BAB 2
LANDASAN TEORI

2.1 Pengenalan Suara

Pengenalan suara merupakan salah satu upaya untuk dapat mengenali atau mengidentifikasi suara sehingga dapat dimanfaatkan untuk berbagai sistem atau aplikasi [2]. Pengenalan suara dapat dikategorikan menjadi dua jenis, yaitu :

2.1.1 Speech recognition

Speech recognition adalah proses identifikasi suara berdasarkan kata yang diucapkan tanpa memperdulikan siapa pembicaranya. Parameter yang dibandingkan ialah tingkat penekanan suara yang kemudian akan dicocokkan dengan *template database* yang tersedia [1].

2.1.2 Speaker recognition

Speaker recognition adalah suatu proses pengenalan pembicara dari informasi yang terkandung dalam gelombang suara yang diinputkan. *Speaker recognition* dibagi menjadi dua bagian, yaitu [3] :

a. Speaker Verification

Speaker verification adalah proses pemeriksaan ulang seorang pembicara, sehingga perlu diketahui terlebih dahulu identitas pembicara tersebut berasarkan data yang telah dimasukkan (misalnya *username* dan *password*). *Speaker verification* membandingkan fitur suara dari seorang pembicara secara langsung dengan fitur pembicara tertentu yang ada didalam *database*. Bila hasil perbandingan tersebut lebih besar atau sama dengan batasan tertentu (*threshold*), maka pembicara tersebut diterima, bila tidak maka ditolak.

b. Speaker Identification

Speaker identification adalah proses untuk mencari dan mendapatkan identitas dari seorang pembicara dengan membandingkan fitur suara yang dimasukkan dengan semua fitur suara pembicara yang ada didalam

database. Berbeda dengan *speaker verification*, proses ini melakukan perbandingan dengan *one-to-many* (1:N). Bila dibandingkan dengan *speaker verification*, *speaker identification* jauh lebih sulit. Hal ini disebabkan, bila jumlah pembicara yang terdaftar dalam *database* semakin besar, maka kemungkinan terjadinya kesalahan dalam pengambilan keputusan semakin besar pula.

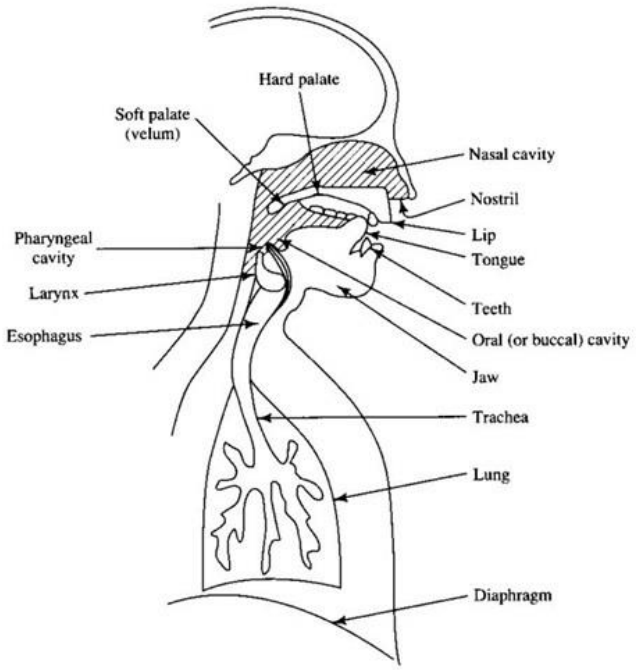
Lebih jauh *speaker recognition* dapat diklasifikasikan menjadi *text-dependent* dan *text-independent*. Pada *text-dependent*, kata-kata yang diucapkan oleh pembicara sudah ditentukan sebelumnya. Sedangkan pada *text-independent*, tidak ada asumsi kata-kata yang akan diucapkan oleh pembicara. Sehingga sistem harus memodelkan fitur-fitur umum dari suara seorang pembicara.

Pada *speaker recognition* ada beberapa faktor yang dapat menyebabkan kesalahan dalam proses verifikasi dan identifikasi, antara lain :

1. Kesalahan dalam pengucapan (*misspoken*) dan pembacaan (*missread*) frasa.
2. Keadaan emosional yang ekstrim (misalnya stres).
3. Pergantian penempatan *microphone* (*intrasession* atau *intersession*).
4. Kekurangan atau ketidak-konsistenan akustik dari ruangan (misalnya *multipath* dan *noise*).
5. *Channel missmatch* (misalnya penggunaan *microphone* yang berbeda channel dalam perekaman dan verifikasi).
6. Sakit (misalnya flu yang dapat meruba *vocal tract*).
7. *Aging* (model *vocal tract* dapat berubah berdasarkan usia).

2.2 Sinyal Suara

Sinyal suara adalah sinyal yang dihasilkan oleh suara manusia dan biasanya mempunyai frekuensi kerja antara 0 sampai 5 KHz. Bentuk gelombang sinyal suara mempunyai bentuk yang unik. Sinyal suara mempunyai unsur bunyi terkecil yang disebut dengan *pitch*. Panjang *pitch* berkisar 10 ms. *Pitch* manusia berbeda antara satu dengan yang lainnya, terutama ditinjau dari jenis kelamin (laki-laki atau perempuan) serta usia [3].



Gambar 2.1 Organ Produksi Suara Manusia

Organ tubuh yang terlibat pada proses produksi suara adalah : paru-paru (*lung*), tenggorokan (*trachea*), laring (*larynx*), faring (*pharynx*), pita suara (*vocal cord*), rongga mulut (*oral cavity*), rongga hidung (*nasal cavity*), lidah (*tongue*), dan bibir (*lips*), seperti yang dapat dilihat pada Gambar 2.1.

Organ tubuh ini dapat dikelompokkan menjadi tiga bagian utama, yaitu : *vocal tract* (berawal di awal bukaan pita suara atau *glottis*, dan berakhir di bibir), *nasal tract* (dari *velum* sampai *nostril*), dan *source generator* (terdiri dari paru-paru, tenggorokan, dan *larynx*). Ukuran *vocal tract* bervariasi untuk setiap individu, namun laki-laki dewasa rata-rata panjangnya sekitar 17 cm. Luas dari *vocal tract* juga bervariasi antara 0 (ketika seluruhnya tertutup) hingga sekitar 20 cm. Ketika *velum*, organ yang memiliki fungsi sebagai pintu penghubung antara *vocal tract* dengan *nasal tract* terbuka, maka secara akustik *nasal tract* akan bergandengan dengan *vocal tract* untuk menghasilkan suara nasal.

2.2.1 Klasifikasi Sinyal Suara

Berdasarkan sinyal yang dihasilkan pada proses produksi suara, sinyal suara ucapan dapat dibagi menjadi tiga bagian yaitu *silence*, *unvoiced*, dan *voiced* [3]:

a. Sinyal *silence* : sinyal pada saat tidak terjadi proses produksi suara ucapan, dan sinyal yang diterima oleh pendengar dianggap sebagai bising latar belakang.
b. Sinyal *unvoiced* : terjadi pada saat pita suara tidak bergetar, dimana sinyal eksitasi berupa sinyal *random*.
c. 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.

2.3 Konversi Sinyal Analog Menjadi Digital

Sinyal suara yang akan diproses masih berbentuk sinyal analog sehingga perlu diubah menjadi sinyal digital sehingga dapat dilakukan pengolahan secara digital, sinyal suara tersebut harus dikonversi menjadi sinyal digital, berupa urutan angka dengan tingkat presisi tertentu yang dinamakan *analog to digital conversion* dengan menggunakan *analog-to-digital converter* (ADC). Konsep Kerja ADC terdiri dari tiga proses [8]:

1. Proses Sampling yaitu mengubah sinyal waktu kontinyu menjadi sinyal waktu diskrit.
2. Proses kuantisasi yaitu mengubah amplitudo sinyal kontinyu menjadi amplitudo diskrit, hasil proses pencuplikan dan kuantisasi adalah sinyal digital (waktu diskrit, amplitudo diskrit) dari sinyal analog (waktu kontinyu, amplitudo kontinyu).
3. Proses pengkodean yaitu mengkodekan sinyal digital dalam representasi binernya (digital).

2.4 Mel Frequency Cepstrum Coefficients

Mel Frequency Cepstrum Coefficients (MFCC) merupakan salah satu metode yang banyak digunakan dalam bidang *speech technology*, baik *speaker recognition* maupun *speech recognition*. Metode ini digunakan untuk melakukan *feature*

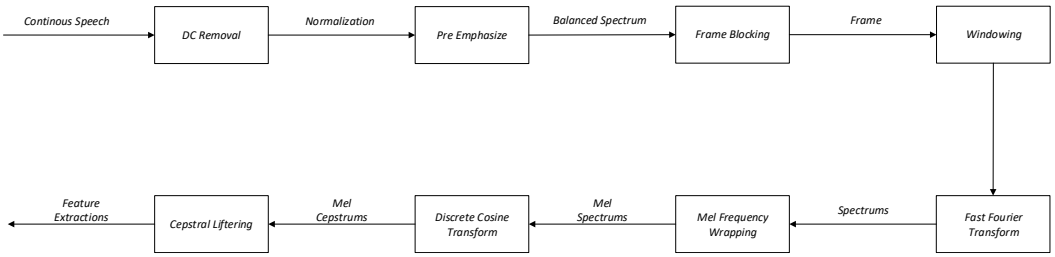
extraction yang merupakan sebuah proses yang mengkonversi sinyal suara menjadi beberapa parameter. Keunggulan dari metode ini adalah [3]:

1. Mampu untuk menangkap karakteristik suara yang sangat penting bagi pengenalan suara. Atau dengan kata lain, mampu menangkap informasi-informasi penting yang terkandung dalam sinyal suara.
2. Menghasilkan data seminimal mungkin tanpa menghilangkan informasi-informasi penting yang ada.
3. Mengadaptasi organ pendengaran manusia dalam melakukan persepsi terhadap sinyal suara.

Perhitungan yang dilakukan MFCC menggunakan dasar dari perhitungan *short-term analysis*. Hal ini dilakukan menginat sinyal suara yang bersifat *quest stationary*. Sinyal suara pada periode waktu yang cukup pendek (sekitar 10-30 ms) akan menunjukkan karakteristik sinyal suara yang stationary. Tetapi bila dilihat 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, dimana sinyal suara akan difilter secara linear untuk frekuensi rendah (dibawah 1KHz) dan secara logaritmik untuk frekuensi tinggi (diatas 1 KHz)

Pada Gambar 2.2 dapat dilihat blok diagram proses MFCC. *Continous Speech* merupakan inputan suara dengan kata yang telah ditentukan oleh sistem, kemudian diproses *DC removal* untuk mendapatkan nilai normal dari sinyal, kemudian diproses *Pre-Emphasize* untuk memperbaiki sinyal suara dari gangguan *noise*, kemudian proses *Frame Blocking* bertujuan untuk membagi sampel sinyal suara menjadi beberapa *frame*, kemudian proses *Windowing* bertujuan untuk mengurangi efek diskontinuitas pada ujung-ujung frame yang dihasilkan oleh proses *Frame Blocking*, kemudian proses *Fast Fourier Transform* bertujuan untuk mendapatkan sampel sinyal suara dalam frekuensi domain, kemudian proses *Filterbank* bertujuan untuk mengetahui ukuran energi dari setiap frekuensi *band*, kemudian proses *Discrete Cosine Transform* untuk mendapatkan *Mel Cepstrums*, langkah terakhir proses *Cepstral Liftering* bertujuan untuk menghaluskan *Spectrum Signal*.



Gambar 2.2 Blok Diagram MFCC

2.4.1 DC Removal

Proses DC removal dilakukan untuk menghitung rata-rata dari data sampel suara, dan mengurangkan nilai setiap sampel suara dengan nilai rata-rata tersebut. Tujuannya adalah mendapat normalisasi dari data suara input [9]. Sehingga untuk mendapatkan nilai hasil DC Removal dapat digunakan persamaan (2.1).

$$dr_i = s_i - \bar{x}, 0 \leq i \leq N - 1 \quad (2.1)$$

Keterangan :

dr_i : titik sinyal hasil proses *DC Removal*.

s_i : titik sinyal ke-i.

$\bar{x}$: nilai rata-rata titik sinyal.

i : 0, 1, 2, …, N-1

N : panjang sinyal (banyak titik sinyal).

Dimana untuk mendapatkan nilai rata-rata dari titik sinyal input dapat digunakan persamaan (2.2) berikut.

$$\bar{x} = \frac{s_0 + s_1 + \cdots + s_{N-1}}{N-1} \quad (2.2)$$

Keterangan :

$\bar{x}$: nilai rata-rata titik sinyal.

s_i : input sinyal ke-i.

i : 0, 1, 2, …, N-1

N : panjang sinyal (banyak titik sinyal).

2.4.2 Pre-Emphasize Filtering

Pre-Emphasize Filtering merupakan salah satu jenis *filter* yang sering digunakan sebelum sebuah sinyal diproses lebih lanjut. *Filter* ini mempertahankan frekuensi-frekuensi tinggi pada sebuah spektrum yang umumnya tereliminasi pada saat proses produksi suara. Tujuan dari *pre-emphasize filter* ini adalah [3]:

1. Mengurangi *Noise Ratio* pada sinyal, sehingga dapat meningkatkan kualitas sinyal.
2. Menyeimbangkan spektrum dari *voice sound*. Pada saat memproduksi *voice sound*, glotis manusia menghasilkan sekitar -12 dB *octave slope*. Namun ketika energi akustik tersebut dikeluarkan melalui bibir, terjadi peningkatan sebesar +6 dB. Sehingga sinyal yang terekam di *microphone* adalah sekitar -6 dB *octave slope*.

Bentuk yang paling umum digunakan dalam *pre-emphasize filter* adalah sebagai berikut :

$$pf_i = dr_i - (dr_{i-1} \times \alpha) \quad (2.3)$$

Keterangan :

pf_i : sinyal hasil *pre-emphasize filter* ke-i.

dr_i : sinyal sebelum *pre-emphasize filter* ke-i.

α : *pre-emphasize filter*.

i : 0, 1, 2, …, N-1

N : panjang sinyal (banyaknya titik sinyal).

Pada umumnya nilai α yang paling sering digunakan adalah antara 0.9 sampai dengan 1. Setelah dilakukan perhitungan *pre-emphasize filter* maka sinyal suara hasil *pre-emphasize filter* dapat direpresentasikan persamaan (2.4).

$$p_i = dr_i + pf_i \quad (2.4)$$

Keterangan :

p_i : titik sinyal baru ke-i.

dr_i : sampel sebelum *pre-emphasize* ke-i (hasil DC Removal)

pf_i : *pre-emphasize filter* pada sampel ke-i.

i : 0, 1, 2, …, N-1

2.4.3 Frame Blocking

Pada langkah ini, sinyal ucapan kontinu diblok menjadi *frame-frame* N sampel. *Frame* pertama terdiri dari N sampel pertama. *Frame* kedua dengan M sampel setelah *frame* pertama, dan *overlap* dengan N-M sampel. Dengan cara yang sama, *frame* ketiga dimulai 2M sampel setelah *frame* pertama (atau M sampel setelah *frame* kedua) dan *overlap* dengan N-2M sampel. Proses ini berlanjut hingga semua sinyal suara dihitung dalam satu atau banyak *frame* [3].

Proses *framing* ini dilakukan secara terus menerus sampai seluruh sinyal dapat terproses. Selain ini, proses ini umumnya dilakukan secara *overlapping* untuk setiap *frame*-nya. Panjang daerah *overlap* yang umumnya digunakan adalah kurang lebih 30% sampai 50% dari panjang *frame*. Adapun proses menghitung jumlah frame blocking dapat dilihat pada persamaan (2.5).

$$jumlah\ frame = ((NPE - SP)/M) + 1) \quad (2.5)$$

Keterangan :

NPE : jumlah sampel (banyak sampel yang akan disegmentasi)

SP : *Sample Point* dalam tiap *frame* (*Sample Rate* × panjang *frame* dalam detik (s))

M : *SP*/2 (*Overlapping*)

Pada persamaan (2.5) tersebut *sample rate* merupakan frekuensi sampling yang menyatakan jumlah titik sampel yang ada dalam file audio. *Sample point* didapatkan dari hasil *sample rate* dikalikan dengan panjang frame yang diinginkan dalam detik, misalnya frame yang diinginkan adalah 30 ms maka *sample point* adalah jumlah *sample rate* × 0,03. *Sample point* (SP) dapat juga disebut sebagai panjang *frame*. Adapun persamaan untuk nilai *sample point* dapat dilihat pada persamaan (2.6).

$$SP = Sample\ Rate \times durasi\ (s) \quad (2.6)$$

2.4.4 Windowing

Proses *windowing* ini dilakukan pada setiap *frame* hasil dari proses *framing*. Hal ini dilakukan untuk mengurangi terjadinya kebocoran spektral atau *aliasing* [3]. *Aliasing* adalah sinyal baru dimana memiliki frekuensi yang berbeda dengan sinyal aslinya. Efek ini dapat terjadi karena rendahnya jumlah *sampling rate*, ataupun karena proses *frame blocking* dimana menyebabkan sinyal menjadi *discontinue*. Untuk mengurangi kemungkinan terjadinya kebocoran spektral, maka hasil dari proses *framing* harus melewati proses *window* [9].

Persamaan (2.7) berikut menunjukkan fungsi *window* terhadap sinyal suara yang diinputkan.

$$w_i = p_i \times fw_i \tag{2.7}$$

Keterangan :

w_i : nilai titik sinyal hasil *windowing*.

p_i : hasil *pre-emphasize* ke-1 dari *frame* ke-i.

fw_i : fungsi *window* ke-1 untuk *frame* ke-i.

NPE : banyaknya *sample point* pada *frame*.

1 : *sample point* ke-1 pada *frame* (dimana nilai n = 0, 1, …, NPE-1).

Fungsi *window* yang paling sering digunakan dalam aplikasi *speaker recognition* adalah *Hamming Window*. Berikut persamaan dari *Hamming Window*:

$$fw_i = 0.54 - 0.46 \cos\frac{2\pi l}{SP-1} \tag{2.8}$$

Keterangan :

fw_i : fungsi *hamming window* untuk titik sinyal ke-1 pada *frame* ke-i.

SP : *Sample Point* (dapat disebut sebagai panjang *frame*).

l : 0, 1, …, SP-1

2.4.5 Fast Fourier Transform (FFT)

Setiap *frame* dikonversi dari domain waktu ke domain frekuensi untuk mendapatkan spektrum frekuensinya. Hal ini dilakukan untuk mempermudah

komputasi dan analisa. Metode yang dilakukan untuk mendapatkan spektrum frekuensi adalah *Fast Fourier Transform* [3].

FFT adalah sebuah algoritma cepat untuk implementasi *Discrete Fourier Transform* (DFT) yang dioperasikan pada sebuah sinyal waktu-diskrit yang terdiri dari N sampel sebagai berikut [2]:

$$f_k = \sum_{i=0}^{SP-1}(w_i \cos\frac{2\pi ik}{SP}) - j\sum_{l=0}^{SP-1}\left(w_i \sin\frac{2\pi ik}{SP}\right), 0 \le k \le SP-1 \qquad (2.9)$$

Keterangan :

f_k : spektrum FFT ke-k.

N : jumlah sampel yang akan diproses (N ∈ N).

l : 0, 1, 2, …, SP-1 (jumlah sampel dalam *frame*).

k : 0, 1, 2, …, SP-1 (variabel frekuensi diskrit, merupakan hasil FFT).

w_i : nilai titik sinyal ke-n (hasil *windowing* ke-n pada *frame*).

j : bilangan imajiner.

Untuk menghitung hasil dari FFT digunakan persamaan (2.10).

$$\left|fft_k = [R^2 + I^2]^{\frac{1}{2}}\right| \qquad (2.10)$$

Keterangan :

R = bilangan real (hasil perhitungan $w_i \cos\frac{2\pi lk}{N}$).

I = bilangan imajiner (hasil perhitungan $j(w_i \sin\frac{2\pi lk}{N})$).

Dengan persamaan (2.10) suatu sinyal suara dalam domain waktu dapat kita cari frekuensi pembentuknya. Hal inilah tujuan penggunaan analisa fourier pada data suara, yaitu untuk merubah data dari domain waktu menjadi data spektrum di domain frekuensi. Untuk pemrosesan sinyal suara, hal ini sangatlah menguntungkan karena data pada domain frekuensi dapat diproses dengan lebih mudah dibandingkan data pada domain waktu, karena pada domain frekuensi, keras lemahnya suara tidak seberapa berpengaruh [8].

2.4.6 Mel-Frequency Wrapping

Studi psikofisik telah menunjukkan bahwa persepsi manusia tentang frekuensi suara untuk sinyal ucapan tidak mengikuti skala linier. Jadi, untuk setiap nada dengan frekuensi sesungguhnya f, dalam Hz, sebuah pola diukur dalam sebuah skala yang disebut 'mel'. Skala 'mel frekuensi' adalah skala frekuensi linier di bawah 1000 Hz dan skala logaritmik di atas 1000 Hz. *Mel Frequency Wrapping* umumnya dilakukan dengan menggunakan *filterbank*. *Filterbank* adalah salah satu bentuk dari *filter* yang dilakukan dengan tujuan untuk mengetahui ukuran energi dari *frequency band* tertentu dalam sinyal suara [2].

Skala ini didefinisikan oleh Stanley Smith, John Volkman dan Edwin Newman sebagai :

$$mel(f) = 2595 \times \log_{10}(1 + \frac{f}{700}) \quad (2.11)$$

Keterangan :

mel(f) : frekuensi *Mel Scale*.

f : frekuensi yang digunakan.

$$mel^{-1}(f) = 700 \times (10^{\frac{f}{2595}} - 1) \quad (2.12)$$

Keterangan :

mel^{-1}*(f)* : frekuensi Mel Scale invers.

f : frekuensi yang digunakan.

Sebuah pendekatan untuk simulasi spektrum dalam skala mel adalah dengan menggunakan *filterbank* yang diletakan secara seragam dalam skala mel.

$$fb_m = \left(\frac{SP}{FS}\right) \times mel^{-1}\left(mel(Nlow) + m\frac{mel(Nhigh) - mel(Nlow)}{Nfbank}\right) \quad (2.13)$$

Keterangan :

fb_m : hasil *boundary point filter ke-m*.

SP : *Sample Point* (banyaknya sampel point).

FS : frekuensi sampling.

m : jumlah *triangular filterbank*.

$$H_{mk} = \begin{cases} 0, untuk\ k < fb_m \\ \frac{k - fb_{m-1}}{fb_m - fb_{m-1}}, untuk\ fb_{m-1 \le k < fb_m} \\ \frac{fb_{m+1} - k}{fb_{m+1} - fb_{m-1}}, untuk fb_m \le k < fb_{m+1} \\ 0, untuk\ k > fb_{m+1} \end{cases} \quad (2.14)$$

Keterangan :

fb_m : hasil *boundary point filter*.

H_{mk} : koefisien dari *mel filterbank*.

$$MF_m = \sum_{k=0}^{SP-1} fft_k \times H_{mk} \quad (2.15)$$

Keterangan :

MF_m : hasil *Mel Frequency Wrapping* ke-m.

fft_k : hasil FFT ke-k.

SP : banyaknya hasil FFT.

m : 0, 1, 2, …, *Nfbank*

2.4.7 Discrete Cosine Transform (DCT)

DCT merupakan langkah terakhir dari proses utama MFCC *feature extraction*. Konsep dasar dari DCT adalah mendekorelasikan *mel cepstrums* sehingga menghasilkan representasi yang baik dari *property* spektral lokal. Pada dasarnya konsep dari DCT sama dengan *inverse fourier transform*. Namun hasil dari DCT mendekati PCA (*Principal Component Analysis*). PCA adalah metode statistika klasik yang digunakan secara luas dalam analisa data dan kompresi. Hal inilah yang menyebabkan seringkali DCT menggantikan inverse fourier transform dalam proses MFCC *feature extraction* [10].

Berikut adalah formula yang digunakan untuk menghitung DCT :

$$C_{koef} = \sqrt{\frac{2}{Nfbank}} \sum_{m=0}^{Nfbank-1} (\log MF_m) \cos\left[koef\left(\frac{2m-1}{2}\right)\frac{\pi}{Nkoef}\right] \quad (2.16)$$

Keterangan :

MF_m : hasil Mel Frequency Wrapping ke-m.

M : 0, 1, 2, …, SP-1 (indeks hasil filterbank ke-m).

Nfbank : jumlah *Mel Filterbank* (40).

Nkoef : jumlah koefisien yang diharapkan (pada penelitian ini 13).

Koefisien ke nol dari DCT pada umumnya akan dihilangkan, walaupun sebenarnya mengindikasikan energi dari *frame* sinyal tersebut. Hal ini dilakukan karena, berdasarkan penelitian-penelitian yang pernah dilakukan, koefisien ke nol ini tidak *reliable* terhadap *speaker recognition* [8].

2.4.8 Cepstral Liftering

Hasil dari fungsi DCT adalah *cepstrum* yang sebenarnya sudah merupakan hasil akhir dari proses MFCC. Tetapi untuk meminimalisasi sensitifitas dari koefisien MFCC yang telah didapat, maka *cepstrum* hasil dari DCT akan diolah lagi pada blok *cepstral liftering*. Sensitifitas tersebut adalah *low order* dari *cepstral coefficients* sangat sensitif terhadap *spectral slope*, sedangkan bagian *high ordernya* sangat sensitif terhadap *noise*.

Cepstral liftering dapat dilakukan dengan mengimplementasikan fungsi window terhadap hasil koefisien MFCC.

$$Cepstral_n = \left\{C \times 1 + \frac{L}{2}\sin\left(\frac{n\pi}{L}\right)\right\} \quad (2.17)$$

Keterangan :

L : jumlah *cepstral coefficients*.

n : indeks dari *cepstral coefficients.*

C : hasil dct.

Cepstral liftering menghaluskan spektrum hasil dari *main processor* sehingga dapat digunakan lebih baik untuk *pattern matching* [9].

2.5 Jaringan Syaraf Tiruan

Jaringan Syaraf Tiruan adalah paradigma pemrosesan suatu informasi yang terinspirasi oleh sistem sel syaraf biologi. JST dibentuk sebagai generalisasi model matematika dari jaringan syaraf biologi, dengan asumsi bahwa :

1. Pemrosesan informasi terjadi pada banyak elemen sederhana (*neuron*).

2. Sinyal dikirimkan diantara *neuron-neuron* melalui penghubung-penghubung.
3. Penghubung antar neuron memiliki bobot yang akan memperkuat atau memperlemah sinyal.
4. Untuk menentukan *output*, setiap *neuron* menggunakan fungsi aktivasi (biasanya bukan fungsi linier) yang dikarenakan pada jumlahan *input* yang diterima. Besarnya *output* ini selanjutnya dibandingkan dengan suatu batas ambang.

Neuron biologi merupakan sistem yang "*fault tolerant*" dalam 2 hal. Pertama, manusia dapat mengenali sinyal input yang agak berbeda dari yang pernah kita terima sebelumnya. Sebagai contoh, manusia sering dapat mengenali seseorang yang wajahnya pernah dilihat dari foto atau dapat mengenali seseorang yang wajahnya agak berbeda karena sudah lama tidak menjumpainya. Kedua, tetap mampu bekerja dengan baik. Jika sebuah *neuron* rusak, *neuron* lain dapat dilatih untuk menggantikan fungsi *neuron* yang rusak tersebut [11].

Hal yang ingin dicapai dengan melatih JST adalah untuk mencapai keseimbangan antara kemampuan memorisasi dan generalisasi. Yang dimaksud kemampuan memorisasi adalah kemampuan JST untuk mengambil kembali secara sempurna sebuah pola yang telah dipelajari. Kemampuan generalisasi adalah kemampuan JST untuk menghasilkan respon yang bisa diterima terhadap pola-pola yang sebelumnya telah dipelajari. Hal ini sangat bermanfaat bila pada suatu saat ke dalam JST itu di inputkan informasi baru yang belum pernah dipelajari, maka JST itu masih akan tetap dapat memberikan tanggapan yang baik, memberikan keluaran yang mendekati.

Jaringan syaraf tiruan menyerupai otak manusia dalam 2 hal, yaitu :

1. Pengetahuan diperoleh jaringan melalui proses belajar.
2. Kekuatan hubungan antar sel syaraf (*neuron*) yang dikenal sebagai bobot-bobot sinaptik digunakan untuk menyimpan pengetahuan.

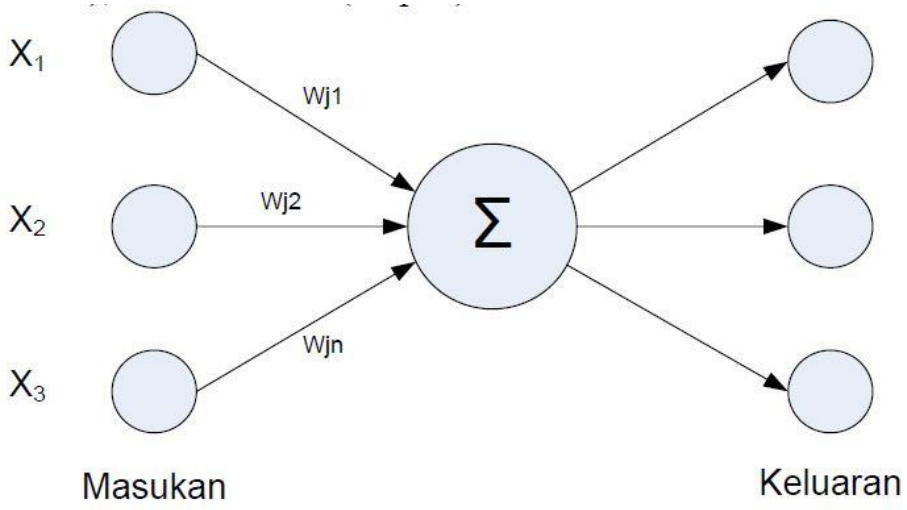
Jaringan syaraf tiruan ditentukan oleh 3 hal :

1. Pola hubungan antar *neuron* (disebut arsitektur jaringan).

2. Metode untuk menentukan bobot penghubung (disebut metode *training/learning*).
3. Fungsi aktivasi, yaitu fungsi yang digunakan untuk menentukan keluaran suatu *neuron*.

2.5.1 Model Neuron

Satu sel syaraf terdiri dari 3 bagian, yaitu : fungsi penjumlahan (*summing function*), fungsi aktivasi (*activation function*), dan keluaran (*output*).



Gambar 2.3 Model Neuron

Jika dilihat pada Gambar 2.3, *neuron* buatan diatas mirip dengan sel *neuron* biologis. Informasi (*input*) akan dikirim ke *neuron* dengan bobot tertentu. *Input* ini akan diproses oleh suatu fungsi yang akan menjumlahkan nilai-nilai bobot yang ada. Hasil penjumlahan kemudian akan dibandingkan dengan suatu nilai ambang (*threshold*) tertentu melalui fungsi aktivasi setiap *neuron*. Apabila *input* tersebut melewati suatu nilai ambang tertentu, maka *neuron* tersebut akan mengirimkan *output* melalui bobot-bobot *output*nya ke semua *neuron* yang berhubungan dengannya.

Sehingga dapat disimpulkan bahwa *neuron* terdiri dari 3 elemen pembentuk, yaitu :

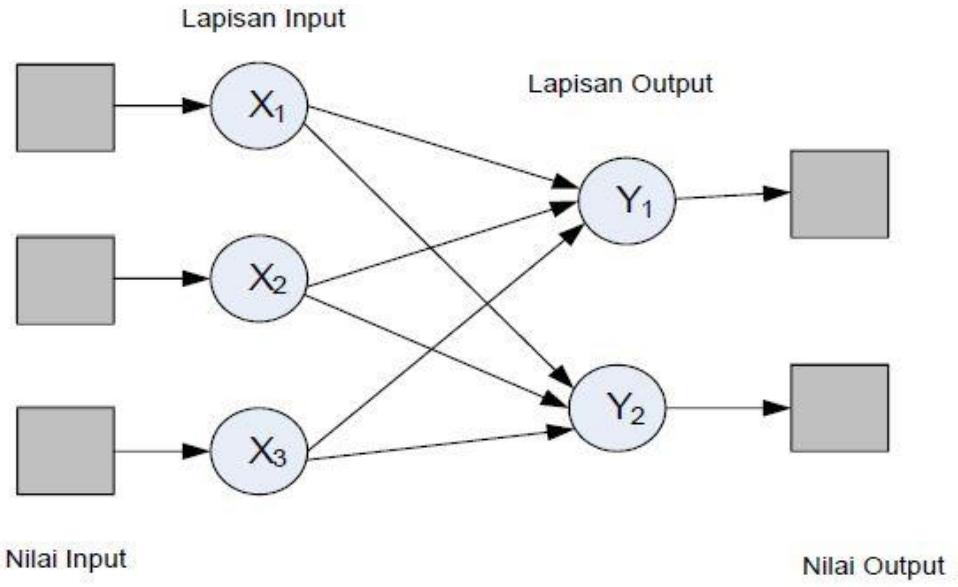
1. Himpunan unit-unit yang dihubungkan dengan jalur koneksi. Jalur-jalur tersebut memiliki bobot yang berbeda-beda. Bobot yang bernilai positif akan memperkuat sinyal dan yang bernilai negatif akan memperlemah sinyal yang dibawa. Jumlah, struktur, dan pola hubungan antar unit-unit tersebut akan menentukan arsitektur jaringan.
2. Suatu unit penjumlahan yang akan menjumlahkan *input-input* sinyal yang sudah dikalikan dengan bobotnya.
3. Fungsi aktivasi yang akan menentukan apakah sinyal dari *input neuron* akan diteruskan ke *neuron* lain atau tidak.

2.5.2 Arsitektur Jaringan Syaraf Tiruan

Jaringan syaraf tiruan memiliki beberapa arsitektur jaringan yang sering digunakan dalam berbagai aplikasi. Arsitektur jaringan syaraf tiruan tersebut, antara lain [11].

1. Jaringan Layar Tunggal (*Single Layer Network*).

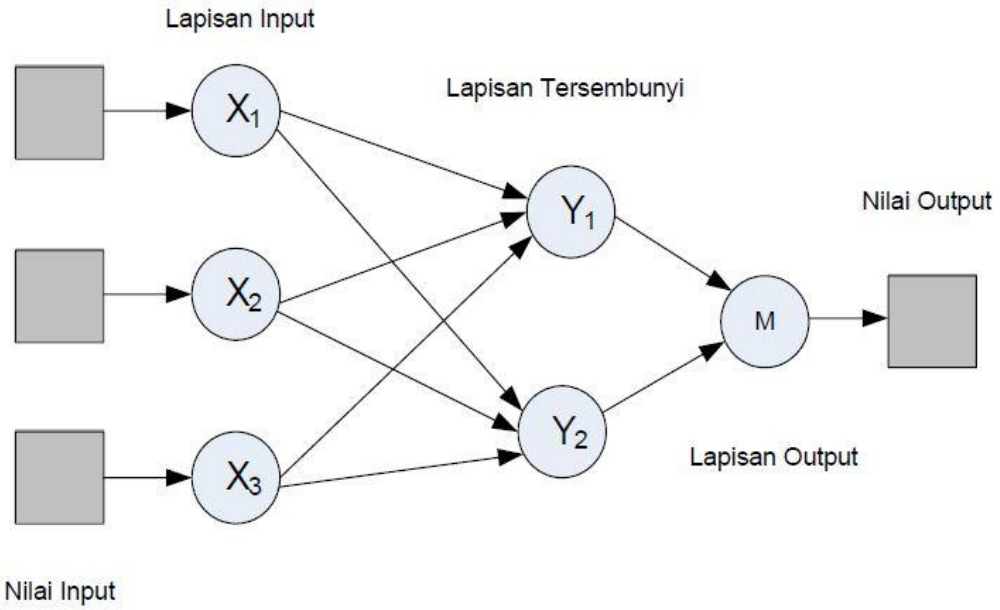
   Jaringan dengan lapisan tunggal terdiri dari 1 layer *input* dan 1 layer *output*. Setiap neuron/unit yang terdapat di dalam lapisan/layer *input* selalu terhubung dengan setiap neural yang terdapat pada layer *output*. Jaringan ini hanya menerima *input* kemudian secara langsung akan mengolahnya menjadi *output* tanpa harus melalui lapisan tersembunyi. Contoh algoritma jaringan syaraf tiruan yang menggunakan metode ini yaitu : ADALINE, Hopfiled, *Perceptron*.



Gambar 2.4 Arsitektur Layer Tunggal

2. Jaringan Layar Jamak (*Multi Layer Network*).

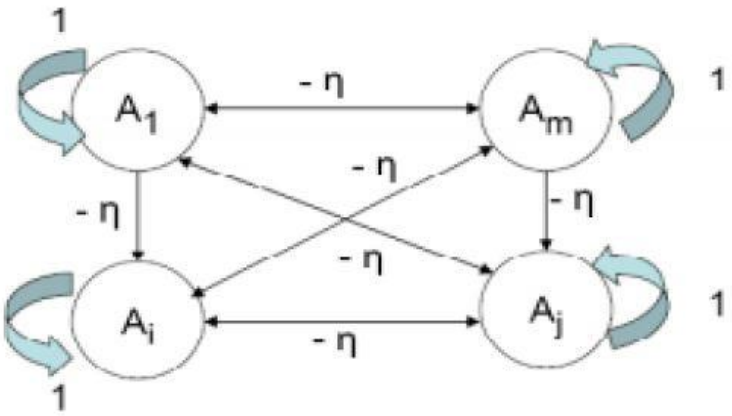
   Jaringan dengan lapisan jamak memiliki ciri khas tertentu yaitu memliki 3 jenis layer yakni layer *input*, layer *output*, layer tersembunyi (*hidden layer*). Jaringan dengan banyak lapisan ini dapat menyelesaikan permasalahan yang kompleks dibandingkan jaringan dengan lapisan tunggal. Namun proses pelatihan sering membutuhkan waktu yang cenderung lama. Contoh algoritma jaringan syaraf tiruan yang menggunkanan metode ini yaitu : MADALINE, *backpropagation*, *neocognitron*.



Gambar 2.5 Arsitektur Layer Jamak

3. Jaringan dengan lapisan kompetitif.

   Pada jaringan ini, sekumpulan neuron bersaing untuk mendapatkan hak menjadi aktif. Contoh algoritma yang menggunakan metode ini adalah : LVQ.



Gambar 2.6 Arsitektur Layer Kompetitif

2.5.3 Metode Pelatihan/Pembelajaran Jaringan Syaraf Tiruan

Cara berlangsungnya pembelajaran atau pelatihan jaringan syaraf tiruan dikelompokkan menjadi 3, yaitu [11]:

1. *Supervised Learning* (Pembelajaran Terawasi).

   Pada metode ini, setiap pola yang diberikan kedalam jaringan syaraf tiruan telah diketahui *output*nya. Selisih antara pola *output* aktual (*output* yang dihasilkan) dengan pola *output* yang dikehendaki (*output target*) yang disebut *error* digunakan untuk mengoreksi bobot jaringan syaraf tiruan sehingga jaringan syaraf tiruan mampu menghasilkan *output* sedekat mungkin dengan pola target yang telah diketahui oleh jaringan syaraf tiruan. Contoh algoritma jaringan syaraf tiruan yang menggunakan metode ini adalah : Hebbian, *Perceptron*, ADALINE, Boltzman, Hopfield, *Backpropagation*.

2. *Unsupervised Learning*(Pembelajaran Tidak Terawasi).

   Pada metode ini, tidak memerlukan target output. Pada metode ini tidak dapat ditentukan hasil seperti apakah yang diharapkan selama proses pembelajaran. Selama proses pembelajaran, nilai bobot disusun dalam suatu jarak (*range*) tertentu pada nilai *input* yang diberikan. Tujuan pembelajaran ini adalah mengelompokkan unit-unit yang hampir sama dalam msuatu area tertentu. Pembelajaran ini biasanya sangat cocok untuk klasifikasi pola. Contoh algoritma jaringan syaraf tiruan yang menggunakan metode ini adalah : *Competitive*, Hebbian, Kohonen, LVQ (*Learning Vector Quantization*), *Neocognitron*.

3. *Hybrid Learning* (Pembelajaran Hibrida).

   Merupakan kombinasi dari metode pembelajaran *Supervised Learning* dan *Unsupervised Learning*, sebagian dari bobot-bobotnya ditentukan melalui pembelajaran terawasi dan sebagian lainnya melalui pembelajaran tak terawasi. Contoh algoritma jaringan syaraf tiruan yang menggunakan metode ini adalah : algoritma RBF. Metode algoritma yang baik dan sesuai dalam melakukan pengenalan pola-pola gambar adalah algoritma *Backpropagation* dan *Perceptron*. Untuk mengenali teks berdasarkan tipe *font* akan digunakan algoritma *Backpropagation*.

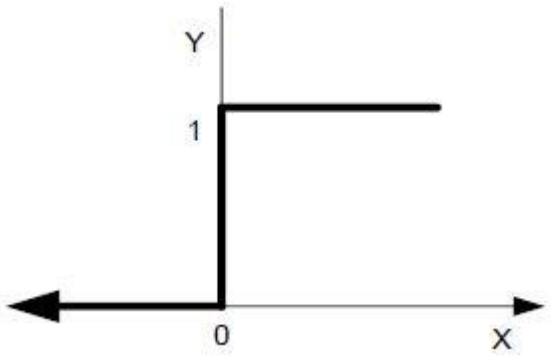
2.5.4 Fungsi Aktivasi Jaringan Syaraf Tiruan

Dalam jaringan syaraf tiruan, fungsi aktivasi digunakan untuk menentukan keluaran suatu *neuron*. Argumen fungsi aktivasi adalah *net* masukan (kombinasi linier masukan dan bobotnya).

Beberapa fungsi aktivasi yang digunakan adalah :

a. Fungsi *Threshold* (batas ambang).

Fungsi *Threshold* merupakan fungsi *threshold* biner. Untuk kasus bilangan bipolar, maka angka 0 diganti dengan angka -1. Adakalanya dalam jaringan syaraf tiruan ditambahkan suatu unit masukkan yang nilainya selalu 1. Unit tersebut dikenal dengan bias. Bias dapat dipandang sebagai sebuah input yang nilainya selalu 1. Bias berfungsi untuk mengubah *threshold* menjadi = 0.



Gambar 2.7 Fungsi aktivasi Threshold

$$F(x) = \begin{cases} 1\ Jika\ x\ \geq a \\ 0\ Jika\ x\ < a \end{cases} \quad (2.18)$$

b. Fungsi Sigmoid.

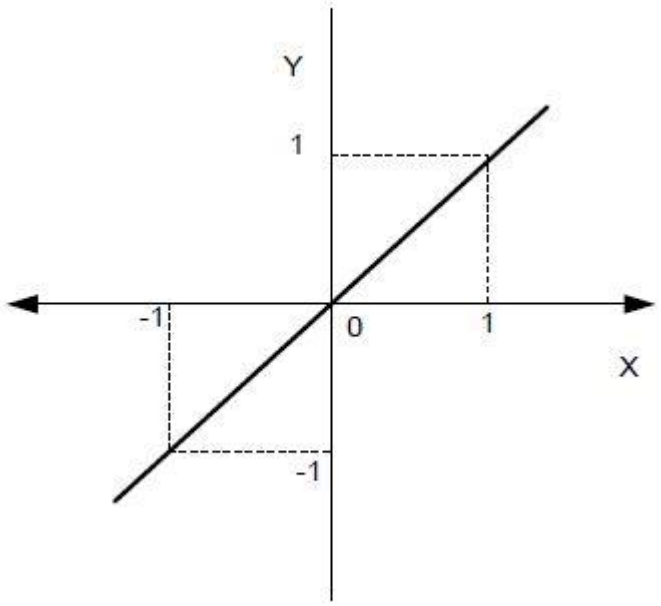
Fungsi ini sering digunakan karena nilai fungsinya yang sangat mudah untuk di diferensiakan.

$$F(x) = \frac{1}{1+e^{-x}} \quad (2.19)$$

c. Fungsi Identitas.

Digunakan jika keluaran yang dihasilkan oleh jaringan syaraf tiruan merupakan sembarang bilangan riil (bukan hanya pada *range* [0,1] atau [1,-1]).

$$Y = X$$



Gambar 2.8 Fungsi aktivasi Identitas

2.5.5 Algoritma Umum Jaringan Syaraf Tiruan

Algoritma pembelajaran/pelatihan jaringan syaraf tiruan.

Dimasukkan n contoh pelatihan kedalam jaringan syaraf tiruan, lakukan :

1. Masukkan kasus kedalam jaringan pada lapisan *input*.
2. Hitung tingkat aktivasi node-node jaringan.
3. Untuk jaringan koneksi umpan maju, jika tingkat aktivasi dari semua unit *output*nya telah dikalkulasi, maka *exit*. Untuk jaringan dengan kondisi balik, jika tingkat aktivasi dari semua unit outputnya menjadi konstan atau mendekati konstan, maka *exit*. Jika tidak, kembali ke langkah 2. Jika jaringan tidak stabil, maka *exit* dan *fail*.

2.5.6 Normalisasi Hasil Ekstraksi Ciri

Normalisasi adalah penskalaan terhadap nilai-nilai yang masuk ke dalam suatu range tertentu. Hal ini dilakukan agar nilai input dan target output sesuai dengan range dari fungsi aktivasi yang digunakan dalam jaringan. Bila fungsi aktivasi yang digunakan adalah sigmoid biner, maka persamaan normalisasi yang dapat digunakan yaitu [15]:

$$normal_i = \frac{0.8(Cepstral_n - a)}{b - a} + 0.1 \qquad (2.20)$$

Keterangan :

$normal_i$: hasil normalisasi ke-i.

$Cepstral_n$: hasil Cepstral Liftering ke-n.

a : nilai minimum pada *frame*.

b : nilai maksimum pada *frame*.

2.6 Backpropagation

Jaringan Syaraf Tiruan *Backpropagation* merupakan suatu metode pembelajaran terawasi (*supervised learning*). Didalam *Backpropagation*, setiap unit yang berada pada lapisan *input* terhubung dengan setiap unit yang ada dilapisan tersembunyi (*hidden layer*). Dan setiap unit pada lapisan *tersembunyi* terhubung pula dengan setiap lapisan *output* [11].

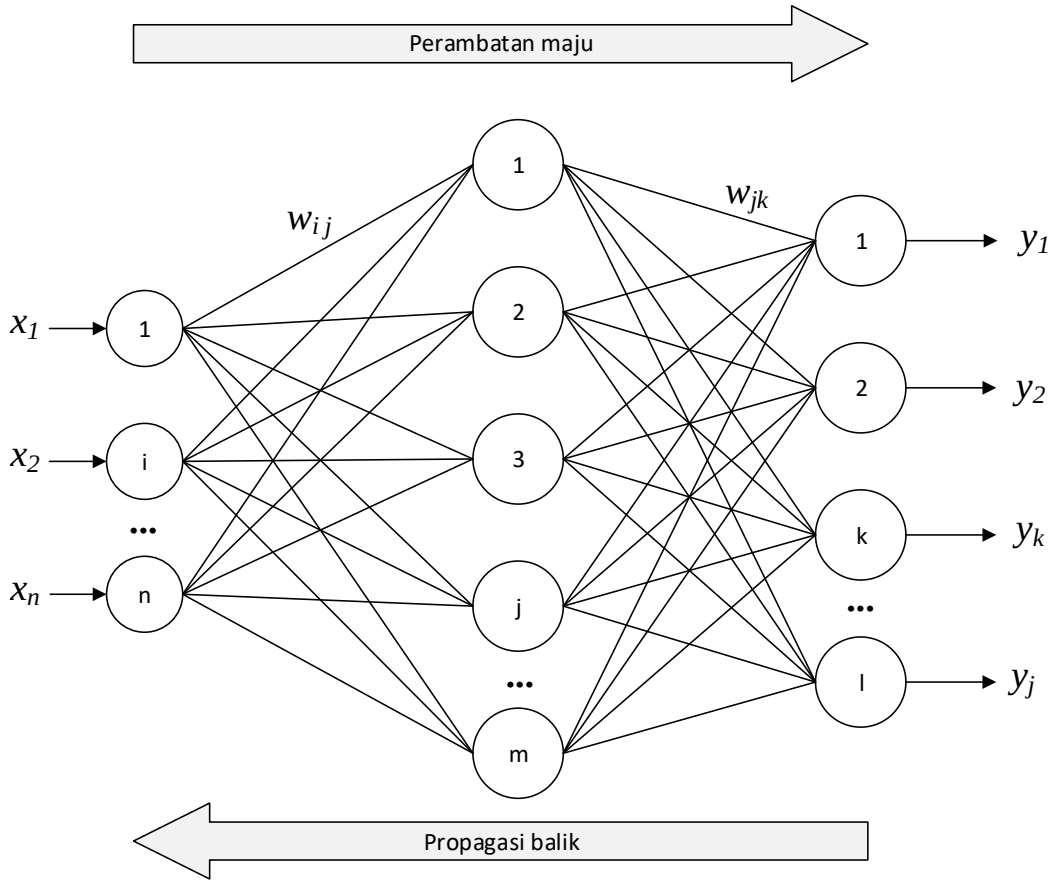
Metode ini merupakan salah satu metode yang sangat baik dalam menangani masalah pengenalan pola-pola kompleks. Didalam jaringan backpropagation, setiap unit yang berada di lapisan *input* berhubungan dengan setiap unit yang ada di lapisan tersembunyi. Setiap unit yang ada di lapisan tersembunyi terhubung dengan setiap unit yang ada di lapisan *output*. Jaringan ini terdiri dari banyak lapisan (*multilayer network*). Ketika jaringan ini diberikan pola masukkan sebagai pola pelatihan, maka pola tersebut menuju unit-unit lapisan tersembunyi untuk selanjutnya diteruskan pada unit-unit di lapisan keluaran. Kemudian unit-unit lapisan keluaran akan memberikan respon sebagai keluaran jaringan syaraf tiruan. Saat hasil keluaran tidak sesuai dengan yang diharapkan, maka keluaran akan

disebarkan mundur (*backward*) pada lapisan tersembunyi kemudian dari lapisan tersembunyi menuju lapisan masukan [12].

Tahap pelatihan ini merupakan langkah untuk melatih suatu jaringan syaraf tiruan, yaitu dengan cara melakukan perubahan bobot, sedangkan penyelesaian masalah akan dilakukan jika proses pelatihan tersebut telah selesai, fase ini disebut dengan fase pengujian.

2.6.1 Arsitektur Jaringan Backpropagation

Setiap unit dari *layer input* pada jaringan *backpropagation* selalu terhubung dengan setiap unit yang berada pada *layer* tersembunyi (*hidden layer*), demikian juga setiap unit *layer* tersembunyi selalu terhubung dengan unit pada *layer output*.



Gambar 2.9 Arsitektur jaringan Backpropagation

Pada Gambar 2.9 terdapat tiga *layer*, yaitu masukan (*input*), tersembunyi (*hidden*) dan keluaran (*output*). Untuk indeks *i*, *j*, *k* masing-masing menyatakan indeks neuron dalam layer masukan, tersembunyi dan keluaran.

2.6.2 Pelatihan Jaringan Backpropagation

Aturan pelatihan jaringan *backpropagation* terdiri dari 2 tahapan, perambatan maju (*feedforward*) dan propagasi balik (*backward propagation*). Pada jaringan diberikan sekumpulan contoh pelatihan yang disebut set pelatihan. Sesi pelatihan ini digambarkan dengan sebuah vektor *feature* yang disebut dengan vektor *input* yang diasosiasikan dengan sebuah *output* yang menjadi target pelatihannya. Dengan kata lain *set* pelatihan terdiri dari jaringan berupa sebuah vektor *input* dan juga vektor *output* target. Keluaran dari jaringan berupa sebuah vektor *output* aktual. Selanjutnya dilakukan perbandingan antara *output* aktual yang dihasilkan dengan *output* target dengan cara melakukan pengurangan diantara kedua *output* tersebut. Hasil dari pengurangan berupa *error*. *Error* dijadikan sebagai dasar dalam melakukan perubahan dari setiap bobot dengan memprogationkannya kembali. Setiap perubahan bobot yang terjadi dapat mengurangi *error*. Siklus perubahan bobot (*epoch*) dilakukan pada setiap set pelatihan sehingga kondisi berhenti dicapai, yaitu bila mencapai jumlah *epoch* yang diinginkan atau hingga sebuah nilai ambang yang ditetapkan terlampaui.

Pada Gambar 2.9 sinyal masukan x_i, x_2, …, x_n dirambatkan maju dari kiri ke kanan, sedangkan sinyal error e_1, e_2, …, e_l dirambatkan balik dari kanan ke kiri. Simbol w_{ij} menyatakan bobot untuk koneksi dari *input layer* ke *hidden layer*, dimana *i* menyatakan neuron dalam *input layer* dan *j* neuron dalam *hidden layer*. Simbol w_{jk} menyatakan bobot dari *hidden layer* ke neuron *k* dalam *output layer*. Untuk merambatkan sinyal error, dimulai dari *output layer* dan berjalan kembali ke *hidden layer* [12].

Sinyal error di neuron keluaran *k* pada iterasi *p* diformulasikan :

$$e_k(p) = y_{dk}(p) - y_k(p) \qquad (2.21)$$

Dimana :

$y_{dk}(p)$: nilai keluaran yang menjadi target untuk neuron *k*.

$y_k(p)$: keluaran nyata yang didapatkan oleh neuron *k* di layer keluaran.

Karena sinyal balik pada semua neuron dalam *output layer* disuplai langsung oleh nilai keluaran yang diharapkannya, maka prosedur untuk memperbaharui bobot w_{jk} menjadi sederhana. Prosedur yang digunakan untuk memperbaharui bobot pada koneksi antara *hidden layer* ke *output layer* sama dengan Perceptron :

$$w_{jk}(p+1) = w_{jk}(p) + \Delta w_{jk}(p) \quad (2.22)$$

Dimana :

$\Delta w_{jk}(p)$: koreksi bobot.

Akan tetapi, perlu diketahui bahwa dalam Perceptron, untuk menghitung koreksi bobot $\Delta w_{jk}(p)$, yang digunakan di dalamnya adalah sinyal masukan x_i, sedangkan dalam MLP yang digunakan adalah sinyal keluaran dari *hidden layer*. Kondisi yang ada adalah masukan neuron pada *output layer* berbeda dari *input* neuron pada *input layer* x_i. oleh karena itu yang digunakan untuk menghitung koreksi bobot dalam MLP dihitung dengan :

$$\Delta w_{jk}(p) = \eta \times y_j(p) \times \delta_k(p) \quad (2.23)$$

Dimana :

η : laju pembelajaran (*learning rate*).

$\delta_k(p)$: *gradien error* pada neuron *k* dalam *output layer* pada iterasi ke *p*.

Gradien error ditentukan dari turunan dari fungsi aktivasi yang dikalikan dengan error pada neuron di *output layer*. Untuk neuron *k* dalam *output layer*, didapatkan :

$$\delta_k(p) = \frac{\partial y_k(p)}{\partial v_k(p)} \times e_k(p) \quad (2.24)$$

Dimana :

$\partial y_k(p)$: keluaran dari neuron *k* pada iterasi *p*.

$\partial v_k(p)$: akumulasi nilai/sinyal input terbobot yang masuk ke neuron *k* pada iterasi yang sama.

Untuk fungsi aktivasi sigmoid biner, turunannya terhadap *v* adalah :

$$\frac{\partial y_k(p)}{\partial v_k(p)} = \frac{\partial \left\{\frac{1}{1+e^{-v_k(p)}}\right\}}{\partial v_k(p)} = \frac{e^{-v_k(p)}}{(1+e^{-v_k(p)})^2}$$

$$\frac{\partial y_k(p)}{\partial v_k(p)} = y_k(p) \times \big(1 - y_k(p)\big) \quad (2.25)$$

Dari formula di atas, untuk menghitung gradien error pada fungsi aktivasi sigmoid biner didapatkan :

$$\delta_k(p) = y_k(p) \times \big(1 - y_k(p)\big) \times e_k(p) \quad (2.26)$$

Dimana :

$$y_k(p) = \frac{1}{1+e^{-v_k(p)}}$$

Untuk menghitung koreksi bobot pada *hidden layer*, cara yang sama juga dapat diterapkan :

$$\Delta w_{ij}(p) = \eta \times x_i(p) \times \delta_j(p) \quad (2.27)$$

$\delta_j(p)$ merepresentasikan gradien error pada neuron *j* dalam hidden layer.

Untuk sigmoid biner :

$$\delta_j(p) = y_j(p) \times \big[1 - y_j(p)\big] + \sum_{k=1}^{l} \delta_k(p) \times w_{jk}(p) \quad (2.28)$$

Dimana :

$$y_j(p) = \frac{1}{1+e^{-v_j(p)}}$$

l adalah jumlah neuron pada *output layer*, sedangkan $v_j(p)$ didapatkan dari akumulasi vektor masukan *x* yang sudah dikalikan bobot:

$$v_j(p) = \sum_{i=1}^{r} x_i(p) \times w_{ij}(p) \quad (2.29)$$

Dimana *r* adalah jumlah neuron pada input layer.

Kriteria kondisi berhenti melalui perhitungan error diformulasikan sebagai berikut:

$$MSE(w) = \frac{1}{N}\sum_{j}^{N}(y_i - y'_i)^2 \quad (2.30)$$

Untuk proses prediksi pada perceptron, dilakukan dengan memproses vektor masukan pada model yang didapat dari proses pelatihan sehingga didapatkan *y'* sebagai kelas hasil prediksi. Secara umum, formula yang digunakan :

$$y' = \begin{cases} 1, jika\ sign\left(\sum_{i=1}^{r} x_i \times w_i\right) \geq 0 \\ 0, jika\ sign\left(\sum_{i=1}^{r} x_i \times w_i\right) < 0 \end{cases}$$

Secara prosedural algorima pelatihan jaringan *backpropagation* dapat diuraikan sebagai berikut :

Langkah 1: Inisialisasi

Inisialisasi semua bobot pada layer tersembunyi (*hidden*) dan layer keluaran (*output*), lalu tetapkan fungsi aktivasi yang digunakan untuk setiap layer.

Langkah 2: Aktivasi

Mengaktivasi jaringan dengan menerapkan masukan $x_1(p), x_2(p), \dots, x_n(p)$ dan keluaran yang diharapkan $y_{d1}(p), y_{d2}(p), \dots, y_{dn}(p)$.

a) Hitung keluaran yang didapatkan dari neuron dalam hidden layer.

$$v_j(p) = \sum_{i=1}^{r} x_i(p) \times w_{ij}(p) \quad (2.31)$$

$$y_j(p) = \frac{1}{1+e^{-v_j(p)}} \quad (2.32)$$

r adalah jumlah masukan (fitur) pada neuron *j* dalam hidden layer.

b) Hitung keluaran yang didapatkan dari neuron dalam output layer.

$$v_k(p) = \sum_{j=1}^{m} x_j(p) \times w_{jk}(p) \quad (2.33)$$

$$y_k(p) = \frac{1}{1+e^{-v_k(p)}} \quad (2.34)$$

m adalah jumlah masukan pada neuron *k* dalam output layer.

Langkah 3: Perbarui Bobot

Bobot diperbarui pada saat error dirambatkan balik (*backward propagation*) dalam ANN, error yang dikembalikan sesuai dengan arah keluaran sinyal keluaran.

a) Hitung gradien error untuk neuron dalam output layer:

$$e_k(p) = y_{dk}(p) - y_k(p) \quad (2.35)$$

$$\delta_k(p) = y_k(p) \times [1 - y_k(p)] \times e_k(p) \quad (2.36)$$

Hitung koreksi bobot:

$$\Delta w_{jk}(p) = \eta \times y_j(p) \times \delta_k(p) \quad (2.37)$$

Perbarui bobot padan neuron output layer:

$$\Delta w_{jk}(p+1) = w_{jk}(p) + \Delta w_{jk}(p) \quad (2.38)$$

b) Hitung gradien error untuk neuron dalam hidden layer:

$$\delta_j(p) = y_j(p) \times \left[1 - y_j(p)\right] + \sum_{k=1}^{l} \delta_k(p) \times w_{jk}(p) \quad (2.39)$$

Hitung koreksi bobot:

$$\Delta w_{ij}(p) = \eta \times x_i(p) \times \delta_j(p) \quad (2.40)$$

Perbarui bobot pada neuron hidden layer:

$$\Delta w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p) \quad (2.41)$$

Langkah 4: Iterasi

Naikkan satu untuk iterasi *p*, kembali ke langkah 2 dan ulangi proses tersebut sampai kriteria error tercapai.

2.7 Quickprop

Quickprop atau *Quickpropagation* adalah salah satu metode pembelajaran dengan JST (Jaringan Syaraf Tiruan). *Quickprop* berasal dari *Backpropagation* tetapi menggunakan cara yang berbeda dalam meng-update bobot sinapsis. *Quickprop* banyak digunakan dalam banyak kasus, salah satu diantaranya adalah *pattern recognition*. Diantara metode pembelajaran dengan JST, *Quickprop* termasuk metode pembelajaran yang cukup cepat dalam pembelajaran dan menghasilkan akurasi yang cukup tinggi dalam pengenalan [4].

Setelah melakukan proses aktivasi terhadap hasil keluaran pada quickprop, maka akan perhitungan *error* dengan (*cost function*) dibawah ini :

$$E = \frac{1}{2}\sum_{j}^{N}(y_{dk} - y_k)^2 \quad (2.42)$$

Keterangan :

y_{dk} : target keluaran yang diinginkan

y_k : keluaran aktual pada *output layer*

2.7.1 Perubahan Bobot pada Quickprop

Perbedaan yang terdapat pada metode *Backpropagation* dengan metode *Quickprop* yaitu rumus perhitungan pada perubahan bobot yang terjadi pada prosesnya. Sebelum dilakukan perubahan, perlu dicari dahulu nilai-nilai setelah dilakukannya proses *feedforward* yaitu :

1. Hitung nilai gradien *k* dengan persamaan (2.43) dibawah ini :

$$\delta_k = y_k(1 - y_k) \times (y_{dk} - y_k) \quad (2.43)$$

Keterangan :

δ_k : gradien *k*

y_k : keluaran aktual pada *output layer*

y_{dk} : nilai keluaran yang diinginkan

2. Hitung nilai derivatif *error* terhadap bobot w_{jk} dengan persamaan (2.44) dibawah ini :

$$\frac{\partial E}{\partial w_{jk}} = y_k \times \delta_k \quad (2.44)$$

Keterangan :

$\frac{\partial E}{\partial w_{jk}}$: derivatif *error* terhadap w_{jk}

3. Hitung nilai gradien *j* dengan persamaan (2.45) dibawah ini :

$$\delta_j = y_j(1 - y_j) \sum_{k \in K} \delta_k w_{jk} \quad (2.45)$$

Keterangan :

δ_j : gradien *j*

y_j : nilai keluaran pada *hidden layer*

4. Hitung nilai derivatif *error* terhadap bobot w_{ij} dengan persamaan (2.46) dibawah ini :

$$\frac{\partial E}{\partial w_{ij}} = y_j \times \delta_j \quad (2.46)$$

Keterangan :

$\frac{\partial E}{\partial w_{ij}}$: derivatif error terhadap w_{ij}

5. Hitung koreksi bobot dari *output layer* ke *hidden layer* dengan persamaan (2.47) dibawah ini :

$$\Delta w_{jk} = -\varepsilon \times \frac{\partial E}{\partial w_{jk}} \quad (2.47)$$

Keterangan :

ε : tingkat pembelajaran (*learning rate*)

6. Hitung koreksi bobot dari *hidden layer* ke *input layer* dengan persamaan (2.48) dibawah ini :

$$\Delta w_{ij} = -\varepsilon \times \frac{\partial E}{\partial w_{ij}} \quad (2.48)$$

7. Berikut ini perubahan bobot yang terjadi pada metode *Quickprop* [13].

Untuk bobot w_{jk} digunakan persamaan

$$w_{jk}(p+1) = w_{jk}(p) + \Delta w_{jk} + \frac{\frac{\partial E}{\partial w_{jk}}(p)}{\frac{\partial E}{\partial w_{jk}}(p-1) - \frac{\partial E}{\partial w_{jk}}(p)} \Delta w_{jk}(p-1) \quad (2.49)$$

Keterangan :

$w_{jk}(p+1)$: Perubahan bobot / bobot baru.

$\Delta w_{jk}(p-1)$: delta bobot pada *epoch* sebelumnya.

$\frac{\partial E}{\partial w_{jk}}(p)$: Derivatif *error*.

$\frac{\partial E}{\partial w_{jk}}(p-1)$: Derivatif *error* pada *epoch* sebelumnya.

Sedangkan untuk bobot w_{ij} digunakan persamaan

$$w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij} + \frac{\frac{\partial E}{\partial w_{ij}}(p)}{\frac{\partial E}{\partial w_{ij}}(p-1) - \frac{\partial E}{\partial w_{ij}}(p)} \Delta w_{ij}(p-1) \quad (2.50)$$

Keterangan :

$w_{ij}(p+1)$: Perubahan bobot / bobot baru.

$\Delta w_{ij}(p-1)$: delta bobot pada *epoch* sebelumnya.

$\frac{\partial E}{\partial w_{ij}}(p)$: Derivatif *error*.

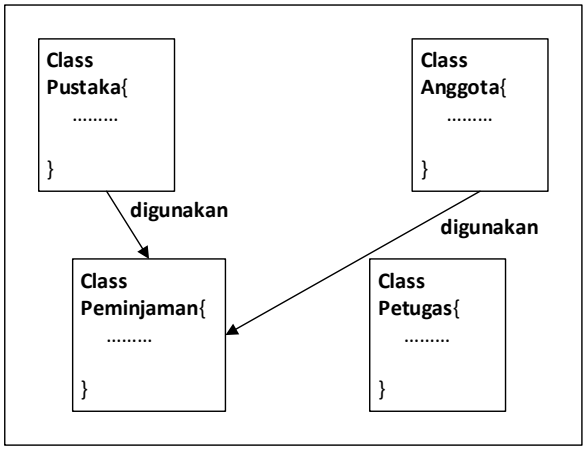
$\frac{\partial E}{\partial w_{ij}}(p-1)$: Derivatif *error* pada *epoch* sebelumnya.

2.8 Pemrograman Berorientasi Objek

Pemrograman berorientasi objek (*Object Oriented Programming*) merupakan suatu paradigma pemrograman yang berorientasikan kepada objek. Pendekatan berorientasi objek adalah cara memandang persoalan menggunakan model-model yang diorganisasikan seputar konsep objek yang mengkombinasikan struktur data dan perilaku suatu entitas. Pada pendekatan ini, organisasi perangkat lunak adalah sebagai kumpulan objek diskrit yang saling bekerja sama, berkomunikasi dan berinteraksi menuju sasaran tertentu [7]. Berikut ini adalah beberapa konsep dasar yang harus dipahami pemrograman berorientasi objek [14]:

1. Kelas (*Class*)

   Kelas merupakan kumpulan objek-objek dengan karakteristik yang sama. Kelas merupakan definisi statik dan himpunan objek yang sama mungkin lahir atau diciptakan dari kelas tersebut. Sebuah kelas akan mempunyai sifat (atribut), kelakukan (operasi/metode), hubungan (*relationship*) dan arti. Suatu kelas dapat diturunkan dan kelas semula dapat diwariskan ke kelas yang baru.



Gambar 2.10 Ilustrasi Kelas

2. Objek (*Object*)

   Objek adalah abstraksi dan suatu yang mewakili dunia nyata seperti benda, manusia, suatu organisasi, tempat, kejadian, struktur, status, atau hal-hal lain yang bersifat abstrak. Objek merupakan suatu entitas yang mampu menyimpan informasi (status) dan mempunyai operasi (metode) yang dapat diterapkan atau dapat berpengaruh pada status objeknya. Objek mempunyai siklus hidup yaitu diciptakan, dimanipulasi, dan dihancurkan.

3. Metode (*Method*)

   Operasi atau metode pada sebuah kelas hampir sama dengan fungsi atau prosedur pada terstruktur. Sebuah kelas boleh memiliki lebih dari satu metode atau operasi. Metode atau operasi yang berfungsi untuk memanipulasi objek itu sendiri.

4. Atribut (*Attribute*)

   Atribut dari sebuah kelas adalah variabel global yang dimiliki sebuah kelas. Atribut dapat berupa nilai atau elemen-elemen data yang dimiliki oleh objek dalam kelas objek. Atribut secara individual oleh sebuah objek, misalnya berat, jenis, nama, dan sebagainya.

5. Abstraksi (*Abstraction*)

   Abstraksi merupakan prinsip untuk merepresentasikan dunia nyata yang kompleks menjadi satu bentuk model yang sederhana dengan mengabaikan aspek-aspek lain yang tidak sesuai dengan permasalahan.

6. Enkapsulasi (*Encapsulation*)

   Enkapsulasi merupakan pembukusan atribut data dan layanan (metode-metode) yang dimiliki objek untuk menyembunyikan implementasi sehingga objek lain tidak mengetahui cara kerja.

7. Pewarisan (*Inheritance*)

   Pewarisan merupakan mekanisme yang memungkinkan satu objek mewarisi sebagian atau seluruh definisi dan objek lain sebagai bagian dari dirinya.

8. Antarmuka (*Interface*)

   Antarmuka atau interface sangat mirip dengan kelas, tetapu tanpa atribut kelas dan tanpa memiliki metode yang dideklarasikan. Antarmuka biasanya digunakan agar kelas lain tidak langsung mengakses ke suatu kelas.

9. *Reusability*

   Rusability atau pemanfaatan kembali objek yang sudah didefinisikan untuk suatu permasalahan pada permasalahan lainnya yang melibatkan objek tersebut.

10. Generalisasi dan Spesialisasi

    Menunjukkan hubungan antara kelas dan objek yang umum dengan kelas dan objek yang khusus. Misalnya kelas yang lebih umum (generalisasi) adalah mamalia dan kelas khususnya (spesialisasi) adalah manusia, lumba-lumba, singa.

11. Komunikasi antar objek

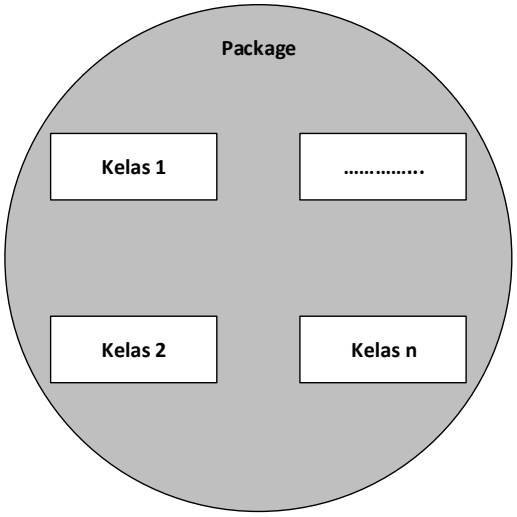
    Komunikasi antar objek dilakukan lewat pesan (*message*) yang dikirim dari satu objek ke objek lainnya.

12. Polimorfisme (*Polymorphism*)

    Polimorfisme merupakan kemampuan suatu objek untuk digunakan dibanyak tujuan yang berbeda dengan nama yang sama sehingga menghemat baris program.

13. Paket (*Package*)

Paket adalah sebuat kontainer atau kemasan yang dapat digunakan untuk mengelompokkan kelas-kelas sehingga memungkinkan beberapa kelas yang bernama sama disimpan dalam paket yang berbeda.



Gambar 2.11 Ilustrasi Package

2.9 Unified Modeling Language

Unified Modeling Language (UML) memiliki arti pemodelan standar. UML adalah bahasa grafis untuk mendokumentasi, menspesifikasikan, dan membangun level abstraksi, tidak bergantung proses pengembangan, tidak bergantung bahasa dan teknologi, pemaduan beberapa notasi di beragam metodologi, usaha dari banyak pihak, didukung oleh kakas-kakas yang diintegrasikan lewat XML. Standar UML dikelola oleh OMG (*Object Management Group*). UML adalah bahasa pemodelan untuk menspesifikasikan, memvisualisasikan, membangun dan mendokumentasikan artifak-artifak dari sistem. UML bukanlah :

1. Bahasa pemrograman visual, tetapi bahasa pemodelan visual.
2. Spesifikasi Kakas, tetapi spesifikasi bahasa pemodelan.
3. Proses, tetapi yang memungkinkan proses-proses.

Teknik pemodelan dalam UML ada dua jenis diagramnya, yaitu :

1. Diagram struktur

   Diagram ini untuk memvisualisasi, menspesifikasikan, membangun dan mendokumentasikan aspek statik dari sistem. Diagram struktur UML terdiri

dari : diagram kelas (*Class Diagram*), diagram objek (*Object Diagram*), diagram komponen (*Component Diagram*), dan diagram deployment (*Deployment Diagram*).

2. Diagram perilaku

   Diagram ini untuk memvisualisasi, menspesifikasikan, membangun dan mendokumentasikan aspek dinamis dari sistem. Diagram perilaku dalam UML terdiri dari : diagram use case (*Use Case Diagram*), diagram sekuen (*Sequence Diagram*), diagram kolaborasi (*Collaboration Diagram*), diagram aktivitas (*Activity Diagram*).

2.9.1 Class Diagram

Class diagram merupakan diagram paling umum dipakai di semua pemodelan berorientasi objek. Pemodelan kelas merupakan pemodelan yang paling utama di pendekatan berorientasi objek. Pemodelan kelas menunjukkan kelas-kelas yang ada di sistem dan hubungan antar kelas-kelas itu, atribut-atribut dan operasi-operasi di setiap kelas. *Class diagram* menunjuk aspek statis sistem terutama untuk mendukung kebutuhan fungsional sistem. Meskipun *class diagram* serupa dengan model data, namun kelas-kelas tidak hanya menunjuk struktur informasi tapi juga mendeskripsikan perilaku. Salah satu maksud *class diagram* adalah untuk mendefinisikan fondasi bagi diagram-diagram lain di mana aspek-aspek lain dari sistem ditunjukkan. Elemen-elemen yang penting dalam *class diagram* adalah sebagai berikut [14]:

1. Kelas

   Kelas merupakan elemen terpenting di sistem berorientasi objek. Kelas mendeskripsikan satu blok pembangunan sistem. Berikut ini adalah bagian dari kelas :

   a. Nama

      Nama kelas harus unik karena akan menjadi *identifier* di program.

   b. Atribut

Atribut adalah properti bernama di kelas yang mendeskripsikan range nilai yang dipunyai instan kelas. Kelas dapat mempunyai sejumlah atribut atau tidak sama sekali.

c. Operasi

Operasi adalah implementasi layanan yang dapat diminta pada sembarang objek kelas itu untuk mempengaruhi perilaku sistem.

d. Access Modifier

Access Modifier dalam kelas dibunakan untuk membatasi akses dari kelas lain yang ingin mengakses atribut atau operasi dari suatu kelas. Ada tiga *Access Modifier* yang digunakan, yaitu *public* (+), *protected* (#) dan *private* (-).

2. Antarmuka

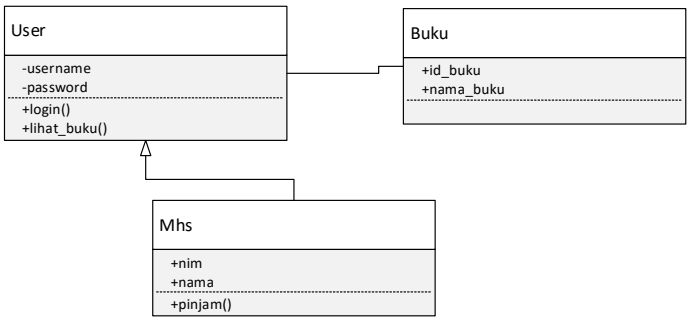
Antarmuka (*Interface*) merupakan korelasi operasi yang menspesifikasikan layanan dari suatu kelas atau komponen. Antarmuka mendeskripsikan perilaku tampak secara eksternal dari elemen.

3. Kolaborasi

Kolaborasi merupakan pendefinisian suatu interaksi dan sekelompok peran elemen-elemen lain yang bekerja sama untuk menyediakan suatu perilaku kooperatif yang lebih besar dari penjumlahan seluruh elemen. Kolaborasi memiliki struktur, perilaku dan dimensi. Kolaborasi ini merepresentasikan implementasi pola tertentu yang membuntuk sistem.

4. Hubungan

Hubungan antar kelas di diagram kelas terdiri dari asosisasi dan generalisasi.



Gambar 2.12 Contoh Class Diagram

2.9.2 Use Case Diagram

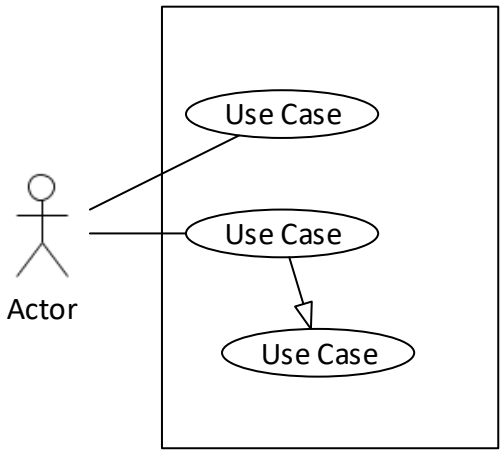
Use case diagram merupakan salah satu diagram untuk memodelkan aspek tingkah laku sistem yang akan dibuat. Masing-masing *use case diagram* menunjukkan sekumpulan *use case*, aktor dan hubungannya. *Use case diagram* penting untuk memvisualisasikan, menspesifikasikan, dan mendokumentasikan kebutuhan perilaku sistem. Diagram-diagram *use case* merupakan pusat pemodelan perilaku sistem, subsistem dan kelas. *Use case* adalah interaksi antara aktor eksternal dan sistem, hasil yang dapat diamati oleh aktor, berorientasi pada tujuan, didekskripsikan di *use case diagram* dan teks. Relasi antar *use case* dengan *use case* lainnya ada tiga jenis, yaitu [14]:

1. Relasi Include

   Merupakan relasi antar *use case* di mana jika *use case* A meng-include *use case* B, maka *use case* B akan diimplementasikan setiap kali *use case* A diimplementasikan. Direpresentasikan dengan garis putus-putus bertuliskan <<*include*>> ke arah *use case* yang akan di-include.
2. Relasi Extend

   Merupakan relasi antar *use case* di mana jika *use case* A di *extend* oleh *use case* B maka *use case* B akan bisa saja diimplementasikan atau tidak setiap kali *use case* A diimplementasikan. Direpresentasikan dengan garis putus-putus bertuliskan <<*extend*>> ke arah *use case* yang akan di *extend*.
3. Relasi Generalization

   Digunakan untuk membuat *use case* lebih spesifik.



Gambar 2.13 Contoh Use Case Diagram

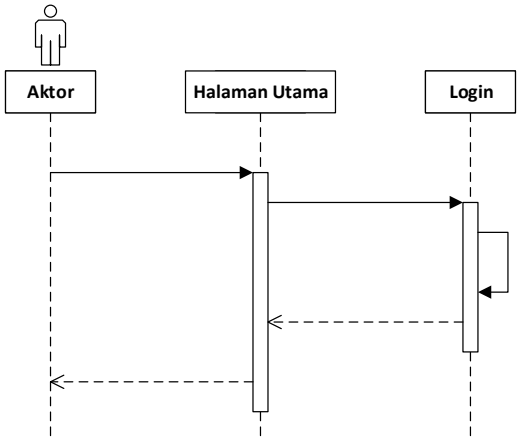
2.9.3 Use Case Scenario

Use case scenario merupakan hasil instansiasi dan penjelasan dari setiap *use case*. *Use case scenario* terbagi menjadi tiga bagian, yaitu :

1. Identifikasi dan inisiasi.
2. *Step performed*.
3. Kondisi, asumsi dan pertanyaan.

2.9.4 Sequence Diagram

Sequence diagram digunakan untuk memodelkan skenario penggunaan. Skenario penggunaan adalah barisan kejadian yang terjadi selama satu eksekusi sistem. Cakupan skenario dapat beragam, dari mulai semua kejadian di sistem atau hanya kejadian pada objek-objek tertentu. *Sequence diagram* menunjukkan objek sebagai garis vertikal dan tiap kejadian sebagai panah horizontal dari objek pengirim ke objek penerima.



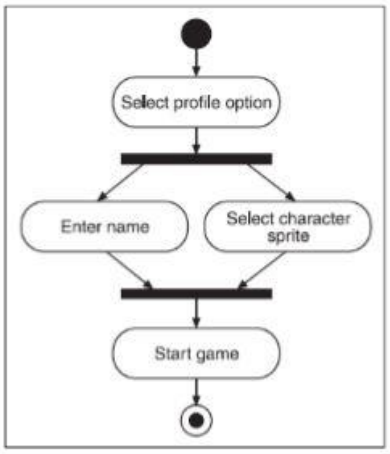
Gambar 2.14 Contoh Sequence Diagram

2.9.5 Activity Diagram

Activity diagram adalah diagram *flowchart* yang diperluas yang menunjukkan aliran kendali satu aktivitas ke aktivitas lain. Kita menggunakan diagram ini memodelkan aspek dinamis sistem. Aktivitas adalah eksekusi notatomik yang berlangsung di *state machine*. *Activity diagram* mendeskripsikan aksi-aksi dah hasilnya. *Activity diagram* berupa operasi-operasi dan aktivitas-aktivitas di *use case*. *Activity diagram* dapat digunakan untuk :

1. Pandangan dalam yang dilakukan di operasi.
2. Pandangan dalam bagaimana objek-objek bekerja.
3. Pandangan dalam di aksi-aksi dan pengaruhnya pada objek-objek.
4. Pandangan dalam dari suatu *use case*.
5. Logik dari proses bisnis.

Diagram aktivitas merupakan jenis khusus dan diagram *statechart*. *State* adalah aksi-aksi yang menuju state berikutnya setelah selesainya aksi itu [14].



Gambar 2.15 Contoh Activity Diagram

2.10 Pengujian Sistem

Pengujian adalah proses pemeriksaan atau evaluasi sistem atau komponen sistem secara manual atau otomatis untuk memverikasi apakah sistem memenuhi kebutuhan-kebutuhan yang dispesifikan atau mengidentifikasi perbedaan perbedaan antara hasil yang diharapkan dengan hasil yang terjadi.

2.10.1 Pengujian Akurasi

Akurasi merupakan seberapa dekat suatu angka hasil pengukuran terhadap angka sebenarnya (*true value* atau *reference value*). Tingkat akurasi diperoleh dengan persamaan (2.42) berikut ini.

$$Akurasi = \frac{jumlah\ karakter\ sama}{jumlah\ seluruh\ karakter} \times 100\% \qquad (2.42)$$

2.11 Bahasa Pemrograman

Bahasa pemrograman diperlukan untuk menjalankan instruksi-instruksi apa saja yang harus dilakukan komputer. Komputer tidak bisa memahami bahasa manusia, sehingga diperlukan penggunaan bahasa komputer di dalam program komputer. Penelitian ini menggunakan bahasa pemrograman *Java*.

2.11.1 Java

Java merupakan bahasa berorientasi objek untuk pengembangan aplikasi mandiri, aplikasi berbasis internet, aplikasi untuk perangkat cerdas yang dapat berkomunikasi lewat internet/ jaringan komunikasi. Melalui teknologi *java*, dimungkinkan perangkat audio stereo dirumah terhubung jaringan komputer. *Java* tidak lagi hanya untuk membuat *applet* yang memperintah halaman *web* tapi *java* telah menjadi bahasa untuk pengembangan aplikasi skala *enterprise* berbasis jaringan besar [7].