BAB 2
LANDASAN TEORI

2.1 Pengenalan suara

Dalam pengenalan suara terdapat 3 kategori yaitu: *speech recognition*, *speaker recognition* dan *language recognition.* Dalam penelitian ini hanya akan membahas *speaker recognition. Speaker recognition* adalah suatu proses yang bertujuan mengenali siapa yang sedang berbicara berdasarkan informasi yang terkandung dalam gelombang suara yang di-*input*-an [8]. *Speaker recognition* dibagi menjadi 2 bagian, yaitu:

2.1.1 *Speaker verification*

Speaker verification adalah proses verifikasi seorang pembicara, di mana sebelumnya telah diketahui identitas pembicara tersebut berdasarkan data yang telah di-*input*-an. *Speaker verification* melakukan perbandingan *one to one* (1:1) dalam arti bahwa fitur-fitur suara dari seorang pembicara dibandingkan secara langsung dengan fitur-fitur seorang pembicara tertentu yang ada dalam sistem. Bila hasil perbandingan (skor) tersebut lebih kecil atau sama dengan batasan tertentu (*threshold*), maka pembicara tersebut diterima, bila tidak maka akan ditolak (dengan asumsi semakin kecil skor berarti kedua sampel semakin mirip) [8].

2.1.2 *Speaker identification*

Speaker identification adalah proses mendapatkan identitas dari seorang pembicara dengan membandingkan fitur-fitur suara yang di-*input*-an dengan semua fitur-fitur dari setiap pembicara yang ada dalam *databas*e. Berbeda dengan pada *speaker verification*, proses ini melakukan perbandingan *one to many* (1:N) [8].

2.2 Teknik-teknik *Speech Recognition*

Dalam proses pengenalan suara terdapat teknik-teknik dalam mengambil informasi dan mengolah data suara. Ada 4 teknik dalam *Speech Recognition* yaitu *Speech Analysis Technique, Feature Extraction Technique, Modeling Technique, Matching Techniques* [9]. Berikut penjelasan 4 teknik tersebut:

2.2.1 *Speech Analysis Technique*

Data *Speech* mengandung berbagai jenis informasi yang menunjukkan identitas pembicara. Tahapan *speech analysis* berkaitan dengan ukuran *frame* yang cocok untuk segmentasi sinyal suara dalam analisa dan ekstraksi yang lebih lanjut.

2.2.2 *Feature Extraction Technique*

Ekstraksi fitur *speech* dalam kategorisasi masalah adalah tentang mengurangi dimensi dari vektor *input* ketika mempertahankan membedakan kekuatan sinyal. Dari pembentukan dasar *speaker identification* dan *speaker verification*, bahwa jumlah pelatihan dan vektor uji diperlukan untuk masalah klasifikasi yang tumbuh dengan dimensi masukan yang diberikan sehingga peneliti membutuhkan fitur ekstraksi dari sinyal suara.

2.2.3 *Modeling Technique*

Tujuan dari *modeling technique* adalah untuk menghasilkan *speaker models* yang menggunakan fitur vektor pembicara khusus. Teknik *Speaker modeling* dibagi menjadi dua klasifikasi yaitu *speaker recognition* dan *speaker identification*. Teknik *Speaker identification* secara otomatis mengidentifikasi siapa yang berbicara berdasarkan informasi individual yang terintegrasi dalam sinyal suara. *Speaker recognition* juga dibagi menjadi dua bagian yaitu *speaker dependent* dan *speaker independent*.

Dalam modus *speaker independent* dari *speech recognition*, komputer harus mengabaikan karakteristik khusus pembicara dari sinyal suara dan mengekstrak pesan yang dimaksudkan. Di sisi lain dalam kasus *speaker recognition machine*

harus mengekstrak karakteristik pembicara dalam sinyal akustik. Tujuan utama dari *speaker identification* adalah membandingkan sinyal pidato dari pembicara tak dikenal ke *database* pembicara yang sudah dikenal. Sistem ini dapat mengenali pembicara, yang telah dilatih dengan sejumlah pembicara. *Speaker recognition* juga dapat dibagi menjadi dua metode, *text dependent* dan *text independent*. Dalam metode *text dependent* pembicara mengatakan kata kunci atau kalimat yang memiliki teks yang sama untuk menguji pelatihan dan pengenalan. Sedangkan *text independent* tidak bergantung pada teks tertentu yang diucapkan.

2.2.4 *Matching Techniques*

Mesin *speech recognition* mencocokkan sebuah kata yang terdeteksi dengan kata yang sudah diketahui salah satu dari teknik-teknik berikut :

1) *Whole Word Matching*

Mesin membandingkan sinyal *digital-audio* yang datang terhadap *template* rekaman kata. Teknik ini membutuhkan waktu lebih sedikit pengolahan dari pencocokan *sub*-kata, tetapi mensyaratkan bahwa pengguna (atau seseorang) merekam setiap kata yang akan dikenali, kadang-kadang beberapa ratus ribu kata. *Template* seluruh kata juga membutuhkan memori penyimpanan yang besar (antara 50 dan 512 *byte* per kata) dan hanya praktis jika pengenalan kosakata tersebut sudah dikenal ketika aplikasi dikembangkan.

2) *Sub Word Matching*

Mesin mencari *sub*-kata, biasanya fonem dan kemudian melakukan pengenalan pola lanjut. Teknik ini membutuhkan lebih banyak pemrosesan dari pencocokan seluruh kata, tetapi membutuhkan penyimpanan lebih sedikit (antara 5 dan 20 *byte* per kata). Selain itu, pengucapan kata dapat ditebak dari teks bahasa Inggris tanpa mengharuskan pengguna untuk berbicara kata yang sebelumnya.

2.3 Klasifikasi sinyal eksitasi

Berdasarkan sinyal eksitasi yang dihasilkan pada proses produksi suara, sinyal suara ucapan dapat dibagi menjadi tiga bagian, yaitu :

1) Sinyal *silence*

   Sinyal pada saat tidak terjadi proses produksi suara ucapan, dan sinyal yang diterima oleh pendengar dianggap sebagai bising latar belakang.

2) Sinyal *unvoiced*

   Sinyal *unvoiced* terjadi pada saat pita suara tidak bergetar, di mana sinyal eksitasi berupa sinyal *random*.

3) Sinyal *voiced*

   Terjadi jika pita suara bergetar, yaitu pada saat sinyal eksitasi berupa sinyal pulsa kuasi-periodik. Selama terjadinya sinyal *voiced* ini, pita suara bergetar pada frekuensi fundamental – inilah yang dikenal sebagai *pitch* dari suara tersebut.

2.4 Konversi Analog Menjadi Digital

Signal – signal yang natural pada umumnya seperti *signal* suara merupakan *signal continue* di mana memiliki nilai yang tidak terbatas. Sedangkan pada komputer, semua signal yang dapat diproses oleh komputer hanyalah *signal discrete* atau sering dikenal sebagai istilah *digital signal*. Agar *signal* natural dapat diproses oleh komputer, maka harus diubah terlebih dahulu dari data *signal continue* menjadi *discrete*. [10] Hal itu dapat dilakukan melalui 3 proses, di antaranya sebagai berikut:

1. Proses sampling

   Proses sampling adalah suatu proses untuk mengambil data sinyal kontinu untuk setiap periode tertentu. Dalam melakukan proses sampling data, berlaku aturan *Nyquist*, yaitu bahwa frekuensi *s*ampling (sampling *rate*) minimal harus dua kali lebih tinggi dari frekuensi maksimum yang akan di sampling. Jika

sinyal sampling kurang dari 2 kali frekuensi maksimum sinyal yang akan di sampling, maka akan timbul efek *aliasing*. *Aliasing* adalah suatu efek di mana sinyal yang dihasilkan memiliki frekuensi yang berbeda dengan sinyal aslinya.

2. Proses kuantisasi

   Proses kuantisasi adalah proses untuk membulatkan nilai data ke dalam bilangan-bilangan tertentu yang telah ditentukan terlebih dahulu. Semakin banyak *level* yang dipakai maka semakin akurat pula data *sinyal* yang disimpan tetapi akan menghasilkan ukuran data besar dan proses yang lama.

3. Proses *coding*

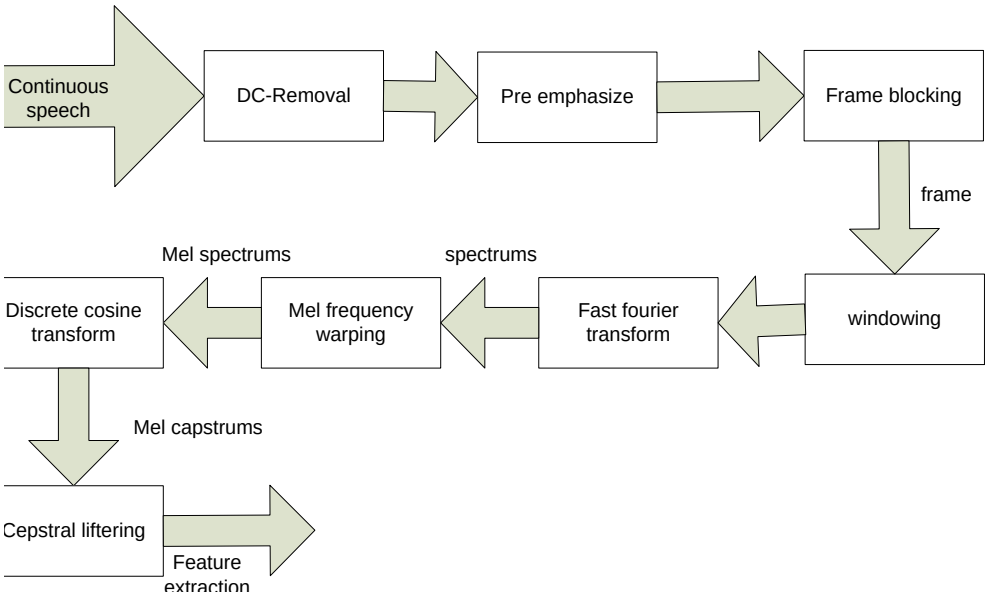
   Proses *coding* adalah proses pemberian kode untuk tiap-tiap data sinyal yang telah ter-kuantisasi berdasarkan level yang ditempati.

2.5 *Mel Frequency Cepstrums Coefficients*

MFCC (*Mel Frequency Cepstrums Coefficients*) merupakan salah satu metode yang banyak digunakan dalam bidang *speech technology*, baik *speaker recognition* maupun *speech recognition.* Metode ini digunakan untuk melakukan *feature extraction*, sebuah proses yang mengonversikan sinyal suara menjadi beberapa parameter [10]. Beberapa keunggulan dari metode ini adalah:

a. Mampu untuk menangkap karakteristik suara yang sangat penting bagi pengenalan suara, atau dengan kata lain dapat menangkap informasi-informasi penting yang terkandung dalam sinyal suara.
b. Menghasilkan data seminimal mungkin, tanpa menghilangkan informasi-informasi penting yang dikandungnya.
c. Meniru organ pendengaran manusia dalam melakukan persepsi terhadap sinyal suara. Pengujian yang dilakukan untuk periode waktu yang cukup pendek akan menunjukkan karakteristik sinyal suara yang *stationary.* Tetapi bila dilakukan dalam periode waktu yang lebih panjang karakteristik sinyal suara akan terus berubah sesuai dengan kata yang diucapkan.

MFCC *feature extraction* sebenarnya merupakan adaptasi dari sistem pendengaran manusia, di mana sinyal suara akan difilter secara linear untuk frekuensi rendah (di bawah 1000 *Hz*) dan secara logaritmik untuk frekuensi tinggi (di atas 1000 *Hz*). Gambar di bawah ini merupakan *block* diagram untuk MFCC [10].



Gambar 2.1 Blok Diagram Untuk MFCC

Pada Gambar 2.1 dapat dilihat proses MFFC, *Continues speech* yaitu masukan suara dengan kata yang telah ditentukan oleh sistem, lalu proses *Dc Removal* untuk mendapatkan nilai normal dari sampel sinyal, lalu proses *Pre-emphasize* untuk memperbaiki sinyal dari gangguan *noise*, lalu proses *Frame Blocking* bertujuan untuk membagi sampel sinyal menjadi beberapa *frame,* lalu proses *Windowing* untuk mengurangi efek diskontinuitas pada ujung-ujung *frame* yang dihasilkan oleh proses *frame blocking*, lalu proses FFT sehingga diperoleh sampel sinyal dalam frekuensi domain, lalu proses *Filter bank* untuk mengetahui ukuran energi dari setiap frekuensi *band*, lalu proses DFT untuk mendapatkan *mel cepstrums*, langkah terakhir proses *Cepstral Liftering*, bertujuan menghaluskan *spectrum signal*.

2.6 *Clustering*

Clustering atau analisis *cluster* adalah proses pengelompokan satu set benda- benda fisik atau abstrak ke dalam kelas objek yang sama [11]. *Clustering* atau pengelompokan adalah salah satu alat bantu pada *data mining* yang bertujuan mengelompokkan obyek-obyek ke dalam *cluster-cluster*. *Cluster* adalah sekelompok atau sekumpulan obyek-obyek data yang *similar*/sama satu sama lain dalam *cluster* yang sama dan *dissimilar*/tidak sama terhadap obyek-obyek yang berbeda *cluster*. Obyek akan dikelompokkan ke dalam satu atau lebih *cluster* sehingga obyek-obyek yang berada dalam satu *cluster* akan mempunyai kesamaan yang tinggi antara satu dengan lainnya. Obyek-obyek dikelompokkan berdasarkan prinsip memaksimalkan kesamaan obyek pada *cluster* yang sama dan memaksimalkan ketidaksamaan pada *cluster* yang berbeda. Kesamaan obyek biasanya diperoleh dari nilai-nilai atribut yang menjelaskan obyek data, sedangkan obyek-obyek data biasanya direpresentasikan sebagai sebuah titik dalam ruang multidimensi [12].

Dengan menggunakan *clustering*, peneliti dapat mengidentifikasi daerah yang padat, menemukan pola-pola distribusi secara keseluruhan, dan menemukan keterkaitan yang menarik antara atribut-atribut data. Dalam *data mining*, usaha difokuskan pada metode-metode penemuan untuk *cluster* pada basis data berukuran besar secara efektif dan efisien. Beberapa kebutuhan *clustering* dalam *data mining* meliputi skalabilitas, kemampuan untuk menangani tipe atribut yang berbeda, mampu menangani dimensionalitas yang tinggi, menangani data yang mempunyai *noise*, dan dapat diterjemahkan dengan mudah.

2.7 *Rough Sets-K-Means Clustering*

Rough Sets-K-Means algoritma adalah penggunaan teori *rough set* dan algoritma *k-means* untuk menangani ketidakpastian yang terlibat pada analisis cluster. Dalam *rough clustering* setiap klaster memiliki dua penaksiran (*approximations*), yaitu *lower approximation* (nilai penaksiran rendah) dan *upper approximation* (nilai penaksiran tinggi). *Lower approximation* adalah bagian dari

upper approximation. Anggota dalam *lower approximation* sudah dipastikan milik sebuah *cluster*, oleh karena itu mereka tidak dapat masuk ke klaster lain. Sedangkan data objek pada *upper approximation* belum dipastikan masuk dalam klaster atau mungkin masuk pada *cluster* lain. Karena keanggotaan *upper approximation* belum dipastikan maka setidaknya mereka masuk ke dalam klaster lain. [13]

Langkah-langkah dalam metode *clustering Rough Sets-K-Means* adalah sebagai berikut:

1. Pilih *cluster* awal n objek ke dalam *cluster* k.
2. Menghitung *Centroid* C_j pada klaster, dengan kondisi-kondisi sebagai berikut.

   $If \quad \underline{U}(K) \neq \emptyset \; and \; \overline{U}(K) - \underline{U}(K) = \emptyset$

   $$C_j = \sum_{x \in \underline{U}(K)} \frac{x_i}{|\underline{U}(K)|} \tag{2.1}$$

   $Else \;\; \underline{U}(K) = \emptyset \; and \; \overline{U}(K) - \underline{U}(K) \neq \emptyset$

   $$C_j = \sum_{x \in \underline{U}(K) - \underline{U}(K)} \frac{x_i}{|\overline{U}(K) - \underline{U}(K)|} \tag{2.2}$$

   $Else$

   $$C_j = W_{lower} \times \sum_{x \in \underline{U}(K)} \frac{x_i}{|\underline{U}(K)|} + W_{upper} \times \sum_{x \in \underline{U}(K) - \underline{U}(K)} \frac{x_i}{|\overline{U}(K) - \underline{U}(K)|} \tag{2.3}$$

3. Tetapkan setiap objek pada batas *lower* $\underline{U}(K)$ atau batas *upper* $\overline{U}(K)$ pada *cluster* masing-masing i. Untuk setiap objek vektor x, tetapkan $d(X, C_j)$ menjadi jarak antara dirinya sendiri dan centroid pada klaster C_j.

   $$d(X, C_j) = min_{1 \leq j \leq K} d(X, C_j) \tag{2.4}$$

   Rasio $d(X, C\_i) \; / \; d(X, C\_j), 1 \leq j \leq K$ digunakan untuk menentukan keanggotaan dari x adalah sebagai berikut $d(X, C_i) \; / \; d(X, C_j) \leq \epsilon$, pada pasangan (i, j), pada $x \epsilon \overline{U}(C_i)$ dan $x \epsilon \overline{U}(C_j)$ dan x tidak akan jadi bagian pada setiap *lower approximation*. Jika tidak, $x \epsilon \overline{U}(C_i)$, seperti $d(X, C_i)$ adalah nilai minimal dari $1 \leq j \leq K$. Sebagai tambahan $x \epsilon \overline{U}(C_i)$.

4. Jika kriteria konvergen terpenuhi, yaitu pusat *cluster* yang sama dengan yang di iterasi sebelumnya maka iterasi akan berhenti, jika tidak maka ulangi langkah 2 dan 3.

2.8 Jaringan Syaraf Tiruan

Jaringan saraf tiruan (JST) atau disebut juga dengan *neural network* (NN), adalah jaringan dari sekelompok unit pemroses kecil yang dimodelkan berdasarkan jaringan saraf manusia. Jaringan syaraf tiruan merupakan sistem adaptif yang dapat mengubah strukturnya untuk memecahkan masalah berdasarkan informasi eksternal maupun internal yang mengalir melalui jaringan tersebut. Secara sederhana, JST adalah sebuah alat pemodelan data statistik *non-linier*. JST dapat digunakan untuk memodelkan hubungan yang kompleks antara *input* dan *output* untuk menemukan pola-pola pada data.

Jaringan syaraf tiruan adalah suatu struktur pemroses informasi yang terdistribusi dan bekerja secara paralel, yang terdiri atas elemen pemroses (yang memiliki memori lokal dan beroperasi dengan informasi lokal) yang berinterkoneksi bersama dengan alur sinyal searah yang disebut koneksi. Setiap elemen pemroses memiliki koneksi keluaran tunggal yang bercabang ke sejumlah koneksi kolateral yang diinginkan (setiap koneksi membawa sinyal yang sama dari keluaran elemen pemroses tersebut). Keluaran dari elemen pemroses tersebut dapat merupakan sebarang jenis persamaan matematis yang diinginkan. Seluruh proses yang berlangsung pada setiap elemen pemroses harus benar-benar dilakukan secara lokal, yaitu keluaran hanya bergantung pada nilai masukan pada saat itu yang diperoleh melalui koneksi dan nilai yang tersimpan dalam memori lokal.

Sebuah jaringan saraf adalah sebuah prosesor yang terdistribusi paralel dan mempunyai kecenderungan untuk menyimpan pengetahuan yang didapatkannya dari pengalaman dan membuatnya tetap tersedia untuk digunakan. Hal ini menyerupai kerja otak dalam dua hal yaitu: 1. Pengetahuan diperoleh oleh jaringan melalui suatu proses belajar. 2. Kekuatan hubungan antar sel saraf yang dikenal dengan bobot sinaps digunakan untuk menyimpan pengetahuan.

Jaringan syaraf tiruan menjadi salah satu pilihan ketika rumusan persoalan-persoalan yang dihadapi tidak bias diselesaikan secara analitis, dan dengan mengasumsikan suatu *black box* yang tidak tahu isinya maka jaringan syaraf tiruan menemukan pola hubungan antara *input* dan *output* melalui tahap pelatihan [14].

Menurut Siang [15], jaringan syaraf tiruan (JST) adalah sistem pemroses informasi yang memiliki karakteristik mirip dengan jaringan syaraf biologi. JST dibentuk sebagai generalisasi model matematika dari jaringan syaraf biologi, dengan asumsi bahwa:

a) Pemrosesan informasi terjadi pada banyak elemen sederhana (neuron).
b) Sinyal terkirimkan di antara neuron-neuron melalui penghubung-penghubung.
c) Penghubung antar neuron memiliki bobot yang akan memperkuat atau memperlemah sinyal.
d) Untuk menentukan *output*, setiap neuron menggunakan fungsi aktivasi (biasanya bukan fungsi linier) yang dikenakan pada jumlahan *input* yang diterima. Besarnya *output* ini selanjutnya dibandingkan dengan suatu batas ambang.

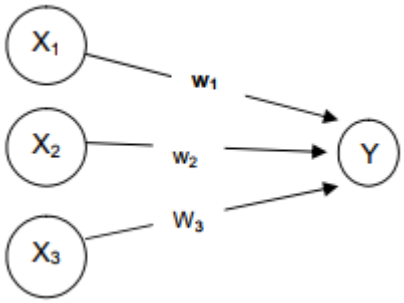
JST ditentukan oleh 3 hal :

1) Pola hubungan antar neuron (disebut arsitektur jaringan)
2) Metode untuk menentukan bobot penghubung (disebut metode *training*/ *learning*/ algoritma)
3) Fungsi aktivasi (fungsi transfer)

Neuron dalam jaringan syaraf tiruan sering diganti dengan istilah simpul. Setiap simpul tersebut berfungsi untuk menerima atau mengirim sinyal dari atau ke simpul-simpul lainnya. Pengiriman sinyal disampaikan melalui penghubung. Kekuatan hubungan yang terjadi antara setiap simpul yang saling terhubung dikenal dengan nama bobot.

Arsitektur jaringan dan algoritma pelatihan sangat menentukan model-model jaringan syaraf tiruan. Arsitektur tersebut gunanya untuk menjelaskan arah perjalanan sinyal atau data di dalam jaringan. Sedangkan algoritma belajar

menjelaskan bagaimana bobot koneksi harus diubah agar pasangan masukan dan keluaran yang diinginkan dapat tercapai. Dalam setiap perubahan harga bobot koneksi dapat dilakukan dengan berbagai cara, tergantung pada jenis algoritma pelatihan yang digunakan. Dengan mengatur besarnya nilai bobot ini diharapkan bahwa kinerja jaringan dalam mempelajari berbagai macam pola yang dinyatakan oleh setiap pasangan masukan dan keluaran akan meningkat. Sebagai contoh, perhatikan neuron Y pada gambar berikut :



Gambar 2.2 Sebuah Sel Syaraf Tiruan

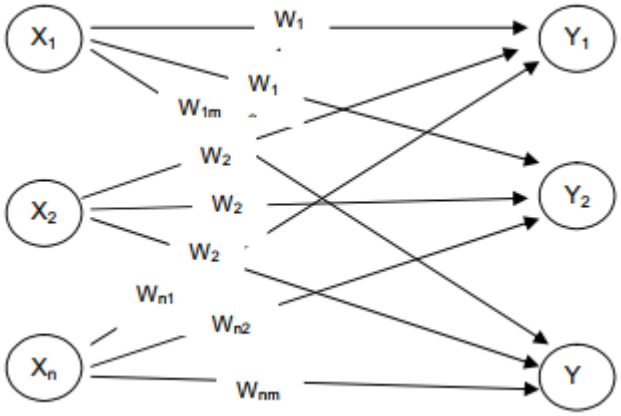
Y menerima *input* dari neuron x_1, x_2, dan x_3 dengan bobot hubungan masing-masing adalah w_1, x_2 dan w_3. Ketiga impuls neuron yang ada dijumlahkan net = $x_1w_1 + x_2w_2 + x_3w_3\ x_1$. Besarnya impuls yang diterima oleh Y mengikuti fungsi aktivasi y = f(net). Apabila nilai fungsi aktivasi cukup kuat, maka sinyal akan diteruskan. Nilai fungsi aktivasi (keluaran model jaringan) juga dapat dipakai sebagai dasar untuk mengubah bobot.

2.8.1 Arsitektur Jaringan

Jaringan syaraf tiruan dapat dibedakan dalam beberapa kategori sesuai sudut pandang yang digunakan. Berdasarkan arsitektur atau pola koneksi yang digunakan dalam pada jaringan syaraf tiruan, maka jaringan syaraf tiruan tersebut dapat dibedakan dalam 3 (tiga) kategori, yaitu Jaringan Layar Tunggal, Jaringan Layar Jamak dan Jaringan *Recurrent* yang dapat diuraikan sebagai berikut.

1. Jaringan Layar Tunggal

Dalam jaringan ini, sekumpulan *input* neuron dihubungkan langsung dengan sekumpulan *output*-nya, seperti gambar berikut ini:

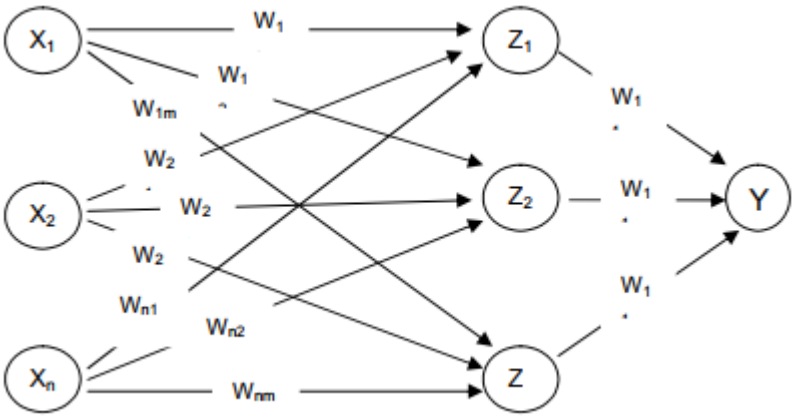


Gambar 2.3 Jaringan Layar Tunggal

Pada Gambar 2.3 diperlihatkan bahwa arsitektur jaringan layar tunggal dengan n buah masukan (X_1, X_2,..., X_n) dan m buah keluaran (Y_1, Y_2,..., Y_m). Dalam jaringan ini semua unit *input* dihubungkan dengan semua unit *output*. Tidak ada unit *input* yang dihubungkan dengan unit *input* lainnya dan unit *output*-pun demikian.

2. Jaringan Layar Jamak

Jaringan ini merupakan perluasan dari layar tunggal. Dalam jaringan ini, selain unit *input* dan *output*, ada unit-unit lain yang sering disebut layar tersembunyi. Layar tersembunyi ini tersebut bisa saja lebih dari satu, sebagai contoh perhatikan Gambar 2.4 di bawah ini:



Gambar 2.4 Jaringan Layar Jamak

Gambar 2.4 Jaringan layar jamak Pada gambar 3 diperlihatkan jaringan dengan n buah unit masukan $(X_1, X_2, \dots, X_n)$, sebuah layar tersembunyi yang terdiri dari m buah unit $(Z_1, Z_2, \dots, Z_n)$ dan 1 buah unit keluaran. Jaringan layar jamak dapat menyelesaikan masalah yang lebih kompleks dibandingkan dengan layar tunggal, meskipun kadang kala proses pelatihan lebih kompleks dan lama.

3. Jaringan *Recurrent*

Model jaringan *recurrent* mirip dengan jaringan layar tunggal ataupun ganda. Hanya saja, ada neuron *output* yang memberikan sinyal pada unit *input* (sering disebut *feedback loop*). Dengan kata lain sinyal mengalir dua arah, yaitu maju dan mundur.

2.8.2 Fungsi Aktivasi

Siang [15] menyebutkan bahwa fungsi aktivasi digunakan untuk menentukan keluaran suatu neuron. Dalam jaringan syaraf tiruan, argumen fungsi aktivasi adalah net masukan (kombinasi linier masukan dan bobotnya). Jika net $\sum x_i w_i$ maka fungsi aktivasinya adalah f (net) = f $(\sum x_i w_i)$.

Fungsi aktivasi yang digunakan dalam penelitian ini adalah fungsi sigmoid biner:

Fungsi *sigmoid*

$$f(x) = \frac{1}{1+e^{-x}} \tag{2.5}$$

fungsi *sigmoid* sering dipakai karena nilai fungsinya yang terletak antara 0 dan 1 dan dapat digunakan dengan mudah.

$$f'(x) = f(x)\,(1 - f(x)) \tag{2.6}$$

2.9 *Backpropagation*

Backpropagation adalah salah satu metode dari jaringan syaraf tiruan yang dapat diaplikasikan dengan baik dalam bidang peramalan (*forecasting*). *Backpropagation* melatih jaringan untuk mendapatkan keseimbangan antara kemampuan jaringan mengenali pola yang digunakan selama *training* serta kemampuan jaringan untuk memberikan balasan yang benar terhadap pola masukan yang serupa namun tidak sama dengan pola yang dipakai selama pelatihan [15].

Menurut Kusumadewi [16] dalam pelatihan dengan *backpropagation* sama halnya seperti pelatihan pada jaringan syaraf yang lain. Pada jaringan *feed forward* (umpan maju), pelatihan dilakukan dalam rangka perhitungan bobot sehingga pada akhir pelatihan akan diperoleh bobot-bobot yang baik. Selama proses pelatihan, bobot-bobot diatur secara iterasi untuk meminimalkan *error* (kesalahan) yang terjadi. Kesalahan dihitung berdasarkan rata-rata kuadrat kesalahan (MSE). Rata-rata kuadrat kesalahan juga dijadikan dasar perhitungan unjuk kerja fungsi aktivasi.

Sebagian besar pelatihan untuk jaringan *feed forward* (umpan maju) menggunakan gradien dari fungsi aktivasi untuk menentukan bagaimana mengatur bobot-bobot dalam rangka meminimalkan kinerja. Gradien ini ditentukan dengan menggunakan suatu teknik yang disebut *backpropagation*. Dan pada dasarnya, algoritma pelatihan standar *backpropagation* akan menggerakkan bobot dengan arah gradien negatif. Prinsip dasar dari algoritma *backpropagation* adalah

memperbaiki bobot-bobot jaringan dengan arah yang membuat fungsi aktivasi menjadi turun dengan cepat.

Ada 3 fase Pelatihan *backpropagation* menurut Siang [15] antara lain:

4. Fase satu, yaitu propagasi maju.

Dalam propagasi maju, setiap sinyal masukan dipropagasi (dihitung maju) ke layar tersembunyi hingga layar keluaran dengan menggunakan fungsi aktivasi yang ditentukan.

5. Fase dua, yaitu propagasi mundur.

Kesalahan (selisih antara keluaran jaringan dengan target yang diinginkan) yang terjadi dipropagasi mundur mulai dari garis yang berhubungan langsung dengan unit-unit di layar keluaran.

6. Fase tiga, yaitu perubahan bobot.

Pada fase ini dilakukan modifikasi bobot untuk menurunkan kesalahan yang terjadi. Ketiga fase tersebut diulang-ulang terus hingga kondisi penghentian dipenuhi.

Algoritma pelatihan untuk jaringan dengan satu layar tersembunyi (dengan fungsi aktivasi *sigmoid* biner) adalah sebagai berikut.

1) Inisialisasi semua bobot dengan bilangan acak kecil
2) Jika kondisi penghentian belum terpenuhi, lakukan langkah 3-11
3) Untuk setiap pasang data pelatihan lakukan langkah 4-10 Fase I : Propagasi maju
4) Tiap unit masukan menerima sinyal dan meneruskannya ke unit tersembunyi di atasnya
5) Hitung semua keluaran di unit tersembunyi z_j (j = 1,2,…,p)

$$z\_in_j = v_{j0} + \sum_{t=1}^{n} x_i v_{ji} \quad (2.7)$$

$$z_j = f\left(z_{in_j}\right) = \frac{1}{1+e^{-z\_in_j}} \quad (2.8)$$

6) Hitung semua keluaran jaringan di unit y_k (k = 1,2,…,m)

$$y\_in_k = w_{k0} + \sum_{j=1}^{n} z_i w_{ji} \quad (2.9)$$

$$y_j = f(y_{in_k}) = \frac{1}{1+e^{-y\_in_k}} \quad (2.10)$$

Fase II : Propagasi mundur

7) Hitung faktor unit keluaran berdasarkan kesalahan di setiap unit keluaran y_k (k= 1,2,…,m)

$$\delta_k = (t_k - y_k) f'(y_{in_k}) = (t_k - y_k) y_k (1 - y_k) \quad (2.11)$$

δ_k merupakan unit kesalahan yang akan dipakai dalam perubahan bobot layar di bawahnya (langkah 8)

8) Hitung suku perubahan bobot w_{kj} (yang akan dipakai nanti untuk mengubah bobot w_{kj}) dengan laju percepatan α ;

$$\Delta w_{kj} = \alpha \delta_k z_j \quad (2.12)$$

9) Hitung faktor unit tersembunyi berdasarkan kesalahan di setiap unit tersembunyi z_j (j = 1,2,…,p)

$$\delta\_in_j = \sum_{k=1}^{m} \delta_k w_{kj} \quad (2.13)$$

Faktor δ unit tersembunyi:

$$\delta_j = \delta\_in_j f'\left(z_{in_j}\right) = \delta_{in_j} z_j (1 - z_j) \quad (2.14)$$

10) Hitung suku perubahan bobot v_{ji} (yang akan dipakai nanti untuk merubah bobot v_{ji})

$$\Delta v_{ji} = \alpha \delta_j x_i \quad (2.15)$$

Fase III : Perubahan bobot

11) Hitung semua perubahan bobot Perubahan bobot garis yang menuju ke unit keluaran:

$$w_{kj}(baru) = w_{kj}(lama) + \Delta w_{kj} \quad (2.16)$$

Perubahan bobot garis yang menuju ke unit tersembunyi:

$$v_{ji}(baru) = v_{ji}(lama) + \Delta v_{ji} \quad (2.17)$$

Setelah pelatihan selesai dilakukan, jaringan dapat dipakai untuk pengenalan pola. Dalam hal ini, hanya propagasi maju (langkah 5 dan 6) saja yang dipakai untuk menentukan keluaran jaringan.

Dalam beberapa kasus pelatihan yang dilakukan memerlukan iterasi yang banyak sehingga membuat proses pelatihan menjadi lama. Untuk mempercepat iterasi dapat dilakukan dengan Parameter α atau laju pemahaman. Nilai α terletak antara 0 dan 1 ($0 \leq \alpha \leq 1$). Jika harga α Semakin besar, maka iterasi yang dipakai semakin sedikit. Akan tetapi jika harga α terlalu besar, maka akan merusak pola yang sudah benar sehingga pemahaman menjadi lambat.

Proses pelatihan yang baik dipengaruhi pada pemilihan bobot awal, karena bobot awal sangat mempengaruhi apakah jaringan mencapai titik minimum lokal atau global, dan seberapa cepat konvergensinya. Oleh karena itu dalam standar *backpropagation*, bobot dan bias diisi dengan bilangan acak kecil dan biasanya bobot awal diinisialisasi secara *random* dengan nilai antara -0,5 sampai 0,5 (atau -1 sampai 1 atau interval yang lainnya).

2.9.1 Momentum

Dalam *backpropagation*, standar perubahan bobot didasarkan atas gradien yang terjadi untuk pola yang dimasukkan saat itu. Modifikasi dilakukan dengan mengubah bobot yang didasarkan atas arah gradien pola terakhir dan pola sebelumnya (momentum) yang dimasukkan. Jadi perhitungannya tidak hanya pola masukan terakhir saja. Momentum ditambahkan untuk menghindari perubahan bobot yang mencolok akibat adanya data yang sangat berbeda dengan data yang lain. Jika beberapa data terakhir yang diberikan ke jaringan memiliki pola serupa

(berarti arah gradien sudah benar), maka perubahan bobot dilakukan secara cepat. Namun jika data terakhir yang dimasukkan memiliki pola yang berbeda dengan pola sebelumnya, maka perubahan bobot dilakukan secara lambat. Dengan penambahan momentum, bobot baru pada waktu ke (t+1) didasarkan atas bobot pada waktu t dan (t-1). Di sini harus ditambahkan dua variabel yang mencatat besarnya momentum untuk dua iterasi terakhir.

Jika μ adalah konstanta ($0 \leq \mu \leq 1$) yang menyatakan parameter momentum maka bobot baru dihitung berdasarkan persamaan berikut ini:

$$W_{kj}(t+1) = W_{kj} + \alpha\delta_k z_i + \mu(W_{kj}(t) - W_{kj}(t-1)) \qquad (2.18)$$

Dengan,

$W_{kj}(t)$= bobot awal pola kedua (hasil dari iterasi pola pertama)

$W_{kj}(t-1)$ = bobot awal pada iterasi pola pertama

Dan

$$V_{ji}(t+1) = V_{ji} + \alpha\delta_k x_i + \mu(V_{ji}(t) - V_{ji}(t-1)) \qquad (2.19)$$

Dengan,

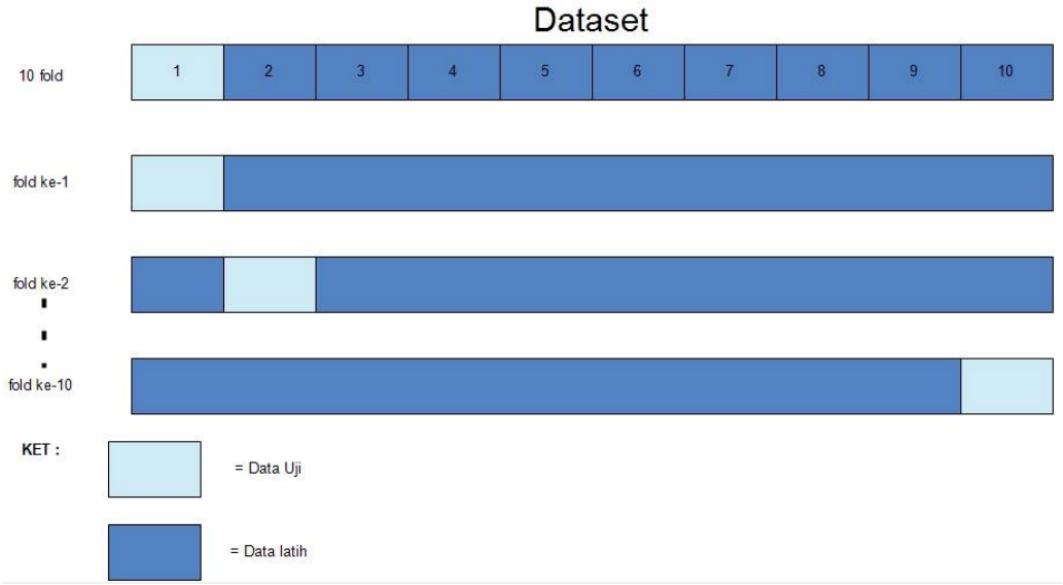
$V_{ji}(t)$= bobot awal pola kedua (hasil dari iterasi pola pertama)

$V_{ji}(t-1)$ = bobot awal pada iterasi pertama

2.10 *K-Fold Cross Validation*

K-fold cross validation adalah salah satu metode untuk mengevaluasi kinerja *classifier*, metode ini dapat digunakan apabila memiliki jumlah data yang terbatas (jumlah *instance* tidak banyak). *K-fold cross validation* merupakan salah satu metode yang digunakan untuk mengetahui rata-rata keberhasilan dari suatu sistem dengan cara melakukan perulangan dengan mengacak atribut masukan sehingga sistem tersebut teruji untuk beberapa atribut *input* yang acak. *K-fold cross validation* diawali dengan membagi data sejumlah n-*fold* yang diinginkan. Dalam proses *cross validation* data akan dibagi dalam n buah partisi dengan ukuran yang sama D1, D2, D3 .. Dan selanjutnya proses uji dan latih dilakukan sebanyak n

kali. Dalam iterasi ke-i partisi Di akan menjadi data uji dan sisanya akan menjadi data latih. Untuk penggunaan jumlah *fold* terbaik untuk uji validitas, dianjurkan menggunakan 10-*fold cross validation* dalam model [17]. Contoh pembagian data set dalam proses 10-*fold cross validation* terlihat pada Gambar 2.5.



Gambar 2.5 Contoh iterasi data dengan *k-fold cross validation*

Cara kerja *K-fold cross validation* adalah sebagai berikut:

1) Total *instance* dibagi menjadi N bagian.
2) *Fold* ke-1 adalah ketika bagian ke-1 menjadi data uji (testing data) dan sisanya menjadi data latih (*training data*). Selanjutnya, hitung akurasi berdasarkan porsi data tersebut. Perhitungan akurasi tersebut dengan menggunakan persamaan sebagai berikut.

$$Akurasi = \frac{\sum data\ uji\ benar\ klasifikasi}{\sum total\ data\ uji} \times 100\%$$

3) *Fold* ke-2 adalah ketika bagian ke-2 menjadi data uji (testing data) dan sisanya menjadi data latih (*training data*). Selanjutnya, hitung akurasi berdasarkan porsi data tersebut.
4) Demikian seterusnya hingga mencapai *fold ke-K*. Hitung rata-rata akurasi dari K buah akurasi di atas. Rata-rata akurasi ini menjadi akurasi final.

2.11 Pemrograman Berbasis Objek

Object oriented programming adalah sebuah metode pemrograman di mana pengembang aplikasi tidak hanya mendefinisikan variabel yang berisi *state* dari sebuah struktur data, tetapi juga mendefinisikan fungsi untuk menunjukkan *behavior* yang diaplikasikan pada struktur data. Dalam hal ini, struktur data merupakan objek. Suatu objek dapat saling berkomunikasi satu sama lain dengan menggunakan fungsi yang ada di dalamnya tanpa perlu mengetahui internal *state* masing-masing objek (*data encapsulation*) [18].

Salah satu keuntungan dari *object oriented programming* dibandingkan *procedural programming* adalah memungkinkan pengembang aplikasi untuk membuat fungsi yang tidak perlu diubah ketika sebuah objek dengan tipe berbeda ditambahkan. Seorang

pengembang aplikasi hanya perlu membuat objek baru yang mewarisi beberapa fungsi atau tipe data dari objek yang sudah ada (*inheritance*). Hal ini membuat *object oriented programming* mudah dalam pengembangannya [18].

Terdapat beberapa konsep dalam *object oriented programming* [19], yaitu:

a) Objek

Objek terdiri dari *state* dan *behavior*. Sebuah objek menyimpan *state* pada variabel dan menunjukkan *behavior* melalui fungsi. Fungsi bertugas mengubah internal *state* dan melayani komunikasi antara satu objek dengan objek lainnya .

b) Class

Class adalah sekumpulan objek yang memiliki *state* dan *behavior* yang sama.

c) Instance

Instance adalah objek tunggal yang memiliki nilai *state* yang secara keseluruhan berbeda dengan objek lain.

d) Generalisasi

Generalisasi adalah konsep yang mengelompokkan *state* dan *behavior* yang sama dari beberapa *class* menjadi *class* tersendiri.

Berikut adalah ciri-ciri dari generalisasi :

1) Inheritance

Mekanisme untuk mengimplementasikan generalisasi dalam bahasa pemrograman berorientasi objek. *Class* yang memiliki *state* dan *behavior* dari hasil pengelompokan disebut *superclass* sedangkan *class* yang memiliki *state* dan *behavior* yang unik karena pengelompokan disebut *subclass*.

2) Operasi transitif

Operasi yang menurunkan *state* dan *behavior* dari *superclass* ke *subclass*.

3) Disjoint nature

Aspek di mana *class* harus memiliki *state* dan *behavior* yang unik dari *class* lain yang satu level.

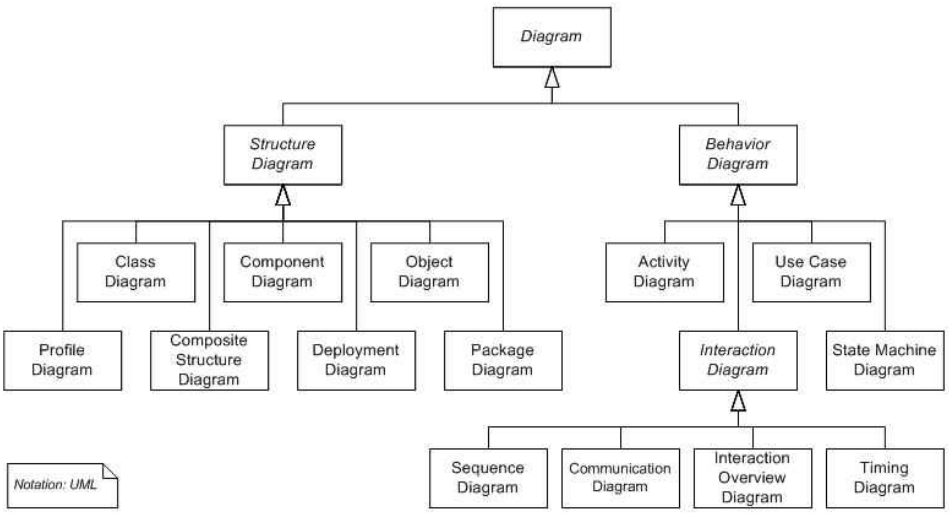
e) Message passing

Message passing adalah komunikasi yang dilakukan objek untuk meminta informasi tertentu dari objek lain dengan menggunakan fungsi dari objek lain.

2.12 *Unified Modeling Language*

UML adalah kumpulan notasi grafis yang membantu dalam mengembangkan dan merancang sistem perangkat lunak, khususnya sistem perangkat lunak yang dibangun dengan *object oriented* [20]. UML (*Unified Modeling Language*) adalah "bahasa" permodelan untuk sistem atau perangkat lunak yang berparadigma "berorientasi objek". Permodelan sesungguhnya digunakan untuk penyederhanaan permasalahan-permasalahan yang kompleks sedemikian rupa sehingga lebih mudah dipelajari dan dipahami. [21]

Berdasarkan pendapat yang dikemukakan di atas dapat ditarik kesimpulan bahwa "*Unified Modeling Language* (UML) adalah sebuah bahasa permodelan yang merepresentasikan dan memvisualkan sebuah sistem pengembangan perangkat lunak berbasis *object oriented*. UML memberikan 13 jenis diagram yang digunakan untuk merepresentasikan bagian penting dari sebuah rancangan perangkat lunak. Penggunaan jenis diagram didasarkan pada kebutuhan dan karakteristik sebuah perangkat lunak. Berikut ini adalah jenis-jenis diagram beserta fungsinya :



Gambar 2.6 Klasifikasi Diagram UML

Tabel 2.1 Tabel Diagram UML

Diagram	Kegunaan	*Learning Priority*
Activity	Menggambarkan proses aplikasi tingkat tinggi, termasuk aliran data, atau untuk model logika yang kompleks dalam sistem.	*High*
Class	Menunjukkan sekumpulan elemen model statis seperti kelas dan jenis, isinya, dan hubungan mereka.	*High*
Communication	Menunjukkan contoh dari kelas, hubungan antar kelas, dan alur pesan antara mereka.	*Low*
Component	Menggambarkan komponen yang membentuk sebuah aplikasi.	*Medium*
Composite Structure	Menggambarkan struktur internal dari sebuah pengklasifikasi.	*Low*
Deployment	Pemindahan artefak ke *node*.	*Medium*
Interaction Overview	Sebuah varian dari diagram aktivitas yang mengontrol aliran dalam proses sistem.	*Low*
Object	Menggambarkan objek dan relasi mereka di sebuah titik waktu tertentu.	*Low*
Package	Menunjukkan bagaimana elemen model akan disusun dalam paket maupun dependensi antara bentuk paket.	*Low*
Sequence	Interaksi antar obyek, penekanan pada *sequence*	*High*

State Machine	Menjelaskan lingkungan sebuah objek atau interaksi di dalamnya, maupun transisi antara lingkungan.	*Medium*
Timing	Menggambarkan perubahan keadaan dari waktu ke waktu.	*Low*
Use Case	Bagaimana pengguna berinteraksi dengan Sistem.	*Medium*

Dalam pembuatan aplikasi pengenalan suara ini, peneliti hanya akan menggunakan 4 macam diagram UML yaitu *Use Case Diagram, Class Diagram, Sequence Diagram* dan *Activity Diagram*. Pemilihan diagram ini didasarkan pada tingkat *Learning Priority* yang tinggi serta dirasa cukup mewakili dari seluruh segmentasi pada diagram UML.

2.12.1 *Use Case Diagram*

Use case adalah sebuah kegiatan yang dilakukan oleh sistem yang biasanya menanggapi permintaan dari pengguna sistem. *Use case* diagram secara grafis menggambarkan interaksi antara sistem, sistem eksternal dan pengguna. Dengan kata lain *use case* diagram secara grafis mendeskripsikan siapa yang akan menggunakan sistem dan dalam cara apa pengguna mengharapkan interaksi dengan sistem itu. [22]

2.12.2 *Class Diagram*

Class Diagram adalah untuk mendokumentasikan dan menggambarkan kelas-kelas dalam pemrograman yang nantinya akan dibangun. *Class diagram* menggambarkan struktur objek sistem. Diagram ini menunjukkan *class object* yang menyusun sistem dan juga hubungan antara *class object* tersebut. [22]

2.12.3 *Sequence Diagram*

Sequence diagram adalah diagram yang digunakan untuk mendefinisikan *input* dan *output* serta urutan interaksi antara pengguna dan sistem untuk sebuah *use case*. *Sequence* diagram secara grafis menggambarkan bagaimana objek

berinteraksi dengan satu sama lain melalui pesan pada *sequence* sebuah *use case* atau operasi. [22]

2.12.4 *Activity Diagram*

Activity Diagram adalah sebuah alur kerja yang menjelaskan berbagai kegiatan pengguna (atau sistem), orang yang melakukan aktivitas, dan aliran sekuensial dari aktivitas-aktivitas tersebut. *Activity diagram* secara grafis digunakan untuk menggambarkan rangkaian aliran aktivitas *use case. Activity diagram* dapat juga digunakan untuk memodelkan aksi yang akan dilakukan saat sebuah operasi dieksekusi, dan memodelkan hasil dari aksi tersebut. [22]

2.13 C#

C# merupakan sebuah bahasa pemrograman yang berorientasi objek yang dikembangkan oleh Microsoft sebagai bagian dari inisiatif kerangka .NET *Framework*. Bahasa pemrograman ini dibuat berbasiskan bahasa C++ yang telah dipengaruhi oleh aspek-aspek ataupun fitur bahasa yang terdapat pada bahasa-bahasa pemrograman lainnya seperti *Java*, *Delphi*, *Visual Basic*, dan lain-lain dengan beberapa penyederhanaan [23].

2.14 DBMS

Database Management System (DBMS) merupakan sistem perangkat lunak yang memungkinkan pengguna untuk mendefinisikan, membuat, memelihara, dan mengatur akses ke *database*. DBMS ini menyediakan fasilitas untuk mendefinisikan *database* melalui DDL (*Data Definition Language*) dan menyediakan fasilitas untuk melakukan *insert, update, delete, select* melalui DML (*Data Manipulation Language*) [24]. Dalam penelitian ini DBMS yang digunakan adalah *MySQL*.

MySQL merupakan DBMS yang bersifat *open source* di mana pengguna dapat melakukan modifikasi pada kodenya yang berorientasi pada performa dan merupakan DBMS yang mendukung beberapa tipe *table* (*storage engine*). Tipe

table yang digunakan secara *default* adalah MyISAM yang memiliki kelebihan akses data yang cepat tetapi tidak mendukung fitur *foreign key*. Jika pengguna ingin ada dukungan *foreign key* pada *table* , pengguna dapat menggunakan tipe *table* InnoDB.

2.15 Microsoft Visual Studio

Microsoft Visual Studio merupakan sebuah perangkat lunak lengkap yang dapat digunakan untuk melakukan pengembangan aplikasi, baik itu aplikasi bisnis, aplikasi personal, ataupun komponen aplikasinya, dalam bentuk aplikasi *console*, aplikasi Windows, ataupun aplikasi *Web*. Visual Studio mencakup *compiler*, SDK, *Integrated Development Environment* (IDE), dan dokumentasi yang umumnya berupa MSDN *Library*. *Compiler* yang dimasukkan ke dalam paket *Visual Studio* antara lain *Visual C++, Visual C#, Visual Basic, Visual Basic .NET, Visual InterDev, Visual J++, Visual J#, Visual FoxPro,* dan *Visual SourceSafe.*

Microsoft Visual Studio dapat digunakan untuk mengembangkan aplikasi dalam *native code* (dalam bentuk bahasa mesin yang berjalan di atas Windows) ataupun *managed code* (dalam bentuk *Microsoft Intermediate Language* di atas .NET *Framework*). Selain itu, Visual Studio juga dapat digunakan untuk mengembangkan aplikasi *Silverlight*, aplikasi Windows Mobile (yang berjalan di atas .NET *Compact Framework*) [23].