BAB 4
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Sistem

Pada tahap ini dilakukan pembangunan perangkat lunak menggunakan bahasa pemrograman Python dengan mengacu pada perancangan yang telah dilakukan. Implementasi sistem terdiri dari implementasi perangkat keras, perangkat lunak dan antarmuka.

4.1.1 Implementasi Perangkat Keras

Spesifikasi dari perangkat keras yang digunakan ketika pembangunan perangkat lunak dapat dilihat pada Tabel 4.1.

Tabel 4.1 Spesifikasi Perangkat Keras

No	Perangkat Keras	Spesifikasi
1	*Processor*	Core i3
2	RAM	4 GB
3	*GPU*	Nvidia GeForce 920MX

4.1.2 Implementasi Perangkat Lunak

Spesifikasi dari perangkat lunak yang digunakan ketika pembangunan perangkat lunak dapat dilihat pada Tabel 4.2.

Tabel 4.2 Spesifikasi Perangkat Lunak

No	Perangkat Lunak	Spesifikasi
1	Sistem Operasi	Linux Mint 19 dan Windows 10
2	*Bahasa Pemrograman*	Python 3.6

4.1.3 Implementasi Antarmuka

Implementasi dari perancangan antarmuka menggunakan HTML dan CSS. Adapun penjelasan masing-masing antarmuka bisa dilihat pada Tabel 4.3.

Tabel 4.3 Implementasi Antarmuka

No	Nama Antarmuka	Deskripsi	Form
1	Halaman Data Masukan	Menampilkan laman untuk mengunggah data *training*	index.html

2	Halaman *Preprocessing Training*	Menampilkan laman untuk melihat hasil *preprocessing training*.	training/preprocessing.html
3	Halaman Pengaturan Parameter *Training*	Menampilkan form untuk mengatur parameter *training*	training/pengaturan_parameter.html
4	Halaman Proses *Training*	Menampilkan hasil proses *training* (informasi setiap epoch)	training/proses_training.html
5	Halaman *Preprocessing Testing*	Menampilkan laman untuk melihat hasil *preprocessing testing*.	testing/preprocessing.html
6	Halaman *Testing*	Menampilkan laman untuk melihat hasil *testing* dan nilai performa.	testing/testing.html

4.2 Pengujian Sistem

Terdapat dua pengujian yang dilakukan, yaitu pengujian fungsionalitas, pengujian parameter dan pengujian performa. Masing-masing penjelasan dan detail pengujian diterangkan sebagai berikut.

4.2.1 Rencana Pengujian

Berikut adalah rencana pengujian untuk pengujian fungsionalitas *white box*, pengujian fungsionalitas *black box*, pengujian parameter dan pengujian performa yang dilakukan penelitian ini.

4.2.1.1 Rencana Pengujian Fungsionalitas *White Box*

Pengujian fungsionalitas *white box* yang dilakukan menggunakan metode *basis path testing* terhadap komponen-komponen yang tidak menerima masukan dari pengguna melainkan dari proses sebelumnya. Adapun rencana pengujian fungsionalitas *white box* dapat dilihat pada Tabel 4.4.

Tabel 4.4 Rencana Pengujian Fungsionalitas *White Box*

No	Nama Proses	Poin Pengujian
1	*Training*	Fungsi *Forward Propagation*
		Fungsi BPTT
		Fungsi SGD
		Fungsi Train
2	*Testing*	Fungsi Predict
		Fungsi Test Predict

4.2.1.2 Rencana Pengujian Fungsionalitas *Black Box*

Pengujian fungsionalitas *black box* dilakukan terhadap komponen-komponen yang menerima masukan dari pengguna. Adapun rencana pengujian fungsionalitas *black box* dapat dilihat pada Tabel 4.5.

Tabel 4.5 Rencana Pengujian Fungsionalitas *Black Box*

No	Nama Proses	Poin Pengujian
1	*Training*	Upload File *Training*
		Menampilkan *Preprocessing Training*
2	*Testing*	Upload File *Testing*
		Menampilkan *Preprocessing Testing*

4.2.1.3 Rencana Pengujian Parameter

Pengujian parameter dilakukan untuk mendapatkan parameter terbaik karena performa sebuah model ditentukan oleh ketepatan dari parameter yang digunakan. Hasil satu pengujian parameter didefinisikan sebagai rata-rata dari hasil pengujian seluruh *validation set* (*development set*) dari 5-fold dataset. Adapun pengujian parameter akan dilakukan terdapat parameter-parameter yang disarankan oleh beberapa penelitian yang nilainya dapat dilihat pada Tabel 4.6. Sedangkan Tabel 4.7 menampilkan nilai parameter yang ingin diuji oleh peneliti disamping nilai parameter yang direkomendasikan oleh penelitian lain.

Tabel 4.6 Rencana Pengujian Parameter

No	Nama Parameter	Nilai	Referensi
1	Learning rate	{0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001}	[31]
2	Dimensi hidden	52	[32]
3	Minibatch size	32	[33]
4	Maks epoch	1000	[34]

Tabel 4.7 Rencana Pengujian Parameter Tambahan

No	Nama Parameter	Nilai
1	Learning rate	{0.5, 0.1}
2	Dimensi hidden	{40, 30, 20, 10}
3	Minibatch size	{64, 16, 8}

Pengujian parameter akan dilakukan skema sebagai berikut. Untuk melakukan pengujian parameter “learning rate” misalnya maka pengujian dilakukan menggunakan parameter dasar dan hanya nilai learning rate yang diubah dari

parameter dasar, menyesuaikan pengujian learning rate yang ingin dilakukan. Selain itu, proses *training* akan menghitung performa hanya saat epoch ke-900 karena menghitung akurasi di setiap epoch akan menguras waktu apalagi terdapat banyak parameter yang ingin diuji.

Tabel 4.8 Parameter Dasar

No	Nama Parameter	Nilai
1	Dimensi hidden	10
2	Bptt truncate	5
3	Freq. unknown token	0
4	Learning rate	0.05
5	Minibatch size	8
6	Jumlah Epoch	900

Alasan nilai-nilai parameter dasar dipilih demikian adalah sebagai berikut. Minibatch size, dimensi hidden dan bptt truncate menggunakan nilai untuk mempercepat proses *training*. Disamping menggunakan nilai parameter kecil untuk mempercepat proses *training*, algoritma tidak akan menggunakan fitur afiks sehingga fitur yang digunakan hanya token; hal ini untuk mempercepat *training* karena dengan hanya menggunakan fitur token akan memungkinkan optimasi pada kode implementasi. Hal yang perlu dicatat baik pengujian parameter dan pengujian performa keduanya menggunakan AdaGrad untuk lebih menstabilkan algoritma.

4.2.1.4 Rencana Pengujian Performa

Pengujian performa dilakukan untuk mendapatkan nilai akurasi dan F_1 score untuk menjawab masalah yang diangkat penelitian ini. Pengujian performa menggunakan parameter terpilih yang telah diseleksi pada pengujian parameter untuk dipasang pada algoritma ERNN. Hal yang perlu dicatat baik pengujian parameter dan pengujian performa keduanya menggunakan AdaGrad untuk menstabilkan algoritma.

4.2.1.5 Rencana Pengujian Waktu Komputasi

Adapun waktu komputasi akan diambil dari waktu komputasi *training* dan *testing* yang dilakukan pada tahap pengujian performa.

4.2.2 Pengujian

Berikut adalah penjabaran dari hasil pengujian yang telah dilakukan berdasarkan rencana pengujian yang telah dibentuk.

4.2.2.1 Pengujian Fungsionalitas *White box*

Pengujian pada subbab ini dilakukan menggunakan metode *white box testing* dengan *basis path testing* dimana prosesnya diawali pembentukan *pseudocode* kemudian *flowgraph* dilanjut identifikasi *independent path* dan terakhir dilakukan *test case*. Adapun berikut adalah pengujian *white box* dari masing-masing poin pengujian pada rencana pengujian.

1. Fungsi *Forward Propagation*

Adapun Gambar 4.1 adalah *pseudocode* dari implementasi kode *forward propagation*.

	Pseudocode:
	`forward_Propagation(x_train, hyper_params, params)`
	`{`
	`  // x_train = satu data train`
	`  // params = bobot`
	`  // hyper_params = hyper parameter`

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

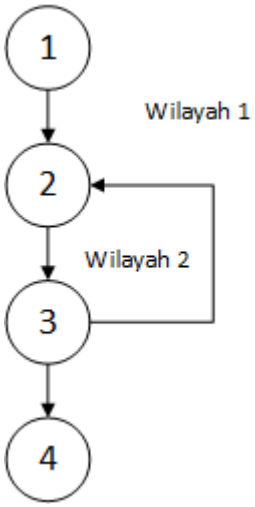
2	`  for t = 1 to T`

3	`  hidden_state[t] = tanh(U.dot(x_train[t]) + W.dot(hidden_state[t-1]))`
	`  output_state[t] = softmax(V.dot(hidden_state[t]))`
	`  endfor`

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

Gambar 4.1 *Pseudocode Forward Propagation*

Selanjutnya *pseudocode* diubah menjadi *flowgraph* seperti yang ditunjukan Gambar 4.2.



Gambar 4.2 *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*; untuk menghitungnya ada beberapa cara dan cara termudah adalah dengan 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. Dalam membentuk *independent path* syaratnya adalah sebuah *independent path* harus memiliki *edge*/garis baru yang tidak dimiliki *independent path* lain. Selain itu *subpath* tidak dihitung. 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 4.9.

Tabel 4.9 ***Test Case Forward Propagation***

No. Path	Path	Data Masukan	Keluaran yang Diharapkan	Pengamatan	Kesimpulan
1	1-2-3-4	*x_train = [[0,0,1,0,0]]*	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]]*	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)

Adapun Gambar 4.3 adalah *pseudocode* dari kode implementasi *backpropagation through time*.

	Pseudocode:

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

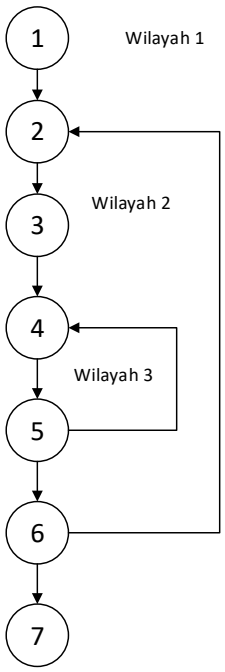
1	`  T = number of token in x_train`
	
	`  output_state, hidden_state = forward_propagation(x_train, hyper_params,`
	`  params)`
	
	`  U = params['U']`
	`  V = params['V']`
	`  W = params['W']`
	
	`  dLdU = np.zeros(U.shape)`
	`  dLdV = np.zeros(V.shape)`
	`  dLdW = np.zeros(W.shape)`
	
	`  delta_output_state = 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 = V.T.dot(delta_output_state[t]) * (1 – (hidden_state[t]^2)) start_iteration = t stop_ iteration = max(1, t-hyper_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 = W.T.dot(delta_t) * (1 – hidden_state[bptt_step – 1]^2) endfor
6	endfor
7	return dLdU, dLdV, dLdW
	}

Gambar 4.3 *Pseudocode BPTT*

Selanjutnya *pseudocode* diubah menjadi *flowgraph* seperti yang ditunjukan Gambar 4.4.



Gambar 4.4 Flowgraph BPTT

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

nilai *cyclomatic complexity*; untuk menghitungnya ada beberapa cara dan cara termudah adalah dengan 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*. Dalam membentuk *independent path* syaratnya adalah sebuah *independent path* harus memiliki *edge*/garis baru yang tidak dimiliki *independent path* lain. Selain itu *subpath* tidak dihitung. 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 4.10.

Tabel 4.10 *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]]*	Menghasilkan nilai turunan W_{ih}, W_{ho}, W_{hh}	Menghasilkan nilai turunan W_{ih}, W_{ho}, W_{hh}	[√] 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]]*	Menghasilkan nilai turunan W_{ih}, W_{ho}, W_{hh}	Menghasilkan nilai turunan W_{ih}, W_{ho}, W_{hh}	[√] Terlewati [ ] Tidak Terlewati

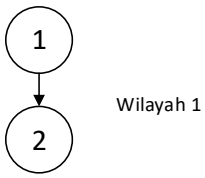
3. Fungsi *Stochastic Gradient Descent* (SGD)

Adapun Gambar 4.5 adalah *pseudocode* dari kode implementasi *stochastic gradient descent*.

	Pseudocode:
	`sgd_step(model, x_train, y_train, learning_rate)`
	`{`
	`// model = mengandung hyper_params dan params`
	`// x_train = satu data train`
	`// y_train = satu data true label train`
1	`dLdU, dLdV, dLdW = bptt(x_train, y_train, model['hyper_params'], model['params'])`
	`model['params']['U'] -= learning_rate * dLdU`
	`model['params']['V'] -= learning_rate * dLdV`
	`model['params']['W'] -= learning_rate * dLdW`
2	`Return model`
	`}`

Gambar 4.5 *Pseudocode SGD*

Selanjutnya *pseudocode* diubah menjadi *flowgraph* seperti yang ditunjukan Gambar 4.6.



Gambar 4.6 *Flowgraph SGD*

Kemudian lakukan identifikasi *independent path*. Adapun jumlah *independent path* sama dengan *cyclomatic complexity*. Oleh karena itu terlebih dahulu hitung nilai *cyclomatic complexity*; untuk menghitungnya ada beberapa cara dan cara termudah adalah dengan 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. Dalam membentuk *independent path* syaratnya adalah sebuah *independent path* harus memiliki *edge*/garis baru yang tidak dimiliki *independent path* lain. Selain itu

subpath tidak dihitung. 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 4.11.

Tabel 4.11 *Test Case SGD*

No. Path	Path	Data Masukan	Keluaran yang Diharapkan	Pengamatan	Kesimpulan
1	1-2	*x_train = [[0,0,1,0,0]]*	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*

Adapun Gambar 4.7 adalah *pseudocode* dari kode implementasi *train*.

	Pseudocode:
	`train_with_sgd_minibatch(model, X_train, Y_train, X_validation, Y_validation, X_test, Y_test, 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`
	`// X_validation = kumpulan data validation`
	`// Y_validation = kumpulan data true label validation`
	`// X_test = kumpulan data test`
	`// Y_test = kumpulan data true label test`
	`// learning rate = learning rate`
	`// max_epoch = maksimal epoch`
	`// evaluate_loss_after = hitung error setiap epoch sekian`
	`// minibatch size = ukuran minibatch`

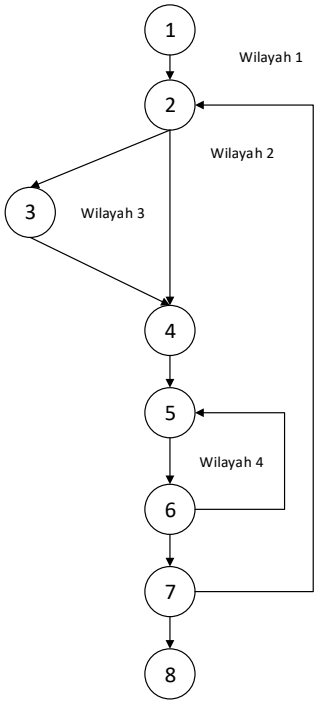
1	`number_of_train_data = number of data in x_train`
2	`for epoch = 1 to max_epoch`

3	`if (epoch mod evaluate_loss_after == 0)`
	`F1_score_val = calculate_f1_score(model, x_val, y_val)`
	`F1_score_test = calculate_f1_score(model, x_test, y_test)`
	`Akurasi_val = calculate_akurasi(model, x_val, y_val)`

	Akurasi_test = calculate_akurasi(model, x_test, y_test) Print("F1 score val = " + f1_score_val + " dan F1 score test = " + f1_score_test) Print("Akurasi val = " + akurasi_val + " dan akurasi test = " + akurasi_test) Print("\n") 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 4.7 *Pseudocode Train*

Selanjutnya *pseudocode* diubah menjadi *flowgraph* seperti yang ditunjukan Gambar 4.8.



Gambar 4.8 *Flowgraph Train*

Kemudian lakukan identifikasi *independent path*. Adapun jumlah *independent path* sama dengan *cyclomatic complexity*. Oleh karena itu terlebih dahulu hitung nilai *cyclomatic complexity*; untuk menghitungnya ada beberapa cara dan cara

termudah adalah dengan 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. Dalam membentuk *independent path* syaratnya adalah sebuah *independent path* harus memiliki *edge*/garis baru yang tidak dimiliki *independent path* lain. Selain itu *subpath* tidak dihitung. 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 4.12.

Tabel 4.12 *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 f1 score dan akurasi pada epoch ke-0 dan menghasilkan	Menghasilkan nilai f1 score dan akurasi pada epoch ke-0 dan menghasilkan	[√] Terlewati [ ] Tidak Terlewati

			model dengan parameter bobot yang telah dioptimasi atau diubah	model dengan parameter bobot yang telah dioptimasi atau diubah	
3	1-2-4-5-6-**7**-**2**-4-5-6-7-8	*x_train = [[0,0,1,0,0]] Minibatch size = 1 Max_epoch = 2 Evaluate_loss_after > 2*	Menghasilkan model dengan parameter bobot yang telah dioptimasi atau diubah	Menghasilkan model dengan parameter bobot yang telah dioptimasi atau diubah	[√] Terlewati [ ] Tidak Terlewati
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*

Adapun Gambar 4.9 adalah *pseudocode* dari kode implementasi *predict*.

	Pseudocode:

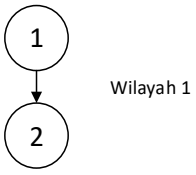
	`predict(model, x_data)`
	`{`
	`  // model = mengandung hyper_params dan params`
	`  // x_data = satu data train`

1	`  hyper_params = model['hyper_params']`
	`  params = model['params']`
	
	`  output_state, hidden_state = forward_propagation(x_train, hyper_params, params)`

2	`  return np.argmax(output_state, axis=1)`
	`}`

Gambar 4.9 *Pseudocode Predict*

Selanjutnya *pseudocode* diubah menjadi *flowgraph* seperti yang ditunjukan Gambar 4.10.



Gambar 4.10 *Flowgraph Predict*

Kemudian lakukan identifikasi *independent path*. Adapun jumlah *independent path* sama dengan *cyclomatic complexity*. Oleh karena itu terlebih dahulu hitung nilai *cyclomatic complexity*; untuk menghitungnya ada beberapa cara dan cara termudah adalah dengan 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. Dalam membentuk *independent path* syaratnya adalah sebuah *independent path* harus memiliki *edge*/garis baru yang tidak dimiliki *independent path* lain. Selain itu *subpath* tidak dihitung. 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 Predict*

No. Path	Path	Data Masukan	Keluaran yang Diharapkan	Pengamatan	Kesimpulan
1	1-2	*x_train = [[0,0,1,0,0]]*	Menghasilkan prediksi algoritma	Menghasilkan prediksi algoritma	[√] Terlewati [ ] Tidak Terlewati

6. Fungsi *Test Predict*

Adapun Gambar 4.11 adalah *pseudocode* dari kode implementasi *test predict*.

Pseudocode:

```
test_predict(model, X_test)
{
        // model = mengandung hyper_params dan params
```

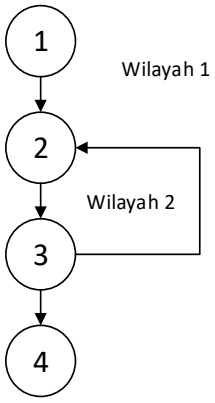
	`// X_test = kumpulan data testing`
1	`predictions = []`
2	`for x in x_test`
3	`predictions.append( Predict(model, x) )` `endfor`

4	`Y_true = [[y_true for y_pred, y_true in sent] for sent in Predictions]`
	`Y_pred = [[y_pred for y_pred, y_true in sent] for sent in Predictions]`
	`f1_score = calculate_f1_score(Y_true, Y_pred)`
	
	`return predictions, f1_score`

	`}`

Gambar 4.11 *Pseudocode Test Predict*

Selanjutnya *pseudocode* diubah menjadi *flowgraph* seperti yang ditunjukan Gambar 4.12.



Gambar 4.12 *Flowgraph Test Predict*

Kemudian lakukan identifikasi *independent path*. Adapun jumlah *independent path* sama dengan *cyclomatic complexity*. Oleh karena itu terlebih dahulu hitung nilai *cyclomatic complexity*; untuk menghitungnya ada beberapa cara dan cara termudah adalah dengan 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. Dalam membentuk *independent path* syaratnya adalah sebuah *independent path* harus memiliki *edge*/garis baru yang tidak dimiliki *independent path* lain. Selain itu

subpath tidak dihitung. 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 4.14.

Tabel 4.14 ***Test Case Test Predict***

No. Path	Path	Data Masukan	Keluaran yang Diharapkan	Pengamatan	Kesimpulan
1	1-2-3-4	*x_test = [[[0,0,0,01], [0,0,1,0,0]]]*	Menghasilkan prediksi algoritma dan f1 score untuk 1 data test	Menghasilkan prediksi algoritma dan f1 score untuk 1 data test	[√] Terlewati [ ] Tidak Terlewati
2	1-2-**3-2**-3-4	*x_test = [[[0,0,0,01], [0,0,1,0,0]], [[0,0,0,01], [0,0,1,0,0]],]*	Menghasilkan prediksi algoritma dan f1 score untuk 2 data test	Menghasilkan prediksi algoritma dan f1 score untuk 2 data test	[√] Terlewati [ ] Tidak Terlewati

4.2.2.2 Pengujian Fungsionalitas *Black box*

Pengujian pada subbab ini dilakukan menggunakan metode *black box*. Adapun berikut adalah pengujian *black box* dari masing-masing poin pengujian pada rencana pengujian.

Tabel 4.15 Pengujian Fungsionalitas *Black box*

Poin Pengujian	Data Masukan	Keluaran yang Diharapkan	Pengamatan	Kesimpulan
Upload File *Training*	Teks data *training*	Laman *preprocessing* menampilkan data *training*	Laman *preprocessing* menampilkan data *training*	[√] Diterima [ ] Ditolak

Menampilkan *Preprocessing Training*	Teks data *training* hasil upload	Laman *preprocessing* menampilkan hasil setiap *preprocessing*.	Laman *preprocessing* menampilkan hasil setiap *preprocessing*.	[√] Diterima [ ] Ditolak

Upload File *Testing*	Teks data *testing*	Laman *preprocessing* menampilkan data *testing*	Laman *preprocessing* menampilkan data *testing*	[√] Diterima [ ] Ditolak
Menampilkan *Preprocessing Testing*	Teks data *testing* hasil upload	Laman *preprocessing* menampilkan hasil setiap *preprocessing*.	Laman *preprocessing* menampilkan hasil setiap *preprocessing*.	[√] Diterima [ ] Ditolak

4.2.2.3 Pengujian Parameter

Sesuai rencana pengujian parameter, berbagai parameter dicoba sehingga didapatkan nilai parameter terbaik. Adapun hasil pengujian dapat dilihat pada Tabel 4.16.

Tabel 4.16 Hasil Pengujian Parameter

Validation Dataset	Parameter	Nilai Parameter	F1 Score	Epoch ke-
dev.01.tsv	learning rate	0.5	94.0597%	1000
		0.1	94.6620%	1000
		0.05	92.7758%	1000
		0.01	60.0560%	1000
		0.005	27.6722%	1000
		0.001	9.4064%	1000
		0.0005	8.6261%	1000
		0.0001	33.9829%	1000
	minibatch	64	95.6869%	1000
		32	95.1429%	1000
		16	94.3520%	1000
		8	92.7924%	1000
	frekuensi unknown token	7	91.3089%	1000
		6	91.5195%	1000
		5	91.8868%	1000
		3	92.6269%	1000
		2	92.7241%	1000
		1	92.6450%	1000
		0	92.9189%	1000
	bptt truncate	*unlimited*	91.6506%	1000

		10	92.7831%	1000
		5	92.7589%	1000
		3	92.8799%	1000
	hidden dimension	52	94.8403%	1000
		40	94.8043%	1000
		30	94.9133%	1000
		20	94.1989%	1000
		10	92.6157%	1000
dev.02.tsv	learning rate	0.5	94.8133%	1000
		0.1	94.8574%	1000
		0.05	92.8200%	1000
		0.01	59.1909%	1000
		0.005	29.0463%	1000
		0.001	8.6099%	1000
		0.0005	22.8847%	1000
		0.0001	34.8155%	1000
	minibatch	64	95.6529%	1000
		32	95.3662%	1000
		16	94.1620%	1000
		8	92.9233%	1000
	frekuensi unknown token	7	91.2240%	1000
		6	91.8202%	1000
		5	92.1233%	1000
		3	92.7113%	1000
		2	92.6501%	1000
		1	92.9740%	1000
		0	92.9926%	1000
	bptt truncate	*unlimited*	92.7527%	1000
		10	92.8760%	1000
		5	92.8925%	1000
		3	92.8796%	1000
	hidden dimension	52	95.2901%	1000
		40	95.1663%	1000
		30	95.0796%	1000
		20	94.3824%	1000
		10	92.6758%	1000
dev.03.tsv	learning rate	0.5	94.5197%	1000
		0.1	94.2103%	1000
		0.05	92.6277%	1000
		0.01	63.1647%	1000
		0.005	40.7527%	1000
		0.001	8.6912%	1000
		0.0005	9.2856%	1000
		0.0001	32.4092%	1000
	minibatch	64	95.2858%	1000
		32	94.8652%	1000
		16	94.0173%	1000
		8	92.5133%	1000

	frekuensi unknown token	7	90.9699%	1000
		6	91.5260%	1000
		5	91.7806%	1000
		3	91.9244%	1000
		2	92.5611%	1000
		1	92.2665%	1000
		0	92.6340%	1000
	bptt truncate	*unlimited*	92.3692%	1000
		10	92.5190%	1000
		5	92.3903%	1000
		3	92.3288%	1000
	hidden dimension	52	94.8426%	1000
		40	94.7174%	1000
		30	94.8266%	1000
		20	94.1622%	1000
		10	92.7024%	1000
dev.04.tsv	learning rate	0.5	94.8327%	1000
		0.1	94.5105%	1000
		0.05	93.1830%	1000
		0.01	56.3801%	1000
		0.005	35.1521%	1000
		0.001	8.7451%	1000
		0.0005	33.6221%	1000
		0.0001	28.4076%	1000
	minibatch	64	95.5634%	1000
		32	95.3083%	1000
		16	94.2008%	1000
		8	92.7295%	1000
	frekuensi unknown token	7	91.9163%	1000
		6	91.9081%	1000
		5	92.1166%	1000
		3	92.5226%	1000
		2	92.9553%	1000
		1	92.7608%	1000
		0	93.0590%	1000
	bptt truncate	*unlimited*	93.3136%	1000
		10	92.9589%	1000
		5	93.1673%	1000
		3	93.2429%	1000
	hidden dimension	52	95.0722%	1000
		40	95.1953%	1000
		30	94.5173%	1000
		20	94.2131%	1000
		10	93.2475%	1000
dev.05.tsv	learning rate	0.5	94.2606%	1000
		0.1	94.3468%	1000
		0.05	92.5381%	1000
		0.01	61.9686%	1000

		0.005	28.4765%	1000
		0.001	8.7297%	1000
		0.0005	44.2558%	1000
		0.0001	29.7765%	1000
	minibatch	64	95.2960%	1000
		32	95.3790%	1000
		16	94.2498%	1000
		8	92.9017%	1000
	frekuensi unknown token	7	91.4577%	1000
		6	91.0218%	1000
		5	91.7336%	1000
		3	92.5847%	1000
		2	92.5577%	1000
		1	92.9267%	1000
		0	92.4978%	1000
	bptt truncate	*unlimited*	92.1191%	1000
		10	92.6905%	1000
		5	92.8423%	1000
		3	92.7722%	1000
	hidden dimension	52	95.0548%	1000
		40	95.1605%	1000
		30	94.6915%	1000
		20	93.7808%	1000
		10	93.0558%	1000

Adapun Tabel 4.17 menunjukkan hasil pengujian parameter yang telah dirata-ratakan terhadap seluruh kelima fold dataset.

Tabel 4.17 Rata-Rata Hasil Pengujian Parameter

Parameter	Nilai Parameter	Rata-Rata F1 Score	Epoch ke-
learning rate	0.5	94.50%	1000
	0.1	**94.52%**	1000
	0.05	92.79%	1000
	0.01	60.15%	1000
	0.005	32.22%	1000
	0.001	8.84%	1000
	0.0005	23.73%	1000
	0.0001	31.88%	1000
minibatch	64	95.50%	1000
	32	**95.21%**	1000
	16	94.20%	1000
	8	92.77%	1000
frekuensi unknown token	7	91.37%	1000
	6	91.56%	1000
	5	91.93%	1000

	3	92.47%	1000
	2	92.69%	1000
	1	92.71%	1000
	0	**92.82%**	1000
bptt truncate	*unlimited*	92.44%	1000
	10	92.76%	1000
	5	92.81%	1000
	3	**92.82%**	1000
hidden dimension	52	**95.02%**	1000
	40	95.01%	1000
	30	94.80%	1000
	20	94.15%	1000
	10	92.86%	1000

Sehingga didapat kesimpulan parameter terbaik adalah seperti yang ditunjukkan Tabel 4.18. Nilai-nilai inilah yang akan digunakan pada tahap pengujian performa.

Tabel 4.18 Nilai Parameter Terbaik

No	Nama Parameter	Nilai
1	Learning rate	0.1
2	Jumlah hidden state	52
3	Ukuran minibatch	32
4	Frek. unknown token	0
5	Bptt truncate	3

4.2.2.4 Pengujian Performa

1. Akurasi

Dengan menggunakan nilai parameter pada Tabel 4.18 dilakukan proses *testing*. Adapun Tabel 4.20 menampilkan hasil prediksi terhadap seluruh token pada dataset test.01.tsv tanpa memisah-misah mereka berdasarkan kalimat. Tabel tersebut menampilkan jumlah hasil prediksi benar yang akan digunakan dalam menghitung nilai akurasi.

Berdasarkan Tabel 4.20, total prediksi benar adalah 51533 dari keseluruhan token 53625 sehingga bila dihitung nilai akurasi menggunakan Persamaan (2.14) seperti berikut

$$akurasi = 100 \times \left(\frac{51533}{53625}\right) = 96.10\%$$

maka didapat akurasi 96.10% untuk dataset test.01.tsv. Mengingat penelitian ini menggunakan 5-fold cross validation maka ada 5 dataset *testing*. Untuk mempersingkat Tabel 4.19 menunjukkan nilai akurasi dari masing-masing dataset.

Tabel 4.19 Akurasi Keseluruhan

Dataset	Akurasi	Epoch ke-
Test.01.tsv	96.10%	700
Test.02.tsv	95.95%	400
Test.03.tsv	96.27%	700
Test.04.tsv	95.93%	800
Test.05.tsv	96.10%	700
Rata-rata	**96.07%**	

Tabel 4.20 Hasil Prediksi Test.01.tsv

No.	Token	Vektor Output	Label Prediksi	Label Sebenarnya	Kesimpulan
1	<START>	$\begin{bmatrix} \mathbf{0.9997} \\ 0.0 \end{bmatrix}$	<START>	<START>	[√]

2	Pemerintah	$\begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0002 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ \mathbf{0.8354} \\ 0.0 \\ 0.1641 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0001 \\ 0.0 \end{bmatrix}$	NN	NNP	[X]
3	AS	$\begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ \mathbf{0.9999} \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \end{bmatrix}$	NNP	NNP	[√]
...	...	...	...	...	...
53625	...	...	...	...	...
				Total Benar	**51533**

Tabel 4.21 *Confusion Matrix* Test.01.Tsv

		Prediksi																							
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**...**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**Total True Label**
True Label	**0**	2007	0	0	0	0	0	0	0	0	0	0	0	0	**...**	0	0	0	0	0	0	0	0	0	2007
	1	0	1518	0	0	0	14	4	0	0	1	0	2	0	**...**	1	0	2	0	0	0	0	0	0	1542
	2	0	0	3616	0	0	1	0	3	0	8	0	3	4	**...**	1	0	0	2	0	1	0	0	0	3640
	3	0	0	0	71	0	0	0	0	0	0	0	0	0	**...**	0	0	4	0	0	0	0	0	0	75
	4	0	0	0	0	333	0	7	0	1	53	0	58	0	**...**	0	0	0	0	0	1	0	1	0	454
	5	0	1	0	0	0	4050	0	1	0	12	0	0	0	**...**	0	0	128	0	0	21	0	0	0	4213
	6	0	1	1	0	2	2	1839	1	0	58	0	27	8	**...**	6	0	2	0	0	18	0	1	0	1966
	7	0	0	1	0	0	0	6	1057	0	2	0	1	0	**...**	1	0	0	0	0	4	0	0	0	1072
	8	0	0	0	0	0	0	0	0	308	0	0	0	0	**...**	5	0	0	0	0	0	0	0	0	313
	9	0	1	4	0	8	9	27	3	0	11953	4	266	1	**...**	2	0	6	2	0	27	0	4	0	12324
	10	0	0	0	0	0	0	0	0	0	30	252	1	0	**...**	0	0	0	0	0	0	0	0	0	283
	11	0	10	15	0	56	10	38	1	0	611	3	6170	5	**...**	1	0	4	2	0	6	0	2	28	6963
	12	0	0	5	0	0	0	7	0	0	2	0	5	119	**...**	0	0	0	0	0	0	0	0	0	138
	...	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**...**
	19	0	0	0	0	0	0	1	0	0	1	0	0	0	**...**	0	0	0	0	3	0	0	0	0	5
	20	0	0	0	0	3	8	14	4	0	23	0	11	0	**...**	2	0	3	0	0	6320	0	0	0	6388
	21	0	0	0	0	0	0	0	0	0	1	0	0	0	**...**	0	0	3	0	0	1	50	1	0	56
	22	0	1	3	0	3	2	1	0	0	19	1	12	0	**...**	0	1	0	0	0	7	0	19	1	70
	23	0	0	0	0	0	0	0	0	0	0	0	14	0	**...**	0	0	0	0	0	0	0	0	5312	5326
Total Prediksi		2007	1537	3647	71	405	4272	1948	1073	312	12807	260	6577	137	**...**	1001	35	2669	442	5	6416	55	30	5341	

2. F1 Score

Terdapat beberapa tipe f1 score yang masing-masing terdapat perbedaan dalam perhitungannya. Penelitian ini menggunakan *weighted macro* F_1 *score* dengan alasan dataset yang tidak seimbang, yaitu banyak token berlabel noun seperti yang diungkap Kurniawan dan Aji [5]. Dalam menghitung f1 score bisa dibantu oleh *Confusion Matrix*. Adapun Tabel 4.21 menunjukkan *confusion matrix* untuk dataset test.01.tsv.

Bagian biru pada Tabel 4.21 menandai *true positive* atau "jumlah prediksi benar kelas X". Angka header 0-23 adalah nomor kelas yang terdapat pada Tabel 4.22. Sebagai contoh label 0 mengindikasikan kelas <START>. Setelah *confusion matrix* terbentuk maka perhitungan f1 score akan lebih mudah. Adapun berikut adalah proses perhitungan f1 score untuk kelas CC yang diawali dengan menghitung recall menggunakan Persamaan (2.16).

$$recall = \frac{1518}{1542} = 0.9844$$

Kemudian menghitung *precision* menggunakan Persamaan (2.17).

$$precision = \frac{1518}{1537} = 0.9876$$

Terakhir menghitung F_1 score menggunakan Persamaan (2.18).

$$F_1\ score = \left(2\ \times \frac{(0.9876 \times 0.9844)}{(0.9876 + 0.9844)}\right) = 0.9860$$

Sehingga f1 score untuk kelas CC sebesar 0.9860. Untuk meringkas maka nilai f1 score setiap masing-masing kelas dapat dilihat pada Tabel 4.22.

Tabel 4.22 F1 Score Seluruh Kelas Test.01.tsv

No	Nama Kelas	F1 Score	Jumlah Anggota Kelas	Prediksi		
				True Positive	True Negative	False Positive
0	<START>	100%	2007	2007	0	0
1	CC	98.60%	1542	1518	24	19
2	CD	99.24%	3640	3616	24	31
3	DT	97.26%	75	71	4	0
4	FW	77.53%	454	333	121	72
5	IN	95.46%	4213	4050	163	222
6	JJ	93.97%	1966	1839	127	109
7	MD	98.55%	1072	1057	15	16
8	NEG	98.56%	313	308	5	4
9	NN	95.12%	12324	11953	371	854
10	NND	92.82%	283	252	31	8

11	NNP	91.14%	6963	6170	793	407
12	OD	86.54%	138	119	19	18
13	PR	99.61%	1041	1036	5	3
14	PRP	99.51%	1540	1532	8	7
15	RB	96.65%	1029	981	48	20
16	RP	94.44%	37	34	3	1
17	SC	93.71%	2703	2517	186	152
18	SYM	98.87%	440	436	4	6
19	UH	60%	5	3	2	2
20	VB	98.72%	6388	6320	68	96
21	WH	90.09%	56	50	6	5
22	X	38%	70	19	51	11
23	Z	99.60%	5326	5312	14	29

Untuk mendapatkan nilai *weighted* F_1 *score* secara keseluruhan maka digunakan Persamaan (2.15). Adapun berikut adalah proses perhitungannya.

$$averaged\ F_1\ score\ =$$

$$\frac{1}{53625} \times \big((2007 \cdot 1) + (1542 \cdot 0.986034) + (3640 \cdot 0.992452) + (75 \cdot 0.972603) + (454 \cdot 0.77532) + (4213 \cdot 0.954626) + (1966 \cdot 0.939704) + (1072 \cdot 0.985548) + (313 \cdot 0.9856) + (12324 \cdot 0.951255) + (283 \cdot 0.928177) + (6963 \cdot 0.911374) + (138 \cdot 0.865455) + (1041 \cdot 0.996154) + (1540 \cdot 0.995128) + (1029 \cdot 0.966502) + (37 \cdot 0.944444) + (2703 \cdot 0.937081) + (440 \cdot 0.988662) + (5 \cdot 0.6) + (6388 \cdot 0.987192) + (56 \cdot 0.900901) + (70 \cdot 0.38) + (5326 \cdot 0.995969)\big)$$

$$= \frac{51508.17}{53625} = 96.05$$

Sehingga didapat f1 score 96.05% untuk dataset test.01.tsv. Mengingat penelitian ini menggunakan 5-fold cross validation maka ada 5 dataset *testing*. Tabel 4.23 menunjukkan nilai f1 score dari masing-masing dataset.

Tabel 4.23 F1 Score Keseluruhan

Dataset	F1 Score	Epoch ke-
Test.01.tsv	96.05%	700
Test.02.tsv	95.91%	400
Test.03.tsv	96.25%	700
Test.04.tsv	95.89%	800
Test.05.tsv	96.06%	700

Rata-rata	**96.03%**	

4.2.2.5 Pengujian Waktu Komputasi

Adapun Tabel 4.24 dan Tabel 4.25 menunjukkan waktu komputasi baik *training* maupun *testing* yang didapatkan dari tahapan pengujian performa yang sudah dilakukan sebelumnya.

Tabel 4.24 Pengujian Waktu Komputasi *Training*

No	Dataset	Dimensi Vektor Input	Jumlah Data (Kalimat)	Waktu *Training*
1	Train.01.tsv	11748	7223	4 jam 22 menit 47 detik
2	Train.02.tsv	11805	7223	4 jam 22 menit 53 detik
3	Train.03.tsv	11842	7223	4 jam 23 menit 54 detik
4	Train.04.tsv	11909	7223	4 jam 16 menit 6 detik
5	Train.05.tsv	11860	7223	4 jam 22 menit 46 detik

Tabel 4.25 Pengujian Waktu Komputasi *Testing*

No	Dataset	Dimensi Vektor Input	Jumlah Data (Kalimat)	Waktu *Testing*
1	Test.01.tsv	11748	2007	7.58 detik
2	Test.02.tsv	11805	2007	8.50 detik
3	Test.03.tsv	11842	2007	11.22 detik
4	Test.04.tsv	11909	2007	8.95 detik
5	Test.05.tsv	11860	2007	8.23 detik

4.2.3 Analisis Hasil Pengujian

Berdasarkan hasil pengujian yang ditampilkan Tabel 4.19 dan Tabel 4.23 didapat rata-rata akurasi sebesar 96.07 persen dan rata-rata f1 score sebesar 96.03 persen. Jika membandingkan hasil tersebut dengan hasil Kurniawan dan Aji [5], mereka unggul ~1%. Dengan melihat tabel yang disajikan penelitian mereka yang dapat dilihat pada Tabel 4.26, perbedaan ~1% ini disebabkan fitur character embedding.

Tabel 4.26 Percobaan Kurniawan dan Aji Terhadap *Validation Set* [5]

Algoritma	Fitur	F1 Score
Bi-LSTM+CRF		96.06%
Bi-LSTM+CRF	Char embed	97.42%
Bi-LSTM+CRF	Char embed + prefiks	97.50%
Bi-LSTM+CRF	Char embed + prefiks + sufiks	97.60%

Untuk melakukan analisis hasil pengujian yang telah dilakukan penelitian ini maka bisa melihat tabel f1 score pada Tabel 4.27 yang menunjukkan nilai rata-rata f1 score setiap kelas pada seluruh data *testing*.

Tabel 4.27 Rata-Rata F1 Score Seluruh Kelas

No	Nama Kelas	F1 Score
0	<START>	100.0%
1	CC	98.5%
2	CD	99.2%
3	DT	98.0%
4	FW	78.5%
5	IN	95.3%
6	JJ	94.0%
7	MD	98.6%
8	NEG	98.7%
9	NN	95.2%
10	NND	92.8%
11	NNP	91.1%
12	OD	86.4%
13	PR	99.5%
14	PRP	99.6%
15	RB	96.2%
16	RP	98.6%
17	SC	93.1%
18	SYM	98.9%
19	UH	70.5%
20	VB	98.9%
21	WH	87.4%
22	X	46.0%
23	Z	99.7%

Berdasarkan f1 score, algoritma cukup kesulitan untuk memprediksi token berlabel FW (*foreign word*) dimana keberhasilan hanya di bawah 80 persen. Hal ini disebabkan tidak ada fitur yang bisa memberitahu algoritma apakah sebuah token adalah kata asing atau bukan dan tak jarang kata asing tergolong ke dalam *unknown token*. Membandingkan hasil dengan Kurniawan dan Aji [5], penelitian mereka cukup berhasil memprediksi FW di atas 80 persen. Kemungkinan penyebab keberhasilan mereka adalah fitur afiks yang digunakan, yang mana berbeda dengan yang digunakan penelitian ini. Mereka menggunakan 2-3 karakter awal dan 2-3

karakter akhir sebagai prefiks dan sufiks, sedangkan penelitian ini menggunakan prefiks dan sufiks baku dari KBBI V Daring. Dengan menggunakan gaya afiks tidak baku, algoritma agaknya akan bisa mengidentifikasi kata asing sebab kata asing—tepatnya bahasa Inggris—umumnya memiliki struktur huruf yang berbeda dengan bahasa Indonesia; misalnya kata asing "single" memiliki 3 karakter akhir "gle" dimana tidak ada satu kata pun dalam bahasa Indonesia yang berakhiran "gle" jika melihat kamus KBBI V Daring. Selain itu character embedding yang mereka gunakan juga membantu untuk mengenali pola huruf kata asing.

Lalu algoritma kesulitan memprediksi token berlabel UH yang merupakan kata-kata tidak baku seperti "nih", "Insya Allah", "ya". Mungkin penyebab memprediksi label UH menjadi sulit adalah karena kemunculan token ini hanya sedikit, yaitu di bawah 20 kemunculan di setiap data *training*.

Algoritma kesulitan memprediksi token berlabel X (*unknown*) dan WH yang dialami pula oleh penelitian Kurniawan dan Aji [5] dimana mereka berpendapat memprediksi label X menjadi sulit karena berupa kata *typo, slang,* singkatan, istilah asing dan jumlah kemunculan label ini sedikit pada dataset. Dan beberapa token berlabel X umumnya berlabel NN atau NNP sehingga membuat bingung algoritma. Beberapa kata asing dan singkatan pun ditandai sebagai label X dan membuat algoritma kesulitan memprediksi label token tersebut karena biasanya token-token tersebut berlabel NN atau NNP seperti yang diungkap Kurniawan dan Aji. Kemudian Kurniawan dan Aji [5] berpendapat label WH menjadi sulit diprediksi karena algoritma kebingungan memprediksi token berlabel WH seperti "apa" dan "siapa" yang bisa juga menjadi berlabel SC (*subordinate clause*) dan pemberian tag oleh Dinarakamani et. al [12] terhadap token-token ini dianggap tidak konsisten oleh Kurniawan dan Aji [5] sehingga menyebabkan algoritma gagal membedakan token berlabel WH dan SC.