	0.1092212074
Model 4: bidirectional + FastText + *pointer-generator*	0.1008057043	0.0069128820	0.0967023352

Secara umum nilai ROUGE yang dihasilkan masih jauh jika dibandingkan dengan keluaran evaluasi pada penelitian referensi. Hal tersebut menandakan konfigurasi *hyperparameter* yang belum optimal. untuk nilai ROUGE-1, ROUGE-2, dan ROUGE-L terbesar didapatkan oleh model 2, meskipun pada grafik loss didapatkan model 2 tidak memiliki progres nilai loss terbaik. Hal tersebut menunjukkan bahwa tingkat progres nilai *loss* tidak berbanding lurus dengan hasil evalusi metrik ROUGE.

4.4.3 Inspeksi Hasil Ringkasan

Inspeksi akan menggunakan satu dokumen *input* untuk dibandingkan hasil keluaran ringkasan tiap model dari dokumen tersebut. Perbandingan hasil keluaran ringkasan dapat dilihat pada Gambar 4.24.

```
article
   liputan6 . com , jakarta : ketua dpr akbar tandjung memimpin rapat
pimpinan di gedung dpr/mpr senayan , jakarta , rabu ( 16/5 ) siang . r
apat yang diikuti seluruh wakil ketua dpr membahas permintaan pemerint
ah agar menunda pelaksanaan sidang paripurna 30 mei mendatang soal mem
orandum dua . namun , sesuai mekanisme yang berlaku , hal itu masih ak
an dibahas di badan musyawarah ( bamus ) . akbar menjelaskan bahwa per
mintaan penundaan itu telah disampaikan lewat surat permohonan dari pe
merintah yang diterimanya selasa kemarin . pemerintah beralasan bahwa
penundaan itu berkenaan dengan konferensi tingkat tinggi g-15 pada 25-
31 mei mendatang yang diikuti 19 negara . permintaan yang sama juga di
sampaikan menteri luar negeri alwi shihab , selasa kemarin . menurut a
kbar , rapat pimpinan dpr belum memutuskan jawaban atas permohonan pem
erintah tersebut . kendati begitu , permohonan itu mendapat cukup dipe
rhatikan fraksi-fraksi . rencananya , kata dia , bamus dpr akan membah
as hal itu pada kamis besok . ( ans/yenny tanda putra dan zakaria ) .

best_hyp abstract model 1
rapat diikuti ketua akbar memimpin pimpinan akbar memimpin pimpinan a
kbar memimpin pimpinan akbar memimpin pimpinan akbar memimpin pimpina
n akbar memimpin pimpinan akbar memimpin pimpinan akbar memimpin pimp
inan akbar memimpin pimpinan akbar memimpin pimpinan akbar memimpin p
impinan akbar memimpin pimpinan

best_hyp abstract model 2
rapat paripurna dpr permintaan rapat paripurna dpr permintaan rapat p
aripurna dpr permintaan rapat paripurna dpr permintaan rapat paripurn
a dpr permintaan rapat paripurna dpr permintaan rapat paripurna dpr p
ermintaan rapat paripurna dpr permintaan rapat paripurna dpr perminta
an rapat paripurna dpr

best_hyp abstract model 3
dpr permintaan ketua akbar memimpin [UNK] ketua akbar memimpin [UNK]
ketua akbar memimpin [UNK] ketua akbar memimpin [UNK] ketua akbar mem
impin [UNK] ketua akbar memimpin [UNK] ketua akbar memimpin [UNK] ket
ua akbar memimpin [UNK] ketua akbar memimpin [UNK] ketua

best_hyp abstract model 4
dpr jawa tengah rapat diikuti wakil ketua akbar memimpin pimpinan akba
r memimpin pimpinan akbar memimpin pimpinan akbar memimpin pimpinan ak
bar memimpin pimpinan akbar memimpin pimpinan akbar memimpin pimpinan
akbar memimpin pimpinan akbar memimpin pimpinan akbar memimpin pimpina
n akbar memimpin
```

Gambar 4.24 Perbandingan hasil ringkasan ke empat model.

Terlihat bahwa tingkat kohesi keseluruhan hasil ringkasan model memiliki awalan ringkasan yang masih dapat dimengerti, tetapi setelah pada poin tertentu hanya menghasilkan repetisi tidak alami. Hal ini dapat dimaklumi mengingat performa *loss* setiap model belum ada yang bisa lebih kecil dari satu. Untuk kualitas informasi, dapat dilihat bahwa model 2 berhasil mengambil kata kunci relevan untuk inti dari artikel *input* yaitu mengenai permintaan seputar rapat paripurna. Hasil tersebut mencerminkan nilai ROUGE dari model 2 yang lebih baik daripada model lainnya. Model 1, model 3, dan model 4 sukses mengambil sebuah poin dari artikel *input* namun bukan merupakan inti dari artikel. Model 3 didapatkan menggunakan token "[UNK]" yang mengindikasikan bahwa kata yang diinginkan tidak terdapat pada *vocabulary*. Secara keseluruhan dapat dilihat bahwa varian model yang dikembangkan dalam penelitian ini belum memiliki kualitas yang sesuai untuk target menyamai kualitas buatan manusia.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan hasil penelitian pengembangan model ATS *attentional encoder decoder* berita bahasa indonesia yang menerapkan metode *bidirectional*, FastText dan *pointer-generator* dapat disimpulkan bahwa:

1. Penggunaan *pointer-generator* mempercepat progres penurunan nilai *loss*.
2. Nilai evaluasi ROUGE tidak berbanding performa pada grafik *loss*.
3. Perbedaan jumlah data, *hyperparameter* dan penerapan strategi belum dapat menyamai hasil optimal seperti pada penelitian referensi.
4. Repetisi kalimat masih menjadi tantangan untuk mencapai hasil model yang kohesif.

Hasil akhir dari penelitian ini mendapatkan bahwa, khusus dalam ruang lingkup konfigurasi *hyperparameter* yang ditetapkan, model 2 yang hanya memanfaatkan *bidirectional* memiliki performa ringkasan terbaik. Hasil tersebut tidak menyamai hasil dari penelitian referensi di mana penggunaan metode lebih seperti *pointer-generator* dan *word embedding* FastText menunjukkan peningkatan performa. Namun, mengingat bahwa performa *loss* tiap model belum dapat mencapai kurang dari 1, masih diperlukan eksplorasi *hyperparameter* lebih lanjut untuk mendapatkan konfigurasi yang menghasilkan performa optimal untuk tiap model.

5.2 Saran

Saran yang diberikan penulis untuk tahap pengembangan selanjutnya adalah:

1. Eksplorasi konfigurasi *optimizer* AdaGrad dan juga bandingkan penggunaan *optimizer* lain seperti Adam, Adadelta atau RMSprop.
2. Eksplorasi penggunaan data, baik lebih sedikit atau lebih banyak. Pada penelitian See et al. (2017), data CNN/ Daily Mail memiliki data sebanyak 287,226, sekitar 80,000 data lebih banyak daripada data yang digunakan pada penelitian ini.
3. Eksplorasi parameter jumlah *batch* dan total *iterasi*. Pada penelitian See et al. (2017) melakukan 600.000 iterasi, dua kali lebih banyak dari penelitian ini.
4. Gunakan metode *coverage* seperti pada penelitian See et al. (2017) untuk mengatasi masalah repetisi.

DAFTAR PUSTAKA

A Visual Guide to FastText Word Embeddings. (n.d.). Retrieved January 10, 2022, from https://amitness.com/2020/06/fasttext-embeddings/

Adelia, R., Suyanto, S., & Wisesty, U. N. (2019). Indonesian abstractive text summarization using bidirectional gated recurrent unit. *Procedia Computer Science*, *157*, 581–588. https://doi.org/10.1016/j.procs.2019.09.017

Al-Sabahi, K., Zuping, Z., & Kang, Y. (2018). *Bidirectional Attentional Encoder-Decoder Model and Bidirectional Beam Search for Abstractive Summarization*.

Attention Mechanism. Attention mechanism was at first unreal… | by Pawandeep Singh | Medium. (n.d.). Retrieved January 11, 2022, from https://medium.com/@developer.pawandeep/attention-mechanism-89b407ec13fc

Bahdanau, D., Cho, K., & Bengio, Y. (2014). *Neural Machine Translation by Jointly Learning to Align and Translate*. http://arxiv.org/abs/1409.0473

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2016). *Enriching Word Vectors with Subword Information*. http://arxiv.org/abs/1607.04606

Cheng, Z., Sun, H., Takeuchi, M., & Katto, J. (2018). *Deep Convolutional AutoEncoder-based Lossy Image Compression*. http://arxiv.org/abs/1804.09535

Cho, K., van Merrienboer, B., Bahdanau, D., & Bengio, Y. (2014). *On the Properties of Neural Machine Translation: Encoder-Decoder Approaches*. http://arxiv.org/abs/1409.1259

Cornegruta, S., Bakewell, R., Withey, S., & Montana, G. (2016). *Modelling Radiological Language with Bidirectional Long Short-Term Memory Networks*. 17–27. https://doi.org/10.18653/v1/w16-6103

Dang, T. A., & Nguyen, T. T. T. (2019). Abstractive text summarization using pointer-generator networks with pre-trained word embedding. *PervasiveHealth: Pervasive Computing Technologies for Healthcare*, 473–478. https://doi.org/10.1145/3368926.3369728

Dehru, V., Tiwari, P. K., Aggarwal, G., Joshi, B., & Kartik, P. (2021). Text Summarization Techniques and Applications. *IOP Conference Series: Materials Science and Engineering*, *1099*(1), 012042. https://doi.org/10.1088/1757-899x/1099/1/012042

Foundations of NLP Explained Visually: Beam Search, How it Works | by Ketan Doshi | Towards Data Science. (n.d.). Retrieved January 10, 2022, from

https://towardsdatascience.com/foundations-of-nlp-explained-visually-beam-search-how-it-works-1586b9849a24

Freitag, M., & Al-Onaizan, Y. (2017). *Beam Search Strategies for Neural Machine Translation*. https://doi.org/10.18653/v1/W17-3207

Gençay, R., & Qi, M. (2001). Pricing and hedging derivative securities with neural networks: Bayesian regularization, early stopping, and bagging. *IEEE Transactions on Neural Networks*, *12*(4), 726–734. https://doi.org/10.1109/72.935086

Grave, E., Bojanowski, P., Gupta, P., Joulin, A., & Mikolov, T. (2018). *Learning Word Vectors for 157 Languages*. http://arxiv.org/abs/1802.06893

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Hu, D. (n.d.). *An Introductory Survey on Attention Mechanisms in NLP Problems*.

Jin, B., Tan, Y., Nettekoven, A., Chen, Y., Topcu, U., Yue, Y., & Vincentelli, A. S. (2019). *An Encoder-Decoder Based Approach for Anomaly Detection with Application in Additive Manufacturing*. http://arxiv.org/abs/1907.11778

Koto, F., Lau, J. H., & Baldwin, T. (2020). *Liputan6: A Large-scale Indonesian Dataset for Text Summarization*. http://arxiv.org/abs/2011.00679

Kramer, M. A. (n.d.). *Nonlinear Principal Component Analysis Using Autoassociative Neural Networks*.

Lamb, A., Goyal, A., Zhang, Y., Zhang, S., Courville, A., & Bengio, Y. (n.d.). *Professor Forcing: A New Algorithm for Training Recurrent Networks*.

Lee, W.-H., Ozger, M., Challita, U., & Sung, K. W. (2021). *Noise Learning Based Denoising Autoencoder*. http://arxiv.org/abs/2101.07937

Lin, C.-Y. (2004). *ROUGE: A Package for Automatic Evaluation of Summaries*.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). *Efficient Estimation of Word Representations in Vector Space*. http://arxiv.org/abs/1301.3781

NLP 101: Word2Vec — Skip-gram and CBOW | by Ria Kulshrestha | Towards Data Science. (n.d.). Retrieved January 10, 2022, from https://towardsdatascience.com/nlp-101-word2vec-skip-gram-and-cbow-93512ee24314

See, A., Liu, P. J., & Manning, C. D. (2017). *Get To The Point: Summarization with Pointer-Generator Networks*. http://arxiv.org/abs/1704.04368

Shi, T., Keneshloo, Y., Ramakrishnan, N., & Reddy, C. K. (2021). Neural Abstractive Text Summarization with Sequence-to-Sequence Models. *ACM/IMS Transactions on Data Science*, *2*(1), 1–37. https://doi.org/10.1145/3419106

Understanding LSTM Networks -- colah's blog. (n.d.). Retrieved January 10, 2022, from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Wang, Y., Yao, H., Zhao, S., & Zheng, Y. (2015). Dimensionality reduction strategy based on auto-encoder. *ACM International Conference Proceeding Series*, *2015-August*, 171–174. https://doi.org/10.1145/2808492.2808555

Widyassari, A. P., Rustad, S., Shidik, G. F., Noersasongko, E., Syukur, A., Affandy, A., & Setiadi, D. R. I. M. (2020). Review of automatic text summarization techniques & methods. In *Journal of King Saud University - Computer and Information Sciences*. King Saud bin Abdulaziz University. https://doi.org/10.1016/j.jksuci.2020.05.006

Yoko, K., Christanti, V., & Hendryli, J. (2018). SISTEM PERINGKAS OTOMATIS ABSTRAKTIF DENGAN MENGGUNAKAN RECURRENT NEURAL NETWORK. In *Computatio: Journal of Computer Science and Information Systems* (Vol. 2, Issue 1). www.detik.com

LAMPIRAN

a. Kode *web scraping* data.

```
import requests
import time
import json, os
import glob
from bs4 import BeautifulSoup
import threading

def get_id(url):
    return url.split('/')[-2]

def get_summary(text):
    target = ''
    for line in text.split('\n'):
        if 'window.kmklabs.channel =' in line:
            target = line
            break
    temp=target.split('window.kmklabs.article = ')[1]
    temp=temp.split(';')[0]
    data = json.loads(temp)
    return data['shortDescription']

def extract_data(text):
    soup = BeautifulSoup(text)
    title = soup.findAll('title')[0].getText().replace(' - News Liputan6.com', '')
    date = soup.findAll('time', {'class': 'read-page--header--author__datetime upd
ated'})[0].getText()
    article = []
    contents = soup.findAll('div', {'class': 'article-content-body__item-content'}
)
    for content in contents:
        article.append(content.getText())
    summary = get_summary(text)
    return title, date, article, summary

def write_file(id, url, title, date, content, summary, target_path):
    json_dict = {}
    json_dict['id']=id
    json_dict['url']=url
    json_dict['title']=title
    json_dict['date']=date
    json_dict['content']='\n'.join(content)
    json_dict['summary']=summary

    with open(f"{target_path}/{id}.json", 'w') as json_file:
        json.dump(json_dict, json_file)

def proceed_one(url, path):
    response = requests.get(url)
    url = response.url
    id = get_id(url)
    title, date, article, summary = extract_data(response.text)
    write_file(id, url, title, date, article, summary, path)

def proceed(urls, path):
    for url in urls:
        try:
            proceed_one(url, path)
        except:
            print('Failed to proceed ', url, '. Potentially the news has been dele
ted from Liputan6.')
```

```
def thread_func(urls, path, num_thread=1):
    os.makedirs(path,exist_ok=True)
    threads = []
    for i in range(num_thread):
        cur_idx = int(i*len(urls)/num_thread)
        cur_urls = urls[cur_idx:cur_idx+int(len(urls)/num_thread)]
        t = threading.Thread(target=proceed, args=(cur_urls, path,))
        threads.append(t)
        t.start()

THREAD = 10
urls = json.load(open('url.json'))

thread_func(urls['dev_urls'], 'data/raw/dev', THREAD)
thread_func(urls['test_urls'], 'data/raw/test', THREAD)
thread_func(urls['train_urls'], 'data/raw/train', THREAD)
```

b. Kode *preprocessing*

```
import json
import glob
import json
import re, os
from tqdm import tqdm

regex_bracket = r"\(([^)]+)\)"
punctuation = '.,!?\'\[]();"'
unusuals_id = set()

def cleanhtml(raw_html):
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, '', raw_html)
    return cleantext

def replace(word):
    word = word.replace('\n', ' ')
    word = word.replace('-', '-')
    word = word.replace('&quot', '"')
    return word

def eliminate(word):
    word = word.replace('__', '')
    word = word.replace('--', '')
    return word

def clean_article(article):
    article = cleanhtml(article)
    article_arr = []
    for word in article.split(' '):
        word = word.lower()
        word = replace(word)
        word = eliminate(word)
        word = word.strip()
        if len(word) > 0:
            tokens = re.findall(r"[\w'\%\&\-\/\=\+\*$£]+|[\[\]().,!?\:;\"\"\”]", w
ord)
            article_arr += tokens
    if not article_arr[-1] in '.!?':
        article_arr += '.'
    return [article_arr]
```

```
def process(PATH, DST, Data_use):
    os.makedirs(DST, exist_ok=True)
    files = glob.glob(PATH)
    for file in tqdm(files, desc = Data_use):
        data = json.load(open(file))
        clean_data = {}
        article = data['content']
        summary = data['summary']
        if(len(article.split())>30 and len(summary.split())>10):
            cleaned_article = clean_article(article)
            cleaned_summary = clean_article(summary)
            clean_data['id'] = data['id']
            clean_data['url'] = data['url']

            if len(cleaned_article[0]) < len(article.split()) or len(cleaned_summ
ary[0]) < len(summary.split()):
                unusuals_id.add(data['id'])

            clean_data['clean_article'] = cleaned_article
            clean_data['clean_summary'] = cleaned_summary
            with open(DST+str(clean_data['id'])+'.json', 'w') as json_file:
                json.dump(clean_data, json_file)

process('data/raw/train/*', 'data/clean/train/', 'Train')
process('data/raw/dev/*', 'data/clean/dev/', 'Val')
process('data/raw/test/*', 'data/clean/test/', 'Test')

print(unusuals_id)
```

c. Kode pembangunan *vocabulary*

```
import json, glob
from tqdm import tqdm
import collections
import time
import re

vocab_size = 50000
data_path = 'data/clean/train/'
save_path = 'vocab'

def get_string(sentences):
    all_sentence = []
    for sentence in sentences:
        all_sentence.append(' '.join(sentence))
    return ' '.join(all_sentence)

def create_vocab(data_path, save_path, vocab_size):
    files = glob.glob(data_path+'*')
    vocab_counter = collections.Counter()
    check = True
    for file in tqdm(files, desc = 'save words'):
        data = json.loads(open(file, 'r').readline())
        article = get_string(data['clean_article'])
        summary = get_string(data['clean_summary'])
        art_tokens = article.split(' ')
        abs_tokens = summary.split(' ')
        tokens = art_tokens + abs_tokens
        tokens = [t.strip() for t in tokens] # strip
        tokens = [t for t in tokens if t!=""] # remove empty
        vocab_counter.update(tokens)
```

```
    i = 0
    with open(save_path, 'w', encoding="utf-8") as writer:
        inputted = []
        for word, count in tqdm(vocab_counter.most_common(), desc = 'Writing voca
b file'):
            check_word = re.sub(r'[^a-z]+[-/]\d*|^\d+%*|^rp\d+|ke-\d+', "", word)
            if len(check_word) != 0:
                if word not in inputted:
                    writer.write(word +" "+ str(count) + '\n')
                    inputted += word
                    i += 1
                    if i == vocab_size:
                        break
    print ("Finished writing vocab file")
```

d. Kode pembangunan FastText

```
import json
import os
from gensim.models.fasttext import load_facebook_model
import tensorflow as tf
from tqdm import tqdm
import numpy as np

data_path = "data/clean/train/"
vocab_path = "data/tfrecords/vocab"
save_path = "embedding/retrained/"

fb_model = load_facebook_model('embedding/original/cc.id.300.bin')

embedding_dim = fb_model.vector_size
vocab_size = 50000

class MyIter():
    def __iter__(self):
        for file in tqdm(os.listdir(data_path)):
            with open(data_path+file, 'r') as data:
                data = json.load(data)
                article = data['clean_article']
                yield(article[0])
                summary = data['clean_summary']
                yield(summary[0])

exist = []
non_exist = []

with open(vocab_path, 'r', encoding = 'utf-8') as f:
  for line in f:
    pieces = line.split()
    if pieces[0] in fb_model.wv.key_to_index:
        exist.append(pieces[0])
    else:
        non_exist.append(pieces[0])

print("exist : "+str(len(exist)))
print("non-exist : "+str(len(non_exist)))
print(non_exist)

total_examples = fb_model.corpus_count
print("Total Examples = "+str(total_examples))
print("TRAINING PHASE")
fb_model.train(corpus_iterable= MyIter(), total_examples=total_examples, epochs=1
0)
```

e. Kode penyimpanan data ke format TFRecords

```
import os
import numpy as np
import tensorflow as tf
import json, glob
from tqdm import tqdm
import time

CHUNK_SIZE = 1000
main_path = 'data/clean/'
data_path = 'data/TFrecords/'
chunks_dir = data_path + 'chunked_files/'

def chunk_file(set_name, chunk_size):
    in_file = data_path+'%s.tfrecords' % set_name
    print(in_file)
    dataset = tf.data.TFRecordDataset(in_file, compression_type = 'GZIP')
    dataset = dataset.batch(chunk_size, drop_remainder = True)
    chunk_idx = 0
    for batch in dataset:
        batch_ds = tf.data.Dataset.from_tensor_slices(batch)
        chunk_fname = os.path.join(chunks_dir, set_name+'/%s_%03d.tfrecords' % (s
et_name, chunk_idx))
        writer = tf.data.experimental.TFRecordWriter(chunk_fname, compression_typ
e = "GZIP")
        start_time = time.time()
        writer.write(batch_ds)
        print("Time needed: ", time.time() - start_time, "s", "\t", chunk_fname)

        chunk_idx += 1


def chunk_all(chunks_dir, chunk_size):
    # Make a dir to hold the chunks
    if not os.path.isdir(chunks_dir):
        os.mkdir(chunks_dir)
    # Chunk the data
    for set_name in ['val', 'test', 'train']:
        if not os.path.isdir(chunks_dir+'/'+set_name):
            os.mkdir(chunks_dir+'/'+set_name)
        print ("Splitting %s data into chunks..." % set_name)
        chunk_file(set_name, chunk_size)
    print ("Saved chunked data in %s" % chunks_dir)


def get_string(sentences):
    all_sentence = []
    for sentence in sentences:
        all_sentence.append(' '.join(sentence))
    return ' '.join(all_sentence)
```

```
def write_tfrecords(in_folder, out_file, data_name):
    checkdata = True
    files = glob.glob(in_folder)
    counter = 0

    files = glob.glob(in_folder, recursive = True)
    tfoptions = tf.io.TFRecordOptions(compression_type = "GZIP")

    with tf.io.TFRecordWriter(out_file, options = tfoptions) as tfrecord:
        for file in tqdm(files, desc = data_name):
            data = json.loads(open(file, 'r').readline())
            article = get_string(data['clean_article'])
            summary = get_string(data['clean_summary'])
            features = {
                'article' : tf.train.Feature(bytes_list=tf.train.BytesList(value=
[article.encode()])),
                'summary' : tf.train.Feature(bytes_list=tf.train.BytesList(value=
[summary.encode()]))
            }
            # Write to tf.Example
            example = tf.train.Example(features = tf.train.Features(feature=featu
res))
            tfrecord.write(example.SerializeToString())

            if checkdata == True:
                print(example)
                checkdata = False

    print ("Finished writing file %s" % out_file)

write_tfrecords(main_path+'dev/*', data_path+'val.tfrecords', 'validation data')
write_tfrecords(main_path+'test/*', data_path+'test.tfrecords', 'test data')
write_tfrecords(main_path+'train/*', data_path+'train.tfrecords', 'train data')

chunk_all(chunks_dir, CHUNK_SIZE)
```

f. Kode analisis data

```
import os
from tqdm import tqdm
import json
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

def check_percentiles(lengths):
    for i in range(0,100,10):
        var = lengths
        var = np.sort(var, axis = None)
        print ("{} percentile value is {}".format(i,var[int(len(var)*(float(i)/10
0))]))
    print("100 percentile value is",var[-1])
```

```
def check_percentiles90(lengths):
    for i in range(90,100):
        var = lengths
        var = np.sort(var, axis = None)
        print ("{} percentile value is {}".format(i,var[int(len(var)*(float(i)/10
0))]))
    print("100 percentile value is",var[-1])

print("\npersentil 100, artikel")
check_percentiles(article_lengths)
print("\npersentil 100, ringkasan")
check_percentiles(summary_lengths)
print("\npersentil 90, artikel")
check_percentiles90(article_lengths)
print("\npersentil 90, ringkasan")
check_percentiles90(summary_lengths)
```

g. Kode *subclassing layer* (layers.py)

```
import tensorflow as tf
import numpy as np

class Encoder(tf.keras.layers.Layer):
  def __init__(self, vocab_size, embedding_dim, enc_units, batch_sz, bidirect=Fal
se, embedding_path=None):
    super(Encoder, self).__init__()
    self.batch_sz = batch_sz
    self.enc_units = enc_units

    if embedding_path is not None:
      print("\nInitialize Preembedding in encoder")
      with open(embedding_path, 'rb') as f:
        embedding_initial = tf.keras.initializers.Constant(np.load(f))
      self.embedding = tf.keras.layers.Embedding(vocab_size+4, embedding_dim, emb
eddings_initializer=embedding_initial, trainable=False, name='encoder_fasttext_em
bedding')
    else:
      print("\nInitialize Embedding in encoder")
      self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim, name=
'encoder_embedding')

    self.LSTM = tf.keras.layers.LSTM(self.enc_units,
                                     return_sequences=True,
                                     return_state=True,
                                     recurrent_initializer='glorot_uniform')
    self.bidirect = bidirect
    if bidirect:
      print("\ninitialize Bidirectional LSTM on encoder")
      self.LSTM = tf.keras.layers.Bidirectional(self.LSTM, name='encoder_bidirect
ional_LSTM')
    else :
      print("\nusing unidirectional on encoder")

  def initialize_hidden_state(self):
    if self.bidirect:
      return [tf.zeros((self.batch_sz, self.enc_units)) for i in range(4)]
    else:
      return [tf.zeros((self.batch_sz, self.enc_units)) for i in range(2)]

  def call(self, x):
    x = self.embedding(x)
    output= self.LSTM(x, initial_state = self.initialize_hidden_state())
    return output
```

```
class BahdanauAttention(tf.keras.layers.Layer):
  def __init__(self, units):
    super(BahdanauAttention, self).__init__()
    self.W1 = tf.keras.layers.Dense(units, name='W1')
    self.W2 = tf.keras.layers.Dense(units, name='W2')
    self.V = tf.keras.layers.Dense(1, name='V')

  def call(self, query, values):

    hidden_with_time_axis = tf.expand_dims(query, 1)
    score = self.V(tf.nn.tanh(
        self.W1(values) + self.W2(hidden_with_time_axis)))
    attention_weights = tf.nn.softmax(score, axis=1)
    context_vector = attention_weights * values
    context_vector = tf.reduce_sum(context_vector, axis=1)

    return context_vector, tf.squeeze(attention_weights,-1)

class Decoder(tf.keras.layers.Layer):
  def __init__(self, vocab_size, embedding_dim, dec_units, batch_sz, bidirect, em
bedding_path=None):
    super(Decoder, self).__init__()
    self.batch_sz = batch_sz
    self.dec_units = dec_units

    if embedding_path is not None:
      print("\nInitialize Preembedding in decoder")
      with open(embedding_path, 'rb') as f:
        embedding_initial = tf.keras.initializers.Constant(np.load(f))
      self.embedding = tf.keras.layers.Embedding(vocab_size+4, embedding_dim, emb
eddings_initializer=embedding_initial, trainable=False, name='decoder_fasttext_em
bedding')
    else:
      print("\nInitialize embedding in decoder")
      self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim, name=
'decoder_embedding')

    self.LSTM = tf.keras.layers.LSTM(self.dec_units, return_sequences=True, retur
n_state=True,
                                     recurrent_initializer='glorot_uniform', name='
decoder_LSTM')
    self.fc = tf.keras.layers.Dense(vocab_size, activation=tf.keras.activations.s
oftmax, name='fully_connected_softmax')


  def call(self, x, hidden, enc_output, context_vector):
    x = self.embedding(x)
    x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)
    output, state_h, state_c = self.LSTM(x, initial_state=hidden)
    output = tf.reshape(output, (-1, output.shape[2]))
    out = self.fc(output)

    return x, out, state_h, state_c


class Pointer(tf.keras.layers.Layer):

  def __init__(self):
    super(Pointer, self).__init__()
    self.w_s_reduce = tf.keras.layers.Dense(1, name='w_s_reduce')
    self.w_i_reduce = tf.keras.layers.Dense(1, name='w_i_reduce')
    self.w_c_reduce = tf.keras.layers.Dense(1, name='w_c_reduce')

  def call(self, context_vector, state, dec_inp):
    return  tf.nn.sigmoid(self.w_s_reduce(state)+self.w_c_reduce(context_vector)+s
elf.w_i_reduce(dec_inp))
```

h. Kode *subclassing model* (model.py)

```
import tensorflow as tf
from utils import _calc_final_dist
from layers import Encoder, BahdanauAttention, BahdanauAttentionCov, Decoder, Poi
nter
import numpy as np

class PGN(tf.keras.Model):
  def __init__(self, params):
    super(PGN, self).__init__()
    self.params = params
    self.encoder = Encoder(params["vocab_size"], params["embed_size"], params["en
c_units"], params["batch_size"], params['bidirect'], params['embed_path'])
    self.attention = BahdanauAttention(params["attn_units"])
    self.decoder = Decoder(params["vocab_size"], params["embed_size"], params["de
c_units"], params["batch_size"], params['bidirect'], params['embed_path'])
    self.pointer = Pointer()
    self.optimizer = tf.keras.optimizers.Adagrad(params['learning_rate'], initial
_accumulator_value=params['adagrad_init_acc'], clipnorm=params['max_grad_norm'])
    self.loss_object = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=
False, reduction='none')
    self.bidirect = params['bidirect']

  def call(self, inputs, training):

    enc_extended_inp = inputs[0]
    dec_inp = inputs[1]
    batch_oov_len = inputs[2]
    enc_output = inputs[3]
    dec_hidden = inputs[4]

    predictions = []
    attentions = []
    p_gens = []

    context_vector, _ = self.attention(dec_hidden[0], enc_output)
    for t in range(dec_inp.shape[1]):
      dec_input = tf.expand_dims(dec_inp[:, t],1)
      dec_x, pred, state_h, state_c = self.decoder(dec_input, dec_hidden, enc_out
put, context_vector)
      dec_hidden = [state_h, state_c]
      context_vector, attn = self.attention(state_h, enc_output)
      p_gen = self.pointer(context_vector, state_h, tf.squeeze(dec_x, axis=1))

      predictions.append(pred)
      attentions.append(attn)
      p_gens.append(p_gen)

    final_dists = _calc_final_dist(enc_extended_inp, predictions, attentions, p_g
ens, batch_oov_len, self.params["vocab_size"], self.params["batch_size"], dec_inp
.shape[1])
    if self.params["mode"] == "train":
      return tf.stack(final_dists, 1), dec_hidden
    else:
      return tf.stack(final_dists, 1), dec_hidden[0], dec_hidden[1], context_vect
or, tf.stack(attentions, 1), tf.stack(p_gens, 1)
```

```
  def loss_function(self, real, pred):
    mask = tf.math.logical_not(tf.math.equal(real, 1))
    dec_lens = tf.reduce_sum(tf.cast(mask, dtype=tf.float32), axis=-1)
    loss_ = self.loss_object(real, pred)
    mask = tf.cast(mask, dtype=loss_.dtype)
    loss_ *= mask
    loss_ = tf.reduce_sum(loss_, axis=-1)/dec_lens
    return tf.reduce_mean(loss_)

  def train_step(self, inputs):
    # print("tracing_train")
    enc_inp = inputs[0]['enc_input']
    enc_extended_inp = inputs[0]['extended_enc_input']
    dec_inp = inputs[1]['dec_input']
    dec_tar = inputs[1]['dec_target']
    batch_oov_len = inputs[0]['max_oov_len']

    loss = 0
    # print('tracing')
    with tf.GradientTape() as tape:
      output = self.encoder(enc_inp)
      if self.bidirect:
        enc_output, state_h_f, state_c_f, state_h_b, state_c_b = output
        state_h = tf.concat([state_h_f, state_h_b], axis=1)
        state_c = tf.concat([state_c_f, state_c_b], axis=1)
      else:
        enc_output, state_h, state_c = output
      dec_hidden = [state_h, state_c]

      predictions, _ = self([enc_extended_inp, dec_inp, batch_oov_len, enc_output
, dec_hidden], training=True)
      loss = self.loss_function(dec_tar, predictions)
    variables = self.trainable_variables
    gradients = tape.gradient(loss, variables)
    self.optimizer.apply_gradients(zip(gradients, variables))
    return {'loss' : loss}

  def test_step(self, inputs):
    enc_inp = inputs[0]['enc_input']
    enc_extended_inp = inputs[0]['extended_enc_input']
    dec_inp = inputs[1]['dec_input']
    dec_tar = inputs[1]['dec_target']
    batch_oov_len = inputs[0]['max_oov_len']

    loss = 0

    output = self.encoder(enc_inp)
    if self.bidirect:
      enc_output, state_h_f, state_c_f, state_h_b, state_c_b = output
      state_h = tf.concat([state_h_f, state_h_b], axis=1)
      state_c = tf.concat([state_c_f, state_c_b], axis=1)
    else:
      enc_output, state_h, state_c = output
    dec_hidden = [state_h, state_c]
    predictions, _ = self([enc_extended_inp, dec_inp, batch_oov_len, enc_output, d
ec_hidden], training=False)
    loss = self.loss_function(dec_tar, predictions)
    return {'loss' : loss}

class Base(tf.keras.Model):

  def __init__(self, params):
    super(Base, self).__init__()
    self.params = params
    self.encoder = Encoder(params["vocab_size"], params["embed_size"], params["en
c_units"], params["batch_size"], params['bidirect'], params['embed_path'])
```

```
    self.attention = BahdanauAttention(params["attn_units"])
    self.decoder = Decoder(params["vocab_size"], params["embed_size"], params["de
c_units"], params["batch_size"], params['bidirect'], params['embed_path'])
    self.optimizer = tf.keras.optimizers.Adagrad(params['learning_rate'], initial
_accumulator_value=params['adagrad_init_acc'], clipnorm=params['max_grad_norm'])
    self.loss_object = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=
False, reduction='none')
    self.bidirect = params['bidirect']

  def call(self, inputs, training):

    dec_inp = inputs[0]
    dec_hidden = inputs[1]
    enc_output = inputs[2]

    predictions = []
    attentions = []

    context_vector, _ = self.attention(dec_hidden[0], enc_output)
    for t in range(dec_inp.shape[1]):

      dec_input = tf.expand_dims(dec_inp[:, t],1)
      dec_x, pred, state_h, state_c = self.decoder(dec_input, dec_hidden, enc_out
put, context_vector)
      dec_hidden = [state_h, state_c]
      context_vector, attn = self.attention(state_h, enc_output)

      predictions.append(pred)
      attentions.append(attn)

    if self.params["mode"] == "train":
      return tf.stack(predictions, 1), dec_hidden
    else:
      return tf.stack(predictions, 1), dec_hidden, context_vector, tf.stack(atten
tions, 1)

  def loss_function(self, real, pred):
    mask = tf.math.logical_not(tf.math.equal(real, 1))
    dec_lens = tf.reduce_sum(tf.cast(mask, dtype=tf.float32), axis=-1)
    loss_ = self.loss_object(real, pred)
    mask = tf.cast(mask, dtype=loss_.dtype)
    loss_ *= mask
    loss_ = tf.reduce_sum(loss_, axis=-1)/dec_lens
    return tf.reduce_mean(loss_)

  def train_step(self, inputs):
    # print('tracing')

    enc_inp = inputs[0]['enc_input']
    dec_inp = inputs[1]['dec_input']
    dec_tar = inputs[1]['dec_target']

    with tf.GradientTape() as tape:
      output = self.encoder(enc_inp)

      if self.bidirect:
        enc_output, state_h_f, state_c_f, state_h_b, state_c_b = output
        state_h = tf.concat([state_h_f, state_h_b], axis=1)
        state_c = tf.concat([state_c_f, state_c_b], axis=1)
      else:
        enc_output, state_h, state_c = output

      dec_hidden = [state_h, state_c]
      predictions, _ = self([dec_inp, dec_hidden, enc_output], training=True)
      loss = self.loss_function(dec_tar, predictions)
    variables = self.trainable_variables
    gradients = tape.gradient(loss, variables)
```

```
    self.optimizer.apply_gradients(zip(gradients, variables))
    return {'loss' : loss}

  def test_step(self, inputs):

    enc_inp = inputs[0]['enc_input']
    dec_inp = inputs[1]['dec_input']
    dec_tar = inputs[1]['dec_target']

    output = self.encoder(enc_inp)

    if self.bidirect:
      enc_output, state_h_f, state_c_f, state_h_b, state_c_b = output
      state_h = tf.concat([state_h_f, state_h_b], axis=1)
      state_c = tf.concat([state_c_f, state_c_b], axis=1)
    else:
      enc_output, state_h, state_c = output

    dec_hidden = [state_h, state_c]

    predictions, _ = self([dec_inp, dec_hidden, enc_output], training=False)
    loss = self.loss_function(dec_tar, predictions)
    return {'loss' : loss}
```

i. Kode train

```
def train(params):
    assert params["mode"].lower() == "train", "change training mode to 'train'"
    tf.compat.v1.logging.info("Building the model ...")
    if params['pointer']:
        print("\nPointer Model")
        model = PGN(params)
    else:
        print("\nNon Pointer Model")
        model = Base(params)

    print("\nCreating the vocab ...")
    vocab = Vocab(params["vocab_path"], params["vocab_size"])

    print("\nCreating the batcher ...")
    b = batcher(params["data_dir"], vocab, params)
    vb = batcher(params["val_data_dir"], vocab, params, training=False)

    # model.compile(run_eagerly=True)
    model.compile()

    tb = tf.keras.callbacks.TensorBoard("output/logs/"+params['model_name']+"_"+d
atetime.datetime.now().strftime("%Y%m%d-%H%M%S"),
                                        histogram_freq=1, write_graph=True, write
_steps_per_second=True, update_freq='batch')
    mc = tf.keras.callbacks.ModelCheckpoint('output/checkpoints/'+params['model_n
ame']+"/"+datetime.datetime.now().strftime("%Y%m%d-%H%M%S"),
                                            monitor='val_loss', verbose=1, save_w
eights_only=True, save_freq='epoch')
    es = tf.keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=0, patien
ce=10, verbose=1,
                                            mode='auto', restore_best_weights=Tru
e)
```

```
    tf.compat.v1.logging.info("Starting the training ...")
    if params['pointer']:
        t0 = time.time()
        print("\nstart time: "+str(datetime.datetime.now()))
        print("\n### Pointer Model Training ####")
        history = model.fit(b, epochs=params['epochs'], steps_per_epoch=params['s
teps_per_epoch'],
                            validation_data=vb, validation_steps=params['val_step
s_per_epoch'], callbacks=[tb, mc, es])
        print("\n train_duration : "+str(time.time()-t0))
    else:
        t0 = time.time()
        print("\nstart time: "+str(datetime.datetime.now()))
        print("\n### Base Model Training ####")
        history = model.fit(b, epochs=params['epochs'], steps_per_epoch=params['s
teps_per_epoch'],
                    validation_data=vb, validation_steps=params['val_steps_per_ep
och'], callbacks=[tb, mc, es])
        print("\ntrain_duration : "+str(time.time()-t0))

    tf.compat.v1.logging.info("Saving Model ...")
    model.save('output/saved_models/'+params['model_name'], overwrite=True,
                include_optimizer=True, save_traces=True)

    tf.compat.v1.logging.info("\nSaving Model Weights ...")
    model.save_weights("output/saved_weights/"+params['model_name']+"/saved_weigh
ts", overwrite=True)

    model.save_weights("output/saved_weights/"+params['model_name']+"/saved_weigh
ts.h5", overwrite=True,
                        save_format="h5")
```

j. Kode beam decoder (test_helper.py)

```
import tensorflow as tf
import numpy as np
from batcher import Data_Helper

def beam_decode(model, batch, vocab, params, show=True):

  def decode_onestep(batch, enc_outputs, dec_state, dec_input):

    final_dists, state_h, state_c, context_vector, attentions, p_gens = model([ba
tch[0]["extended_enc_input"], dec_input, batch[0]["max_oov_len"], enc_outputs, de
c_state], training=False)
    top_k_probs, top_k_ids = tf.nn.top_k(tf.squeeze(final_dists), k = params["bea
m_size"]*2)
    top_k_log_probs = tf.math.log(top_k_probs)
    results = {"last_context_vector" : context_vector,
               "state_h" : state_h,
               "state_c" : state_c,
               "attention_vec" :attentions,
               "top_k_ids" : top_k_ids,
               "top_k_log_probs" : top_k_log_probs,
               "p_gen" : p_gens}
    return results


    enc_outputs, state_h, state_c = output
```

```
  class Hypothesis:

    def extend(self, token, log_prob, state_h, state_c, attn_dist, p_gen):
      return Hypothesis(tokens = self.tokens + [token],
                        log_probs = self.log_probs + [log_prob],
                        state_h = state_h,
                        state_c = state_c,
                        attn_dists = self.attn_dists + [attn_dist],
                        p_gens = self.p_gens + [p_gen]
                        )

    @property
    def latest_token(self):
      return self.tokens[-1]

    @property
    def tot_log_prob(self):
      return sum(self.log_probs)

    @property
    def avg_log_prob(self):
      return self.tot_log_prob/len(self.tokens)

  output = model.encoder(batch[0]["enc_input"])

  if params['bidirect']:
    enc_outputs, state_h_f, state_c_f, state_h_b, state_c_b = output
    state_h = tf.concat([state_h_f, state_h_b], axis=1)
    state_c = tf.concat([state_c_f, state_c_b], axis=1)
  else:
    enc_outputs, state_h, state_c = output

  hyps = [Hypothesis(tokens=[vocab.word_to_id('[START]')],
                     log_probs = [0.0],
                     state_h = state_h,
                     state_c = state_c,
                     attn_dists=[],
                     p_gens = [],
                     ) for _ in range(params['batch_size'])]


  results = []
  steps=0

  while steps < params['max_dec_steps'] and len(results) < params['beam_size'] :


    latest_tokens = [h.latest_token for h in hyps]
    latest_tokens = [t if t in range(params['vocab_size']) else vocab.word_to_id(
'[UNK]') for t in latest_tokens]
    states_h = [h.state_h for h in hyps]
    states_c = [h.state_c for h in hyps]
    states = [state_h, state_c]

    returns = decode_onestep( batch, enc_outputs, states, tf.expand_dims(latest_t
okens, axis=1))
    topk_ids, topk_log_probs, new_states_h, new_states_c, attn_dists , p_gens=  r
eturns['top_k_ids'], returns['top_k_log_probs'], returns['state_h'], returns['sta
te_c'], returns['attention_vec'], np.squeeze(returns["p_gen"])
    all_hyps = []
    num_orig_hyps = 1 if steps == 0 else len(hyps)
    for i in range(num_orig_hyps):

      h, new_state_h, new_state_c, attn_dist, p_gen = hyps[i], new_states_h[i], n
ew_states_c[i], attn_dists[i], p_gens[i]
```

```
      for j in range(params['beam_size']*2):

        new_hyp = h.extend(token=topk_ids[i,j].numpy(),
                           log_prob=topk_log_probs[i,j],
                           state_h = new_state_h,
                           state_c = new_state_c,
                           attn_dist=attn_dist,
                           p_gen=p_gen)
        all_hyps.append(new_hyp)

    hyps = []
    sorted_hyps = sorted(all_hyps, key=lambda h: h.avg_log_prob, reverse=True)
    for h in sorted_hyps:
      if h.latest_token == vocab.word_to_id('[STOP]'):
        if steps >= params['min_dec_steps']:
          results.append(h)
      else:
        hyps.append(h)
      if len(hyps) == params['beam_size'] or len(results) == params['beam_size']:
        break

    steps += 1

  if len(results)==0:
    results=hyps

  hyps_sorted = sorted(results, key=lambda h: h.avg_log_prob, reverse=True)
  best_hyp = hyps_sorted[0]
  best_hyp.abstract = " ".join(Data_Helper.output_to_words(best_hyp.tokens, vocab
, batch[0]["article_oovs"][0], params['pointer'])[1:-1])
  best_hyp.text = batch[0]["article"].numpy()[0].decode()
  if params["mode"] == "eval":
    best_hyp.real_abstract = batch[1]["abstract"].numpy()[0].decode()
    print("\ntokens")
    print(best_hyp.tokens)
    print("\noovs")
    print(batch[0]["article_oovs"][0].numpy())
    print("\narticle")
    print(best_hyp.text)
    print("\nbest_hyp real abstract")
    print(best_hyp.real_abstract)
    print("\nbest_hyp abstract")
    print(best_hyp.abstract)
  return best_hyp
```

k. Kode test

```
def test(params):
        assert params["mode"].lower() in ["test","eval"], "change training mode t
o 'test' or 'eval'"
        assert params["beam_size"] == params["batch_size"], "Beam size must be eq
ual to batch_size, change the params"

        tf.compat.v1.logging.info("Building the model ...")
        model = PGN(params)

        print("Creating the vocab ...")
        vocab = Vocab(params["vocab_path"], params["vocab_size"])

        print("Creating the batcher ...")
        b = batcher(params["data_dir"], vocab, params)

        print("Creating the checkpoint manager")
        checkpoint_dir = "{}".format(params["checkpoint_dir"])
        ckpt = tf.train.Checkpoint(step=tf.Variable(0), PGN=model)
        ckpt_manager = tf.train.CheckpointManager(ckpt, checkpoint_dir, max_to_ke
ep=11)

        path = params["model_path"] if params["model_path"] else ckpt_manager.lat
est_checkpoint
        ckpt.restore(path)
        print("Model restored")

        for batch in b:
                yield  beam_decode(model, batch, vocab, params)
```

l. Kode eval.

```
def evaluate(params):
        gen = test(params)
        reals = []
        preds = []
        with tqdm(total=params["max_num_to_eval"],position=0, leave=Tr
ue) as pbar:
                for i in range(params["max_num_to_eval"]):
                        trial = next(gen)
                        reals.append(trial.real_abstract)
                        preds.append(trial.abstract)
                        pbar.update(1)
        r=Rouge()
        scores = r.get_scores(preds, reals, avg=True)
        print("\n\n")
        pprint.pprint(scores)
```

Berikut alamat akses program, model yang telah dilatih dan data TFRecords, https://drive.google.com/drive/folders/1d9Nj1_keuQSv3-vnTpmCRos-CcrfV4vP?usp=sharing

Beberapa kode program adalah hasil modifikasi dari https://github.com/fajri91/sum_liputan6 dan https://github.com/steph1793/Pointer_Generator_Summarizer.