BAB II
LANDASAN TEORI

2.1 Kecerdasan Buatan

Kecerdasan Buatan (*Artificial Intelligence* atau AI) didefinisikan sebagai kecerdasan yang ditunjukkan oleh suatu *entitas* buatan. Sistem seperti ini umumnya dianggap komputer. Kecerdasan diciptakan dan dimasukkan kedalam suatu mesin (komputer) agar dapat melakukan pekerjaan seperti yang dapat dilakukan manusia [6].

2.1.1 Definisi Kecerdasan Buatan

Kecerdasan buatan merupakan salah satu bagian ilmu komputer yang membuat agar mesin (komputer) dapat melakukan pekerjaan seperti dan sebaik yang dilakukan oleh manusia. Pada awal diciptakannya, komputer hanya difungsikan sebagai alat hitung saja. Namun seiring dengan perkembangan zaman, maka peranan komputer semakin mendominasi kehidupan umat manusia. Komputer tidak lagi hanya digunakan sebagai alat hitung, lebih dari itu komputer diharapkan dapat diberdayakan untuk mengerjakan segala sesuatu yang bisa dikerjakan manusia.

Manusia bisa menjadi pandai dalam menyelesaikan segala permasalahan di dunia ini karena manusia mempunyai pengetahuan dan pengalaman. Pengetahuan diperoleh dari belajar. Semakin banyak bekal pengetahuan yang dimiliki seseorang tentu saja diharapkan akan lebih mampu dalam menyelesaikan permasalahan. Namun bekal pengetahuan saja tidak cukup, manusia juga diberi akal untuk melakukan penalaran, mengambil kesimpulan berdasarkan pengetahuan dan pengalaman yang mereka miliki. Tanpa memiliki kemampuan untuk menalar dengan baik, manusia dengan segudang pengalaman dan pengetahuan tidak akan dapat menyelesaikan masalah dengan baik. Demikian pula, dengan kemampuan menalar yang sangat baik, namun tanpa bekal pengetahuan dan pengalaman yang memadai, manusia juga tidak akan bisa menyelesaikan masalah dengan baik.

Agar komputer bisa bertindak seperti dan sebaik manusia, maka komputer juga harus diberi bekal pengetahuan dan mempunyai kemampuan untuk menalar. Untuk itu pada *Artificial Intelligence* (AI) akan mencoba untuk memberikan beberapa metoda untuk membekali komputer dengan kedua komponen tersebut agar komputer bisa menjadi mesin yang pintar.

Pengertian kecerdasan buatan dapat dipandang dari berbagai sudut pandang, antara lain :

a. Sudut pandang kecerdasan

   Kecerdasan buatan akan membuat mesin menjadi „cerdas‟ (mampu berbuat seperti apa yang dilakukan oleh manusia).

b. Sudut pandang penelitian

   Kecerdasan buatan adalah suatu studi bagaimana membuat agar komputer dapat melakukan sesuatu sebaik yang dikerjakan oleh manusia.

c. Sudut pandang bisnis

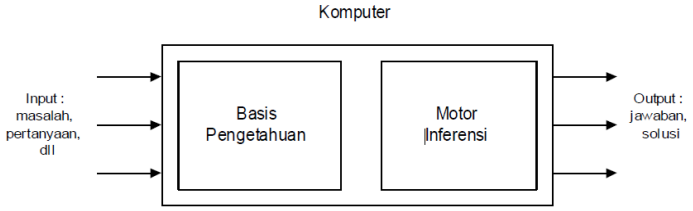
   Kecerdasan buatan adalah kumpulan peralatan yang sangat powerful dan metodologis dalam menyelesaikan masalah-masalah bisnis.

d. Sudut pandang pemrograman

   Kecerdasan buatan meliputi studi tentang pemrograman simbolik, penyelesaian masalah (problem solving) dan pencarian (searching).

Untuk membuat aplikasi kecerdasan buatan ada 2 bagian utama yang sangat dibutuhkan, yaitu :

1. Basis Pengetahuan (*Knowledge Base*), berisi fakta-fakta, teori, pemikiran dan hubungan antara satu dengan lainnya.
2. Motor Inferensi (*Inference Engine*), yaitu kemampuan menarik kesimpulan berdasarkan pengalaman.



Gambar 2.1 Penerapan Konsep Kecerdasan Buatan Pada Komputer

2.1.2 Kecerdasan Buatan dan Kecerdasan Ilmiah

Kecerdasan alamiah adalah kecerdasan yang dimiliki oleh manusia. Jika dibandingkan dengan kecerdasan buatan, ada beberapa keuntungan kecerdasan buatan dibanding kecerdasan alamiah, yaitu [6]:

1. Lebih permanen

   Kecerdasan alamiah akan cepat mengalami perubahan. Hal ini dimungkinkan karena sifat manusia yang pelupa. Kecerdasan buatan tidak akan berubah sepanjang sistem komputer dan program tidak diubah.

2. Memberikan kemudahan dalam duplikasi dan penyebaran

   Mentransfer pengetahuan manusia dari satu orang ke orang lain membutuhkan proses yang sangat lama dan keahlian itu juga tidak akan pernah dapat diduplikasi dengan lengkap. Oleh karena itu, jika pengetahuan terletak pada suatu sistem komputer, pengetahuan tersebut dapat disalin dari komputer tersebut dan dapat dipindahkan dengan mudah ke komputer yang lain.

3. Relatif lebih murah dari kecerdasan alamiah

   Menyediakan layanan komputer akan lebih mudah dan lebih murah dibandingkan dengan harus mendatangkan seseorang untuk mengerjakan sejumlah pekerjaan dalam jangka watu yang sangat lama.

4. Konsisten dan teliti

   Hal ini disebabkan karena kecerdasan buatan adalah bagian dari teknologi komputer. Sedangkan kecerdasan alami akan senantiasa berubah-ubah.

5. Dapat didokumentasi

   Keputusan yang dibuat oleh komputer dapat didokumentasi dengan mudah dengan cara melacak setiap aktivitas dari sistem tersebut. Kecerdasan alami sangat sulit untuk direproduksi.

6. Dapat mengerjakan beberapa *task* dengan lebih cepat dan lebih baik dibanding manusia.

Sedangkan keuntungan kecerdasan alamiah dibanding kecerdasan buatan, yaitu:

1) Bersifat lebih kreatif.

   Kemampuan untuk menambah ataupun memenuhi pengetahuan itu sangat melekat pada jiwa manusia. Pada kecerdasan buatan, untuk menambah pengetahuan harus dilakukan melalui sistem yang dibangun.

2) Dapat melakukan proses pembelajaran secara langsung, sementara AI harus mendapatkan masukan berupa simbol dan representasi.

3) Fokus yang luas sebagai referensi untuk pengambilan keputusan, sebaiknya AI menggunakan fokus yang sempit.

Komputer dapat digunakan untuk mengumpulkan informasi tentang objek, kegiatan (*events*), proses dan dapat memproses sejumlah besar informasi dengan lebih efisien dari yang dapat dikerjakan manusia. Namun disisi lain, manusia dengan menggunakan insting dapat melakukan hal yang sulit diprogram pada komputer, yaitu kemampuan mengenali (*recognize*) hubungan antara hal-hal tersebut, menilai kualitas dan menemukan pola yang menjelaskan hubungan tersebut.

2.1.3 Hubungan Kecerdasan Buatan Dengan Pengenalan Suara

Pemahaman Ucapan/Suara (*Speech/Voice Understanding*), adalah teknik agar komputer dapat mengenali dan memahami bahasa ucapan. Proses ini mengijinkan seseorang berkomunikasi dengan komputer dengan cara berbicara kepadanya. Istilah “pengenalan suara” mengandung arti bahwa tujuan utamanya adalah mengenai kata yang diucapkan tanpa harus tahu artinya, di mana bagian itu merupakan tugas “pemahaman suara”.

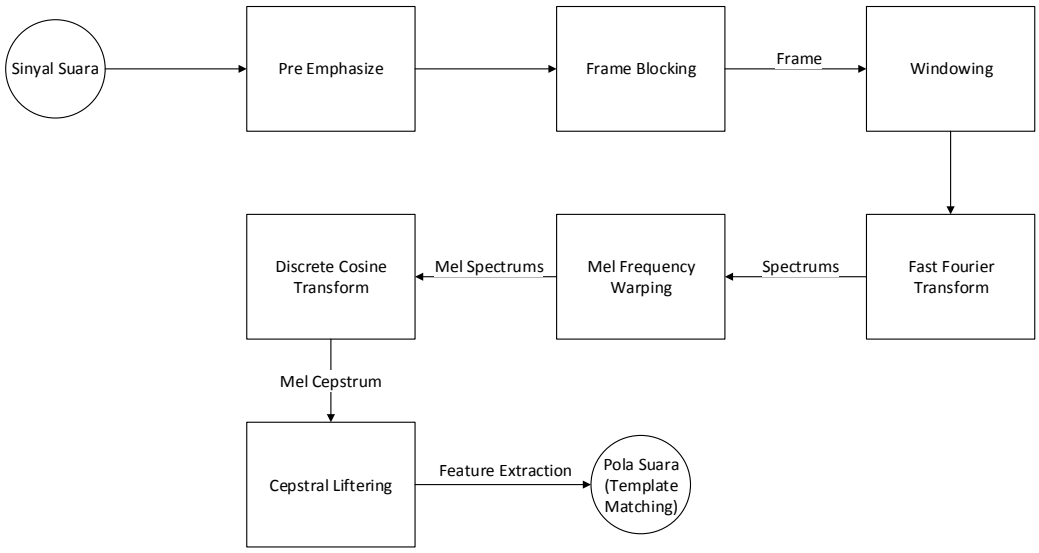
2.2 MFCC (Mel Frequency Cepstrum Coefficient)

MFCC (*Mel Frequency Cepstrum Coefficients*) merupakan salah satu metode yang banyak digunakan dalam bidang *speech technology*, baik *speaker recognation* maupun *speech recognation*. Metode ini digunakan untuk melakukan *feature extraction*, sebuah proses yang mengkonversikan sinyal suara menjadi beberapa parameter. Keunggulan dari metode ini adalah:

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 mengingat sinyal suara yang bersifat *quasistationary*. Pengujian yang dilakukan untuk periode waktu yang cukup pendek (sekitar 0 sampai 10 milidetik) akan menunjukan 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 dimana sinyal suara akan di-filter secara linear untuk frekuensi rendah (dibawah 1000 Hz) dan secara logaritmik untuk frekuensi tinggi [3]. Berikut ini blok diagram untuk proses yang dilakukan pada metode MFCC:



Gambar 2.2 Blok Diagram Alur Proses MFCC

2.2.1 Konversi Analog Menjadi Digital

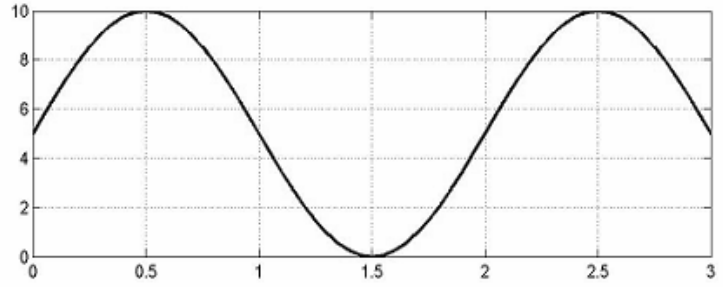
Sinyal–sinyal yang natural pada umumnya, seperti sinyal suara merupakan sinyal *continue* dimana memiliki nilai yang tidak terbatas. Sedangkan pada

komputer, semua sinyal yang dapat diproses oleh komputer hanyalah sinyal *discrete* atau sering dikenal dengan istilah digital *signal*. Agar sinyal natural dapat diproses oleh komputer kita harus dapat mengubah data sinyal *continue* menjadi *discrete*. Hal itu dapat melalui tiga proses, diantaranya adalah proses sampling data, proses kuantisasi, dan proses pengkodean.

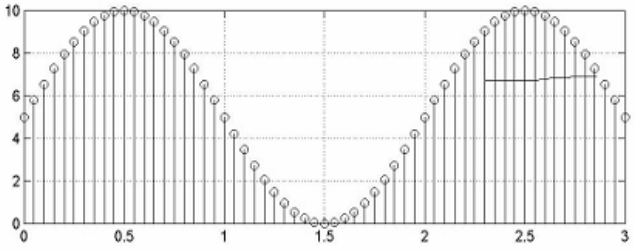
Proses sampling adalah suatu proses untuk mengambil data sinyal *continue* untuk setiap periode tertentu. Dalam melakukan proses sampling data, berlaku aturan Nquist, yaitu bahwa frekuensi sampling (sampling rate) minimal harus dua kali lebih tinggi dari frekuensi maksimum yang akan disamplingkan. Jika sinyal sampling kurang dari dua kali frekuensi maksimum sinyal yang akan disampling, maka akan timbul efek aliasing. Aliasing adalah suatu efek dimana sinyal yang dihasilkan memiliki frekuensi yang berbeda dengan sinyal aslinya,

Proses kuantisasi adalah proses untuk membulatkan nilai data kedalam 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 yang besar dan proses lama.

Proses pengkodean adalah proses pemberian kode untuk tiap tiap data sinyal yang telah terkuantisasi berdasarkan level yang ditempati.



Gambar 2.3 Sinyal Sinus



Gambar 2.4 Sinyal Sinus Setelah Tersampling

2.2.2 Pre-Empasis

Pre-emphasis dilakukan untuk memperbaiki signal dari gangguan *noise*, sehingga dapat meningkatkan tingkat akurasi dari proses feature extraction. Default dari nilai alpha yang digunakan dalam proses pre-emphasis filtering adalah 0,9.

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

$$H(z) = 1 - az^{-1} \qquad (2.1)$$

Dimana $0.9 \leq \alpha \leq 1.0$ dan $\alpha \in R$, formula diatas dapat diimplementasikan sebagai *filter order differentiator*, sebagai berikut

$$y[n] = s[n] - a\, s[n-1] \qquad (2.2)$$

Keterangan:

y[*n*] = signal hasil *pre-emphasis filter*

s[*n*] = signal sebelum *pre-emphasis filter*

2.2.3 Frame Blocking

Hasil perekaman suara merupakan sinyal analog yang berada dalam domain waktu yang bersifat variant time, yaitu suatu fungsi yang bergantung waktu. Oleh karena itu sinyal tersebut harus dipotong-potong dalam slot-slot waktu tertentu agar dapat dianggap invariant. Setiap potongan sinyal suara tersebut disebut frame.

Untuk menghitung jumlah frame digunakan rumus:

$$Jumlah\ Frame\ =\ ((I - N)/M + 1) \qquad (2.3)$$

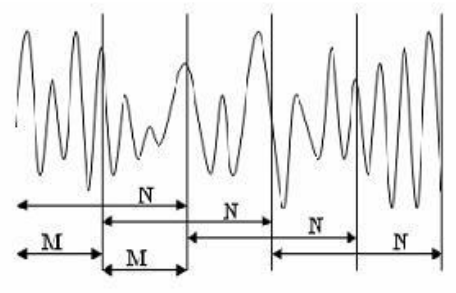
Keterangan:

I = *Sample rate*

N = *Sample point* (*Sample rate* * waktu framing (s))

M = N/2

Potongan *frame* digambarkan seperti gambar dibawah ini:



Gambar 2.5 Framing

2.2.4 Windowing

Fungsi *window* yang paling sering digunakan dalam aplikasi *speaker recognation* adalah Hamming Window. Fungsi ini menghasilkan sidelobe level yang tidak terlalu tinggi (kurang lebih -43dB) selain itu *noise* yang dihasilkan pun tidak terlalu besar (kurang lebih 1.36 BINS).

Window Hamming:

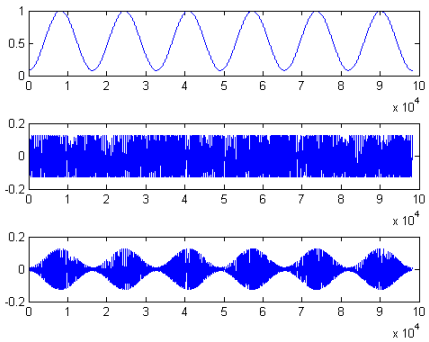
$$W_{ham}(n) = 0.52 - 0.46\cos\left[\frac{2\pi n}{(N-1)}\right] 0 \leq n \leq N-1 \quad (2.4)$$

Keterangan:

$W_{ham}(n)$ = *Window Hamming*

n = Sinyal ke [1 ... n]

N = *Sample Point*



Gambar 2.6 Window Hamming

2.2.5 Fast Fourier Transform

FFT (*Fast Fourier Transform*) adalah teknik perhitungan cepat dari DFT. FFT adalah DFT dengan teknik perhitungan yang cepat dengan memanfaatkan sifat periodikal dari *transformasi fourier*. Perhatikan definisi dari FFT:

$$F(k) = \sum_{n=1}^{N} f(n)e^{-j2\pi knT/N} \quad (2.5)$$

Atau dapat dituliskan dengan:

$$F(k) = \sum_{n=1}^{N} f(n) \cos\left(\frac{2\pi nkT}{N}\right) - j \sum_{n=1}^{N} f(n) \sin(2\pi nkT/N) \quad (2.6)$$

Untuk melihat nilai hasil FFT digunakan rumus:

$$|f(u)| = [R^2 + I^2]^{1/2} \quad (2.7)$$

2.2.6 Mel Frequency Warping (FilterBank)

Mel Frequency Warping umumnya dilakukan dengan menggunakan filterbank. Filterbank menggunakan representasi konvolusi dalam melakukan filter terhadap sinyal. Konvolusi dapat dilakukan dengan melakukan multiplikasi antara spektrum sinyal dengan koeffisien *filterbank*.

Berikut ini adalah rumus yang digunakan dalam perhitungan *filterbank*.

$$Y[i] = \sum_{j=1}^{N} S[j] H_i[j] \quad (2.8)$$

Keterangan:

N = jumlah magnitude spectrum

S[j] = magnitude spectrum pada frekuensi j

$H_i[j]$ = koefisien *filterbank* pada frekuensi j ($1 \leq i \leq M$)

M = jumlah channel dalam *filterbank*

Dimana $H_i = \frac{mel\ f}{x_{i/2}}$

2.2.7 Discrete Cosine Transform

Proses ini merupakan langkah akhir dari *feature extraction*. Hasil dari DCT ini adalah fitur-fitur yang dibutuhkan untuk melakukan proses analisa terhadap pengenalan suara tersebut. Menggunakan rumus:

$$T_n = \sum_{k=1}^{K} (log\ S_k) \cos[n(k - \tfrac{1}{2}) \tfrac{\pi}{K}] \quad (2.9)$$

Keterangan:

S_k = keluaran dari proses *filterbank* pada indeks *k*

K = jumlah koefisien yang diharapkan

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 recognation*. Tetapi koefisien ke nol dari DCT

sangatlah berguna dalam menentukan *end point* dari suatu suku kata maupun kata. Hal ini disebabkan karena pada *end point* dari suatu suku kata maupun kata mempunyai energi yang lebih rendah daripada *point-point*.

2.2.8 Cepstral Liftering

Cepstral, hasil dari proses utama MFCC *feature extraction*, memiliki beberapa kelemahan. *Low-order* dari *cepstral coefficients* sangat sensitif terhadap *spectral slope*, sedangkan bagian *high order*-nya sangat sensitif terhadap *noise*. Oleh karena itu, maka *cepstral liftering* menjadi salah satu standar teknik yang diterapkan untuk meminimalisasi sensitifitas tersebut.

Cepstral liftering dapat dilakukan dengan mengimplementasikan fungsi window terhadap *cepstral features*.

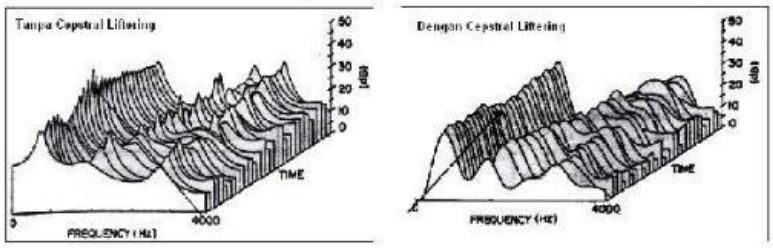
$$W[n] = \{1 + \frac{L}{2}\sin(\frac{n\pi}{L})\} \quad (2.10)$$

Keterangan:

L = jumlah cepstral coefficients

n= index dari cepstral coefficients

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



Gambar 2.7 Perbandingan Spectrum dengan dan Tanpa Cepstral liftering

Berdasarkan gambar diatas dapat dilihat bahwa pola spektrum setelah dilakukannya *cepstral liftering* lebih halus daripada spektrum yang tidak melalui tahap *cepstral liftering*. Dalam beberapa jurnal dikatakan bahwa *cepstral liftering* dapat meningkatkan akurasi dari aplikasi pengenalan suara, baik *speaker recognation*, maupun *speech recognation.*

2.3 Jaringan Syaraf Tiruan

Jaringan saraf tiruan (JST) adalah sistem pemrosesan informasi yang memiliki karakteristik mirip dengan jaringan saraf biologi. Hal ini menyerupai kerja otak dalam dua hal yaitu [7]:

1. Pengetahuan diperoleh oleh jaringan melalui suatu proses belajar,
2. Kekuatan hubungan antar sel saraf yang dikenal dengan bobot sinapsis digunakan untuk menyimpan pengetahuan.

JST merupakan sebuah model komputasi dari otak manusia yang mampu melakukan perhitungan, pengenalan, pengamatan serta pengambilan keputusan. Jaringan syaraf tiruan memanfaatkan struktur pengolahan paralel atas sejumlah pengolah sederhana dan hubungan antar pengolah tersebut. Dalam jaringan syaraf tiruan, terdapat elemen pengolah yang merupakan model dari neuron. Setiap hubungan elemen pengolah menentukan kemampuan JST. Seperti halnya jaringan saraf biologis, JST juga memiliki kemampuan untuk belajar dan beradaptasi terhadap masukan-masukannya. JST tidak perlu diprogram secara eksplisit, karena JST dapat belajar dari beberapa contoh pelatihan. Secara umum proses pembelajaran pada jaringan syaraf tiruan dibagi menjadi dua macam, yaitu:

1. Belajar dengan pengawasan (*Supervised Learning*)

   Pada metode *supervised learning* jaringan diberikan vektor target yang harus dicapai sebagai dasar untuk mengubah hubungan interkoneksi atau bobot pada jaringan. Contoh jaringan yang belajar dengan pengawasan adalah *Backpropagation* dan Perceptron.

2. Belajar Tanpa Pengawasan (*Unsupervised Learning*)

   Berbeda dengan *supervised learning* yang masih membutuhkan pengawasan dan sangat tergantung pada *knowledge* yang diberikan pada pembangunan jaringan, metode *unsupervised learning* ini melatih jaringan terhadap suatu *input*an tanpa adanya pengawasan, dimana vektor target tidak ditentukan. Vektor masukan dimasukkan ke dalam jaringan, kemudian sistem akan mengatur dirinya sendiri sedemikian rupa sehingga dihasilkan keluaran yang konsisten bilamana pola yang menyerupai vektor masukan tersebut diberikan. Contoh jaringan yang belajar tanpa pengawasan adalah

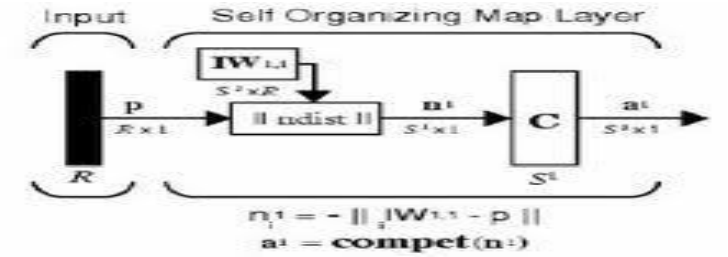
Adaptive Resonance Theory 2 (ART 2), *Self Organizing Map* (SOM), *dan Learning Vektor Quantization* (LVQ).

2.3.1 Jaringan Syaraf Tiruan *Self Organizing Maps* (SOM)

JST Kohonen - SOM merupakan salah satu model JST yang menggunakan metode *unsupervised learning*, yaitu mempelajari distribusi himpunan pola-pola tanpa informasi kelas. Jaringan ini terdiri dari dua lapisan (*layer*), yaitu lapisan input dan lapisan output. Setiap neuron dalam lapisan input terhubung dengan setiap neuron pada lapisan output. Setiap neuron dalam lapisan output merepresentasikan kelas (*cluster*) dari input yang diberikan.

Prinsip kerja dari algoritma SOM adalah pengurangan node-node tetangganya (*neighbor*), sehingga pada akhirnya hanya ada satu node output yang terpilih (*winner node*). Pertama kali yang dilakukan adalah melakukan inisialisasi bobot untuk tiap-tiap node dengan nilai random. Setelah diberikan bobot random, maka jaringan diberi input sejumlah dimensi node/neuron input (10x10). Setelah input diterima jaringan, maka jaringan mulai melakukan perhitung jarak vektor yang didapatkan dengan menjumlah selisih/jarak antara vektor input dengan vektor bobot.

Pada jaringan ini, suatu lapisan yang berisi neuron-neuron akan menyusun dirinya sendiri berdasarkan input nilai tertentu dalam suatu kelompok yang dikenal dengan istilah *cluster*. Selama proses penyusunan diri, *cluster* yang memiliki vector bobot paling cocok dengan pola input (memiliki jarak yang paling dekat) akan terpilih sebagai pemenang. Neuron yang menjadi pemenang beserta neuron-neuron tetangganya akan memperbaiki bobot-bobotnya. Berikut ilustrasi arsitektur JST SOM:



Gambar 2.8 Arsitektur JST-SOM

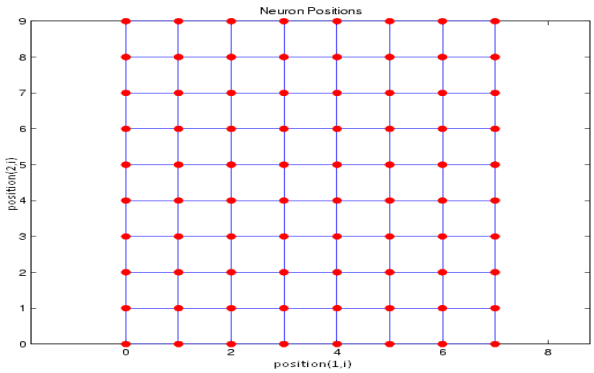
Bobot vektor-vektor contoh berfungsi sebagai penentu kedekatan antara vektor tersebut dengan masukan yang diberikan. Selama proses pengaturan, vektor contoh pada saat itu paling dekat dengan masukan akan muncul sebagai pemenang, kemudian vektor pemenang dan vektor-vektor sekitarnya akan dimodifikasi bobotnya.

Dalam JST SOM, neuron target tidak diletakkan dalam sebuah baris seperti layaknya model jaringan syaraf tiruan yang lain. Neuron target diletakkan dalam dua dimensi yang bentuk/topologinya dapat diatur.

Ada tiga macam topologi yang dapat dibentuk yaitu [7]:

1. Gridtop

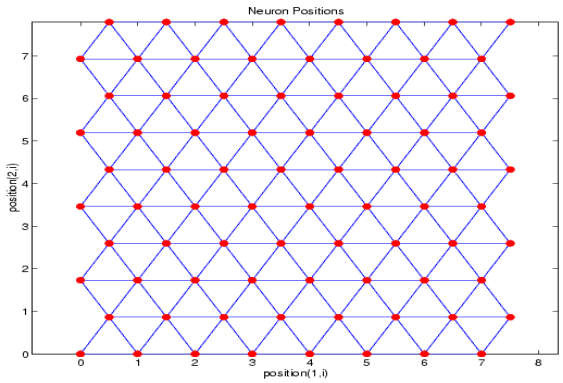
   Dalam gridtop neuron disusun dalam array dua dimensi dengan bentuk persegi.



Gambar 2.9 Topologi Gridtop

2. Hextop

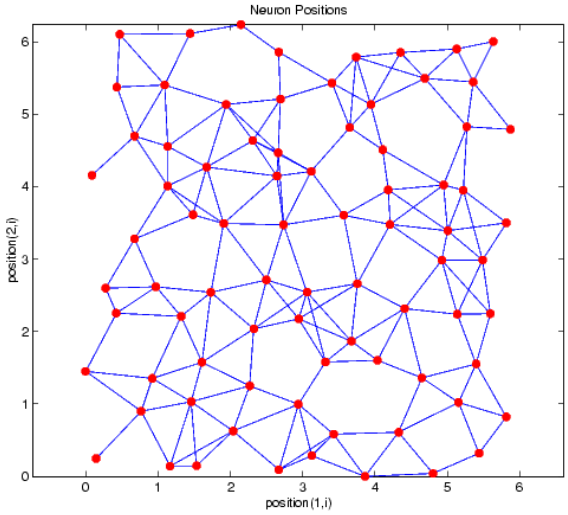
   Dalam hextop neuron disusun dalam array dua dimensi dengan bentuk heksagonal.



Gambar 2.10 Topologi Hextop

3. Randtop

   Dalam randtop neuron disusun dalam array dua dimensi dengan susunan acak.



Gambar 2.11 Topologi Randtop

2.3.2 Jarak antara Neuron (Fungsi Jarak)

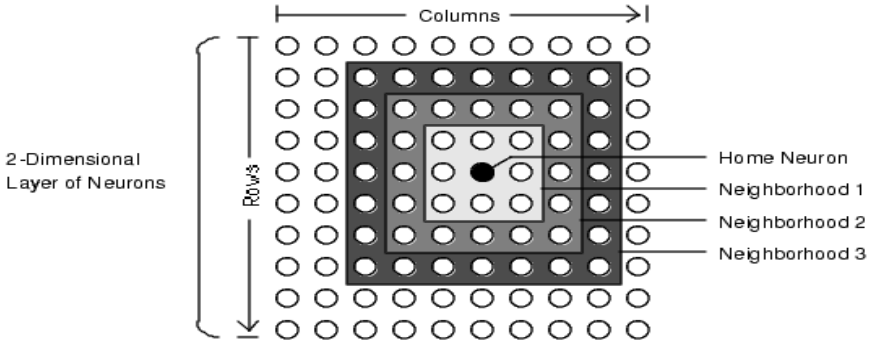
Dalam SOM, perubahan bobot tidak hanya dilakukan pada bobot garis yang terhubung ke neuron pemenang saja, tetapi juga pada bobot garis ke neuron-neuron di sekitarnya. neuron di sekitar neuron pemenang ditentukan berdasarkan jaraknya dari neuron pemenang. Ada 4 macam definisi jarak antara 2 neuron yaitu [7]:

1. Jarak *Euclidean distance* (*Euclidist*)

   Jarak *Euclidean distance* adalah jarak antara 2 titik dalam posisi berbeda yang biasa kita kenal. Misal (x1,y1) dan (x2,y2) adalah koordinat 2 neuron, maka jarak kedua neuron tersebut didefinisikan sebagai:

$$d = \sqrt{(x2 - x1)^2 + y2 - y1^2} \qquad (2.11)$$

2. Jarak Persegi (*Boxdist*)

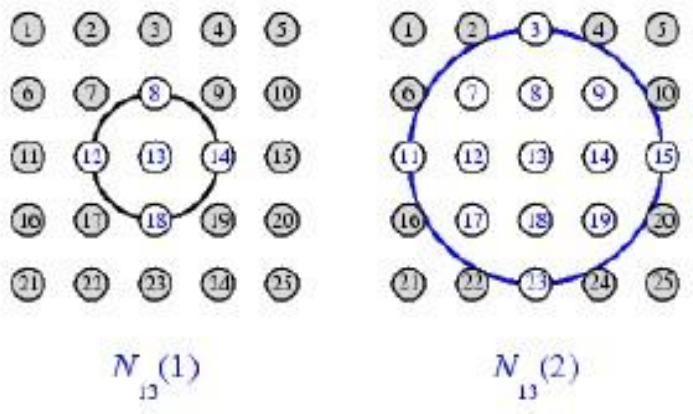


Gambar 2.12 Boxdist pada topologi Gridtop

Jarak persegi adalah jarak langsung antara neuron pemenang dengan neuron tetangganya secara langsung. Sebagai contoh apabila topologi neuron adalah gridtop, maka paling banyak terdapat 8 buah neuron tetangga dengan boxdist=1, dan terdapat paling banyak 16 tetangga jika boxdist=2. Jika neuron pemenang terletak di pinggir maka hanya ada 5 neuron tetangga dengan boxdist=1.

3. Jarak Link (*Linkdist*)

Jarak link adalah jumlah langkah yang dibutuhkan untuk menuju neuron tersebut. Jika dalam jaringan SOM menggunakan topologi gridtop dengan linkdist=1, berarti hanya neuron-neuron yang berhubungan langsung dengan neuron pemenang saja yang diubah bobotnya (jumlah neuron tetangga=4). Jika linkdist=2 maka jumlah neuron tetangga=12.



Gambar 2.13 LinkDist pada topologi gridtop

4. Jarak Manhattan (mandist)

   Jarak Manhattan antara vector x (x1,x2,...,xn) dan vector y (y1,y2,...,yn) didefinisikan sebagai:

$$sum\big(abs(x-y)\big) = \sum_{i=1}^{n} |xi - yi| \qquad (2.12)$$

   Jika x=(x1,x2) dan y=(y1,y2) menyatakan koordinat neuron yang dibentuk melalui topologi tertentu, maka jarak Manhattan antara neuron x dan y adalah

$$D = |x1 - y1| + |x2 - y2| \qquad (2.13)$$

2.4 Basis Data

Sistem basis data adalah sistem terkomputerisasi yang tujuan utamanya adalah memelihara data yang sudah diolah atau informasi dan membuat informasi tersedia saat dibutuhkan . Pada intinya basis data adalah media untuk menyimpan data agar dapat diakses dengan mudah dan cepat [8]. Untuk basis data yang digunakan dalam Matlab yaitu berformat *.mat.

2.5 Analisis Terstruktur

2.5.1 Pengertian Analisis Terstruktur

Pemrograman terstruktur adalah konsep atau paradigma atau sudut pandang pemrograman yang membagi-bagi program berdasarkan fungsi-fungsi atau prosedur-prosedur yang dibutuhkan program komputer. Modul-modul biasanya dibuat dengan mengelompokan fungsi-fungsi dan prosedur-prosedur yang diperlukan sebuah proses tertentu.

Fungsi dan prosedur ditulis secara sekuensial atau terurut dari atas kebawah sesuai dengan kebergantungan antar fungsi atau prosedur. Pemodelan pada pemrograman terstruktur dibagi berdasarkan fungsi-fungsi dan prosedur-prosedur. Oleh karena itu, pemodelan pada pemrograman terstruktur lebih fokus tentang bagaimana memodelkan data dan fungsi-fungsi dan prosedur-prosedur yang harus dibuat. Jenis paradigma pemrograman yang digunakan dapat dideteksi dari bahasa pemrograman yang digunakan [8].

2.5.2 Perangkat Pemodelan Analisis Terstruktur

Perangkat Pemodelan Analisis Terstruktur adalah alat bantu pemodelan yang digunakan untuk menggambarkan hasil pelaksanaan Analisis Terstruktur. Perangkat Analisis Terstruktur antara lain adalah [8]:

1) Diagram Konteks

   Diagram konteks adalah diagram yang terdiri dari suatu proses dan menggambarkan ruang lingkup suatu sistem. Diagram konteks merupakan level tertinggi dari DFD yang menggambarkan seluruh *input* ke sistem atau *output* dari sistem. Ia akan memberi gambaran tentang keseluruhan sistem. Dalam diagram konteks hanya ada satu proses. Tidak boleh ada *store* dalam diagram konteks. Diagram konteks berisi gambaran umum (secara garis besar) sistem yang akan dibuat. Secara kalimat, dapat dikatakan bahwa diagram konteks ini berisi siapa saja yang memberi data (dan data apa saja) ke sistem, serta kepada siapa saja informasi (dan informasi apa saja) yang harus dihasilkan sistem.

2) Diagram Aliran Data atau *Data Flow Diagram* (DFD)

   Diagram untuk menggambarkan aliran data dalam sistem, sumber dan tujuan data, proses yang mengolah data tersebut, dan tempat penyimpanan datanya. Keuntungan dari diagram arus data adalah memungkinkan pengembangan sistem dari level yang paling tinggi dan memecah menjadi level yang lebih rendah. Beberapa simbol yang digunakan DFD adalah sebagai berikut :

   a) Entitas Luar (*External Entity*)

      Suatu yang berada diluar sistem, tetapi ia memberikan data ke dalam sistem atau memberikan data dari sistem, disimbolkan dengan suatu kotak notasi. *External Entity* tidak termasuk bagian dari sistem. Bila sistem informasi dirancang untuk suatu bagian lain yang masih terkait menjadi *external entity*.

   b) Proses (*Process*)

      Proses merupakan apa yang dikerjakan oleh sistem. Proses dapat mengolah data atau aliran data masuk menjadi aliran data keluar. Proses berfungsi mentransformasikan suatu atau beberapa data keluaran sesuai dengan spesifikasi yang diinginkan. Setiap proses memiliki satu atau beberapa

masukan serta menghasilkan satu atau beberapa data kelurahan. Proses sering juga disebut *bubble*.

c) Arus Data (*Data Flow*)

Arus data merupakan tempat mengalirnya informasi dan digambarkan dengan garis yang menghubungkan komponen dari sistem. Arus data ditunjukan dengan arah panah dan garis diberi nama atas arus data yang mengalir. Arus data ini mengalir diantara proses, *data store* dan menunjukan arus data dari data yang berupa masukan untuk sistem atau hasil proses sistem.

d) Simpanan Data (*Data Store*)

Simpanan data merupakan tempat penyimpanaan data yang ada dalam sistem. *Data store* dapat disimbolkan dengan dua garis sejajar atau dua garis dengan salah satu sisi samping terbuka. Proses dapat mengambil data dari atau memberikan data ke simpanan data (*database*).

3) Kamus Data (*Data Dictionary*)

Merupakan suatu tempat penyimpanan (gudang) dari data dan informasi yang dibutuhkan oleh suatu sistem informasi. Kamus data digunakan untuk mendeskripsikan rincian dari aliran data atau informasi yang mengalir dalam sistem, elemen-elemen data, *file* maupun basis data (tempat penyimpanan) dalam DFD.

2.6 Rekayasa Perangkat Lunak

Rekayasa perangkat lunak (*software engineering*) merupakan pembangunan dengan menggunakan prinsip atau konsep rekayasa dengan tujuan menghasilkan perangkat lunak yang bernilai ekonomi, dipercaya dan bekerja secara efisien menggunakan mesin. Rekayasa perangkat lunak lebih fokus pada praktik pengembangan perangkat lunak dan mengirimkan perangkat lunak yang bermanfaat bagi pengguna. Rekayasa perangkat lunak fokus pada bagaimana membangun perangkat lunak yang memenuhi kriteria berikut:

1. Dapat terus dipelihara setelah perangkat lunak selesai dibuat seiring berkembangnya teknologi dan lingkungan.
2. Dapat diandalkan dengan proses bisnis yang dijalankan dan perubahan yang terjadi.

3. Efisien dari segi sumber daya dan penggunaan.
4. Kemampuan untuk dipakai sesuai dengan kebutuhan.

Pekerjaan yang terkait dengan rekayasa perangkat lunak dapat dikategorikan menjadi tiga buah kategori umum tanpa melihat area dari aplikasi, ukuran proyek atau kompleksitas perangkat lunak yang akan dibangun, yaitu:

1. Fase Pendefinisian (*Definition Phase*)

   Fase pendefinisian fokus pada "*what*" yang artinya harus mencari tahu atau mengidentifikasi informasi apa yang harus diproses, seperti apa fungsi dan performansi yang diinginkan, seperti apa prilaku sistem yang diinginkan, apa kriteria validasi yang dibutuhkan untuk mendefinisikan sistem.
2. Fase Pengembangan (*Development Phase*)

   Fase pengembangan fokus pada "*how*" yang artinya selama tahap pengembangan perangkat lunak seorang perekayasa perangkat lunak (*software engineer*) berusaha untuk mendefinisikan bagaimana data diinstrukturkan dan bagaimana fungsi-fungsi yang dibutuhkan diimplementasikan, bagaimana tampilan karakter antarmuka, bagaimana desain ditranslasikan ke dalam bahasa pemrograman dan bagaimana pengujian dijalankan.
3. Fase Pendukung (*Support Phase*)

   Fase pendukung fokus pada perubahan yang terasosiasi pada perbaikan kesalahan (*error*), adaptasi yang dibutuhkan pada lingkungan perangkat lunak yang terlibat dan perbaikan yang terjadi akibat perubahan kebutuhan pengguna. Fase pendukung terdiri dari empat tipe perubahan antara lain:

   a. Koreksi (*Correction*), yaitu pemeliharaan dengan melakukan perbaikan terhadap kecacatan perangkat lunak.
   b. Adaptasi (*Adaptation*), yaitu merupakan tahap untuk memodifikasi perangkat lunak guna mengakomodasi perubahan lingkungan luar dimana perangkat lunak dijalankan.
   c. Perbaikan (*Enchancement*), pemeliharaan perfektif atau penyempurnaan melakukan eksekusi atau penambahan pada kebutuhan fungsional sebelumnya.

d. Pencegahan (*Prevention*), pencegahan atau sering disebut juga dengan rekayasa ulang sistem (*software reengineering*) harus dikondisikan untuk mempu melayani kebutuhan yang diinginkan pemakainya.

2.7 Perangkat Lunak Pendukung

Adapun perangkat lunak pendukung yang digunakan dalam implementasi metode di aplikasi pengenalan dan identifikasi suara burung berkicau sebagai berikut :

2.7.1 MATLAB

Matlab dikembangkan sebagai bahasa pemograman sekaligus alat visualisasi, yang menawarkan banyak kemampuan untuk menyelesaikan berbagai kasus yang berhubungan langsung dengan disiplin ilmu matematika. Matlab dibangun dari bahasa induknya yaitu bahasa C, namun tidak dapat dikatakan sebagai varian dari bahasa C, karena dalam sintak maupun cara kerjanya sama sekali berbeda dari bahasa C. Namun dengan hubungan langsungnya dengan bahasa C, Matlab memiliki kelebihan-kelebihan dari bahasa C bahkan mampu berjalan pada semua *platform* Sistem Operasi tanpa mengalami perubahan sintak sama sekali. Bahasa ini mengintegrasikan kemampuan komputasi, visualisasi dan pemograman dalam sebuah lingkungan yang tunggal dan mudah digunakan. Matlab memberikan sistem interaktif yang menggunakan konsep array/matrik sebagai standar variabel elemennya tanpa membutuhkan pendeklarasian array seperti pada bahasa lainnya.

Matlab dikembangkan oleh MathWorks, yang pada awalnya dibuat untuk memberikan kemudahan mengakses data matrik pada proyek LINPACK dan EISPACK. Selanjutnya menjadi sebuah aplikasi untuk komputasi matrik. Dari sejak awal dipergunakan, matlab memperoleh masukan ribuan pemakai. Dalam lingkungan pendidikan ilmiah menjadi alat pemrograman standar bidang matematika, rekayasa dan keilmuan terkait. Dan dalam lingkungan industri dapat menjadi pilihan paling produktif untuk riset, pengembangan dan analisa [9].

2.8 Pengujian Perangkat Lunak

Pengujian perangkat lunak merupakan suatu investigasi yang dilakukan untuk mendapatkan informasi mengenai kualitas dari produk atau layanan yang

sedang diuji. Pengujian perangkat lunak juga memberikan pandangan mengenai perangkat lunak secara obyektif dan *independen*, yang bermanfaat dalam operasional bisnis untuk memahami tingkat risiko pada implementasinya. Teknik-teknik pengujian mencakup, namun tidak terbatas pada, proses mengeksekusi suatu bagian program atau keseluruhan aplikasi dengan tujuan untuk menemukan "*bug*" perangkat lunak. *Bug* merupakan suatu kesalahan desain pada suatu perangkat keras komputer atau perangkat lunak komputer yang menyebabkan peralatan atau program itu tidak berfungsi semestinya. *Bug* umumnya lebih umum dalam dunia perangkat lunak dibandingkan dengan perangkat keras [10].

Pengujian perangkat lunak merupakan suatu tahapan penting dalam pembangunan perangkat lunak. Pengujian dilakukan dengan cara mengevaluasi konfigurasi perangkat lunak yang terdiri dari spesifikasi kebutuhan, deskripsi perancangan, dan program yang dihasilkan. Hasil evaluasi kemudian dibandingkan dengan hasil uji yang diharapkan. Jika ditemukan kesalahan, maka perbaikan perangkat lunak harus dilakukan untuk kemudian diuji kembali.

Pengujian perangkat lunak adalah proses menjalankan dan mengevaluasi sebuah perangkat lunak secara manual maupun otomatis untuk menguji apakah perangkat lunak sudah memenuhi persyaratan atau belum, atau untuk menentukan perbedaan antara hasil yang diharapkan dengan hasil sebenarnya. Peksanaan pengujian perangkat lunak biasanya disesuaikan dengan metodologi pembangunan perangkat lunak yang digunakan.

Pengujian dikatakan berhasil apabila pengujian yang dilakukan memiliki probabilitas tinggi untuk menemukan kesalahan yang belum pernah ditemukan sebelumnya.

2.8.1 Teknik Pengujian

Teknik pengujian yang dapat digunakan dalam menguji perangkat lunak sebagai berikut :

a. Pengujian *White Box*

Pengujian *white box* adalah pengujian yang didasarkan pada pengecekan terhadap detail perancangan, menggunakan struktur kontrol dari desain program secara procedural untuk membagi pengujian ke dalam beberapa kasus pengujian.

Secara sekilas dapat diambil kesimpulan *white box testing* merupakan petunjuk untuk mendapatkan program yang benar secara 100%.

Penggunaan metode pengujian *white box* dilakukan untuk :

1) Memberikan jaminan bahwa semua jalur independen suatu modul digunakan minimal satu kali
2) Menggunakan semua keputusan logis untuk semua kondisi true atau false
3) Mengeksekusi semua perulangan pada batasan nilai dan operasional pada setiap kondisi.
4) Menggunakan struktur data internal untuk menjamin validitas jalur keputusan.

Persyaratan dalam menjalankan strategi *White Box Testing* sebagai berikut:

1) Mendefinisikan semua alur logika
2) Membangun kasus untuk digunakan dalam pengujian
3) Mengevaluasi semua hasil pengujian
4) Melakukan pengujian secara menyeluruh

b. Pengujian *Black Box* :

Pengujian *black box* digunakan untuk menguji fungsi-fungsi khusus dari perangkat lunak yang dirancang. Pada teknik ini, kebenaran perangkat lunak yang diuji hanya dilihat berdasarkan keluaran yang dihasilkan dari data atau kondisi masukan yang diberikan untuk fungsi yang ada tanpa melihat bagaimana proses untuk mendapatkan keluaran tersebut. Dari keluaran yang dihasilkan, kemampuan program dalam memenuhi kebutuhan pemakai dapat diukur sekaligus dapat diiketahui kesalahan-kesalahannya.

Beberapa jenis kesalahan yang dapat diidentifikasi :

1) Fungsi tidak benar atau hilang
2) Kesalahan antar muka
3) Kesalahan pada struktur data (pengaksesan basis data)
4) Kesalahan inisialisasi dan akhir program
5) Kesalahan performasi

Walaupun sulit untuk menelusuri kesalahan yang mungkin didapat, teknik pengujian *black box* lebih sering dipilih untuk menguji perangkat lunak karena kemudahan dalam pelaksanaannya.