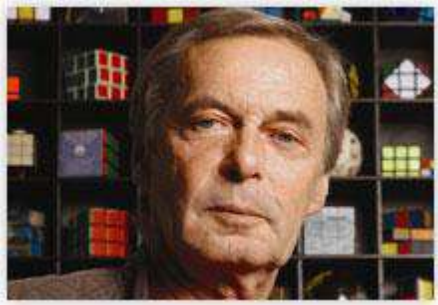
BAB 2
TINJAUAN PUSTAKA

2.1 Rubik's CUBE

Rubik's CUBE adalah sebuah *twisty puzzle*, yaitu permainan *puzzle* mekanik tiga dimensi yang ditemukan pada tahun 1974 oleh seorang pemahat dan profesor arsitektur dari *Hungaria* bernama *Erno Rubik* [1]. *Rubik's* memberi nama hasil temuannya itu *Magic CUBE*, yang kemudian dipatenkan di *Hungaria* dan dijual pertama kali melalui perusahaan *Ideal Toy Corporation*. Pada tahun 1980, perusahaan *Ideal Toy* mengubah nama *magic CUBE* tersebut menjadi "*Rubik's CUBE*". Dan hingga saat ini, lebih dari 350 juta *Rubik's CUBE* telah dijual di seluruh dunia. **Gambar** Error! No text of specified style in document.**-1** adalah *Erno Rubik*, pencipta *Rubik's CUBE*.

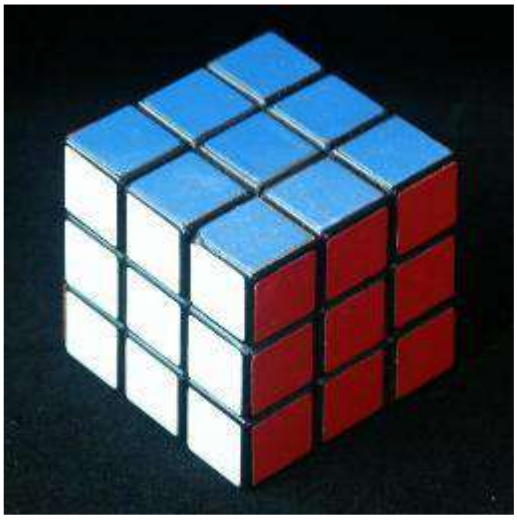


Gambar Error! No text of specified style in document.**-1 *Erno Rubik* [1]**

2.1.1 Varian-varian *Rubik's CUBE*

Sebuah *Rubik's CUBE* standard (3x3x3) terbentuk dari 26 kubus kecil yang disebut juga *CUBElets* atau *cubies*, terdiri dari 6 *cubies* yang diletakkan pada setiap ujung dari sebuah kerangka yang mempunyai 6 axis. 20 *cubies* lainnya diapasang sedemikian rupa pada kerangka tersebut sehingga terbentuk sebuah *Rubik's CUBE*. Setiap sisi *Rubik's CUBE* memiliki 9 permukaan yang terdiri dari enam warna yang

berbeda **Gambar** Error! No text of specified style in document.**-2** memperlihatkan sebuah *Rubik's CUBE* standard.



Gambar Error! No text of specified style in document.**-2 *Rubik's CUBE* Standar [1]**

Standard *CUBE* merupakan model yang paling populer hingga dibuat berbagai macam varian-nya [5] antara lain:

1. *Master CUBE* (4x4x4), dikeluarkan oleh *Rubik's* dengan nama *Rubik's Revenge*, diciptakan oleh *Péter Sebestény*, pengembangan *Rubik's CUBE* dengan tingkat kesulitan yang lebih tinggi, memiliki 2 macam mekanisme berbeda yaitu keluaran *Rubik's Revenge* & *Eastsheen*. Ukuran standar adalah 6.5 cm^3.
2. *Professor's CUBE* (5x5x5), diciptakan oleh *Udo Krell*, karena memiliki *center pieces, puzzle* ini memiliki metode mirip seperti 3x3x3 namun dengan kesulitan jauh lebih tinggi. 2 variasi lain dari mekanisme 5x5x5 selain keluaran *Rubik's* adalah *V-CUBE* & *Eastsheen*. Ukuran standar jenis ini adalah 7 cm^3.
3. *V-CUBE* 6 (6x6x6), diciptakan oleh *Panagiotis Verdes*, *twisty puzzle* dengan total 152 keping bila dipisah2, karya genius dari *Verdes Innovations SA*. Ukuran standar yang dikeluarkan adalah 6.9 cm^3.
4. *V-CUBE* 7 (7x7x7), penciptanya sama seperti *V-CUBE* 6, juga dengan mekanisme yang sama, *twisty puzzle* jenis *cubic* original dengan tingkat kesulitan paling kompleks. Ukuran standar jenis ini adalah 7.2 cm^3.
5. *Pocket CUBE* (2x2x2), disebut juga *Mini CUBE*.

Disamping bentuk kubus standar terdapat pula berbagai varian *twisty puzzle* seperti yang dikembangkan oleh *Tony Fisher* dimulai dengan *Fisher's CUBE* yang memiliki metode penyelesaian & mekanisme mirip dengan 3x3x3 namun bisa diputar secara diagonal, serta berbagai varian lainnya. **Gambar** Error! No text of specified style in document.**-3** memperlihatkan berbagai varian *Rubik's CUBE* oleh *Tony Fisher*.



Gambar Error! No text of specified style in document.**-3 Variansi *Rubik's CUBE* oleh *Tony Fisher* [5]**

2.1.2 Aturan Permainan

Sampai saat ini aturan permainan *Rubik's CUBE* masih belum berubah. Hal ini dikarenakan tujuan dari permainan ini sama dengan tujuan permainan *puzzle* lainnya. Dalam permainan *Rubik's CUBE*, kondisi permasalahan telah selesai yaitu ketika setiap sisi telah tersusun atas warna yang sama dengan cara menggerakkan sisi-sisinya.

2.1.3 Heuristik Untuk *Rubik's CUBE*

Heuristik yang digunakan untuk memperkirakan jarak dari suatu konfigurasi *Rubik's CUBE* acak ke konfigurasi yang terselesaikan dapat menggunakan 3D *Manhattan Distance* yaitu jumlah langkah minimum yang dibutuhkan untuk mengembalikan posisi setiap *cubies* ke posisi dan orientasi yang sebenarnya [6]. Nilai tersebut harus dibagi 8 agar *admissible*, karena setiap langkah menggerakkan 4 *corner cubies* (sudut) dan 4 *edge cubies* (rusuk). Fungsi heuristik yang lebih baik adalah dengan mengambil jumlah maksimum 3D *Manhattan Distance* dari *corner cubies* dan *edge cubies* masing masing dibagi 4. Nilai

Manhattan Distance yang diharapkan adalah 5.5 untuk *edge cubies* dan sekitar 3 untuk *corner cubies*, karena terdapat 12 *edge cubies* dan 8 *corner cubies*.

2.1.4 God's Number

Setiap metode untuk menyelesaikan *Rubik's CUBE* menggunakan suatu Algoritma tertentu, yaitu serangkaian langkah untuk menyelesaikan *Rubik's CUBE*. Terdapat banyak Algoritma yang berbeda beda, baik dari segi kompleksitas maupun jumlah langkah yang dibutuhkan, tetapi umumnya Algoritma yang dapat dihafalkan oleh manusia biasa membutuhkan lebih dari 40 langkah *face turn* [7].

Bila diasumsikan bahwa Tuhan pasti menggunakan Algoritma yang lebih efisien yaitu Algoritma yang selalu dan pasti menggunakan serangkaian langkah terpendek, maka Algoritma tersebut disebut sebagai *God's Algorithm*, Jumlah langkah yang dibutuhkan oleh Algoritma tersebut pada *worst case* disebut *God's Number*.

Pada tahun 2010 *Tomas Rokicki*, *Herbert Kociemba*, *Morley Davidson*, dan *John Dethridge* membuktikan bahwa *God's Number* untuk *Rubik's CUBE* adalah 20, dengan menggunakan sekitar 35 tahun *CPU* dari komputer-komputer yang disediakan oleh Google [7].

2.2 Kecerdasan Buatan

Kecerdasan buatan merupakan salah satu bidang ilmu komputer yang didefinisikan sebagai kecerdasan yang dibuat untuk suatu sistem dengan menggunakan Algoritma - Algoritma tertentu sehingga sistem tersebut seolah-olah dapat berpikir seperti manusia [9]. Kecerdasan buatan merupakan cabang dari ilmu komputer yang dalam merepresentasi pengetahuan, lebih banyak menggunakan bentuk simbol-simbol dari pada bilangan, dan memproses informasi berdasarkan metode heuristik atau dengan berdasarkan sejumlah aturan.

Dari beberapa pengertian di atas, maka dapat ditarik suatu kesimpulan bahwa kecerdasan buatan ialah salah satu bagian dari ilmu komputer yang mempelajari perancangan sistem komputer yang cerdas. Maksud dari sistem cerdas yaitu suatu sistem yang dapat memperlihatkan karakteristik yang ada pada tingkah

laku manusia, seperti mengerti suatu bahasa, mempelajari, mempertimbangkan, dan memecahkan suatu masalah.

Kecerdasan buatan telah memberikan suatu kemampuan baru kepada komputer untuk memecahkan masalah yang lebih besar dan lebih luas, tidak hanya terbatas pada soal-soal perhitungan, penyimpanan data, pengambilan data atau pengendalian yang sederhana saja.

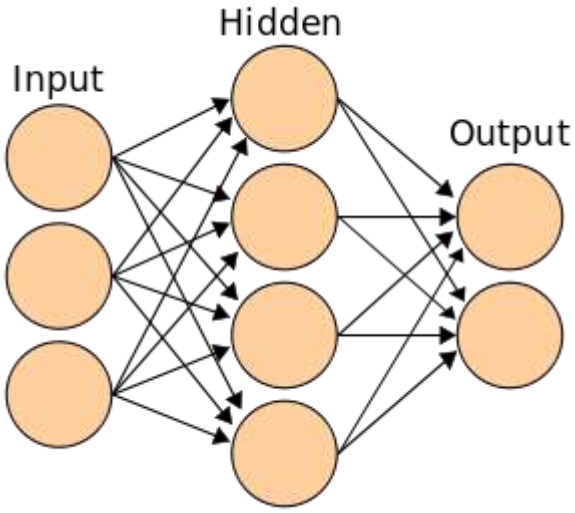
2.2.1 Tujuan Akhir Kecerdasan Buatan

Menurut *Lenat* dan *Feigenbaum* , terdapat sembilan tujuan akhir dari kecerdasan buatan, yaitu :

1. Memahami pola pikir manusia, mencoba untuk mendapatkan pengetahuan ingatan manusia yang mendalam, kemampuan dalam memecahkan masalah, belajar, dan mengambil keputusan.
2. *Otomatisasi* menciptakan sistem yang dapat menggantikan manusia dalam tugas-tugas *intelegensi*. Menggunakan sistem yang performanya sebaik manusia dalam melakukan pekerjaan.
3. Penguatan *intelegensi* membangun sistem untuk membantu manusia agar mampu berpikir lebih baik dan lebih cepat
4. *Intelegensi* manusia super, membangun sistem yang mempunyai kemampuan untuk melebihi intelegensi manusia
5. Menyelesaikan permasalahan, sistem mampu menyelesaikan berbagai masalah yang luas.
6. Wacana *koheren* mampu berkomunikasi dengan manusia dengan menggunakan bahasa alami.
7. Belajar mampu memperoleh data sendiri dan mengetahui bagaimana cara memperoleh data. Sistem mampu membuat hipotesis, penerapan atau pembelajaran secara *heuristik* dan membuat alasan dengan analogi.
8. Otonomi mempunyai sistem intelegensi yang beraksi atas inisiatif sendiri.
9. Informasi mampu menyimpan informasi dan mengetahui cara mengambil informasi.

2.2.2 Jaringan Saraf Tiruan

Jaringan saraf tiruan (JST) (Bahasa Inggris A*rtificial Neural Network* (*ANN*) atau juga disebut *Simulated Neural Network* (*SNN*), atau umumnya hanya disebut *Neural Network* (*NN*) adalah jaringan dari sekelompok *unit* pemroses kecil yang dimodelkan berdasarkan jaringan saraf manusia. JST merupakan sistem adaptif yang dapat mengubah strukturnya untuk memecahkan masalah berdasarkan informasi eksternal maupun internal yang mengalir melalui jaringan tersebut. Secara sederhana, JST adalah sebuah alat pemodelan data statistik non-linier. JST dapat digunakan untuk memodelkan hubungan yang kompleks antara *input* dan *output* untuk menemukan pola-pola pada data diperlihatkan pada **Gambar** Error! No text of specified style in document.**-4** dibawah ini.



Gambar Error! No text of specified style in document.**-4 Pola *Input* dan *Output* JST**

Ada beberapa efinisi dari dari Jaringan Saraf Tiruan diantaranya :

1. *Hecht-Nielsend* (1988) mendefinisikan sistem saraf buatan sebagai berikut :
   "Suatu *Neural Network* (*NN*) adalah suatu struktur pemroses informasi yang terdistribusi dan bekerja secara paralel, yang terdiri atas elemen pemroses (yang

memiliki memori lokal dan beroperasi dengan informasi lokal) yang diinterkoneksi bersama dengan alur sinyal searah yang disebut koneksi. Setiap elemen pemroses memiliki koneksi keluaran tunggal yang bercabang (*fan out*) ke sejumlah koneksi kolateral yang diinginkan (setiap koneksi membawa sinyal yang sama dari keluaran elemen pemroses tersebut). Keluaran dari elemen pemroses tersebut dapat merupakan sebarang jenis persamaan matematis yang diinginkan. Seluruh proses yang berlangsung pada setiap elemen pemroses harus benar-benar dilakukan secara lokal, yaitu keluaran hanya bergantung pada nilai masukan pada saat itu yang diperoleh melalui koneksi dan nilai yang tersimpan dalam memori lokal".

2. Menurut *Haykin, S*. (1994), *Neural Networks* : *A Comprehensive Foundation, NY, Macmillan,* mendefinisikan jaringan saraf sebagai berikut:

   "Sebuah jaringan saraf adalah sebuah prosesor yang terdistribusi paralel dan mempuyai kecenderungan untuk menyimpan pengetahuan yang didapatkannya dari pengalaman dan membuatnya tetap tersedia untuk digunakan. Hal ini menyerupai kerja otak dalam dua hal yaitu:

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

3. Dan menurut *Zurada, J.M.* (1992), *Introduction To Artificial Neural Systems, Boston* : *PWS Publishing Company,* mendefinisikan sebagai berikut :

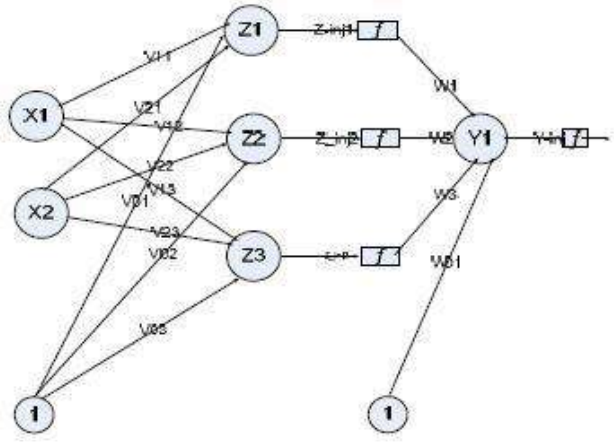
   "Sistem saraf tiruan atau jaringan saraf tiruan adalah sistem selular fisik yang dapat memperoleh, menyimpan dan menggunakan pengetahuan yang didapatkan dari pengalaman".

4. *DARPA Neural Network Study* (1988, *AFCEA International Press, p.* 60) mendefinisikan jaringan syaraf buatan sebagai berikut :

   "Sebuah jaringan syaraf adalah sebuah sistem yang dibentuk dari sejumlah elemen pemroses sederhana yang bekerja secara paralel di mana fungsinya ditentukan oleh stuktur jaringan, kekuatan hubungan, dan pegolahan dilakukan pada komputasi elemen atau nodes".

2.2.3 Backpropagation

Salah satu arsitektur jaringan saraf tiruan yang sangat populer adalah multilayer *feedforward networks*. Secara umum, jaringan seperti ini terdiri dari sejumlah *unit neuron* sebagai lapisan masukan, satu atau lebih lapisan simpul - simpul *neuron* komputasi lapisan tersembunyi, dan sebuah lapisan simpul - simpul *neuron* komputasi keluaran. Sinyal masukan dipropagasikan ke arah depan (arah lapisan keluaran), lapisan demi lapisan. Jenis jaringan ini adalah hasil generalisasi dari arsitektur *perceptron* satu lapisan, jadi biasa disebut sebagai multilayer *perceptron* (*MLPs*). *Error Backpropagation* adalah Algoritma *MLPs* yang menggunakan prinsip pembelajaran terawasi. Propagasi balik (ke arah lapisan masukan) terjadi setelah jaringan menghasilkan keluaran yang mengandung *error*. Pada fase ini seluruh bobot *synaptic* (yang tidak memiliki aktivasi nol) dalam jaringan akan disesuaikan untuk mengkoreksi / memperkecil *error* yang terjadi (*error correction rule*). Untuk pelatihan jaringan, pasangan fase propagasi ke depan dan balik dilakukan secara berulang untuk satu set data latihan, kemudian diulangi untuk sejumlah *Epoch* (satu sesi lewatan untuk seluruh data latihan dalam sebuah proses pelatihan jaringan) sampai *error* yang terjadi mencapai batas kecil toleransi tertentu atau nol.



Gambar Error! No text of specified style in document.**-5 Bentuk Jaringan *Backpropagation***

Dapat dilihat bahwa yang bertindak sebagai dendrit adalah X1 dan X2, yaitu data masukan pada jaringan tersebut. Terdapat 2 sinapsis atau bobot yaitu V dan W, sedangkan Z dan Y merupakan bagian dari *soma* atau badal sel dari jaringan

tersebut. Dan yang bertindak sebagai akson atau data keluaran adalah Y. Berikut ini Algoritma pembelajaran metode *Backpropagation*.

1. Menentukan Nilai Bobot

   Langkah 0 :

   Pemberian inisialisasi penimbang (diberi nilai kecil secara acak)

   Langkah 1 :

   Jika kondisi berhenti tidak tercapai, Ulangi langkah 2 hingga 9 sampai kondisi akhir iterasi dipenuhi(kondisi berhenti jika perulangan mencapai maksimal perulangan atau M*ean Square Error* (*MSE*) sudah lebih kecil dari rasio pembelajaran)

   Langkah 2 :

   Untuk masing-masing pasangan data pelatihan (*Training* data) lakukan langkah 3 hingga 8

2. Propagasi maju (*Feedforward*)

   Langkah 3 :

   Masing-masing *unit* masukan (Xi, I = 1,…,n) menerima sinyal masukan Xi dan sinyal tersebut disebarkan ke *unit-unit* bagian berikutnya (*unit-unit* lapisan tersembunyi)

   Langkah 4 :

   Menghitung nilai *Hidden* dan *Net*, Untuk menghitung nilai *Net*, *input*an harus dinormalisasi dulu dengan menggunakan rumus *sigmoid bipolar*, harus dinormalisasi dengan menggunakan persamaan

   $$\boldsymbol{y = f(x) = \frac{1-e^{-x}}{(1+e^{-x})}}$$ **(**Error!

No text of specified style in document.**-1)**

   Masing-masing *unit* dilapisan tersembunyi dikalikan dengan faktor penimbang dan dijumlahkan serta ditambahkan dengan biasnya :

   $$\boldsymbol{Z_{in_j} = V_{oj} + \sum_{I=1}^{n} X_i V_{ij}}$$ **(**Error!

No text of specified style in document.**-2)**

   Kemudian menghitung sesuai dengan fungsi aktifasi yang digunakan :

$$\boldsymbol{Z_j = f(Z_{in_j})}$$ **(**Error!

No text of specified style in document.**-3)**

Bila yang digunakan adalah fungsi *sigmoid* maka bentuk fungsi tersebut adalah

$$\boldsymbol{Z_j = \frac{1}{1+exp^{-z_in_j}}}$$ **(**Error!

No text of specified style in document.**-4)**

Masing-masing *unit* keluaran (Yk, k=1,2,3,…m) dikalikan dengan faktor penimbang dan dijumlahkan :

$$\boldsymbol{Y_{in_k} = W_{ok} + \sum_{j=1}^{p} Z_j W_{jk}}$$ **(**Error!

No text of specified style in document.**-5)**

Menghitung kembali sesuai dengan fungsi aktifasi

$$\boldsymbol{y_k = f(y_{in_k})}$$ **(**Error!

No text of specified style in document.**-6)**

3. Propagasi Mundur *(Backpropagation)* dan Galatnya

Langkah 5 :

Masing-masing *unit* keluaran (Yk, k=1,…,m) menerima pola target sesuai dengan pola masukan saat pelatihan dan dihitung galatnya :

$$\boldsymbol{\delta_k = (t_k - y_k) f'(y_in_k)}$$ **(**Error!

No text of specified style in document.**-7)**

Karena $f'(y_in_k) = y_k$ menggunakan fungsi sigmoid, maka :

$$\boldsymbol{f'(y_in_k) = f(y_in_k)(1 - f(y_in_k))}$$

$$\boldsymbol{= y_k(1 - y_k)}$$ **(**Error!

No text of specified style in document.**-8)**

Menghitung perbaikan faktor penimbang (kemudian untuk memperbaiki $\boldsymbol{w_{jk}}$

$$\boldsymbol{\Delta W_{kj} = \alpha . \partial_k . Z_j}$$ **(**Error!

No text of specified style in document.**-9)**

Menghitung perbaikan koreksi :

$\Delta W_{0j} = \alpha . \partial_k$ **(**Error!

No text of specified style in document.**-10)**

Dan menggunakan nilai δ_k pada semua *unit* lapisan sebelumnya.

Langkah 6:

Masing-masing penimbang yang menghubungkan *unit-unit* l;apisan keluaran dengan *unit-unit* pada lapisan tersembunyi $Z_j, j = 1, \dots, p$) dikalikan delta dan dijumlahkan sebagai masukan ke *unit-unit* lapisan berikutnya.

$\delta_in_j = \sum_{k=1}^{n} \delta_k W_{jk}$ **(**Error!

No text of specified style in document.**-11)**

Selanjutnya dikalikan dengan turunan dari fungsi aktifasinya untuk menghitung galat.

δ_j=δ_in_j f'(y _in_j) **(**Error!

No text of specified style in document.**-12)**

Kemudian menghitung perbaikan penimbang (digunakan untuk memperbaiki **V_ij)**

ΔV_ij=αδ_j X_i **(**Error!

No text of specified style in document.**-13)**

Kemudian menghitung perbaikan bias (untuk memperbaiki V_oj)

ΔV_oj=αδ_j **(**Error!

No text of specified style in document.**-14)**

Memperbaiki penimbang dan bias

Langkah 7 :

Masing-masing keluaran *unit* (yk, k=1,…,m) diperbaiki bias dan penimbangnya (j=0,…,p)

W_jk (baru)=W_jk (lama)ΔW_jk **(**Error!

No text of specified style in document.**-15)**

Masing-masing *unit* tersembunyi (Zj, j:1,…,p) diperbaiki bias dan penimbangnya (j=0,…,n)

V_jk (baru)=V_jk (lama)∆V_jk **(**Error! No text of specified style in document.**-16)**

Langkah 8 :

Uji kondisi pemberhentian (akhir iterasi) pengujian pemberhentian dilakukan dengan menghitung nilai mean *Square Error* (*MSE*). Di mana nilai *MSE* ini akan mengambil rata-rata kuadrat *error* yang terjadi antara *output* jaringan dan target, berikut adalah rumus *MSE*

$$\mathbf{MSE} = \frac{\sum_{i=1}^{n} e_j^2}{n}$$ **(**Error! No text of specified style in document.**-17)**

Tabel Error! No text of specified style in document.**-1 Daftar Keterangan Notasi *Backpropagation***

Notasi		Keterangan
X^p	=	Pola masukan pelatihan ke-p, p=1,2,…,p<=1. (X_1,X_2,X_3, …,X_n)
t^p	=	Pola keluaran target dari pelatihan. (t_1, t_2, t_3, …, t_n)
xi	=	*Unit* ke-I pada lapisan masukan
Xi	=	Nilai aktifasi dari *unit* Xi
Z_j	=	*Unit* ke-j pada lapisan tersembunyi
Z_in_j	=	Keluaran untuk *unit* Z_j
z_j	=	Nilai aktifasi dari *unit* Z_j
Y_k	=	*Unit* ke-k pada lapisan keluaran
Y_in_k	=	Net masukan untuk *unit* Y_k
W_{k0}	=	Nilai penimbang sambungan pada bias untuk *unit* Y_k

Notasi		Keterangan
W_{kj}	=	Nilai penimbang sambungan dari Z_{ij} ke *unit* Y_k
∇W_{kj}	=	Selisih $W_{kj}(t)$ dengan $W_{kj}(t+1)$
V_{io}	=	Nilai penimbang sambungan pada bias untuk *unit* Z_i
∇W_{ij}	=	Selisih antara $V_{ij}(t)$ dengan $V_{ij}(t+1)$
δ_k	=	Faktor pengaturan nilai penimbang sambungan pada lapisan keluaran
δ_j	=	Faktor pengaturan nilai penimbang sambungan pada lapisan tersembunyi
α	=	Konstanta laju pelatihan (*Learning Rate*) $0<\alpha<1$
E	=	Total galat

2.3 Pemodelan *UML*

UML (*Unified Modeling Language*) adalah salah satu alat bantu yang sangat handal di dunia pengembangan sistem yang berorientasi obyek. Hal ini disebabkan karena *UML* menyediakan bahasa pemodelan visual yang memungkinkan bagi pengembang sistem untuk membuat cetak biru atas visi mereka dalam bentuk yang baku, mudah dimengerti serta dilengkapi dengan mekanisme yang efektif untuk berbagi dan mengkomunikaskan rancangan mereka dengan yang lain.

UML merupakan kesatuan dari bahasa pemodelan yang dikembangkan oleh *Booch, Object Modeling Technique* (*OMT*) dan *Object Oriented Software Engineering* (*OOSE*). Metode *Booch* dari *Grady Booch* sangat terkenal dengan nama metode *Design Object Oriented*. Metode ini menjadikan proses analisis dan design ke dalam empat tahapan iterative, yaitu : identifikasi kelas-kelas dan obyek-obyek, identifikasi semantic dari hubungan obyek dan kelas tersebut, perincian

interface dan implementasi. Keunggulan metode *Booch* adalah pada detail dan kayanya dengan notasi dan elemen. Pemodelan *Object Modeling Technique* yang dikembangkan *Rumbaugh* didasarkan pada analisis terstruktur dan permodelan *entity-relationship*. Tahapan utama dalam metodologi ini adalah analisis, *design sistem*, *design obyek* dan implementasi. Keungulan metode ini aladah dalam penotasian yang mendukung semua konsep *Object Oriented Software Engineering*. Metode *Object Oriented Software Engineering Jacobson* lebih memberikan penekanan pada *Use Case*. *Object Oriented Software Engineering* memiliki tiga tahapan yaitu membuat *model requirement* dan analisis, *design* dan implementasi, dan model pengujian. Keungulan metode ini adalah mudah dipelajari karena memiliki notasi yang sederhana namun mencakup seluruh tahapan dalam rekayasa perangkat lunak.

Dengan *UML*, metode *Booch*, *Object Modeling Technique* dan *Object Oriented Software Engineering* digabungkan dengan membuang elemen-elemen yang tidak praktis ditambah dengan elemen-elemen dari metode lain yang lebih efektif dan elemen-elemen baru yang belum ada pada metode terdahulu sehingga *UML* lebih ekspresif dan seragam dari pada metode lainnya.

2.3.1 Diagram-Diagram *UML*

Didalam *UML* terdapat beberapa macam Diagram yang dapat menggambarkan suatu sistem, berikut adalah beberapa Diagram yang terdapat di dalam *UML*.

1. *Use Case* dan *Use Case* Diagram
    a. *Use Case* adalah deskripsi fungsi dari sebuah sistem dari perspektif pengguna. *Use Case* bekerja dengan cara mendeskripsikan tipikal antara pengguna sebuah sistem dengan sistemnya sendiri melalui sebuah cerita bagaimana sebuah sistem dipakai. Urutan langkah-langkah yang menerangkan antara pengguna dan sistem disebut scenario. Setiap scenario mendeskripsikan urutan kejadian. Setiap urutan diinisialisasi oleh orang, sistem yang lain, perangkat keras atau urutan waktu. Dengan demikian

secara singkat bisa dikatakan, *Use Case* adalah serangkaian scenario yang digabungkan bersama-sama oleh tujuan umum pengguna.

b. *Use Case* Diagram adalah penggambaran interaksi pengguna sistem (*Actor*) dengan kasus (*Use Case*) yang telah disesuaikan dengan langkahlangkah (*scenario*). Diagram *Use Case* menunjukan 3 aspek dari system yaitu *Actor, Use Case* dan sistem, sub sistem *boundary*. *Actor* mewakili perang orang, sistem yang lain atau alat ketika berkomunikasi dengan *Use Case*.

2. *Class* Diagram

Class dalam notasi *UML* digambarkan dengan kotak. *Class* adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah obyek dan merupakan inti dari pengembangan dan desain berorientasi obyek. *Class* menggambarkan keadaan (atribut atau properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda atau fungsi). *Class* mempunyai beberapa bagian yang menjelaskan isi dari *Class*:

a. *Attribute* adalah property dari sebuah *Class*. *Attribute* ini melukiskan batas nilai yang mungkin ada pada obyek *Class*. Sebuah *Class* mungkin mempunyai nol atau lebih *Attribute*.

b. *Operation* adalah sesuatu yang bisa dilakukan oleh sebuah *Class* atau *Class* yang lain dapat lakukan untuk sebuah *Class*.

c. *Responsibility* adalah keterangan tentang apa yang akan dilakukan *Class* yaitu apa yang akan dicapai oleh *Attribute* dan *Operation*.

3. *Statechart* Diagram

Menelusuri individu - individu obyek melalui keseluruhan daur hidupnya, menspesifikasikan semua urutan yang mungkin dari pesan-pesan yang akan diterima obyek tersebut, bersama-sama dengan tanggapan atas pesan-pesan tersebut. *Statechart* Diagram menggambarkan transisi dan perubahan keadaan (dari satu state ke *state* lainnya) suatu obyek pada sistem sebagai akibat dari stimulasi yang diterima. Pada umunya *Statechart* Diagram menggambarkan *Class* tertentu (satu *Class* dapat memiliki lebih dari satu *Statechart* Diagram). Simbol *UML* untuk *State* Diagram adalah segiempat yang tiap pojoknya dibuat

rounded. Titik awalnya menggunakan lingkaran solid yang diarsir dan diakhiri dengan mata.

4. *Activity* Diagram

   Activity Diagram adalah teknik untuk mediskripsikan logika *procedural*, proses bisnis dan aliran kerja dalam banyak kasus. *Activity* Diagram mempunyai peran seperti halnya *Flowchart*, akan tetapi perbedaannya dengan *Flowchart* adalah *Activity* Diagram bisa mendukung perilaku parallel sedangkan *Flowchart* tidak bisa. *Activity* Diagram tidak menunjukan apa yang terjadi, tetap tidak menunjukan siapa yang melakukan apa. *Activity* Diagram menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *Decision* yang mungkin terjadi dan bagaimana mereka berakhir, *Activity* Diagram juga dapat menggambarkan proses paralel yang mungking terjadi pada beberapa eksekusi. Simbol-simbol yang sering digunakan pada pembuatan *Activity* Diagram.

5. *Sequence* Diagram

   Sequence Diagram digunakan untuk menggambarkan perilaku pada sebuah *scenario*. Diagram ini menunjukan sejumlah contoh obyek dan *Message* pesan yang diletakan diantara obyek-obyek ini dalam *Use Case*. *Sequence* Diagram menggambarkan interaksi antar obyek di dalam dan di sekitar sistem berupa *Message* yang digambarkan terhadap waktu. *Sequence* Diagram terdiri atas dimensi *vertical* (waktu) dan dimensi *horizontal* (obyek-obyek yang terkait). *Sequence* Diagram biasa digunakan untuk menggambarkan *scenario* atau rangkaian langkah-langkah yang dilakukan sebagai respons dari sebuah event untuk menghasilkan *output* tertentu. Diawali dari apa yang men-*trigger* aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internal dan *output* apa yang dihasilkan. Komponen utama *Sequence* Diagram terdiri atas obyek yang dituliskan dengan kotak segiempat bernama. *Message* diwakili oleh garis dengan tanda panah dan waktu yang ditunjukan dengan progress *vertical*. Penjelasan dari komponen utama *Sequence* Diagram yaitu :

   a. Obyek atau *Participant*

Obyek atau *Participant* diletakan di dekat bagian atas Diagram dengan urutan dari kiri ke kanan. Mereka diatur dalam urutan guna menyederhanakan Diagram. Setiap *Participant* terhubung dengan garis titiktitik yang disebut *Life-line*. Sepanjang *Life-line* ada kotak yang disebut *Activation*. *Activation* mewakili sebuah eksekusi operation dari participiant. Panjang kotak ini berbanding lurus dengan durasi *Activation*.

b. Message

Sebuah *Message* bergerak dari satu *Participant* ke *Participant* yang lain dan dari satu *Life-line* ke *Life-line* yang lain. Sebuah participant bisa mengirim sebuah *Message* kepada dirinya sendiri.

c. Time

Time adalah Diagram yang mewakili waktu pada arah *vertical*. Waktu dimulai dari atas kebawah. *Message* yang lebih dekat dari atas akan dijalankan terlebih dahulu dibandingkan *Message* yang lebih dekat ke bawah.

6. Collaboration Diagram

Collaboration Diagram adalah perluasan dari obyek Diagram. Obyek Diagram menunjukan obyek-obyek dan hubungannya satu dengan yang lain. *Collaboration* Diagram menunjukan *Message* - *Message* obyek yang dikirimkan satu sama lain. *Collaboration* Diagram juga menggambarkan interaksi antar obyek seperti *Sequence* Diagram, tetapi lebih menekankan pada peran masing-masing obyek dan bukan pada waktu penyampaian *Message*. Setiap *Message* memiliki *Sequence number*, di mana *Message* dari level tertinggi memiliki nomor satu *Message* dari level yang sama memiliki *prefix* yang sama. Dengan *Collaboration* Diagram memungkinkan untuk memodelkan pengiriman sebuah *Message* ke banyak obyek pada *Class* yang sama. Demikian juga halnya untuk menunjukan adanya obyek aktif yang mengendalikan aliran *Message*.

7. Component Diagram

Component Diagram merepresentasikan dunia nyata item yaitu *Component Software*. *Component* Diagram menggambarkan struktur dan hubungan antar

komponen piranti lunak, termasuk ketergantungan (*dependency*) diantaranya. *Component Software* adalah modul berisi *code*, baik berisi *source code* maupun *binary code*, baik *library* maupun *executable*, baik yang muncul pada *compile time*, *link time*, maupun *run time*. Umumnya komponen terbentuk dari beberapa *Class* dan atau *package*, tapi dapat juga dari komponen-komponen yang lebih kecil. Komponen dapat juga berupa I*nterface*, yaitu kumpulan layanan yang disediakan sebuah komponen untuk komponen lain. *Component* Diagram mengandung *Component*, *Interface* dan *Relationship*.

8. *Deployment* Diagram

Deployment Diagram menunjukan tata letak sebuah sistem secara fisik, menampakan bagian-bagian *software* yang berjalan pada bagian-bagian *hardware*. *Deployment* Diagram menyediakan gambaran bagaimana sistem secara fisik akan terlihat. Sistem terdiri dari node-node di mana setiap node diwakili untuk sebuah kubus. Garis yang menghubungkan antara dua kubus menunjukan hubungan diantara kedia node tersebut. Tipe node bisa berupa device yang berwujud *hardware* dan bisa juga *processor* (yang mengeksekusi *component*) atau *execution environment* (*software* yang menjadi *host* atau mengandung *software* yang lain).

2.4 Teknologi *Java*

Java adalah bahasa pemrograman yang disusun oleh *James Gosling* yang dibantu oleh rekan-rekannya seperti *Patrick Naughton*, *Chris Warth*, *Ed Rank*, dan *Mike Sheridan* di suatu perusahaan perangkat lunak yang bernama *Sun Microsystems* pada tahun 1991. Bahasa pemrograman ini mula-mula diinisialisasi dengan nama "*Oak*", namun pada tahun 1995 diganti namanya menjadi "*Java*".

2.4.1 Sejarah *Java*

Pada awal rilisnya, versi lama *Java* masih dikenal dengan sebutan JDK (*Java Development Kit*). Semua kebutuhan untuk pengembangan dan eksekusi program dalam *JDK* masih tergabung menjadi satu. Penamaan ini berlaku sampai *Java* 1.1. Setelah *Java* 1.2 rilis, *Sun Microsystems* menamainya dengan *JSDK* (*Java Software Development Kit*). Pada *JSDK*, kebutuhan untuk pengembangan program

dipisahkan dengan kebutuhan eksekusi program. Bagian *software* yang digunakan untuk kebutuhan eksekusi program disebut dengan *JRE* (*Java-Runtime Environment*). Pada perkembangan selanjutnya *Sun Microsystems* memperkenalkan *Java* versi 1.2 atau lebih dikenal dengan nama *Java* 2 yang terdiri atas *JDK* dan *JRE* versi 1.2. Aplikasi-aplikasi *Java* yang kompatibel dengan *Java* 2 ini dikenal dengan *Java* 2 *Compliant*. Pada *Java* 2 ini, *Java* dibagi menjadi tiga kategori, yaitu:

1. *Java* 2 *Standard Edition* (*J2SE*)

   Kategori ini digunakan untuk menjalankan dan mengembangkan aplikasiaplikasi *Java* pada level *PC* (*Personal Computer*).
2. *Java* 2 *Enterprise Edition* (*J2EE*)

   Kategori ini digunakan untuk menjalankan dan mengembangkan aplikasiaplikasi *Java* pada lingkungan *enterprise*, dengan menambah fungsionalitas fungsionalitas *Java* semacam *EJB* (*Enterprise Java Bean*), *Java COBRA*, *Servlet* dan *JSP*, serta *Java XML* (*Extensible Markup Language*)
3. *Java* 3 *Micro Edition* (*J2ME*)

   Kategori ini digunakan untuk menjalankan dan mengembangkan aplikasi-aplikasi *Java* pada *handled devices* atau perangkat-perangkat semacam handphone, *PDA*, dan *Pocket PC*.

2.4.2 Arsitektur *Java*

Secara arsitektur, *Java* tidak berubah sedikit pun semenjak awal mula bahasa tersebut dirilis. Kompiler *Java* (yang disebut dengan *Javac* atau *Java Compiler*) akan mentransformasikan kode-kode dalam bahasa *Java* ke dalam suatu *bytecode*. *Bytecode* adalah sekumpulan perintah hasil kompilasi yang kemudian dapat dieksekusi melalui sebuah mesin komputer abstrak, yang disebut dengan *JVM* (*Java Virtual Machine*). *JVM* juga sering dinamakan sebagai *interpreter*, karena sifatnya yang selalu penerjemahkan kode-kode yang tersimpan dalam *bytecode* dengan cara baris demi baris.

2.4.3 Kelebihan *Java*

Beberapa kelebihan *Java* adalah :

1. *Multiplatform*

   Kelebihan utama dari *Jav*a ialah dapat dijalankan di beberapa *platform*/sistem operasi komputer, sesuai dengan prinsip tulis sekali, jalankan di mana saja. Dengan kelebihan ini pemrogram cukup menulis sebuah program *Java* dan dikompilasi (diubah, dari bahasa yang dimengerti manusia menjadi bahasa mesin/*bytecode*) sekali lalu hasilnya dapat dijalankan di atas beberapa platform tanpa perubahan. Kelebihan ini memungkinkan sebuah program berbasis java dikerjakan diatas operating sistem linux tetapi dijalankan dengan baik di atas *Microsoft Windows*. *Platform* yang didukung sampai saat ini adalah *Microsoft Windows, linux, Mrac OS* dan *Sun Solaris*. Penyebabnya adalah setiap sistem operasi menggunakan programnya sendiri-sendiri untuk menginterpretasikan *bytecode* tersebut.

2. *OOP* (*Object Oriented Programming*)

   Yang artinya semua aspek yang terdapat di *Java* adalah objek. *Java* merupakan salah satu bahasan pemrograman berbasis objek secara murni. Semua tipe data diturunkan dari kelas dasar yang disebut Object. Hal ini sangat memudahkan pemrogram untuk mendesain, membuat, mengembangkan, dan mengalokasi kesalahan sebuah program dengan basis java secara cepat, tepat, mudah dan terorganisir. Kelebihan ini menjadikan *Java* sebagai salah satu bahasa pemrograman termudah, bahkan untuk fungsi yang *advance* seperti komunikasi antara komputer sekalipun.

3. *Library* yang lengkap

   Di mana *Java* terkenal dengan kelengkapan *library*/perpustakaan (kumpulan program-program yang disertakan dalam pemrogram *Java*) yang sangat memudahkan dalam penggunaan oleh para pemrogram untuk membangun aplikasinya. Kelengkapan perpustakaan ini ditambah dengan keberadaan kom*unit*as *Java* yang besar yang terus menerus membuat perpustakaan-perpustakaan baru untuk melingkupi seluruh kebutuhan pembangunan aplikasi.

4. Bergaya *C++*

Memiliki sintaks seperti bahasa pemrograman *C++* sehingga menarik banyak pemrogram *C++* untuk pindah ke *Java*. Saat ini pengguna java sangat banyak, sebagian besar adalah pemrogram *C++* yang pindah ke *Java*.

5. Pengumpulan sampah otomatis

   Memiliki fasilitas pengaturan penggunaan memori sehingga para pemrogram tidak perlu melakukan pengaturan memori secara langsung (seperti halnya dalam bahasa *C++* yang dipakai secara luas).

2.4.4 Kekurangan *Java*

Beberapa kekurangan *Java* antara lain :

1. Mudah didekompilasi.

   Dekompilasi adalah proses membalikkan dari kode jadi menjadi kode sumber. Ini memungkinkan karena kode jadi *Java* merupakan *bytecode* yang menyimpan banyak atribut bahasa tingkat tinggi seperti namanama kelas, metode dan tipe data. Hal yang sama juga terjadi pada *Microsoft.Net Platform.* Dengan demikian, Algoritma yang digunakan program akan lebih sulit disembunyikan dan mudah dibajak / *di-reverse-engineer*.

2. Penggunaan memori yang banyak.

   Penggunaan memori untuk program berbasis *Java* jauh lebih besar daripada bahasa tingkat tinggi generasi sebelumnyaseperti *C/C++* dan *Pascal* (lebih spesifik lagi *delphi* dan *Object Pascal*). Biasanya ini bukan merupakan masalah bagi pihak yang menggunakan teknologi terbaru, tetapi menjadi masalah bagi mereka yang masih harus berkutat dengan mesin komputer berumur lebih dari 4 tahun.

2.5 Pengujian *Black Box*

Pengujian *Black Box* merupakan pendekatan komplementer dari teknik W*hite Box*, karena pengujian *Black Box* diharapkan mampu mengungkap kelas kesalahan yang lebih luas dibandingkan teknik *White Box*. Pengujian *Black Box* berfokus pada pengujian persyaratan fungsional perangkat lunak, untuk mendapatkan serangkaian kondisi *input* yang sesuai dengan persyaratan fungsional

suatu program. Pengujian *Black Box* adalah pengujian aspek fundamental sistem tanpa memperhatikan struktur logika internal perangkat lunak. Metode ini digunakan untuk mengetahui apakah perangkat lunak berfungsi dengan benar. Pengujian *Black Box* merupakan metode perancangan data uji yang didasarkan pada spesifikasi perangkat lunak. Data uji dibangkitkan, dieksekusi pada perangkat lunak dan kemudian keluaran dari perangkat lunak dicek apakah telah sesuai dengan yang diharapkan. Pengujian *Black Box* berusaha menemukan kesalahan dalam kategori :

1. Fungsi-fungsi yang tidak benar atau hilang
2. Kesalahan interface
3. Kesalahan dalam struktur data atau akses database eksternal
4. Kesalahan kinerja
5. Inisialisasi dan kesalahan terminasi.

2.5.1 Cross Validation

Cross Validation merupakan salah satu teknik untuk menilai / memvalidasi keakuratan sebuah model yang dibangun berdasarkan *Data Set* tertentu. Pembuatan model biasanya bertujuan untuk melakukan prediksi maupun klasifikasi terhadap suatu data baru yang boleh jadi belum pernah muncul di dalam *Data Set*. Data yang digunakan dalam proses pembangunan model disebut data latih/*Training*, sedangkan data yang akan digunakan untuk memvalidasi model disebut sebagai data *test*.

K-fold Cross Validation adalah sebuah teknik intensif komputer yang menggunakan keseluruhan data yang ada sebagai traning set dan *test* set. Seluruh data secara acak dibagi menjadi K buah subset dengan ukurang yang sama di mana merupakan himpunan bagian dari (1..n) sedemikian sehingga dan = ∅ setelah itu dilakukan iterasi sebanyak k kali. Pada iterasi ke k subset bk menjadi *test test*, sedangkan subset yang lain menjadi *Training* set. Setelah itu dihitung nilai rata-rata *error* dengan menggunakan hasil dari k buah iterasi.

Kelebihan dari metode ini adalah tidak adanya masalah dalam pembagian data. Setiap data akan menajdi *test* set sebanyak satu kali dan akan menjadi *Training* set sebanyak K-1 kali. Kekurangan dari metode ini adalah Algoritma pembelajaran

harus dilakukan sebanyak K kali yang berarti menggunakan K kali waktu komputasi.