# BAB 4

# IMPLEMENTASI DAN PENGUJIAN

## 4.1 Implementasi Sistem

Tahap implementasi sistem ini dilakukan penerapan berdasarkan analisis ke dalam bahasa pemrograman PHP dan *Python*. Adapun implementasi sistem terdiri dari implementasi perangkat keras, implementasi perangkat lunak, dan implementasi antarmuka

## 4.2 Implementasi Perangkat Keras

Perangkat keras yang digunakan dalam pembangunan sistem *Coreference Resolution* Bahasa Indonesia pada kata ganti kepemilikan menggunakan *Recurrent neural network* dengan pertimbangan ketersediaan dan kemampuannya yang dapat mengakses program *Coreference Resolution* dengan spesifikasi pada Tabel 4. 1. Spesifikasi Perangkat Keras.

**Tabel 4. 1. Spesifikasi Perangkat Keras**

| No. | Perangkat Keras | Spesifikasi |
|---|---|---|
| 1 | Processor | Intel Core i5-2410M CPU 2.30GHz |
| 2 | Memori (RAM) | 4.00 GB |
| 3 | Hard disk | 500 GB |

### 4.2.1 Implementasi Perangkat Lunak

Spesifikasi perangkat lunak yang digunakan dalam implementasi sistem dengan spesifikasi minimal dapat dilihat pada Tabel 4. 2 Spesifikasi Perangkat Lunak.

**Tabel 4. 2 Spesifikasi Perangkat Lunak**

| No. | Perangkat lunak | spesifikasi |
|---|---|---|
| 1 | Sistem Operasi | Windows 8.1 Pro |
| 2 | Text Editor (HTML, PHP, *Python*) | Sublime Text 3, Jupyter Notebook |
| 3 | Package Tool (*python*) | Anaconda |
| 3 | *Web Server* | XAMPP |
| 4 | *Web Browser* | Google Chrome |

### 4.2.2 Implementasi Antarmuka

Implementasi antarmuka dilakukan untuk setiap tampilan aplikasi yang dibangun. Penjelasan implementasi antarmuka pada aplikasi *Coreference Resolution* dapat dilihat pada Tabel 4. 3 Implementasi Antarmuka.

**Tabel 4. 3 Implementasi Antarmuka**

| No. | Nama Antarmuka | Deskripsi | Nama File |
|---|---|---|---|
| 1 | *Input* data latih | Menampilkan fitur unggah data latih | *input*_data_latih.html |
| 2 | *Input* data uji | Menampilkan fitur unggah data uji | *input*_data_uji.html |
| 3 | proses | Menampilkan *drop box* untuk memilih data latih dan data uji yang tersedia serta memasukkan nilai parameter | index.html |
| 4 | Hasil uji | Menampilkan hasil proses dari data uji | hasil_1.html |

## 4.3 Pengujian Sistem

Pengujian sistem dilakukan untuk memasikan bahwa setiap sistem yang dibuat sesuai dengan perancangan sistem yang telah dilakukan sebelumnya. Pada tahap pengujian sistem terdiri dari pengujiann fungsionalitas dan pengujian akurasi.

### 4.3.1 Skenarion Pengujian

Pada penelitian ini dilakukan beberapa tahapan pengujian yang akan menjelaskan tentang pengujian fungsionalitas *white box*, *black box*, dan pengujian parameter dan akurasi.

#### 4.3.1.1 Skenario Pengujian *White Box*

Pengujian fungsionalitas *white box* dilakukan untuk mengecek implementasi setiap fungsi dari metode RNN. Pengujian *white box* ini akan menguji setiap source code fungsi yang ada pada metode RNN dan diubah menjadi bentuk flowgraph dengan identifikasi menggunakan independent path dan terakhir menggunakan test case. Adapun fungsi yang akan diuji pada metode RNN seperti pada Tabel 4. 5 Skenario Pengujian *Black Box*Tabel 4. 4 Skenario Pengujian *White Box*.

**Tabel 4. 4 Skenario Pengujian *White Box***

| No. | Tahapan | Fungsi yang diuji |
|---|---|---|
| 1 | Training | *Forward Propagation* |
| | | Backward Propagation |
| | | SGD |
| | | Train |
| 2 | Testing | Predict |

### 4.3.1.2 Skenario Pengujian *Black Box*

Pengujian *Black Box* berfokus pada persyaratan fungsional dari sebuah sistem yang dibangun dan menemukan kesalahan program. Pengujian *Black Box* yang digunakan pada aplikasi ini adalah menggunakan teknik sample testing dimana pengujian bertujuan untuk memeriksa apakah sistem dapat berjalan sesuai dengan harapan pengguna[17]. Berikut skenario pengujian menggunakan *black box* pada Tabel 4. 5 Skenario Pengujian *Black Box*.

**Tabel 4. 5 Skenario Pengujian *Black Box***

| No. | Komponen Uji | Item Uji | Jenis Uji |
|---|---|---|---|
| 1 | Unggah Data | Training.tsv | *Black box* |
| | | Testing.tsv | *Black box* |
| | | Coref.ipynb | *Black box* |
| 2 | *Input* parameter minibatch dan epoch | Bilangan bulat positif : 64, 32, 1000, 500 | *Black box* |
| | | Bilangan desimal : 0.1, 0.5, 0.0001 | *Black box* |
| | | Bilangan bulat negatif : -1, -0.1 | *Black box* |
| 3 | *Input* parameter learning rate | Bilangan bulat positif : 64, 32, 1000, 500 | *Black box* |
| | | Bilangan desimal : 0.1, 0.5, 0.0001 | *Black box* |
| | | Bilangan bulat negatif : -1, -0.1 | *Black box* |

### 4.3.1.3 Skenario Pengujian Parameter

Pengujian parameter dilakukan untuk mengetahui nilai akurasi terbaik dari setiap parameter yang akan diuji dengan nilai setiap parameter yang berbeda-beda. Berikut adalah skenario pengujian parameter yang akan diuji pada Tabel 4. 6 Skenario Pengujian Parameter.

**Tabel 4. 6 Skenario Pengujian Parameter**

| Pengujian ke-i | Parameter | | |
|---|---|---|---|
| | n(epoch) | Minibatch size | Learning rate |
| 1 | 1000 | 64 | 0.001 |
| 2 | 1000 | 64 | 0.0001 |
| 3 | 1000 | 32 | 0.001 |
| 4 | 1000 | 32 | 0.0001 |

Dari tabel tersebut, berikut adalah alasan mengapa menggunakan nilai tersebut untuk pengujian parameter.

1. Nilai epoch sebanyak 1000 dengan alasan untuk mengetahui pada epoch ke berapa agar didapatkan nilai akurasi tertinggi yang dihasilkan dalam satu kali pengujian
2. Nilai minibatch hanya dikisaran 64 hingga 32 karena ketika menggunakan minibacth yang lebih kecil dari 32 atau lebih besar dari 64, hanya menghasilkan perbedaan akurasi sedikit saja pada saat di awal iterasi.
3. Nilai learning rate yang diberikan tidak bisa lebih besar dari $10^{-3}$ seperti $10^{-2}$ dan seterusnya karena akan menghasilkan error yang besar di epoch awal.

### 4.3.2 Hasil Pengujian *White Box*

Berikut adalah hasil pengujian beberapa fungsi pada metode RNN dengan menggunakan pengujian fungsionalitas *white box*.

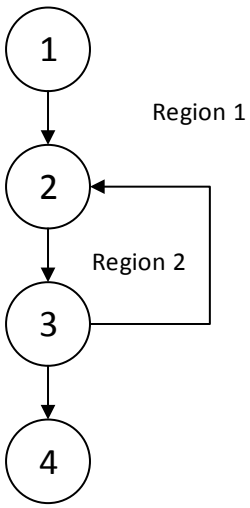
## 1. Fungsi *Forward Propagation*

Berikut adalah source code dari fungsi *Forward Propagation* pada Gambar 4. 1. Source Code *Forward Propagation*.

| | |
|---|---|
| | def numba_forward_propagation(x, hyper_params, params): |
| 1 | # The total number of time steps<br>T = len(x) |
| 2 | s = np.zeros((T + 1, hyper_params['*hidden*_dim']), dtype=np.float64)<br>s[-1] = np.zeros(hyper_params['*hidden*_dim'])<br>o = np.zeros((T, hyper_params['*output* _dim'])) |
| 3 | for t in np.arange(T): |
| | s[t] = np.tanh(params['U'].dot(x[t]) + params['W'].dot(s[t-1]))<br>o[t] = numba_sigmoid(params['V'].dot(s[t]))<br>#endfor |
| 4 | return [o, s] |

**Gambar 4. 1. Source Code *Forward Propagation***

Selanjutnya source code dari *Forward Propagation* akan diubah menjadi flowgraph seperti pada Gambar 4. 2 Flow Graph *Forward Propagation*.




**Gambar 4. 2 Flow Graph *Forward Propagation***

Selanjutnya melakukan identifikasi dengan independent path yang mana jumlahnya sama dengan *cyclomatic complexity* yaitu dengan menghitung jumlah wilayah/region atau bisa saja dengan mengurangi jumlah edge/garis dengan node/simpul kemudian dijumlahkan dengan 2. Sehingga digunakanlah rumus perhitungan berupa selisih edge dan node. Dan berikut adalah menggunakan rumus menghitung jumlah region tertutup dan region terbuka.

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

= (1)+1

= 2

Dari hasil tersebut dapat disimpulkan bahwa terdapat 2 independent path yang ada pada flowgraph *Forward Propagation* baik dengan meggunakan rumus penjumlahan region atau selisih antara node dengan edge. Independent path diharuskan memiliki edge atau garis baru yang tidak dimiliki path lainnya, sehingga nilai yang didapat dari 2 independent path adalah sebagai berikut.

Independent path(1) = 1-2-3-4

Independent path(2) = 1-2-3-2-3-4

Setelah ditentukan nilai masing-masing independent path, maka dilakukan tes kasus yang akan diuji sesuai independent path pada Tabel 4. 7 Tes Kasus Dari *Forward Propagation*.

**Tabel 4. 7 Tes Kasus Dari *Forward Propagation***

| No. | Path | Data Masukan | Keluaran yang diharapkan | Pengamatan | kesimpulan |
|---|---|---|---|---|---|
| 1 | 1-2-3-4 | x = [0,1,1,0,0.32,0,0]<br><br>Nilai x berupa1 token hasil ekstraksi fitur berdimensi 7 | Menghasilkan keluaran 1 vektor dan 1 *hidden state* | Menghasilkan keluaran 1 vektor dan 1 *hidden state* | [√] Diterima<br>[ ] Ditolak |
| 2 | 1-2-3-2-3-4 | x = [[0,1,1,0,0.32,0,0], [0,1,1,0,0.42,0.5,1]]<br><br>Nilai x berupa 2 atau lebih token hasil ekstraksi fitur berdimensi 7 | Menghasilkan keluaran 2 atau lebih vektor dan 2 atau lebih *hidden state* sebanyak nilai T (banyaknya data x>1) | Menghasilkan keluaran 2 atau lebih vektor dan 2 atau lebih *hidden state* sebanyak nilai T (banyaknya data x>1) | [√] Diterima<br>[ ] Ditolak |

**2. Fungsi Backward Propagation(BPTT)**

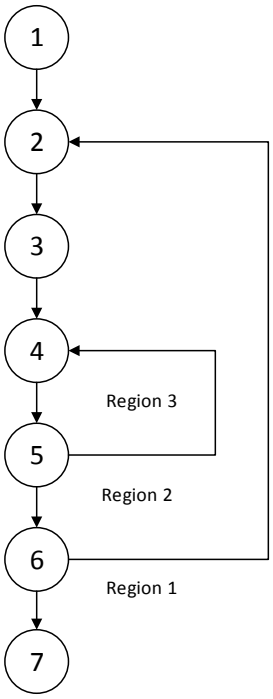
Berikut adalah source code dari fungsi *backward propagation* pada Gambar 4. 3 Source Code Backward Propagation.

```
    def numba_bptt(x, y, hyper_params, params):
1       T = len(y)
        o, s = numba_forward_propagation(x, hyper_params, params)
        delta_o = o
        delta_o[np.arange(len(y)), y] -= 1.

        dLdU = np.zeros(params['U'].shape)
        dLdV = np.zeros(params['V'].shape)
        dLdW = np.zeros(params['W'].shape)
2       for t in np.arange(T)[::-1]:
3           dLdV += np.outer(delta_o[t], s[t].T)
            # Initial delta calculation
            delta_t = params['V'].T.dot(delta_o[t]) * (1 - (s[t] ** 2))
4                   for bptt_step in np.arange(max(0, t-
hyper_params['bptt_truncate']),
                    t+1)[::-1]:
5                           dLdW += np.outer(delta_t, s[bptt_step-1])
                            dLdU += np.outer(delta_t, x[bptt_step])
                            # Update delta for next step
                            delta_t = params['W'].T.dot(delta_t) * (1 - s[bptt_step-1] **
2)
                    #endfor
6               #endfor
7               return dLdU, dLdV, dLdW
```

**Gambar 4. 3 Source Code Backward Propagation**

Selanjutnya source code dari *backward propagation* akan diubah menjadi flowgraph seperti pada Gambar 4. 4 Flow Graph Backward Propagation.




**Gambar 4. 4 Flow Graph Backward Propagation**

Selanjutnya melakukan identifikasi dengan independent path yang mana jumlahnya sama dengan *cyclomatic complexity* yaitu dengan menghitung jumlah selisih edge dan node. Sehingga digunakanlah rumus perhitungan sebagai berikut.

V(G) = Edge – Node + 2

= 8 – 7 + 2

= 3

Dari hasil tersebut dapat disimpulkan bahwa terdapat 3 independent path yang ada pada flowgraph backward propagation. Independent path diharuskan memiliki edge atau garis baru yang tidak dimiliki path lainnya, sehingga nilai yang didapat dari 2 independent path adalah sebagai berikut.

Independent path(1) = 1-2-3-4-5-6-7

Independent path(2) = 1-2-3-4-5-4-5-6-7

Independent path(3) = 1-2-3-4-5-4-5-6-2-3-4-5-6-7

Setelah ditentukan nilai masing-masing independent path, maka dilakukan tes kasus yang akan diuji sesuai independent path pada Tabel 4. 8 Tes Kasus Dari Backward Propagation.

**Tabel 4. 8 Tes Kasus Dari Backward Propagation**

| No. | Path | Data Masukan | Keluaran yang diharapkan | Pengamatan | kesimpulan |
|---|---|---|---|---|---|
| 1 | 1-2-3-4-5-6-7 | x = [0,1,1,0,0.32,0,0]<br>T = 1<br><br>Nilai x berupa1 token hasil ekstraksi fitur berdimensi 7 | Menghasilkan nilai turunan bobot U, V, dan W | Menghasilkan nilai turunan bobot U, V, dan W | [√] Diterima<br>[ ] Ditolak |
| 2 | 1-2-3-4-5-4-5-6-7 | x = [[0,1,1,0,0.32,0,0], [0,1,1,0,0.42,0.5,1]]<br>T=1<br>t > 1<br><br>Nilai x berupa 2 atau lebih token hasil ekstraksi fitur berdimensi 7 | Menghasilkan nilai turunan bobot U V dan W | Menghasilkan nilai turunan bobot U V dan W | [√] Diterima<br>[ ] Ditolak |
| 2 | 1-2-3-4-5-4-5-6-2-3-4-5-6-7 | x = [[0,1,1,0,0.32,0,0], [0,1,1,0,0.42,0.5,1]]<br><br>T > 1<br>t > 1<br><br>Nilai x berupa 2 atau lebih token hasil ekstraksi fitur berdimensi 7 | Menghasilkan nilai turunan bobot U V dan W | Menghasilkan nilai turunan bobot U V dan W | [√] Diterima<br>[ ] Ditolak |

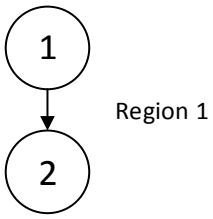
### 3. Fungsi *Stochastic Gradient Descent* (SGD)

Berikut adalah source code dari fungsi Stochastic Gradient Descent(SGD).

```
    def sgd_step(model, x, y, learning_rate):
1       # Calculate the gradients
            dLdU, dLdV, dLdW = numba_bptt(x, y, 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 Source Code SGD**

Selanjutnya source code dari SGD akan diubah menjadi flowgraph seperti pada Gambar 4. 6 Flow Graph SGD.




**Gambar 4. 6 Flow Graph SGD**

Selanjutnya melakukan identifikasi dengan independent path yang mana jumlahnya sama dengan *cyclomatic complexity* yaitu dengan menghitung selisih edge dan node. Sehingga digunakanlah rumus perhitungan sebagai berikut.

V(G) = Edge – Node + 2

= 1-2+2

= 1

Dari hasil tersebut dapat disimpulkan bahwa terdapat 1 independent path. Independent path diharuskan memiliki edge atau garis baru yang tidak dimiliki path lainnya, sehingga nilai yang didapat dari 1 independent path adalah sebagai berikut.

Independent path(1) = 1-2

Setelah ditentukan nilai masing-masing independent path, maka dilakukan tes kasus yang akan diuji sesuai independent path pada Tabel 4. 9 Tes Kasus Dari SGD.

**Tabel 4. 9 Tes Kasus Dari SGD**

| No. | Path | Data Masukan | Keluaran yang diharapkan | Pengamatan | kesimpulan |
|---|---|---|---|---|---|
| 1 | 1-2 | Model parameter<br>x = [0,1,1,0,0.32,0,0]<br>y = [1,0,0,1,0.45,0.5,0]<br>learning rate = 0.001<br><br>Nilai x berupa 1 token hasil ekstraksi fitur berdimensi 7 | Menghasilkan model dengan nilai bobot yang telah dioptimasi sesuai parameternya | Menghasilkan model dengan nilai bobot yang telah dioptimasi sesuai parameternya | [√] Diterima<br>[ ] Ditolak |

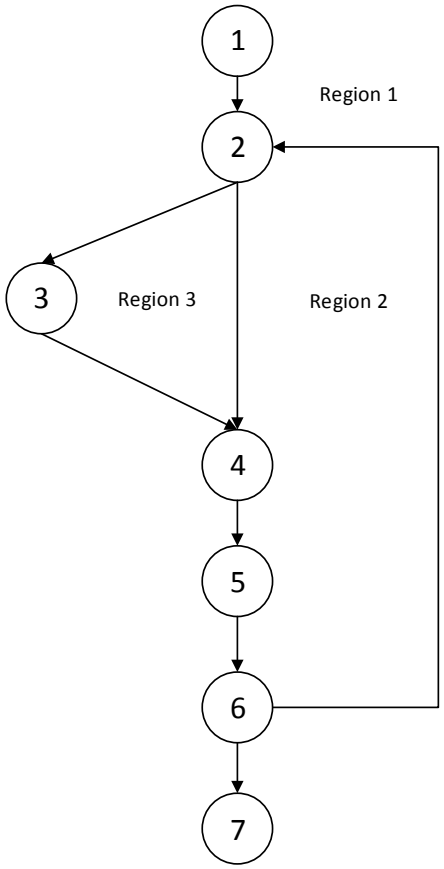
**4. Fungsi Train**

Berikut adalah source code dari fungsi train pada Gambar 4. 7 Source Code Fungsi Train.

| | |
|---|---|
| | def train_with_sgd_minibatch(<br>model,<br>X_train,<br>y_train,<br>X_test,<br>y_test,<br>learning_rate=0.005,<br>nepoch=50,<br>evaluate_loss_after=1,<br>minibatch_size=32,<br>skip_loss_after=0): |
| 1 | losses =<br>num_examples_seen = 0 |
| 2 | for epoch in range(nepoch): |
| 3 | if (epoch % evaluate_loss_after == 0) and (epoch >=<br>skip_loss_after):<br>loss_without_mean, loss = calculate_loss(model, X_train,<br>y_train)<br>akurasi_train = calculate_akurasi(model, X_train, y_train)<br>akurasi_test = calculate_akurasi(model, X_test, y_test)<br>losses.append((num_examples_seen, loss))<br>time=datetime.now().strftime('%Y-%m%d%H:%M:%S')<br>print("%s: *Error* pada epoch=%d: %f (akurasi train: %f -<br>akurasi test: %f)" % (time, epoch, loss, akurasi_train,<br>akurasi_test))<br>sys.stdout.flush() |
| 4 | #endif |
| 5 | model = sgd_step(model, X_train, y_train, learning_rate) |
| 6 | #endfor |
| 7 | return model |

**Gambar 4. 7 Source Code Fungsi Train**

Selanjutnya source code dari SGD akan diubah menjadi flowgraph seperti pada Gambar 4. 8 Flow Graph Fungsi Train.




**Gambar 4. 8 Flow Graph Fungsi Train**

Selanjutnya melakukan identifikasi dengan independent path yang mana jumlahnya sama dengan *cyclomatic complexity* yaitu dengan menghitung selisih edge dan node. Sehingga digunakanlah rumus perhitungan sebagai berikut.

V(G) = 8 – 7 + 2

= 1+2

= 3

Dari hasil tersebut dapat disimpulkan bahwa terdapat 1 independent path. Independent path diharuskan memiliki edge atau garis baru yang tidak dimiliki path lainnya, sehingga nilai yang didapat dari 1 independent path adalah sebagai berikut.

Independent path(1) = 1-2-4-5-6-7

Independent path(2) = 1-2-3-4-5-6-7-8

Independent path(3) = 1-2-4-5-6-2-3-4-5-6-7

Setelah ditentukan nilai masing-masing independent path, maka dilakukan tes kasus yang akan diuji sesuai independent path pada Gambar 4. 10 Flow Graph Fungsi Predict.

**Tabel 4. 10 Tes Kasus Dari Train**

| No. | Path | Data Masukan | Keluaran yang diharapkan | Pengamatan | kesimpulan |
|---|---|---|---|---|---|
| 1 | 1-2-4-5-6-7 | x = [0,1,1,0,0.32,0,0]<br>*Minibatch size = 1*<br>*Max_epoch = 1*<br>*Evaluate_loss_after > 1*<br><br>Nilai x berupa token hasil ekstraksi fitur berdimensi 7 dan nilai epoch% evaluate loss after != 0 | Menghasilkan model dengan nilai bobot yang telah dioptimasi sesuai parameternya | Menghasilkan model dengan nilai bobot yang telah dioptimasi sesuai parameternya | [√] Diterima<br>[ ] Ditolak |
| 2 | 1-2-3-4-5-6-7-8 | x = [0,1,1,0,0.32,0,0]<br>*Minibatch size = 1*<br>*Max_epoch = 1*<br>*Evaluate_loss_after = 1*<br>Nilai x berupa token hasil ekstraksi fitur berdimensi 7 dan nilai epoch% evaluate loss after = 0 dan epoch = 1 | Menghasilkan nilai akurasi pada epoch ke-0 dan menghasilkan model dengan nilai bobot yang telah dioptimasi sesuai parameternya | Menghasilkan nilai akurasi pada epoch ke-0 dan menghasilkan model dengan nilai bobot yang telah dioptimasi sesuai parameternya | [√] Diterima<br>[ ] Ditolak |
| 3 | 1-2-4-5-6-2-3-4-5-6-7 | x = [0,1,1,0,0.32,0,0], [1,0,0,1,0.45,0]<br>*Minibatch size = 1*<br>*Max_epoch = 2*<br>*Evaluate_loss_after = 1*<br>Nilai x berupa token hasil ekstraksi fitur berdimensi 7 dan nilai epoch% evaluate loss after = 0 dan epoch > 1 | Menghasilkan nilai akurasi pada epoch ke-n dan menghasilkan model dengan nilai bobot yang telah dioptimasi sesuai parameternya | Menghasilkan nilai akurasi pada epoch ke-n dan menghasilkan model dengan nilai bobot yang telah dioptimasi sesuai parameternya | [√] Diterima<br>[ ] Ditolak |

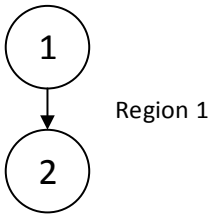
**5. Fungsi Predict**

Berikut adalah source code dari fungsi predict pada Gambar 4. 9 Source Code Fungsi Predict.

| | |
|---|---|
| | def predict(model, x):<br># Perform *Forward Propagation* and return index of the highest score |
| 1 | o, s = numba_forward_propagation(x, model['hyper_params'], model['params']) |
| 2 | return np.argmax(o, axis=1) |

**Gambar 4. 9 Source Code Fungsi Predict**

Selanjutnya source code dari SGD akan diubah menjadi flowgraph seperti Gambar 4. 10 Flow Graph Fungsi Predict.




**Gambar 4. 10 Flow Graph Fungsi Predict**

Selanjutnya melakukan identifikasi dengan independent path yang mana jumlahnya sama dengan *cyclomatic complexity* yaitu dengan menghitung selisih edge dan node. Sehingga digunakanlah rumus perhitungan sebagai berikut.

V(G) = Edge – Node + 2

= 1-2+2

= 1

Dari hasil tersebut dapat disimpulkan bahwa terdapat 1 independent path. Independent path diharuskan memiliki edge atau garis baru yang tidak dimiliki path lainnya, sehingga nilai yang didapat dari 1 independent path adalah sebagai berikut.

Independent path(1) = 1-2

Setelah ditentukan nilai masing-masing independent path, maka dilakukan tes kasus yang akan diuji sesuai independent path seperti pada Tabel 4. 11 Tes Kasus Dari Predict.

**Tabel 4. 11 Tes Kasus Dari Predict**

| No. | Path | Data Masukan | Keluaran yang diharapkan | Pengamatan | kesimpulan |
|---|---|---|---|---|---|
| 1 | 1-2 | x = [0,1,1,0,0.32,0,0] Nilai x berupa token hasil ekstraksi fitur berdimensi 7 | Menghasilkan Prediksi algoritma | Menghasilkan prediksi algoritma | [√] Diterima [ ] Ditolak |

### 4.3.3 Hasil Pengujian *Black Box*

Berikut adalah hasil pengujian dari fungsionalitas *Black Box* seperti pada Tabel 4. 12 Pengujian Fungsionalitas *Black Box*.

**Tabel 4. 12 Pengujian Fungsionalitas *Black Box***

| **Unggah Data** | | | | |
|---|---|---|---|---|
| **No.** | **Data Masukan** | **Hasil yang diharapkan** | **Pengamatan** | **Kesimpulan** |
| 1 | Training.tsv | Menampilkan modal alert berupa upload file training berhasil | Menampilkan modal alert berupa unggah file berhasil dan list file data training bertambah | [√] Diterima [ ] Ditolak |
| 2 | Testing.tsv | Menampilkan modal alert berupa upload file testing berhasil | Menampilkan modal alert berupa unggah file berhasil dan list file data testing bertambah | [√] Diterima [ ] Ditolak |
| 3 | Corpus.ipynb | Menampilkan modal alert berupa upload file testing harus berformat .tsv | Menampilkan modal alert berupa upload file testing harus berformat .tsv | [√] Diterima [ ] Ditolak |
| ***Input* Parameter Minibatch dan Epoch** | | | | |
| **No.** | **Data Masukan** | **Hasil yang diharapkan** | **Pengamatan** | **Kesimpulan** |
| 1 | Bilangan bulat positif : 64, 32, 1000, 500 | Sistem menerima *input*an tersebut | Sistem menerima *input*an | [√] Diterima [ ] Ditolak |

| | | | | |
|---|---|---|---|---|
| | | | tersebut dan bisa diproses | |
| 2 | Bilangan desimal : 0.1, 0.5, 0.0001 | Sistem tidak merespon *input*an dari keyboard selain angka | Sistem tidak merespon *input*an dari keyboard selain angka | [√] Diterima [ ] Ditolak |
| 3 | Bilangan bulat negatif : -1, -2, -10 | Sistem tidak merespon *input*an dari keyboard selain angka | Sistem tidak merespon *input*an dari keyboard selain angka | [√] Diterima [ ] Ditolak |
| ***Input* Parameter Learning Rate** | | | | |
| **No.** | **Data Masukan** | **Hasil yang diharapkan** | **Pengamatan** | **Kesimpulan** |
| 1 | Bilangan bulat positif : 64, 32, 1000, 500 | Sistem menerima *input*an tersebut | Sistem menerima *input*an tersebut dan bisa diproses | [√] Diterima [ ] Ditolak |
| 2 | Bilangan desimal : 0.1, 0.5, 0.0001 | Sistem menerima *input*an tersebut | Sistem menerima *input*an tersebut dan bisa diproses | [√] Diterima [ ] Ditolak |
| 3 | Bilangan bulat negatif : -1, -2, -10 | Sistem menolak nilai yang lebih kecil dari 0 | Sistem menolak nilai yang lebih kecil dari 0 | [√] Diterima [ ] Ditolak |

## 4.4 Pengujian Akurasi dan Parameter

Pengujian akurasi pada penelitian *Coreference Resolution* ini dilakukan untuk mengetahui nilai akurasi untuk dapat menentukan apakah metode *Recurrent neural network* cocok digunakan untuk *Coreference Resolution* pada Dataset Indonesian Tagged Manually Corpus. Jumlah data yang digunakan adalah 1 dataset training.tsv yang memiliki jumlah baris datanya sebanyak 545 baris yang digunakan sebagai data latih karena semakin banyak data yang menjadi data latih maka akan mempengaruhi kenaikan nilai akurasinya. Jumlah pasangan *coreference* dari dataset training.tsv tersebut adalah 4851 pasangan. Sedangkan untuk data uji akan digunakan 1 dataset testing.tsv yang memiliki jumlah baris datanya sebanyak 220 baris yang memiliki jumlah pasangan *coreference* sebanyak 1225 pasangan.

**Tabel 4. 13 Data Klasifikasi *Coreference***

| Dataset | Jumlah baris kata | Pasangan *coreference* |
|---|---|---|
| Training.tsv | 545 | 4851 |
| Testing.tsv | 220 | 1225 |

Dari data training dan data testing tersebut selanjutnya dilakukan tahap pengklasifikasian. Pada tahap pengklasifikasian *coreference resolution* menggunakan metode *Recurrent neural network*, digunakan tiga parameter utama pada saat pengujian yaitu epoch, minibatch size, dan learning rate. dalam pengujian ketiga parameter tersebut, dilakukan beberapa kali perubahan nilai parameter untuk mengetahui pengaruhnya terhadap nilai akurasi yang menunjukan kedekatan antara nilai prediksi atau model dengan nilai aktual. Dari nilai akurasi tersebut dapat disimpulkan seberapa besar nilai akurasi yang didapat dari ketiga parameter tersebut dan pada pengujian ke berapa didapatkan akurasi terbesar.

Perhitungan pengujian nilai akurasi pada data testing yang dihasilkan menggunakan metode *Recurrent neural network* adalah sebagai berikut.

$$akurasi = \frac{Jumlah\ Fitur\ Benar}{Jumlah\ Seluruh\ Fitur}\ X\ 100\%$$

$$= \frac{1036}{1225}\ X\ 100\% = 84{,}57\%$$

Berikut adalah data beberapa percobaan pengujian data testing dengan mengubah-ubah nilai parameter yang ada beserta hasil pengujiannya.

**Tabel 4. 14 Pengujian Parameter dan Akurasi Pada Data Testing**

| Pengujian ke-i | Parameter | | | Hasil Pengujian | | | | |
|---|---|---|---|---|---|---|---|---|
| | n(epoch) | Minibatch size | Learning rate | Epoch ke-n | Nilai *Error* | Jumlah Prediksi Benar | Jumlah Prediksi Salah | Nilai Akurasi |
| 1 | 1000 | 64 | 0.001 | 134 | 0.1683 | 1036 | 189 | 0.845714 (84.57%) |
| 2 | 1000 | 64 | 0.0001 | 310 | 0.1450 | 1025 | 200 | 0.836734 (83.67%) |
| 3 | 1000 | 32 | 0.001 | 170 | 0.1698 | 1032 | 193 | 0.842448 (84.24%) |
| 4 | 1000 | 32 | 0.0001 | 645 | 0.1416 | 1026 | 199 | 0.837551 (83.76%) |

Berikut adalah perhitungan untuk akurasi keseluruhan pada klasifikasi dataset indonesian tagged manually corpus dengan metode RNN.

$$Rata - rata\ akurasi = \frac{Total\ akurasi\ setiap\ pengujian}{Jumlah\ pengujian}$$

$$= \frac{84.57\% + 83.67\% + 84.24\% + 83.76\%}{4} = 84.06\%$$

## 4.5 Pembahasan Pengujian

Hasil pengujian merupakan proses dimana sampel data masukan berhasil melewati proses *preprocessing* (tokenisasi kalimat, tokenisasi kata, ekstraksi fitur) dan pengklasifikasian menggunakan metode *Recurrent neural network.* pengujian yang dilakukan pada penelitian ini adalah pengujian *white box*, *black box*, akurasi dan parameter.

Hasil pengujian *white box* menyatakan bahwa semua fungsi algoritma RNN berhasil dilewati sesuai harapan, dan untuk pengujian *black box* menyatakan bahwa semua sistem yang ada pada program berhasil dilewati sesuai harapan. Hasil pengujian parameter dan akurasi menyatakan dengan jumlah pasangan kata *coreference* pada dataset testing.tsv sebanyak 1225 pasangan menggunakan *Recurrent neural network* menghasilkan nilai akurasi terbesar yaitu 84.57% dengan rata-rata akurasi 84.06%. Berikut ini adalah penjelasan penyebab nilai akurasi *coreference* resolution yang dihasilkan.

1. Belum adanya penggunaan *Named Entity Recognition*(NER) dikarenakan kata ganti yang berimbuhan –NYA tidak memiliki pasangan *coreference* yang berlabel selain NNP, sedangkan NER itu merupakan entitas nama yang biasanya memiliki tag NN atau entitas nama yang sudah dikategorikan seperti nama tempat, organisasi, grup, dan lain-lain[22].
2. Sistem lebih banyak menghasilkan kesimpulan bernilai benar pada prediksi pasangan yang labelnya bukan *coreference*, dan hanya sedikit yang bernilai benar pada prediksi pasangan berlabel *coreference*. Hal ini dikarenakan kemungkinan datanya imbalance yaitu data yang bukan *coreference* lebih banyak sehingga diperlukan penyeimbang data training agar algoritma lebih memahami pola pasangan yang termasuk kelas *coreference*.


UNIVERSITAS KOMPUTER INDONESIA
UNIKOM