BAB 4
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Sistem

Pada tahap ini dilakukan implementasi sistem yang merupakan penerapan hasil perancangan yang telah dilakukan ke dalam bentuk program. Implementasi sistem terdiri dari implementasi perangkat keras, implementasi perangkat lunak, dan implementasi antarmuka.

4.1.1 Implementasi Perangkat Keras

Berikut ini merupakan perangkat keras yang digunakan untuk mengimplementasikan sistem ABSA dengan menggunakan ERNN ini dapat dilihat pada Tabel 4.1.

Tabel 4.1 Implementasi Perangkat Keras

No	Perangkat Keras	Spesifikasi
1	*Processor*	AMD A8-6410 APU
2	RAM	8 Gb
3	*Harddisk*	500 Gb
4	*VGA*	AMD Radeon R5

4.1.2 Implementasi Perangkat Lunak

Adapun untuk perangkat lunak yang digunakan dalam pembangunan sistem ABSA dengan menggunakan ERNN ini dapat dilihat pada Tabel 4.2.

Tabel 4.2 Implementasi Perangkat Lunak

No	Jenis Perangkat Lunak	Perangkat Lunak
1	Sistem Operasi	Windows 10 Pro 64-bit
2	*Web browser*	Firefox Quantum
3	*Tekt Editor*	Visual Studio Code

4.1.3 Implementasi Antarmuka

Berikut ini merupakan hasil dari perancangan antarmuka untuk prototype sistem ABSA dengan menggunakan ERNN. Penjelasan implementasi antarmuka pada prototype sistem ABSA dengan menggunakan ERNN dapat dilihat pada Tabel 4.3.

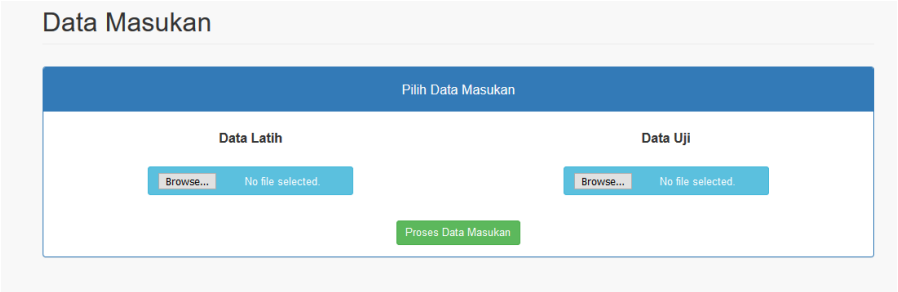
Tabel 4.3 Deskripsi Antarmuka

No	Nama Antarmuka	Deskripsi
1	Data Masukan	Menampilkan halaman untuk memilih data latih dan data uji yang akan digunakan
2	Praproses Data Latih	Menampilkan halaman praproses data latih yang terdiri dari proses *case folding, filtering, word normalization, tokenization, stopword removal,* penambahan token aspek, pembangunan kamus kata, dan *one hot encoding*
3	Proses Pelatihan	Menampilkan halaman pelatihan dimana user akan memasukan parameter pelatihan
4	Praproses Data Uji	Menampilkan halaman praproses data uji yang terdiri dari proses *case folding, filtering, word normalization, tokenization, stopword removal,* penambahan token aspek, dan *one hot encoding*
5	Proses Pengujian	Menampilkan halaman hasil pengujian token aspek

Berikut adalah tampilan antarmuka prototype sistem ABSA menggunakan metode ERNN.

1. Implementasi Antarmuka Halaman Data Masukan

   Halaman data masukan yang dibangun akan menampilkan dua buah panel untuk memilih file data latih dan file data uji yang akan digunakan. Adapun implementasi antarmuka halaman data masukan ditunjukan

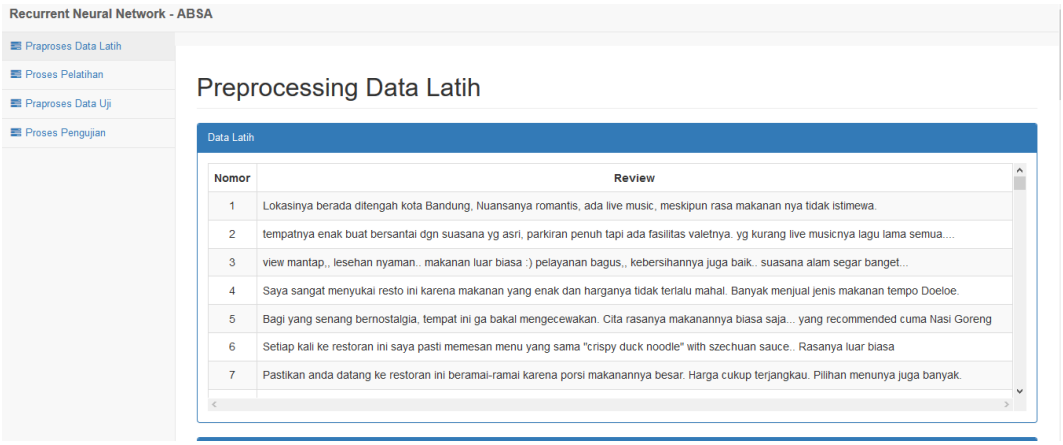


Gambar 4.1 Antarmuka Halaman Data Masukan

   Gambar 4.1.

2. Implementasi Antarmuka Halaman Praproses Data Latih

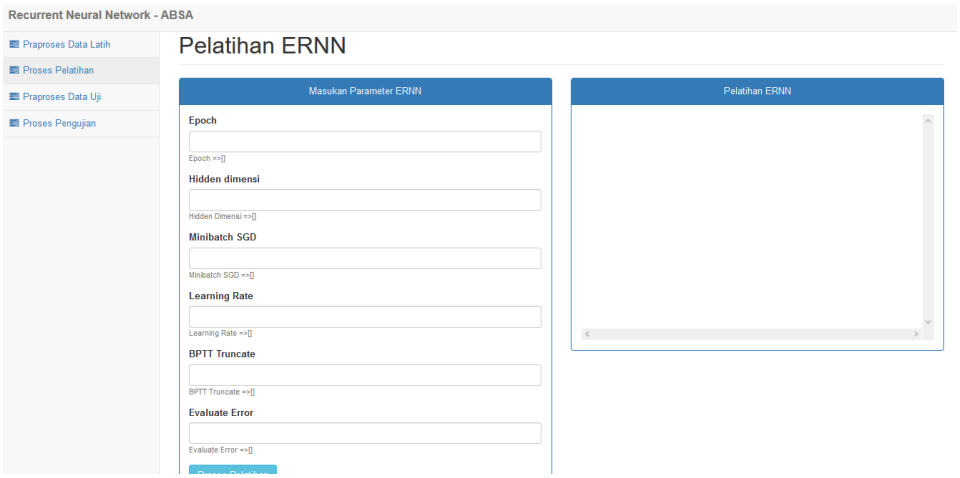
   Halaman praproses data latih akan menampilkan proses-proses *preprocessing* yang terdiri dari *case folding*, *filtering*, *word normalization*, *tokenization*, *stopword removal*, penambahan token aspek, pembangunan kamus kata, dan *one hot encoding*. Implementasi antarmuka halaman prarproses data latih dapat dilihat pada Gambar 4.2.



Gambar 4.2 Antarmuka Halaman Praproses Data Latih

3. Implementasi Antarmuka Halaman Pelatihan

   Halaman pelatihan akan menampilkan parameter-parameter yang akan digunakan dalam proses pelatihan data latih. Adapun antarmuka halaman

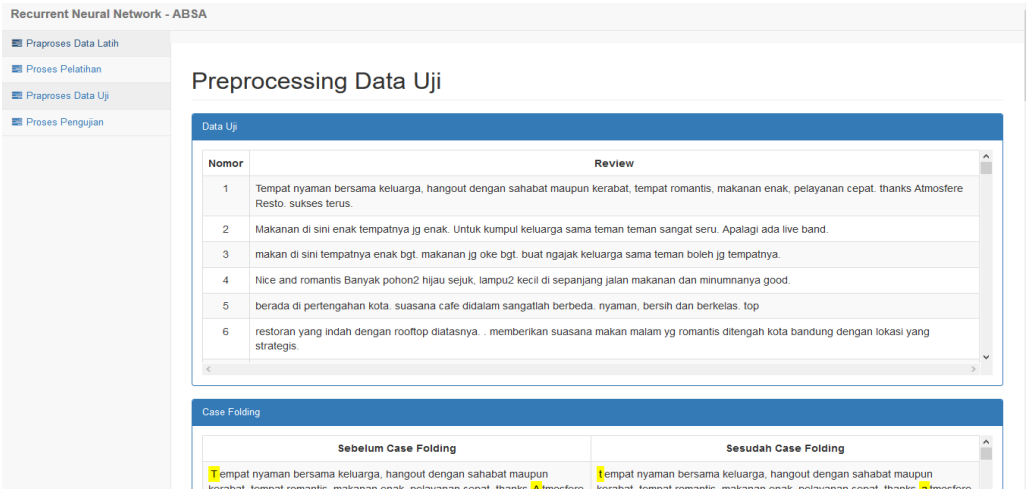


Gambar 4.3 Antarmuka Halaman Pelatihan

pelatihan dapat dilihat pada Gambar 4.3.

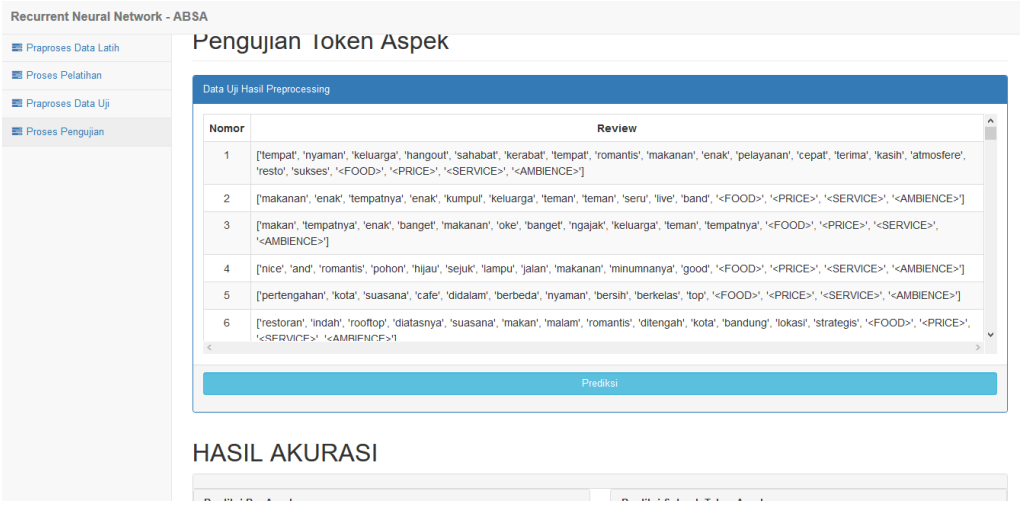
4. Implementasi Antarmuka Halaman Praproses Data Uji

   Halaman praproses data uji akan menampilkan proses-proses *preprocessing* terhadap data uji yang terdiri dari *case folding*, *filtering, word normalization, tokenization, stopword removal*, penambahan token aspek, dan *one hot encoding*. Implementasi antarmuka halaman prarproses data uji dapat dilihat pada Gambar 4.4.



5. Implementasi Antarmuka Halaman Pengujian

   Halaman pengujian akan menampilkan hasil akhir praproses data uji selain itu pada halaman ini juga menampilkan hasil prediksi algoritma terhadap token aspek. Adapun antarmuka halaman pengujian ditunjukan Gambar



Gambar 4.5 Antarmuka Halaman Pengujian

4.5

4.2 Pengujian Sistem

Pengujian sistem dilakukan untuk memastikan bahwa sistem yang dibuat sesuai dengan rancangan sistem yang telah dibangun sebelumnya. Adapun pengujian sistem yang dilakukan adalah sebagai berikut.

4.2.1 Skenario Pengujian

Pada penelitian ini skenario pengujian meliputi skenario pengujian fungsionalitas *black box*, skenario pengujian fungsionalitas *white box*, skenario pengujian parameter, dan skenario pengujian performa.

4.2.1.1 Skenario Pengujian Fungsionalitas *Black Box*

Berikut adalah skenario pengujian fungsionalitas *black box* dimana pengujian berfokus pada meneliti fungsionalitas perangkat lunak tanpa mengintip

struktur atau pengkodean internal. Adapun skenario pengujian fungsionalitas *black box* dapat dilihat pada Tabel 4.4.

Tabel 4.4 Skenario Pengujian Fungsionalitas *Black Box*

No	Nama Proses	Poin Pengujian	Jenis Pengujian
1	Praproses Data Latih	*Case Folding*	*Black Box*
		Filtering	*Black Box*
		Word Normalization	*Black Box*
		Tokenization	*Black Box*
		Stopword Removal	*Black Box*
		Penambahan Token Aspek	*Black Box*
		Pembangunan Kamus Kata	*Black Box*
		One Hot Encoding	*Black Box*
2	Praproses Data Uji	*Case Folding*	*Black Box*
		Filtering	*Black Box*
		Word Normalization	*Black Box*
		Tokenization	*Black Box*
		Stopword Removal	*Black Box*
		Penambahan Token Aspek	*Black Box*
		One Hot Encoding	*Black Box*

4.2.1.2 Skenario Pengujian Fungsionalitas *White Box*

Pada penelitian ini pengujian fungsionalitas *white box* menggunakan metode *basis path testing* dimana pengujian berfokus pada meneliti fungsionalitas perangkat lunak dengan malihat struktur atau pengkodean internal. Adapun skenario pengujian fungsionalitas white box dapat dilihat pada Tabel 4.5.

Tabel 4.5 Skenario Pengujian Fungsionalitas *White Box*

No	Nama Proses	Poin Pengujian
1	Pelatihan ERNN	Fungsi Forward Propagation
		Fungsi BPTT
		Fungsi SGD
		Fungsi Train
2	Pengujian ERNN	Fungsi Predict

4.2.1.3 Skenario Pengujian Parameter

Pengujian parameter bertujuan untuk mendapatkan model yang terbaik karena performa model ditentukan oleh pemilihan parameter yang tepat. Pada penelitian ini pengujian parameter akan dilakukan terhadap data latih sebanyak 1584 kalimat dengan ketentuan pada Tabel 4.6.

Tabel 4.6 Jumlah Kelas Aspek

Aspek	Bukan Sentimen	Positif	Negatif
Food	1030	493	61
Price	1209	232	143
Service	1291	239	54
Ambience	1120	401	63

Pada pemilihan parameter terdapat parameter-parameter yang disarankan dari beberapa penelitian disamping parameter yang akan diuji oleh peneliti. Adapun nilai parameter-parameter yang akan diuji dapat dilihat pada Tabel 4.7.

Tabel 4.7 Nilai Parameter

No	Nama Parameter	Nilai
1	*Learning rate*	{0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001}[23] {0.1, 0.5}
2	*Dimensi hidden*	{52}[24] {12, 22, 32, 42}
3	*Minibatch size*	{256}[25] {512, 1024}
4	*Epoch*	{100}[26]
5	*BPTT Truncated*	{2, 4, 6, 8, 10}

Adapun untuk memilih nilai parameter terbaik dari masing-masing parameter akan menggunakan parameter dasar sebagai acuan utama, misalnya untuk melakukan pengujian parameter "*learning rate*" maka hanya nilai *learning rate* yang diubah dari parameter dasar. Nilai parameter dasar dapat dilihat pada Tabel 4.8 dan skenario pengujian parameter ditunjukan pada Tabel 4.9.

Tabel 4.8 Parameter Dasar

No	Nama Parameter	Nilai
1	*Learning rate*	0.1
2	*Dimensi hidden*	32
3	*Minibatch size*	256
4	*Epoch*	100
5	*BPTT Truncated*	4

Tabel 4.9 Skenario Pengujian Parameter

No	Learning rate	Dimensi hidden	Minibatch size	BPTT	Epoch
1	0.5	32	256	4	100
2	0.001	32	256	4	100
3	0.005	32	256	4	100
4	0.001	32	256	4	100
5	0.0005	32	256	4	100
6	0.0001	32	256	4	100
7	0.1	32	256	4	100
8	0.5	32	256	4	100
9	0.1	32	256	4	100
10	0.1	42	256	4	100
11	0.1	12	256	4	100
12	0.1	22	256	4	100
13	0.1	52	256	4	100
14	0.1	32	512	4	100
15	0.1	32	256	4	100
16	0.1	32	1024	4	100
17	0.1	32	256	8	100
18	0.1	32	256	6	100
19	0.1	32	256	4	100
20	0.1	32	256	10	100

21	0.1	32	256	2	100

Dalam melakukan pengujian parameter menggunakan optimisasi Adagrad untuk mengstabilkan kinerja algoritma.

4.2.1.4 Skenario Pengujian Performa

Pengujian performa bertujuan untuk melihat seberapa bagus algoritma dalam menyelesaikan kasus ABSA dilihat dari nilai akurasi dan f1 *score* yang didapat. Pada pengujian performa nilai parameter yang digunakan adalah parameter yang telah diseleksi pada proses pengujian parameter. Dalam melakukan pengujian performa menggunakan optimisasi Adagrad untuk mengstabilkan kinerja algoritma.

4.2.2 Pengujian

Berikut adalah hasil pengujian yang telah dilakukan berdasarkan skenario pengujian yang telah dibuat.

4.2.2.1 Pengujian Fungsionalitas *Black Box*

Pada subbab ini dilakukan pengujian fungsionalitas menggunakan metode *black box*. Adapun pengujian fungsionalitas *black box* yang telah dilakukan terhadap poin pada skenario pengujian dapat dilihat pada Tabel 4.10.

Tabel 4.10 Hasil Pengujian Fungsionalitas

Aktivitas yang Dilakukan	Data Masukan	Keluaran yang Diharapkan	Pengamatan	Kesimpulan
Case Folding	Kalimat *Review*	Kalimat dengan semua karakter berupa *lowercase*	Menampilkan kalimat dengan semua karakter *lowercase*	[√] Diterima [ ] Ditolak
Filtering	Kalimat *Review*	Kalimat tanpa adanya simbol dan tanda baca	Menampilkan kalimat tanpa adanya simbol dan tanda baca	[√] Diterima [ ] Ditolak
Word Normalization	Kalimat *Review*	Kalimat dengan kata-kata baku	Menampilkan kalimat dengan kata-kata baku	[√] Diterima [ ] Ditolak
Tokenization	Kalimat *Review*	Array token kata	Menampilkan array token kata	[√] Diterima [ ] Ditolak
Stopword Removal	Kalimat *Review*	Array token kata tanpa *stopword*	Menampilkan array token kata tanpa *stopword*	[√] Diterima [ ] Ditolak
Penambahan Token Aspek	Kalimat *Review*	Array token kata dengan token aspek	Menampilkan array token kata dengan token aspek	[√] Diterima [ ] Ditolak

Pembangunan Kamus Kata	Kalimat *Review*	Kamus kata unik/berbeda	Menampilkan kamus kata unik	[√] Diterima [ ] Ditolak
One Hot Encoding	Kalimat *Review*	Vektor setiap token kata	Menampilkan indeks vektor setiap token kata	[√] Diterima [ ] Ditolak

4.2.2.2 Pengujian Fungsionalitas *White Box*

Pengujian pada subbab ini dilakukan menggunakan metode *white box* dengan *basis path testing* dimana prosesnya meliputi *pseudocode*, *flowgraph* , *independent path*, dan *test case*. Adapun berikut adalah pengujian whitebox dari masing-masing poin pengujian pada skenario pengujian.

1. Fungsi Forward Propagation

Implementasi *pseudocode* dari kode *forward propagation* ditunjukan Gambar 4.6.

	Pseudocode:

	`forward_propagation(x, params, weight)`
	`{`
	`// x_train = satu data train`
	`// weight = bobot`
	`// params = parameter`

1	`T = number of token in x_train`
	`hidden_state = numpy.zeros((T + 1, params['hidden_dim']))`
	`output_state = numpy.zeros((T, params['hidden_dim']))`

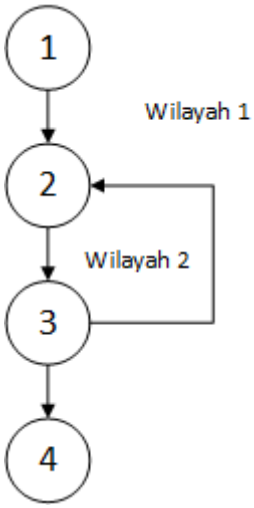
2	`for t = 1 to T`

3	`hidden_state[t] = tanh(weight['U'].dot(x_train[t]) + weight['W'].dot(hidden_state[t-1]))`
	`output_state[t] = softmax(weight['V'].dot(hidden_state[t]))`
	`endfor`

4	`return [output_state, hidden_state]`
	`}`

Gambar 4.6 *Pseudocode Forward Propagation*

Selanjutnya *pseudocode* diubah menjadi flowgraph seperti yang ditunjukan Gambar 4.7.



Gambar 4.7 Flowgraph Forward Propagation

Kemudian lakukan identifikasi *independent path*. Adapun jumlah *independent path* sama dengan *cyclomatic complexity*. Oleh karena itu terlebih dahulu hitung nilai *cyclomatic complexity*, perhitungan *cyclomatic complexity* dengan cara menghitung jumlah wilayah (*region*). Adapun berikut adalah proses perhitungannya dimana V(G) = *cyclomatic complexity*.

V(G) = (jumlah wilayah tertutup) + wilayah terbuka
= (wilayah 2) + wilayah 1
= 1 + 1
= 2

Sehingga terdapat 2 *independent path* yang ada dalam *flowgraph* tersebut. Adapun *independent path* dari *flowgraph* tersebut adalah sebagai berikut dimana nilai yang ditebalkan menandakan *edge/*garis baru.

Independent path ke-1 = 1-2-3-4

Independent path ke-2 = 1-2-**3-2**-3-4

Dengan 2 *independent path* maka terdapat 2 *test case* yang akan dilakukan seperti yang ditunjukkan Tabel 4.11.

Tabel 4.11 *Test Case Forward Propagation*

No. Path	Path	Data Masukan	Keluaran yang	Pengamatan	Kesimpulan

			Diharapkan		
1	1-2-3-4	*x_train = [[0,0,1,0,0]]* yaitu sebuah data train dengan 1 token dimana token adalah vektor onehot berdimensi 5	Menghasilkan 1 output vektor dan 1 *hidden state* .	Menghasilkan 1 output vektor dan 1 *hidden state* .	[√] Terlewati [ ] Tidak Terlewati
2	1-2-**3**-**2**-3-4	*x_train = [[0,0,1,0,0], [0,0,0,0,1]]* yaitu sebuah data train dengan 2 token dimana token adalah vektor onehot berdimensi 5	Menghasilkan 2 output vektor dan 2 *hidden state* .	Menghasilkan 2 output vektor dan 2 *hidden state* .	[√] Terlewati [ ] Tidak Terlewati

2. Fungsi Backpropagation Through Time (BPTT)

Implementasi *pseudocode* dari kode *backpropagation through time* ditunjukan Gambar 4.8.

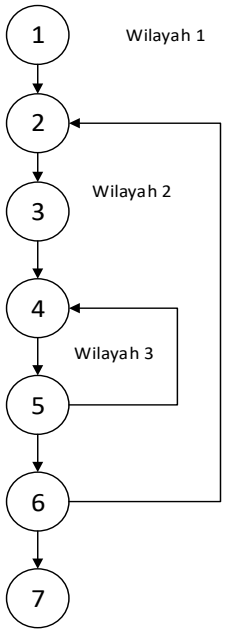
	Pseudocode:

```
bptt(x_train, y_train, params, weight)
{
        // x_train = 1 data train
        // y_train = 1 data true label train
        // weight = bobot
        // params = parameter
```

1	T = number of token in x_train output_state, hidden_state = forward_propagation(x_train, params, weight) dLdU = np.zeros(weight['U'].shape) dLdV = np.zeros(weight ['V'].shape) dLdW = np.zeros(weight ['W'].shape) delta_output_state = output_state
2	for t = T to 1
3	delta_output_state[t] = delta_output_state[t] – y_train[t] dLdV = dLdV + np.outer(delta_output_state[t], hidden_state[t].T) delta_t = weight ['V'].T.dot(delta_output_state[t]) * (1 – (hidden_state[t]^2)) start_iteration = t stop_iteration = max(1, t-params['bptt_truncate'])
4	for bptt_step = start_iteration to stop_iteration
5	dLdW = dLdW + np.outer(delta_t, hidden_state[bptt_step – 1]) dLdU = dLdU + np.outer(delta_t, x_train[bptt_step]) delta_t = weight ['W'].T.dot(delta_t) * (1 – hidden_state[bptt_step – 1]^2) endfor
6	endfor
7	return dLdU, dLdV, dLdW
	}

Gambar 4.8 Pseudocode BPTT

Selanjutnya *pseudocode* diubah menjadi flowgraph seperti yang ditunjukan Gambar 4.9.



Gambar 4.9 Flowgraph BPTT

Kemudian lakukan identifikasi *independent path*. Adapun jumlah *independent path* sama dengan *cyclomatic complexity*. Oleh karena itu terlebih dahulu hitung nilai *cyclomatic complexity*, perhitungan *cyclomatic complexity* dengan cara menghitung jumlah wilayah (*region*). Adapun berikut adalah proses perhitungannya dimana V(G) = *cyclomatic complexity*.

V(G) = (jumlah wilayah tertutup) + wilayah terbuka
= (wilayah 2 + wilayah 3) + wilayah 1
= 2 + 1
= 3

Sehingga seharusnya terdapat 3 *independent path* yang ada dalam *flowgraph* tersebut namun karena ada *nested loop* dimana *inner loop* bergantung pada *outer loop* sehingga menjadi 2 *independent path*. Adapun *independent path* dari *flowgraph* tersebut adalah sebagai berikut dimana nilai yang ditebalkan menandakan *edge/*garis baru.

Independent path ke-1 = 1-2-3-4-5-6-7

Independent path ke-2 = 1-2-3-4-5-**4-5-6-2-3-4-5**-6-7

Dengan 2 *independent path* maka terdapat 2 *test case* yang akan dilakukan seperti yang ditunjukkan Tabel 4.12.

Tabel 4.12 Test Case BPTT

No. Path	Path	Data Masukan	Keluaran yang Diharapkan	Pengamatan	Kesimpulan
1	1-2-3-4-5-6-7	*x_train = [[0,0,1,0,0]]* yaitu sebuah data train dengan 1 token dimana token adalah vektor onehot berdimensi 5	Menghasilkan nilai turunan U, W, V	Menghasilkan nilai turunan U, W, V	[√] Terlewati [ ] Tidak Terlewati
2	1-2-3-4-5-**4-5-6-2-3-4-5**-6-7	*x_train = [[0,0,1,0,0], [0,0,0,0,1]]* yaitu sebuah data train dengan 2 token dimana token adalah vektor onehot berdimensi 5	Menghasilkan nilai turunan U, W, V	Menghasilkan nilai turunan U, W, V	[√] Terlewati [ ] Tidak Terlewati

3. Fungsi Stochastic Gradient Descent (SGD)

Implementasi *pseudocode* dari kode *stochastic gradient descent* ditunjukan Gambar 4.10.

	Pseudocode:

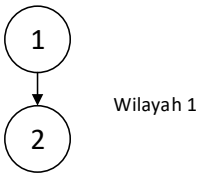
	`sgd_step(model, x_train, y_train, learning_rate)`
	`{`
	`  // model = mengandung params dan weight`
	`  // x_train = satu data train`
	`  // y_train = satu data true label train`

1	`  dLdU, dLdV, dLdW = bptt(x_train, y_train, model['params'],`

	model[‘weight’]) model[weight']['U'] -= learning_rate * dLdU model[weight']['V'] -= learning_rate * dLdV model[weight']['W'] -= learning_rate * dLdW
2	Return model
	}

Gambar 4.10 Pseudocode SGD

Selanjutnya *pseudocode* diubah menjadi flowgraph seperti yang ditunjukan Gambar 4.11.



Gambar 4.11 Flowgraph SGD

Kemudian lakukan identifikasi *independent path*. Adapun jumlah *independent path* sama dengan *cyclomatic complexity*. Oleh karena itu terlebih dahulu hitung nilai *cyclomatic complexity*, perhitungan *cyclomatic complexity* dengan cara menghitung jumlah wilayah (*region*). Adapun berikut adalah proses perhitungannya dimana V(G) = *cyclomatic complexity*.

V(G) = (jumlah wilayah tertutup) + wilayah terbuka
= (0) + wilayah 1
= 0 + 1
= 1

Sehingga terdapat 1 *independent path* yang ada dalam *flowgraph* tersebut.. Adapun *independent path* dari *flowgraph* tersebut adalah sebagai berikut dimana nilai yang ditebalkan menandakan *edge*/garis baru.

Independent path ke-1 = 1-2

Dengan 1 *independent path* maka terdapat 1 *test case* yang akan dilakukan seperti yang ditunjukkan Tabel 4.13.

Tabel 4.13 Test Case SGD

No. Path	Path	Data Masukan	Keluaran yang	Pengamatan	Kesimpulan

			Diharapkan		
1	1-2	*x_train = [[0,0,1,0,0]]* yaitu sebuah data train dengan 1 token dimana token adalah vektor onehot berdimensi 5	Menghasilkan model dengan parameter bobot yang telah dioptimasi atau diubah	Menghasilkan model dengan parameter bobot yang telah dioptimasi atau diubah	[√] Terlewati [ ] Tidak Terlewati

4. Fungsi Train

Implementasi *pseudocode* dari kode *train* ditunjukan Gambar 4.12.

	Pseudocode:

	`train_with_sgd_minibatch(model, X_train, Y_train, learning_rate, max_epoch, evaluate_loss_after, minibatch_size)`
	`{`
	`// model = mengandung hyper_params dan params`
	`// X_train = kumpulan data train`
	`// Y_train = kumpulan data true label train`
	`// learning rate = learning rate`
	`// max_epoch = maksimal epoch`
	`// evaluate_loss_after = hitung error setiap epoch sekian`
	`// minibatch size = ukuran minibatch`

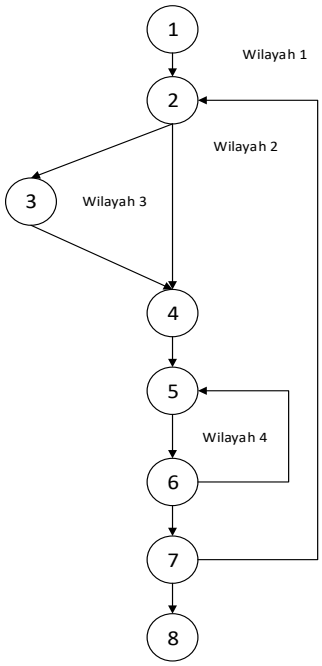
1	`number_of_train_data = number of data in x_train`
2	`for epoch = 1 to max_epoch`

3	`if (epoch mod evaluate_loss_after == 0)`
	`aspek_akurasi_cal = akurasi_per_aspek(model,X_train, Y_train)`
	`time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')`
	`print("%s: epoch=%d: akurasi:%f f1-score:%f " % (time,epoch, aspek_akurasi_cal['akurasi'],aspek_akurasi_cal['f1-score']))`
	`endif`

4	`random_index = np.random.randint(number_of_train_data, size=minibatch_size)`
5	`For i = 1 to minibatch_size`
6	`Model = sgd_step(model, x_train[random_index[i], y_train[i], learning_rate)` `endfor`
7	`endfor`
8	`return model`
	`}`

Gambar 4.12 Pseudocode Train

Selanjutnya *pseudocode* diubah menjadi flowgraph seperti yang ditunjukan Gambar 4.13.



Gambar 4.13 Flowgraph Train

Kemudian lakukan identifikasi *independent path*. Adapun jumlah *independent path* sama dengan *cyclomatic complexity*. Oleh karena itu terlebih dahulu hitung nilai *cyclomatic complexity*, perhitungan *cyclomatic complexity* dengan cara menghitung jumlah wilayah (*region*). Adapun berikut adalah proses perhitungannya dimana V(G) = *cyclomatic complexity*.

V(G) = (jumlah wilayah tertutup) + wilayah terbuka

= (wilayah 2 + wilayah 3 + wilayah 4) + wilayah 1

= 3 + 1

= 4

Sehingga terdapat 4 *independent path* yang ada dalam *flowgraph* tersebut.. Adapun *independent path* dari *flowgraph* tersebut adalah sebagai berikut dimana nilai yang ditebalkan menandakan *edge/*garis baru.

Independent path 1 : 1-2-4-5-6-7-8

Independent path 2 : 1-**2**-**3**-**4**-5-6-7-8

Independent path 3 : 1-2-4-5-6-**7-2**-4-5-6-7-8

Independent path 4 : 1-2-4-5-**6-5**-7-8

Dengan 4 *independent path* maka terdapat 4 *test case* yang akan dilakukan seperti yang ditunjukkan Tabel 4.14.

Tabel 4.14 Test Case Train

No. Path	Path	Data Masukan	Keluaran yang Diharapkan	Pengamatan	Kesimpulan
1	1-2-4-5-6-7-8	*x_train = [[0,0,1,0,0]] Minibatch size = 1 Max_epoch = 1 Evaluate_loss_after > 1*	Menghasilkan model dengan parameter bobot yang telah dioptimasi atau diubah	Menghasilkan model dengan parameter bobot yang telah dioptimasi atau diubah	[√] Terlewati [ ] Tidak Terlewati
2	1-**2-3-4**-5-6-7-8	*x_train = [[0,0,1,0,0]] Minibatch size = 1 Max_epoch = 1 Evaluate_loss_after = 1*	Menghasilkan nilai akurasi pada epoch ke-0 dan menghasilkan model dengan parameter bobot yang telah dioptimasi atau diubah	Menghasilkan nilai akurasi pada epoch ke-0 dan menghasilkan model dengan parameter bobot yang telah dioptimasi atau diubah	[√] Terlewati [ ] Tidak Terlewati
3	1-2-4-5-6-**7-2**-4-	*x_train = [[0,0,1,0,0]] Minibatch size = 1 Max_epoch = 2*	Menghasilkan model dengan parameter bobot yang	Menghasilkan model dengan parameter bobot yang telah	[√] Terlewati [ ] Tidak Terlewati

	5-6-7-8	*Evaluate_loss_after > 2*	telah dioptimasi atau diubah	dioptimasi atau diubah	
4	1-2-4-5-**6**-**5**-7-8	*x_train = [[0,0,1,0,0], [0,0,0,0,1]]* *Minibatch size = 2* *Max_epoch = 1* *Evaluate_loss_after > 1*	Menghasilkan model dengan parameter bobot yang telah dioptimasi atau diubah	Menghasilkan model dengan parameter bobot yang telah dioptimasi atau diubah	[√] Terlewati [ ] Tidak Terlewati

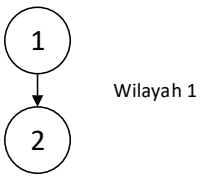
5. Fungsi Predict

Implementasi *pseudocode* dari kode *predict* ditunjukan Gambar 4.14.

	Pseudocode:
	`predict(model, x_data)`
	`{`
	`// model = mengandung params dan weight`
	`// x_data = satu data train`
1	`hyper_params = model['params']`
	`params = model['weight']`
	`output_state, hidden_state = forward_propagation(x_train, params, weight)`
2	`return np.argmax(output_state, axis=1)`
	`}`

Gambar 4.14 Pseudocode Predict

Selanjutnya *pseudocode* diubah menjadi flowgraph seperti yang ditunjukan Gambar 4.15.



Gambar 4.15 Flowgraph Predict

Kemudian lakukan identifikasi *independent path*. Adapun jumlah *independent path* sama dengan *cyclomatic complexity*. Oleh karena itu terlebih dahulu hitung nilai *cyclomatic complexity*, perhitungan *cyclomatic complexity*

dengan cara menghitung jumlah wilayah (*region*). Adapun berikut adalah proses perhitungannya dimana V(G) = *cyclomatic complexity*.

V(G) = (jumlah wilayah tertutup) + wilayah terbuka
= (0) + wilayah 1
= 0 + 1
= 1

Sehingga terdapat 1 *independent path* yang ada dalam *flowgraph* tersebut.. Adapun *independent path* dari *flowgraph* tersebut adalah sebagai berikut dimana nilai yang ditebalkan menandakan *edge/*garis baru.

Independent path ke-1 = 1-2

Dengan 1 *independent path* maka terdapat 1 *test case* yang akan dilakukan seperti yang ditunjukkan Tabel 4.15.

Tabel 4.15 Test Case Predict

No. Path	Path	Data Masukan	Keluaran yang Diharapkan	Pengamatan	Kesimpulan
1	1-2	*x_train = [[0,0,1,0,0]]* x_train yaitu sebuah data train dengan 1 token dimana token adalah vektor onehot berdimensi 5	Menghasilkan prediksi algoritma	Menghasilkan prediksi algoritma	[√] Terlewati [ ] Tidak Terlewati

4.2.2.3 Pengujian Parameter

Pengujian parameter dilakukan sesuai dengan skenario pengujian parameter terhadap data latih yang berjumlah 1584 kalimat dengan acuan

parameter dasar pada Tabel 4.8 untuk memilih parameter terbaik. Adapun hasil pengujian parameter dapat dilihat pada Tabel 4.16.

Tabel 4.16 Hasil Pengujian Parameter

Parameter	Nilai Parameter	Rata-Rata Akurasi	Epoch ke-
Learning rate	**0.05**	**95.92%**	100
	0.01	85.75%	100
	0.005	73.34%	100
	0.001	68.06%	100
	0.0005	64.55%	100
	0.0001	73.39%	100
	0.1	95.45%	100
	0.5	74.34%	100
Hidden dimensi	**32**	**97.60%**	100
	42	97.01%	100
	12	93.62%	100
	22	97.27%	100
	52	96.70%	100
Minibatch size	512	99.05%	100
	256	96.62%	100
	1024	**99.65%**	100
Bptt truncate	8	98.53%	100
	6	98.39%	100
	4	98.10%	100
	10	**99.22%**	100
	2	88.62%	100

Berdasarkan Tabel 4.16 didapat nilai parameter terbaik dari masing-masing parameter dan akan digunakan pada proses pengujian performa. Adapun nilai parameter terbaik ditujukan Tabel 4.17.

Tabel 4.17 Nilai Parameter Terbaik

No	Nama Parameter	Nilai
1	Learning rate	0.05
2	Jumlah *hidden state*	32
3	Ukuran minibatch	1024
4	Epoch	100
5	Bptt truncate	10

4.2.2.4 Pengujian Performa

1. Akurasi

Setelah mendapatkan parameter terbaik yang terdapat pada Tabel 4.17 maka akan dilakukan proses pengujian terhadap data uji yang berjumlah 422 kalimat. Hasil prediksi sentimen terhadap 4 aspek (FOOD, PRICE, SERVICE, AMBIENCE) pada setiap kalimat ditunjukan Tabel 4.18.

Tabel 4.18 Hasil Prediksi Data Uji

No	Kalimat	Aspek	Vektor Output	Label Prediksi	Label Sebenarnya	Kesimpulan
1	Dahulu pernah menjadi tempat favorit kumpul-kumpul anak muda dan keluarga.	FOOD	[4.098.. **9.999..** 7.847.. 4.976..]	Bukan sentimen	Bukan sentimen	[√]
		PRICE	[8.093.. **9.999..** 5.212.. 1.390..]	Bukan sentimen	Bukan sentimen	[√]
		SERVICE	[1.510.. 1.500.. **9.997..** 6.519..]	Positif	Bukan sentimen	[X]
		AMBIENCE	[6.875.. **8.493..** 1.439.. 6.730..]	Bukan sentimen	Positif	[X]
2	Baik secara	FOOD	[1.069.. 4.770.. **9.946..** 4.438..]	Positif	Positif	[√]

	rasa dan tempat yang nyaman untuk bersantai.	PRICE	[6.049.. **9.988..** 1.144.. 1.238..]	Bukan sentimen	Bukan sentimen	[√]
		SERVICE	[1.232.. **9.999..** 4.100.. 1.766..]	Bukan sentimen	Bukan sentimen	[√]
		AMBIENCE	[2.461.. 3.946.. **9.999..** 4.457..]	Positif	Positif	[√]
…	……	……..	……	……..	…….	……
422	Tetapi untuk piknik kesini rasanya setimpal iya.	FOOD	[8.220.. 1.114.. **9.993..** 6.840..]	Positif	Bukan sentimen	[X]
		PRICE	[9.032.. **9.998..** 1.151.. 1.982..]	Bukan sentimen	Bukan sentimen	[√]
		SERVICE	[9.402.. **9.999..** 8.568.. 6.220..]	Bukan sentimen	Bukan sentimen	[√]
		AMBIENCE	[3.828.. **9.999..** 1.850.. 1.316..]	Bukan sentimen	Bukan sentimen	[√]

Adapun jumlah prediksi benar setiap aspek dari data uji yang berjumlah 422 dapat dilihat pada Tabel 4.19.

Tabel 4.19 Jumlah Prediksi Aspek

Aspek	Jumlah Prediksi Benar
<FOOD>	306
<PRICE>	367
<SERVICE>	388
<AMBIENCE>	310

Sehingga untuk menghitung nilai akurasi per aspek menggunakan persamaan (2.16) sebagai berikut.

$$akurasi\ food = 100 \times \left(\frac{306}{422}\right) = 72.51\%$$

$$akurasi\ price = 100 \times \left(\frac{367}{422}\right) = 86.97\%$$

$$akurasi\ service = 100 \times \left(\frac{388}{422}\right) = 91.94\%$$

$$akurasi\ ambience = 100 \times \left(\frac{310}{422}\right) = 73.46\%$$

Untuk menghitung rata-rata akurasi yang didapat dari kesuluruhan token aspek adalah sebagai berikut.

$$akurasi\ rata - rata = \frac{72.51 + 86.97 + 91.94 + 73.46}{4} = 81.22\%$$

Sehingga didapat nilai akurasi rata-rata sebesar 81.22%.

2. F1 score

Pada penelitian ini menggunakan *weighted macro f1 score* dalam menghitung f1 score dikarenakan dataset yang digunakan mempunyai label yang tidak seimbang, yaitu token aspek yang mempunyai label "BUKAN SENTIMEN" lebih banyak . Sebelum menghitung f1 score maka dibuat *confusion matrix* untuk memudahkan dalam perhitungan f1 score. Confusion matrix dapat dilihat pada Tabel 4.20.

Tabel 4.20 Confusion Matrix

		Prediksi			
True Label		BUKAN SENTIMEN	NEGATIF	POSITIF	Total True Label
	BUKAN SENTIMEN	1202	45	153	1400
	NEGATIF	30	12	14	56
	POSITIF	71	4	157	232

Total Prediksi	1303	61	324	

Terdapat bagian biru pada Tabel 4.20 untuk menandakan *true positive* atau "jumlah prediksi benar kelas i". Sebelum menghitung f1 score , terlebih dahulu harus menghitung recall dengan persamaan (2.17) dan precision dengan persamaan (2.18) sehingga dapat menghitung nilai f1 score dengan menggunakan persamaan (2.19). Adapun perhitungan untuk kelas "BUKAN SENTIMEN" adalah sebagai berikut.

$$recall = \frac{1202}{1400} = 0.8586$$

$$precision = \frac{1202}{1282} = 0.9376$$

$$F_1\ score = \left(2 \times \frac{(0.9376 \times 0.8586)}{(0.9376 + 0.8586)}\right) = 0.8963$$

Berikut adalah ringkasan perhitungan f1 score terhadap seluruh kelas ditunjukan pada Tabel 4.21.

Tabel 4.21 f1 Seluruh Kelas

No	Nama Kelas	f1 Score	Jumlah Anggota Kelas	Prediksi		
				True Positive	True Negative	False Positive
1	BUKAN SENTIMEN	89%	1400	1202	198	101
2	NEGATIF	21%	56	12	44	49
3	POSITIF	56%	232	157	75	167

Untuk mendapatkan nilai *weighted* f_1 *score* secara keseluruhan maka digunakan Persamaan (2.20). Adapun berikut adalah proses perhitungannya.

$$averaged\ F_1\ score = \frac{1}{1688}.\big((1400 * 0.8963) + (56 * 0.2051) + (232 * 0.5647)\big) = \frac{1397.3159}{1688} = 0.8278$$

Sehingga nilai *weighted f1 score* secara keseluruhan adalah 82.78%.

4.2.3 Pembahasan Hasil Pengujian

Berdasarkan hasil pengujian performa menggunakan akurasi dan f1 score didapat rata-rata akurasi 81.22% dan rata-rata f1 score 82.78%.

1. Informasi antar kata banyak yang hilang dikarenakan ada proses *stopword removal* sehingga agak kesulitan dalam memprediksi.
2. Algoritma kesulitan dalam memprediksi label “NEGATIF” dimana keberhasilan algoritma memprediksi hanya 21 persen ini dikarenakan sering ditemukannya dalam kalimat token yang tidak terdapat pada kamus kata (UNKNOWN TOKEN) sebagai contoh kalimat “porsi di menu tidak ada explainnya” kata “explainnya” tidak terdapat pada kamus kata sehingga diganti dengan UNKNOWN TOKEN yang harusnya prediksi pada aspek <FOOD> “NEGATIF” menjadi “POSITIF”.
3. Sulit memprediksi label “NEGATIF” karena perbedaan jumlah label antara label “NEGATIF” dan “BUKAN SENTIMEN” terlalu besar.
4. Perlu adanya proses *aspect extraction* untuk mengetahui aspek yang terdapat pada kalimat.