BAB 5
IMPLEMENTASI

Pada bab ini akan dijelaskan tentang proses implementasi program. Implementasi terdiri dari penjelasan tentang spesifikasi program, batasan-batasan pada proses implementasi, implementasi algoritme, dan implementasi antarmuka. Proses implementasi berdaarkan hasil dari proses perancangan yang telah dibuat.

5.1 Spesifikasi Sistem

Sub-bab ini akan menjelaskan spesifikasi sistem yang digunakan saat proses implementasi. Spesifikasi yang dibahas pada sub-bab ini ada 2, yaitu spesifikasi perangkat keras dan spesifikasi perangkat lunak.

5.1.1 Spesifikasi Perangkat Keras

Spesifikasi perangkat keras yang digunakan untuk membuat sistem penentuan durasi nyala lampu lalu lintas adalah sebagai berikut:

1. Prosesor Intel(R) Core(TM) i5-4210U CPU @ 1.70GHz – 2.40GHz
2. RAM 8GB
3. ROM 1TB
4. Nvidia GeForce 820M 2GB

5.1.2 Spesifikasi Perangkat Lunak

Spesifikasi perangkat lunak yang digunakan untuk membuat sistem penentuan durasi nyala lampu lalu lintas adalah sebagai berikut:

1. Windows 10
2. Microsoft Office Excel 2016
3. IDE Netbeans 8
4. Bahasa Pemrograman Java

5.2 Batasan Implementasi

Pada sub-bab ini dijelaskan batasan-batasan pada implementasi sistem penentuan durasi nyala lampu lalu lintas, antara lain adalah sebagai berikut:

1. Sistem penentuan durasi nyala lampu lalu lintas yang dibangun hanya sebatas program console
2. Data yang diproses pada sistem disimpan pada file dengan format .csv.
3. Keluaran sistem berupa durasi nyala lampu lalu lintas berdasarkan panjang antrian kendaraan yang ditentukan menggunkan metode *backpropagation*.

4. Durasi nyala lampu hasil dari keluaran sistem akan di bandingkan dengan durasi sebenarnya menggunakan simulasi untuk melihat mana yang lebih baik.

5.3 Implementasi Algoritme

Pada sub-bab ini akan dijelaskan mengenai hasil implementasi dari perancangan yang sudah dijelaskan pada bab sebelumnya ke dalam sebuah program. Proses penentuan durasi nyala lampu lalu lintas ini memiliki beberapa proses utama, diantaranya proses pelatihan, setelah proses pelatihan selesai maka proses menentukan durasi nyala lampu lalu lintas. Didalam proses pelatihan juga terdapat proses utama, yaitu proses normalisasi data, proses *feedforward*, proses *backpropagation*, dan proses *weightupdating*. Sedangkan dalam proses penentuan durasi nyala lampu lalu lintas hanya terdapat satu proses tambahan, aitu proses denormalisasi data.

5.3.1 Implementasi Pelatihan

Proses pertama yang perlu dilakukan setelah jaringan dibangun adalah pelatihan. Proses pelatihan berfungsi untuk melatih jaringan untuk mengupdate bobotnya. Dalam proses pelatihan terdapat beberapa sub-proses didalamnya seperti proses *feeforward*, *backpropagation*, dan *weight updating*. Implementasi dari proses pelatihan dapat dilihat pada Kode Program 5.1.

```
1  public void training() {
2
3      if (isBuild()) {
4          //Inisialisasi bobot awal (random)
5          for (int i=1;i<=penghubungLookUp.size();i++){
6              double random =(0.1*Math.random()+0.01);
7              penghubungLookUp.get(i).setWeight(
8                      (0.15 * Math.random() + 0.051));
9          }
10         while (this.iterasi != 0) {
11             this.RMSE = 0;
12             this.sumD = 0;
13             for (int i = 0; i < this.dataLatihAntrian
14                     .length; i++) {
15                 this.dKuadrat = 0;
16                 feedForward(this.dataLatihAntrian[i]
17                         .nilai, this.dataLatihLampu[i]
18                                 .nilai);
19                 backpropagation(this.dataLatihLampu[i]
20                         .nilai);
```

```
21                  if(i < this.dataLatihAntrian.length - 1){
22                      weightUpdate(this.dataLatihAntrian
23                              [i + 1].nilai);
24                  }
25
26                  this.sumD += this.dKuadrat;
27              }
28              this.RMSE = Math.sqrt(this.sumD /
29                      this.dataLatihAntrian.length);
30              allRMSE.add(RMSE);
31              this.iterasi--;
32          }
33      } else {
34          build();
35      }
36  }
```

Kode Program 5.1 Proses Pelatihan

Berikut ini penjelasan implementasi proses pelatihan:

1. Baris 1: inisialisasi method pelatihan
2. Baris 3: DI cek, apakah jaringan sudah dibuat atau belum. Jika sudah maka akan lanjut ke baris berikutnya.
3. Baris 5 – 9: Dilakukan inisialisasi semua bobt secara random
4. Baris 10: Masuk ke iterasi yang jumlah nya sebanyak nilai yang sudah di tentukan sebelumnya
5. Baris 11 – 12: Set nilai variabel-variabel yang dibutuhkan
6. Baris 13 – 14: *Looping* proses pada setiap iterasi untuk memproses semua baris data
7. Baris 15: Set nilai variabel yang dibutuhkan
8. Baris 16 – 18: Proses *Feedforward*
9. Baris 19 – 20: Proses *Backpropagation*
10. Baris 21 – 24: Proses *Weight Updating*
11. Baris 26: Hitung nilai *sumD*, yaitu selisih dari *output* asli dengan *output* sistem;
12. Baris 27: Selesai looping dan ulangi prose ke-6 sebanyak data yang akan di proses

13. Baris 28 – 29: Hitung nilai RMSE pada iterasi tersebut
14. Baris 30 : Simpan hasil perhitungan RMSE
15. Baris 31 – 32: Masuk ke iterasi berikutnya, ulangi langkah 4
16. Baris 33 – 35: Di cek, jika jaringan belum dibuat, maka harus dibuat terlebih dahulu
17. Baris ke 36: Selesai

5.3.2 Implementasi Normalisasi Data

Sebelum diproses lebih lanjut menggunakan algoritme *backpropagation*, data yang tersedia harus dinormalisasi terlebih dahulu. Ini bertujuan untuk menyamakan *range* nilai semua data sehingga tidak ada salah satu data yang dominan. Hasil dari normalisasi data akan berada pada range 0 sampai dengan 1. Implementasi proses normalisasi data dapat dilihat pada Kode Program 5.2.

```
1  private double normalisasi(int nilai, int max, int min) {
2      double n = 0;
3      n = 0.8 * (nilai-min)/(max-min);
4      return n;
5  }
```

Kode Program 5.2 Proses Normalisasi Data

Berikut ini penjelasan implementasi proses normalisasi data:

1. Baris 2: Inisialisasi variabel lokal *n*
2. Baris 3: Hitung hasil dari normalisasi data menggunakan rumus yang sudah ditentukan dengan nilai berdasarkan parameter. Kemudian simpan hasil pada variabel *n*
3. Baris 4: Kembalikan nilai *n*.

5.3.3 Implementasi *Feedforward*

Proses *feedforward* merupakan proses pertama dari 3 proses utama algoritme *backpropagation*. *Feedforward* merupakan proses untuk menghitung keluaran jaringan berdasarkan masukan yang diterima. Implementasi dari proses *feedforward* dapat dilihat pada Kode Program 5.3.

```
1  private void feedForward(int[] nilaiInput,
2              int[] nilaiOutputAsli) {
3      double temp = 0;
4      double[] y = new double[4];
5      double[] yAksen = new double[4];
6      if (nilaiInput.length == this.input.length) {
```

```
7           for (int i = 0; i < nilaiInput.length; i++) {
8               this.input[i].setY(normalisasi(
9                       nilaiInput[i],
10                      this.input[i].getMax(),
11                      this.input[i].getMin()));
12          }
13      } else {
14          return;
15      }
16      for (int i = 0; i < this.hidden.length; i++) {
17          this.hidden[i].setY(getY(this.hidden[i]));
18      }
19      for (int i = 0; i < this.output.length; i++) {
20          this.output[i].setY(getY(this.output[i]));
21          yAksen[i] = this.output[i].getY();
22      }
23      for (int i = 0; i < nilaiOutputAsli.length; i++) {
24          y[i] = normalisasi(nilaiOutputAsli[i],
25                  this.output[i].getMax(),
26                  this.output[i].getMin());
27          this.d[i] = y[i] - yAksen[i];
28      }
29      for (int i = 0; i < d.lenght; i++) {
30          temp += this.d[i];
31      }
32      this.dKuadrat += (Math.pow(temp / 4, 2));
33  }
```

Kode Program 5.3 Proses *Feedforward*

Berikut ini adalah penjelasan implementasi proses *feedforward*:

1. Baris 1 – 2: Inisialisasi method *feedforward* dengan 2 parameter yaitu nilai *input* dan nilai *output* asli
2. Baris 3 – 5: Set nilai variabel-variabel yang dibutuhkan
3. Baris 6 – 15: Normalisasi semua data *input*
4. Baris 16 – 18: Hitung *output* pada *hidden layer*
5. Baris 19 – 22: Hitung *output* pada *output layer*, dengan menyimpan nilai *output* tersebut pada variabel *yAksen*

6. Baris 23 – 28: Hitung niali *d*, yaitu niali selisih dari *output* sistem dengan *output* asli
7. Baris 29 – 31: Simpan semua nilai yang ada di *d* pada variabel *temp*
8. Baris 32: Hitung nilai pada variabel *dKuadrat*. Yaitu nilai dari *temp* dibagi dengan jumlah node pada *output layer* kemudian dipangkat 2
9. Baris 33: Selesai

5.3.4 Implementasi *Backpropagation*

Proses *backpropagation* merupakan proses kedua dari 3 proses utama algoritme *backpropagation*. *Backpropagation* merupakan proses untuk menghitung nilai eror berdasarkan keluaran jaringan dari proses *feedforward*. Penghitungan nilai eror dilakukan dengan cara mundur dari *output layer* ke *hidden layer*. Implementasi dari proses *backpropagation* dapat dilihat pada Kode Program 5.4.

```
1  private void backpropagation(int[] nilaiOutputAsli) {
2      for (int i = 0; i < this.output.length; i++) {
3          double yAksen = this.output[i].getY();
4          double y = normalisasi(nilaiOutputAsli[i],
5                  this.output[i].getMax(),
6                  this.output[i].getMin());
7          this.output[i].setError(
8                  yAksen * (1 - yAksen) * (y - yAksen));
9      }
10     for (Node hidden1 : this.hidden) {
11         int counter = 0;
12         double temp = 0;
13         double yAksen = hidden1.getY();
14         for (int j=1;j<= penghubungLookUp.size();j++){
15             if (penghubungLookUp.get(j).getNodeAsal()
16                     .getLabel().equalsIgnoreCase(
17                             hidden1.getLabel())) {
18                 double w = penghubungLookUp
19                         .get(j).getWeight();
20                 double err = this.output[counter]
21                         .getError();
22                 temp += (w * err);
23                 counter++;
24             }
25         }
```

```
26          hidden1.setError(yAksen*(1 - yAksen)+temp);
27      }
28  }
```

Kode Program 5.4 Proses *Backpropagation*

Berikut ini adalah penjelasan implementasi proses *backpropagation*:

1. Baris 1: Inisialisasi method backpropagation dengan parameter output asli
2. Baris 2 – 9: Hitung semua eror yang ada pada *node hidden layer*
3. Baris 10 – 27: Hitung semua eror yang ada pada *node output layer*.
4. Baris 28: Selesai

5.3.5 Implementasi *Weight Updating*

Proses *weight updating* merupakan proses terakhir dari 3 proses utama algoritme *backpropagation*. *Weight updating* merupakan proses untuk mengupdate semua bobot yang ada pada jaringan berdasarkan nilai eror yang didapatkan pada proses *backprpagation*. Implementasi dari proses *weight updating* dapat dilihat pada Kode Program 5.5.

```
1   private void weightUpdate(int[] nilaiInputAsli) {
2       for (int i = 1; i <= penghubungLookUp.size(); i++){
3           double wSebelum = penghubungLookUp
4                   .get(i).getWeight();
5           double w;
6           int asal = Integer.parseInt(penghubungLookUp
7                   .get(i).getLabel().substring(1, 2));
8           double x;
9           double alfa = this.alfa;
10          double err = penghubungLookUp
11                  .get(i).getNodeTujuan().getError();
12          //bobot pada input - hidden
13          if (penghubungLookUp.get(i).getLabel()
14                  .substring(0, 1).equalsIgnoreCase("H")) {
15              x = normalisasi(nilaiInputAsli[asal - 1],
16                      this.input[asal - 1].getMax(),
17                      this.input[asal - 1].getMin());
18              w = wSebelum + (alfa * err * x);
19              penghubungLookUp.get(i).setWeight(w);
20          } else { //bobot pada hidden - output
21              x = penghubungLookUp
```

```
22                          .get(i).getNodeAsal().getY();
23                  w = wSebelum + (alfa * err * x);
24                  penghubungLookUp.get(i).setWeight(w);
25              }
26          }
27      }
```

Kode Program 5.5 Proses *Weight Updating*

Berikut ini adalah penjelasan implementasi proses *weight update*:

1. Baris 1: Inisialisasi method *weight updating* dengan parameter nilai inputan. Parameter ini digunakan untuk mendapatkan nilai *x*
2. Baris 2: *Looping* untuk melakukan proses *update* ke semua bobot
3. Baris 3 – 11: Set nilai variabel-variabel yang dibutuhkan
4. Baris 13 – 19: Ubah bobot yang menghubungkan *input layer* dengan *hidden layer*. Ini ditandai dengan label bobot yang diawali dengan huruf 'H'
5. Baris 20 – 25: Ubah bobot yang menghubungkan *hidden layer* dengan *output layer*.
6. Baris 26: Keluar dari *looping*
7. Baris 27: Selesai

5.3.6 Implementasi Penentuan Durasi Nyala Lampu

Setelah semua proses pelatihan selesai dilakukan, proses selanjutnya adalah menghitung durasi nyala lampu berdasarkan data uji yang sudah disiapkan menggunakan variabel – variabel yang didaptakan dari proses pelatihan. Alurnya sama seperti proses *feedforward*. Implementasi dari proses Penentuan Durasi Nyala Lampu dapat dilihat pada Kode Program 5.6.

```
1   public void hitungDurasi(Data[] dataUjiAntrian,
2           Data[] dataUjiLampu){
3       int jumlahDataAntrian = dataUjiAntrian.length;
4       double sumD = 0;
5       for (int i = 0; i < jumlahDataAntrian; i++) {
6           double[] y;
7           double[] t;
8           double[] denormalisasi;
9           y = feedForward(dataUjiAntrian[i].nilai);
10          t = new double[y.length];
11          for (int j = 0; j < t.length; j++) {
12              t[j] = normalisasi(dataUjiLampu[i]
```

```
13                      .nilai[j],
14                      output[j].getMax(),
15                      output[j].getMin());
16          }
17
18          double temp = 0;
19          for (int j = 0; j < t.length; j++) {
20              temp += (t[j]- y[j]);
21          }
22          sumD += Math.pow((temp/4), 2);
23          denormalisasi = new double[y.length];
24          for (int j = 0; j<denormalisasi.length; j++) {
25               denormalisasi[j] = denormalisasi(y[j],
26                      output[j].getMax(),
27                      output[j].getMin());
28          }
29          for (int j = 0; j<denormalisasi.length; j++) {
30              System.out.printf("%.2f  ",
31                      denormalisasi[j]);
32          }
33          System.out.print("\tAsli: ");
34          for (int j = 0; j <= 3; j++) {
35              System.out.print(dataUjiLampu[i]
36                      .nilai[j]+ " ");
37          }
38          System.out.println("");
39      }
40      System.out.println("RMSE: " +
41              Math.sqrt(sumD/dataUjiLampu.length));
42  }
```

Kode Program 5.6 Proses Penentuan Durasi Nyala Lampu

Berikut ini penjelasan implementasi proses Penentuan Durasi Nyala Lampu:

1. Baris 3-4: inisialisasi variabel yang dibutuhkan
2. Baris 5: lakukan perulangan sebanyak data yang ada
3. Baris 6-10: inisialisasi variabel yang dibutuhkan. Variabel y[i] merupakan hasil *output* sistem dari *dataUjiAntrian* ke-*i*

4. Baris 11-16: lakukan perulangan sebanyak data yang ada pada *t[i]* yang ada. Satu data *t[i]* memiliki 4 nilai, yaitu 4 *output* target yang ada pada database. Normalisasi semua nilai pada *t*
5. Baris 18-22: hitung variabel d, yaitu variabel untuk menyimpan selisih *output* target (*t*) dengan *output* sistem (*y*)
6. Baris 23-28: Denormalisasi variabel *y* dan simpan di variabel baru bernama *denormalisasi*
7. Baris 29-40: mencetah semua hasil. Hasil berupa variabel *denormalisasi* dan *output* target
8. Baris 41: hitung *RMSE* dan tampilkan
9. Kembali

5.3.7 Implementasi Denormalisasi Data

Setelah data diproses, data perlu didenormalisasi terlebih dahulu sebelum data ditampilkan. Proses ini bertujuan untuk mendapatkan nilai asli dari nilai yang dinormalisasi sebelumnya. Implementasi dariproses denormalisasi data dapat dilihat pada Kode Program 5.7.

```
1  private double denormalisasi(int nilai, int max, int min) {
2      double n = 0;
3      n = (((max-min)*(nilai-0.1))/0.8)+min;
4      return n;
5  }
```

Kode Program 5.7 Proses Denormalisasi Data

Berikut ini penjelasan implementasi proses Denormalisasi Data:

1. Baris 2: Inisialisasi variabel lokal n
2. Baris 3: Hitung hasil dari normalisasi data menggunakan rumus yang sudah ditentukan dengan nilai berdasarkan parameter. Kemudian simpan hasil pada variabel *n*
3. Baris 4: kembalikan nilai *n*.