BAB 2

LANDASAN TEORI

2.1 Sejarah *Artificial Intelligence*

Pada musim panas tahun 1956, sebuah konferensi kampus Perguruan tinggi Dartmouth, bidang riset *Artificial Intelligence* dilahirkan. Di tahun tersebut, mereka yang menghadiri konferensi, akan menjadi para pemimpin riset *Artificial Intelligence* untuk beberapa dekade, terutama John Mccarthy, Marvin Minsky, Allen Newell dan Herbert Simon. Mereka dan para siswa menulis program yang mengejutkan banyak orang. Karena didalam program itu, komputer sedang memecahkan permasalahan kata di dalam aljabar, pembuktian teori logika dan berbicara bahasa inggris. Pada pertengahan tahun 60an riset mereka dibiayai oleh DARPA dan mereka optimis tentang masa depan dari bidang baru tersebut. Beberapa hal yang diutarakan antara lain :

- "*Selama 20 tahun mendatang mesin akan mampu membuat beberapa pekerjaan yang dapat dilakukan oleh seorang manusia*". (H.A.Simon, 1965)
- "*Didalam suatu generasi, permasalahan dalam menciptakan 'Artificial Intelligence' pada akhirnya akan dipecahkan*". (Marvin Minsky, 1967)

Banyak prediksi mereka yang tidak dapat tercapai. Mereka telah gagal untuk mengenali beberapa kesulitan dari permasalahan yang mereka hadapi. Di tahun 1974, sebagai jawaban atas kritikan dari inggris (Tn.James Lighthill) dan tekanan berkelanjutan dari kongres untuk membiayai proyek yang lebih produktif, DARPA memotong semua biaya yang tidak produktif untuk riset penyelidikan didalam *Artificial*

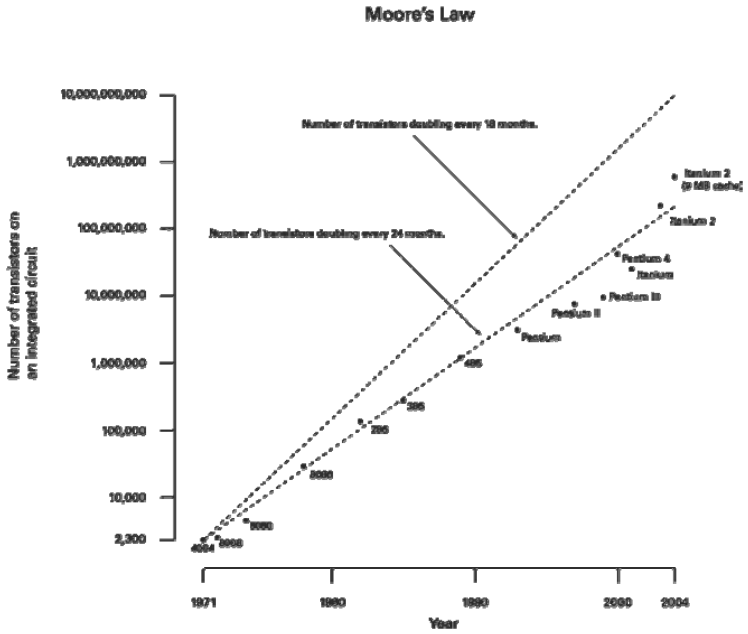
Intelligence. Ini adalah Penyelidikan *Artificial Intelligence* yang pertama kalinya pada Musim dingin.

Pada awal 80an, bidang ini dihidupkan kembali oleh komersial sukses dari sistem pakar dan Pada tahun 1985, pasar untuk *Artificial Intelligence* telah mencapai lebih dari milyaran dolar. *Minsky* dan orang yang lain memperingatkan kepada masyarakat yang minat akan *Artificial Intelligence,* mempunyai spiral yang tak bisa dikuasai dan kekecewaan itu pasti untuk diikuti. *Minsky* Benar, mulai bersama dengan robohnya market *Mesin Lips* di tahun 1987, *Artificial Intelligence* sekali lagi jatuh masuk kedalam tahun keburukan.

Di tahun 90an dan awal abad 21, *Artificial Intelligence* mencapai sukses terbesar nya, sekalipun hanya di belakang layar. *Artificial Intelligence* diadopsi diseluruh industri yang berteknologi, menyediakan pengangkatan yang berat untuk logistik, data mining, hasil diagnosa medis dan berbagai macam area.

Beberapa factor yang mempengaruhi kesuksesannya di tahun 90an yaitu :

- kuasa komputer yang luar biasa dari komputer saat ini



Gambar 2.1 Moore's Law

Gambar diatas menunjukan trend hardware yang terjadi pada tahun 90an.

- suatu penekanan lebih besar pada pemecahan subprogram yang spesifik.
- ciptaan dari spesies baru antara *Artificial Intelligence* dan beberapa bidang lain yang memecahkan masalah yang sama.
- dan di atas semuanya suatu komitmen baru dari peneliti ke metode mathematical dan standard rigorous ilmiah kaku.

Sebegitu banyak masalah yang dapat di pecahkan dengan menggunakan *Artificial Intelligence* ini sehingga banyak orang yang mengganggap bahwa *Artificial Intelligence* merupakan teknologi yang dapat membantu memecahkan masalah yang rumit dan tidak semua orang dapat mengerjakan masalah tersebut.

Beberapa pengertian *Artificial Intelligence* antara lain :

"*salah satu bagian ilmu komputer yang membuat agar komputer dapat melakukan pekerjaan seperti dan sebaik yang dilakukan oleh manusia*"**.** (Kusumadewi , 2003)

"sebuah pembelajaran dimana komputer belajara untuk mengerjakan sesuatu yang dapat dikerjakan manusia". (Elaine Rich, 1991).

2.2 Neural Network

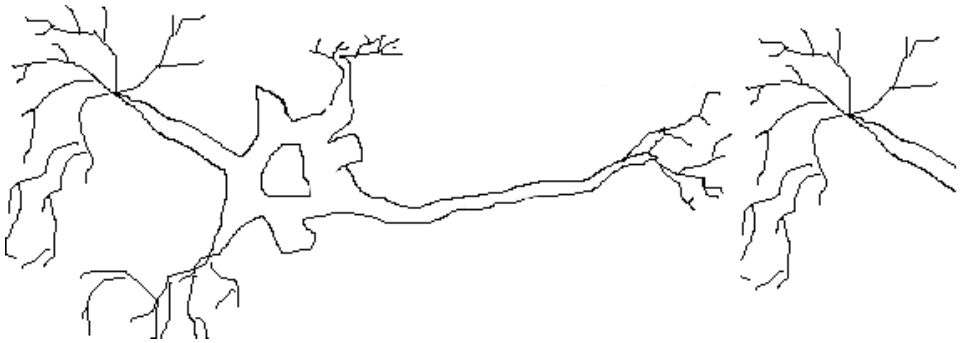
Dengan teknologi yang semakin berkembang dan maju secara pesat, para ilmuwan terus memikirkan untuk membuat suatu sistem yang biasanya dikerjakan oleh manusia dapat digantikan oleh komputer. Dengan begitu, terbentuklah sistem tersebut.

2.2.1. Jaringan Syaraf Biologi

Otak manusia mempunyai sifat yang sangat unik, sehingga terkadang ada masalah yang dapat dipecahkan oleh otak manusia, sedangkan tidak dapat dipecahkan oleh komputer. Otak manusia terdiri atas jaringan sistem saraf yang sangat kompleks.

Neural berasal dari kata neuron, yaitu sel-sel penyusun sistem saraf di dalam otak manusia yang memiliki fungsi utama mengumpulkan, memproses, dan menyebarkan sinyal-sinyal elektris. Kapasitas pemrosesan informasi otak muncul terutama dari jaringan-jaringan neuron.

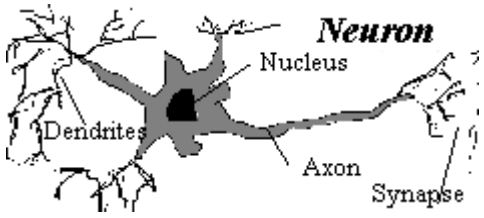
Biological neuron memiliki tiga komponen utama, yaitu dendrit, soma, dan axon. Dendrit bertugas menerima sinyal dari neuron-neuron yang lain. Sinyal merupakan rangsangan elektris yang ditransmisikan melalui celah sinapsis (*synaptic gap*) yang disebabkan oleh proses kimiawi tubuh. Proses transmisi secara kimiawi ini mengubah sinyal yang masuk atau sinyal input (dengan mengukur besarnya frekuensi dari sinyal yang diterima). Cara inilah yang diadopsi oleh proses perubahan bobot dalam *Artificial Neural Network*.



Gambar 2.2 Neuron

Gambar ini menunjukkan bahwa Neuron menembakkan suatu sinyal ketengah lewat suatu isyarat kepada yang berikutnya di dalam rantai itu.

Berikut ini merupakan gambar structure dari neural network itu sendiri.



Gambar 2.3 Struktur Neural Network

Pada tahun 1854 dimana George Boole menyusun suatu dalil mengenai hukum pemikiran yang bersifat dasar tentang benar atau salahnya nomor *binary.* Prinsip Boole's menyusun apa yang dikenal sebagai aljabar Boolean, koleksi logika mengenai *AND, OR, NOT operands*. Untuk lebih jelasnya kita dapat melihat contoh dibawah ini yang berdasarkan peraturan dari statemen :

sebagai contoh ini mempertimbangkan semua buah apel yang merah

- Buah apel merah – Benar
- Buah apel adalah merah DAN jeruk adalah ungu – Salah
- Buah apel adalah merah ATAU jeruk adalah warna ungu – Benar
- Buah apel adalah merah DAN jeruk adalah tidak ungu - juga Benar

Menurut hukum ini, Boole juga mengira bahwa pekerjaan yang dilakukan oleh pemikiran manusia melaksanakan operasi logis yang bisa diberi alasan. Sembilan puluh tahun kemudian, Claude Shannon menerapkan prinsip Boole'S didalam sirkuit, dan *blueprint* untuk komputer elektronik.

Artificial Neural Network disini merupakan distribusi *processor parallel* besar-besaran yang dibuat dari unit simple *processing*, dimana kecenderungan alami untuk menyimpan pengetahuan, pengalaman dan membuatnya dapat digunakan kembali. Hal ini mendekati otak manusia dalam 2 aspek :

1. Pengetahuan diperlukan oleh jaringan dari lingkungannya melalui proses belajar.
2. Kekuatan koneksi antar neuron, dikenal dengan berat synaptic, yang digunakan untuk menyimpan pengetahuan yang didapatkan.

2.2.2. *Artificial Neural Network*

Bekerja dalam *Artificial Neural Network*, biasanya diarahkan pada *"Neural Network*", dimotivasi pada awalnya dari pengenalan bahwa otak manusia menghitung dalam cara-cara yang berbeda dari pada *digital computer conventional*. Otak sesungguhnya kompleks, nonlinear, dan parallel computer(sistem proses informasi). Otak juga memiliki kemampuan untuk menyusun bentuknya, yang dikenal dengan neuron-neuron, jadi untuk menampilkan perhitungan yang tepat (contohnya pengenalan pola, penerapan, dan kendali motor) lebih cepat berkali-kali dari pada *computer digital* tercepat yang ada saat ini. Contohnya visi manusia, dimana adalah bertugas memproses informasi (*Marr*, 1982 dan *Levine*, 1985 dan *Churchland and Sejnowski*, 1992). Ini berfungsi sebagai sistem visual untuk menyediakan representasi dari lingkungan sekitar kita dan lebih penting lagi, untuk mensuplai informasi yang kita butuhkan untuk interaksi dengan lingkungan. Untuk lebih spesifik, otak secara rutin menyelesaikan tugas-tugas pengenalan perceptual (contohnya mengenali wajah yang familiar dalam pemandangan yang tidak familiar) dalam rata-rata 100-200 ms, dimana tugas-tugas kurang kompleks dapat dikerjakan berhari-hari oleh *computer conventional*.

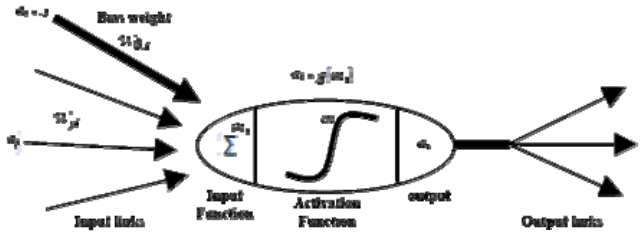
Neural network terdiri atas node atau unit yang terhubung dengan link yang tearah. Sebuah link dari unit $_j$ ke unit $_i$ melayani untuk menyebarkan pengaktifan node a_j dari unit $_j$ ke unit $_i$. Masing – masing link mempunyai sebuah berat numerik $W_{j,i}$ yang dihubungkan dengan node tersebut dan yang menentukan kekuatan node tersebut. Masing – masing unit I, pertama–tama menghitung suatu penjumlahan berat dari *inputan*.

$$in = \sum_{j=0}^{n} W_{j,i} a_j$$

kemudian menggunakan suatu fungsi pengaktifan g ke penjumlahan ini untuk memperoleh *output*.

$$a_i = g(in_i) = g(\sum_{j=0}^{n} W_{j,i} a_j)$$

Dibawah ini akan diperlihatkan gambar fungsi aktifasi yang terdiri dari *input* dan *output link*, *bias weigth*, *input* dan *output* serta *activation function*.



Gambar 2.4 Fungsi Aktifasi

Penjelasan gambar :

Sebuah model matematika untuk neuron. Unit output aktifasi adalah $a_i = g(\sum_{j=0}^{n} W_{j,i} a_j)$,dimana a_j adalah aktifasi output dari unit j dan $W_{j,i}$ adalah Berat didalam sebuah link dari unit j ke unit ini.

2.2.3. Sejarah *Neural Network*

Neural Network pertama kali diperkenalkan oleh McCulloch dan Pitts di tahun 1943. McCulloch dan Pitts menyatakan bahwa sebuah neuron dapat di gabungkan menjadi sebuah jaringan untuk menghasilkan output yang dapat di wakili sebagai sebuah kombinasi dari fungsi logik.

Setelah *Neural Network* diperkenalkan, kemudian mulailah terbentuk aturan-aturan pembelajaran yang pertama kalinya untuk *Neural Network* yang di temukan oleh Donald Hebb pada tahun 1949.

Dengan landasan pemikiran yaitu jika dua neurons adalah aktif secara serempak, kemudian kekuatan koneksi antara (b/w) bobot / weight nya harus ditingkatkan.

2.2.4. Keuntungan dan Kerugian dari *Neural Network*

2.2.4.1. Keuntungan dari *Neural Network*

Dapat dilihat bahwa *Neural Network* mengembangkan kekuatan menghitungnya melalui :

- secara besar-besaran struktur parallel distribusi.
- kemampuan untuk belajar dan lalu digeneralisasi.

Generalisasi mengacuh pada *Neural Network* yang menghasilkan output yang masuk akal untuk input yang tidak dapat ditemukan selama latihan (belajar). Kedua kemampuan pemrosesan informasi ini memungkinkan *Neural Network* untuk menyelesaikan masalah yang kompleks (dalam skala besar) yang sesungguhnya tidak dapat diselesaikan dengan perhitungan algoritma biasa. Dalam latihan, bagaimanapun juga *Neural Network* tidak dapat menyediakan solusi dengan bekerja secara individu.

Penggunaan *Neural Network* menawarkan pengguna kemampuan yang berguna yaitu

1. *NonLinearity.*

   Nonlinerarity merupakan jaringan saraf buatan nonlinear. *Neural Network* membuat koneksi dalam dari saraf nonlinear. Terlebih lagi, nonlinear adalah jenis khusus dalam merasakan yang didistribusikan melalui jaringan. Nonlinear adalah bagian yang penting, terutama jika mekanisme fisik bertanggung jawab atas bagian sinyal input (contoh: sinyal suara) diturunkan menjadi nonlinear.

2. *Input-OutPut Mapping.*

   Paradikma yang populer dari belajar disebut belajar dengan guru(*supervised learning*) atau belajar yang diawasi termasuk modifikasi dari berat synaptic dari *Neural Network* dengan menerapkan sejumlah label contoh-contoh latihan dan contoh-contoh tugas. Tiap contoh terdiri dari sinyal input yang unik dan koresponden respon yang diinginkan. Jaringan dipresentasikan dengan contoh-contoh yang diambil secara acak dari sejumlah data dan berat *synaptic* (parameter bebas) dari jaringan yang dimodifikasi untuk meminimalisasi perbedaaan antara respon yang diinginkan dan respon yang aktual dari jaringan yang dihasilkan oleh sinyal input menurut dengan kriteria yang pantas dengan statistik. Latihan dari jaringan, diulang untuk banyak contoh dalam sejumlah data sampai jaringan mencapai titik yang tetap dimana tidak ada lagi mengalami perubahan yang signifikan dalam berat *synaptic*. Lalu jaringan belajar dari contoh-contoh dengan kontruksi input-output mapping untuk masalah di tangan.

3. *Adaptivity.*

Neural Network memiliki dasar kemampuan untuk mengadaptasi berat *synaptic* untuk berubah dalam lingkungannya. Biasanya, *Neural Network* dilatih untuk beroperasi dalam lingkungan yang spesifik, dapat dengan mudah dilatih lagi untuk menghadapi perubahan kecil saat kondisi lingkungan telah beroperasi. Terlebih lagi, pada saat beroperasi dalam lingkungan yang tidak statik, kerja *Neural Network* dapat didesain untuk mengubah berat *synaptic*nya dalam waktu nyata. Arsitektur yang alami dari *Neural Network* untuk pengenalan pola, sinyal prosessing, dan kendali aplikasi, digabungkan dengan kemampuan adaptasi dari jaringan, membuatnya menjadi alat yang berguna dalam adaptasi pengenalan pola, adaptasi sinyal prosessing, dan adaptasi kendali. Menurut aturan umum, dapat dikatakan bahwa semakin adaptasi kita membuat suatu sistem, sepanjang waktu dapat menjamin bahwa sistem tetap stabil, semakin tetap penampilannya akan lebih baik saat suatu sistem perlu dioperasikan pada lingkungan yang tidak statik.

4. *Evidental Response.*

Dalam konteks pengenalan pola, *Neural Network* dapat didesain untuk menyediakan informasi tidak hanya sebatas pola biasa yang dipilih, tetapi juga kepercayaan dalam pembuatan keputusan. Informasi terkini ini dapat digunakan untuk menolak pola yang ambigu, dan meningkatkan penampilan pengenalan dari jaringan.

5. *Contextual information.*

Pengetahuan direpresentasikan oleh struktur dan titik aktivasi dari *Neural Network*. Setiap neuron dalam jaringan sangat berpotensial dipengaruhi oleh kegiatan global dari semua neuron yang lain dalam jaringan. Akibatnya, informasi yang berhubungan diatasi dengan alami oleh *Neural Network*.

6. *Fault Tolerance.*

Neural Network, diimplementasikan dalam bentuk hardware, memiliki potensial untuk menurunkan *fault tolerance*, atau mampu menghitung tetap.

7. *VLSI implementability.*

Sejumlah besar paralel alami dari *Neural Network* membuat *Neural Network* berpotensial cepat untuk menghitung tugas-tugas tertentu. Fungsi yang sama ini membuat *Neural Network* cukup baik untuk implementasi menggunakan teknologi *very-large-scale-integrated*(VLSI). Salah satu keuntungan kegunaan dari VLSI adalah menyediakan alat dari penangkap sifat yang benar-benar kompleks dalam bentuk hirarki.

8. *Uniformity of Analysis and Design*.

Dasarnya, *Neural Network* cocok secara universal untuk *processor* informasi. Fungsi ini membuktikan diri sendiri dalam berbagai cara:

- Neuron, dalam satu bentuk atau bentuk lainnya, memrepresentasikan sejumlah bagian-bagian yang sama ke seluruh *Neural Network*.
- Persamaan ini membuat mungkin untuk membagi teori dan algoritma belajar dalam aplikasi-aplikasi berbeda dari *Neural Network*.

- Jaringan modular dapat dibangun melalui *seamless integration of modules*.

9. *Neurobiological Analogy*.

Desain dari *Neural Network* dimotivasi oleh persamaan dengan otak, yang telah dibuktikan bahwa kesalahan toleransi prosessing paralel adalah tidak hanya kemungkinan secara fisik, tapi juga kecepatan dan kekuatan. Neurobiologist melihat artificial *Neural Network* sebagai alat riset untuk pengertian dari fenomena neurobiological. Disisi lain, para ahli menggunakan neurobiology untuk ide baru dalam menyelesaikan masalah yang lebih kompleks.

2.2.4.2. Kerugian dari *Neural Network*

Beberapa kelemahan *Neural Network* itu antara lain :

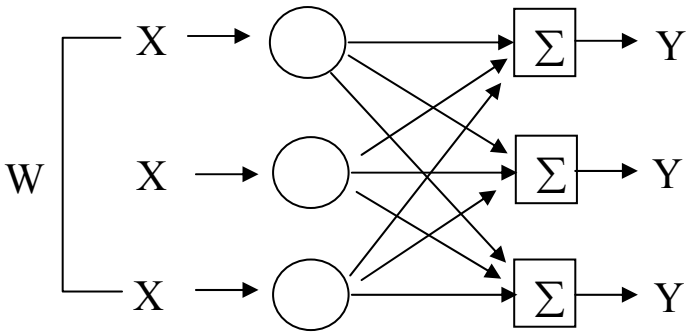
- Mengecilkan kelebihan - kelebihan yang memerlukan banyak perhitungan.
- Hubungan individu diantara variabel input dan variabel output tidak dibangun dari pertimbangan rancangan sehingga model cenderung.
- Menjadi kotak hitam atau input/output tanpa berbasis analitis.
- Ukuran contoh harus besar

2.2.5. Arsitektur *Neural Network*

Cara di mana neuron dalam suatu *Neural Network* terstruktur adalah dengan link bersama dengan algoritma pembelajaran digunakan untuk mentraining *network* nya. Secara umum kita boleh mengidentifikasi tiga kelas yang pada dasarnya berbeda tentang arsitektur jaringan :

1. *Single layer feedforward networks*

   Didalam sebuah layer *Neural Network*, neuron di atur didalam sebuah format layer. Didalam form yang sederhana ada layer network, kita mempunyai sebuah input layer dari node sumber itu dirancang ke sebuah output layer dari neuron, tapi bukan sebaliknya. Dengan kata lain, network ini dengan keras adalah feedforward atau acyclic tipe.

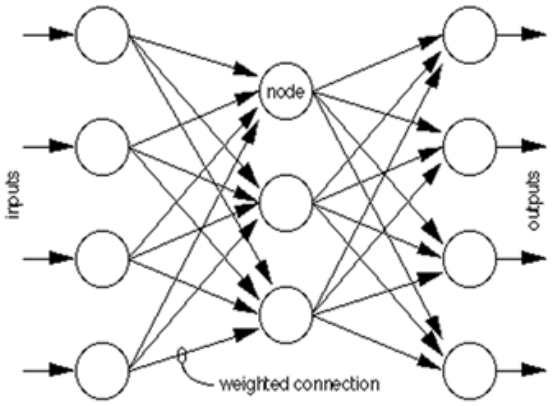


Gambar 2.5 Arsitektur Single layer *Network*

2. *Multi layer feedforward networks*

   Kelas kedua dari feedforward neural network mencirikan dirinya sendiri dengan kehadiran dari 1 atau lebih hidden layer, penghitungan node yang menghubungkan disebut hidden neuron atau hidden unit. Fungsi dari

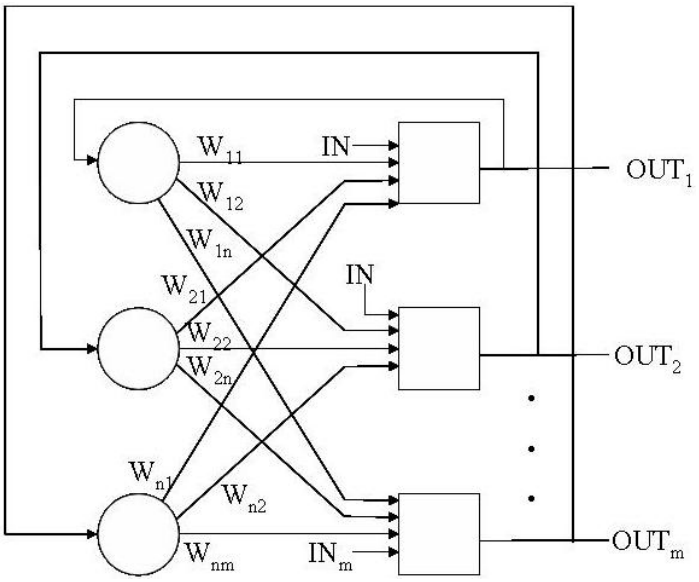
hidden neuron adalah menjadi batas diantara inputan *eksternal* dan *output network*.



Gambar 2.6 Arsitektur *Multi layer Network*

3. *Recurrent networks*

Sebuah *recurrent network* mencirikan dirinya sendiri dari sebuah feedforward neural network dalam suatu pengulangan yang sedikitnya cuma ada satu umpan balik. Dengan kata lain *network* ini dapat memberikan nilai output yang telah didapat nya ke nilai input dari semua neuron lainnya.

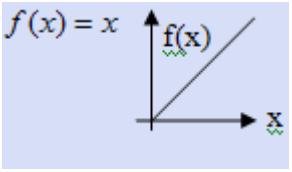


Gambar 2.7 Arsitektur *Recurrent Network*

2.2.6. Fungsi Aktifasi

1. Fungsi identitas

Fungsi ini umumnya digunakan pada neuron – neuron yang berada pada input unit.

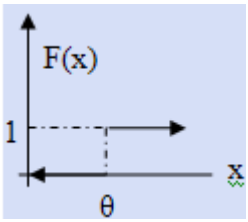


Gambar 2.8 Kurva Fungsi Identitas

2. Fungsi tangga biner

$$f(x) = \begin{cases} 1, jika\ x \geq \theta \\ 0, jika\ x < \theta \end{cases}$$

Dimana θ adalah suatu nilai threshold.



Gambar 2.9 Kurva Tangga Biner

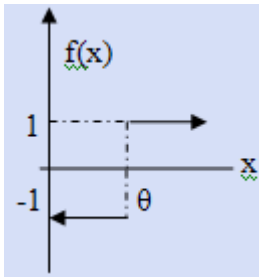
Fungsi ini sering digunakan pada single layer network . Biasanya digunakan untuk mengubah input yang merupakan variabel kontinyu, menjadi input bernilai bias (0 atau 1). Nilai threshold (θ) menjadi garis pemisah antara daerah dengan respon aktifasi positif dan daerah dengan respon aktifasi negatif.

3. Fungsi tangga bipolar

Fungsi tangga bipolar serupa dengan fungsi tangga biner, hanya saja daerah {-1,1}.

$$f(x) = \begin{cases} 1, jika\, x \geq \theta \\ -1, jika\, x < \theta \end{cases}$$

Dimana θ adalah suatu nilai threshold



Gambar 2.10 Kurva Tangga Bipolar

4. Fungsi signoid biner

fungsi ini mencakup fungsi – fungsi kurva S yang mana fungsi ini juga mempunyai sifat – sifat sinyal pada otak manusia.

Contoh – contoh yang sering digunakan adalah fungsi logistic, dimana fungsi ini memiliki kelebihan dalam melatih *Neural Network* dengan model backpropagation, karena memiliki hubungan sederhana antara nilai fungsi pada suatu titik dengan nilai turunannya, sehingga meminimumkan waktu selama pembelajaran.

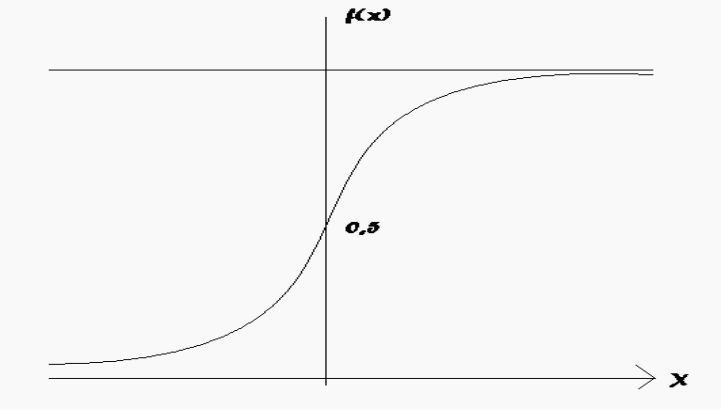
$$f(x) = \frac{1}{1 + e^{-\sigma x}}$$

dimana σ merupakan parameter persamaan yang diberikan, dan umumnya nilai ini dipilih σ =1. fungsi ini memiliki turunan pertama.

$$f(x)' = \sigma f(x)[1 - f(x)]$$

dimana daerah hasil adalah interval 0 sampai dengan 1. fungsi ini khususnya digunakan sebagai aktifasi untuk *Neural Network* dimana outputnya terletak pada interval 0-1.

Fungsi ini khususnya digunakan sebagai aktifasi untuk *Neural Network* dimana outputnya terletak pada interval 0-1



Gambar 2.11 Kurva Signoid Biner

5. Fungsi signoid bipolar

perhatikan posisi d, tentang fungsi sigmoid biner, yang bisa di skalakan sehingga memiliki daerah hasil pada sembarang interval sesuai dengan permasalahan yang di berikan yang paling umum adalah daerah hasil dari -1 sampai dengan 1. Fungsi yang merupakan hasil dari proses skala tersebut dinamakan fungsi signoid bipolar. Sesuai dengan daerah hasilnya, jika f(x) adalah fungsi signoid biner dimana $f(x) = \frac{1}{1+e^{-x}}$

maka g(x) adalah fungsi signoid bipolar sebagai berikut.

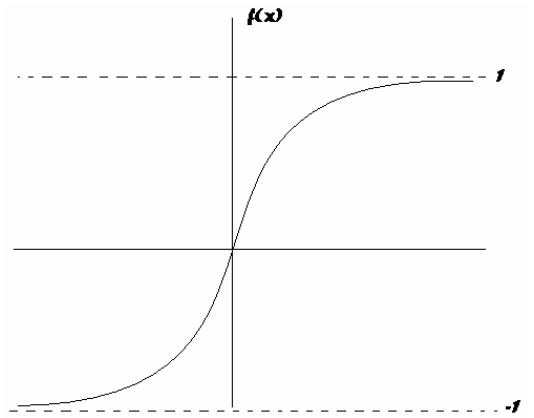
$$g(x) = 2f(x) - 1$$

$$= \frac{2}{1+e^{-x}} - 1$$

$$g(x) = \frac{1-e^{-x}}{1+e^{-x}}$$

turunan pertama dari g(x) adalah

$$g'(x) = \sigma/2\,[1 + g(x)][1 - g(x)]$$



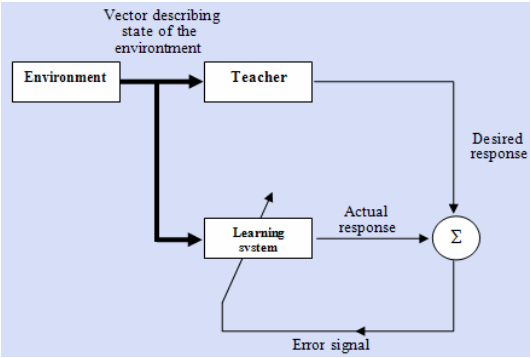
Gambar 2.12 Kurva Signoid Bipolar

2.2.7. Paradigma Pembelajaran dalam *Neural Network*

Neural network mempunyai 3 macam paradigma pembelajaran yaitu learning *supervised learning* (*learning with a teacher*)*, unsupervised leaning (learning without a teacher),* dan *Reinforcement Learning (learning with a critic*). Untuk pengertian dan penjelasan lebih lanjut akan dipaparkan dibawah ini.

2.2.7.1. *Supervised Learning*

Supervised learning (*learning with a teacher*) yaitu menggunakan seorang guru. Dalam istilah konsep, guru pasti mempunyai pengetahuan dari lingkungannya, bersama dengan pengetahuan yang diperkenalkan oleh sebuah set contoh dari masukan dan keluaran.



2.13 Gambar Blok Diagram *Supervised Learning*

Gambar blok diagram merupakan salah satu contoh supervised learning.

Supervised learning merupakan suatu cara learning yang menggunakan guru atau memerlukan pengetahuan yang di ajarkan terlebih dahulu.

Supervised learning mempunyai beberapa kriteria antara lain :

- Sebuah "Profesor" boleh menyediakan contoh dan mengajar *Neural Network* bagaimana cara memenuhi suatu tugas yang tertentu.
- Tanggapan yang diinginkan dari *Neural Network* didalam fungsi dari masukan yang tertentu sungguh baik dikenal.

2.2.7.2. *Unsupervised Learning*

Pada paradigma *unsupervised learning* (*learning without a teacher*), tidak ada guru yang mengawasi proses pembelajaran. Dengan kata lain, target output tidak diberikan. *Training set* hanya terdiri dari vektor-vektor input, tanpa pasangan output. Sejauh ini, paradigma ini dianggap sebagai model dalam konsep sistem biologis.

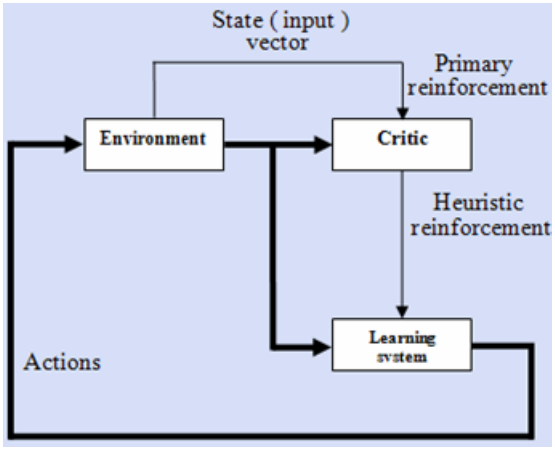
Pada *unsupervised learning* atau *self-organized learning*, proses pembelajaran dilakukan tanpa adanya *external supervisor* atau *external teacher* yang mengawasi

proses pembelajaran atau dengan kata lain tidak adanya target output untuk dibandingkan dengan vector output sebagai hasil komputasi dari vector input seperti pada supervised learning. Secara garis besar, *unsupervised learning* mencoba untuk mengambil fitur-fitur yang umum atau paling mewakili dari suatu input data. Jaringan akan mengubah bobot sehingga beberapa vektor input yang memiliki kesamaan fitur akan dikelompokan ke dalam output yang sama.

Keuntungan dari *unsupervised learning* adalah kemampuannya untuk mengelompokkan input data yang hilang atau mengandung *error* (*noise*) dengan baik. Sistem dapat menggunakan fitur yang telah diambil dan dipelajari dari training data, untuk membangun kembali pola input data dari input data yang rusak (*corrupted*).

2.2.7.3. *Reinforcement Learning*

Didalam Reinforcement Learning *(learning with a critic*)., agen diberikan evaluasi atas tindakan yang telah dilakukan, namun dengan tidak memberitahukan tindakan apa yang benar dan harus dilakukan. Agen akan diberikan penghargaan (*reward*) atau hukuman (*punishment*) atas tindakannya. Penghargaan dan hukuman inilah yang disebut dengan *reinforcement*.



Gambar 2.14 Gambar Blok Diagram *Reinforcement Learning*

Gambar blok diagram diatas merupakan salah satu bentuk dari sebuah sistem pembuatan *Reinforcement Learning* yang mempunyai sebuah kritikus yang mengubah sebuah primary *reinforcement* signal yang di dapat dari lingkungannya, kemudian menjadi signal *reinforcement* yang nenpunyai kualitas yang tinggi yang disebut *heuristic reinforcement signal.* Kemudian sistem didisain untuk belajar dibawah penundaan *reinforcement*. Didalam *reinforcement learning* mempunyai beberapa macam metode-metode, salah satu metodenya adalah Q- learning.

2.2.7.3.1 *Q-learning*

Watkins [*Watkins*, 1989] memperkenalkan metode dari Reinforcement Learning yang disebut *Q-learning*. *Q-learning* adalah suatu prosedur programming incremental yang dinamis yang menentukan kebijakan optimal didalam sebuah cara langkah per langkah.

Q-Learning mempunyai training information. Didalam *training information* yang tersedia untuk agen, hanya berupa serangkaian *immediate reward* $r(s_i, a_i)$ untuk i = 0, 1, 2, 3, dst. Berdasarkan informasi training (training *information*) ini, akan lebih mudah untuk mempelajari fungsi evaluasi numerik (*numerical evaluation function)*.

Evaluation function yang harus dipelajari oleh agen adalah V^*. Melalui *evaluation function* ini, agen dapat memilih *state-state* yang dapat memberikan *cumulative reward* yang lebih besar.

$$\pi^*(s) = \arg\max_a \left[r(s,a) + \gamma V^*(\delta(s,a)) \right]$$

di mana:

$\delta(s,a) \rightarrow$ memberikan state baru hasil dari *action a* terhadap *state s*

Berdasarkan persamaan di atas, agen dapat memperoleh optimal *policy* dengan mempelajari V^*, dengan kondisi agen memiliki pengetahuan yang sempurna terhadap *fungsi immediate reward r* dan fungsi trasisi *state* δ. Akan tetapi, pada awalnya agen tidak memiliki pengetahuan apapun tentang fungsi-fungsi tersebut, sehingga agen tidak dapat memilih *action* yang optimal. Oleh karena itu, dibutuhkanlah sebuah *evaluation function Q*, guna mendapatkan *evalutation function* yang optimal (V^*).

2.2.7.3.2 *Q-function*

Q-function dinotasikan sebagai *Q*(*s,a*) dan nilai dari *Q* adalah *reward* yang diterima secara langsung ketika agen mengeksekusi *action a* terhadap *state s*, ditambah nilai (didiskon oleh γ) dari optimal *policy* setelah *action-state* tersebut dipilih.

$$Q(s,a) \equiv r(s,a) + \gamma V^*(\delta(s,a))$$

Berdasarkan persamaan di atas, maka dapat ditulis kembali persamaan π^* menjadi:

$$\pi^*(s) = \arg\max_a Q(s,a)$$

Persamaan di atas menunjukkan bahwa agen mempelajari *Q-function* bukan mempelajari fungsi V^*. Dengan mempelajari *Q-function*, agen dapat memilih optimal *action* ketika agen tidak memiliki pengetahuan apapun tentang fungsi *r* dan δ. Persamaan di atas juga memperjelas bahwa untuk mendapatkan optimal *policy*, agen harus memilih *action a* terhadap *state s* yang dapat memaksimalkan $Q(s,a)$.

2.2.7.3.3 Algoritma *Q-learning*

Algoritma Q learning banyak digunakan di dalam pencarian sebuah jalur. Untuk lebih jelas mengenai algoritma Q learning dibawah ini merupakan pseudocode Q learning.

1. Set parameter γ, and environment reward (reward function)
2. Initialize the table entry $\hat{Q}(s,a)$ to zero
3. For each episode:
   a. Select random initial state
   b. Do while not reach goal state:
      - Select action a from s using ε-greedy strategy for the current state
      - Receive immediate reward r
      - Observe the new state s'
      - Update the table entry for $\hat{Q}(s,a)$ as follows:

        $$\hat{Q}(s,a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s',a')$$

      - Set the next state as the current state

   End Do
End For

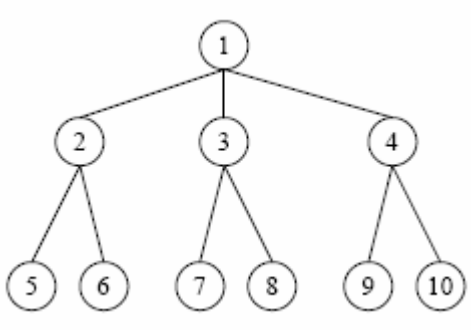
2.2.8. Backtracking

Algoritma *backtracking* sangat berguna untuk mencari solusi-solusi jumlah kombinasi jalan yang diperlukan untuk mencari solusi tersebut selalu berubah terhadap waktu ataupun respon *user* (dinamik). Dalam penerapannya algoritma *backtracking* juga mudah diimplementasikan dengan bahasa pemograman yang sudah mendukung pemanggilan fungsi atau prosedur secara rekursif. Untuk masalah - masalah yang mempunyai kemungkinan solusi yang kompleks, seperti permainan catur, sebaiknya kedalaman pohon dibatasi sehingga tidak menghabiskan waktu yang sangat lama (tentu saja batas kedalaman pohon ini relatif - bergantung pada kecepatan *cycle clock* prosesor yang tersedia). Secara umum, semakin dalam pohon yang ditelusuri, semakin akurat pula jalan menuju solusi (dalam hal *board games* maka tingkat kesulitan *Artificial*

Intelegence semakin bertambah). Untuk alokasi memori yang akan dipakai untuk menyimpan jalan solusi sebaiknya menggunakan *dynamic array* mengingat sebagian besar program yang menggunakan algoritma ini menghasilkan solusi yang tidak dapat diprediksi berhubung dengan sifat program yang akan sangat bergantung pada *state* program yang berubah setiap saat. Saran akhir dalam penggunaan algoritma *backtracking*, sebagaimana dengan algoritma-algoritma yang lain adalah kombinasikan dengan algoritma yang lain. Beberapa metoda dari *exhaustive search* bisa digunakan untuk menentukan apakah solusi yang sedang dijelajahi (*explore*) menuju ke solusi yang diharapkan.

2.2.8.1. Pengertian Algoritma Backtracking

Algoritma *backtracking* mempunyai prinsip dasar yang sama seperti *brute-force* yaitu mencoba segala kemungkinan solusi. Perbedaan utamanya adalah pada ide dasarnya, semua solusi dibuat dalam bentuk pohon solusi (pohon ini tentunya berbentuk abstrak) dan algoritma akan menelusuri pohon tersebut secara *DFS* (*depth field search)* sampai ditemukan solusi yang layak. Nama *backtracking* didapatkan dari sifat algoritma ini yang memanfaatkan karakteristik himpunan solusinya yang sudah disusun menjadi suatu pohon solusi. Agar lebih jelas bisa dilihat pada pohon solusi berikut:



Gambar 2.15 Tree

Misalkan pohon diatas menggambarkan solusi dari suatu permasalahan. Untuk mecapai solusi (5), maka jalan yang ditempuh adalah (1,2,5), demikian juga dengan solusi-solusi yang lain. Algoritma *backtracking* akan memeriksa mulai dari solusi yang pertama yaitu solusi (5). Jika ternyata solusi (5) bukan solusi yang layak maka algoritma akan melanjutkan ke solusi (6). Jalan yang ditempuh ke solusi (5) adalah (1,2,5) dan jalan untuk ke solusi (6) adalah (1,2,6). Kedua solusi ini memiliki jalan awal yang sama yaitu (1,2). Jadi daripada memeriksa ulang dari (1) kemudian (2) maka hasil (1,2) disimpan dan langsung memeriksa solusi (6). Pada pohon yang lebih rumit, cara ini akan jauh lebih efisien daripada *brute-force*.

Pada beberapa kasus, hasil perhitungan sebelumnya harus disimpan, sedangkan pada kasus yang lainnya tidak perlu.

2.2.8.2. Prinsip Pencarian Solusi dengan Metode *Backtracking*

Langkah - langkah pencarian solusi pada pohon ruang status yang dibangun secara dinamis yaitu :

1. Solusi dicari dengan membentuk lintasan dari akar ke daun. Aturan pembentukan yang dipakai adalah mengikuti metode pencarian mendalam (DFS). Simpul - simpul yang sudah dilahirkan dinamakan **simpul hidup** (*live node*). Simpul hidup yang sedang diperluas dinamakan **simpul-E** (*Expand-node*). Simpul dinomori dari atas ke bawah sesuai dengan urutan kelahirannya.
2. Tiap kali simpul-E diperluas, lintasan yang dibangun olehnya bertambah panjang. Jika lintasan yang sedang dibentuk tidak mengarah ke solusi maka simpul-E tersebut "dibunuh" sehingga menjadi **simpul mati** (*deadnode*).

   Fungsi yang digunakan untuk membunuh simpul-E adalah dengan menerapkan **fungsi pembatas** (*bounding function*). Simpul yang sudah mati tidak akan pernah diperluas lagi.
3. Jika pembentukan lintasan berakhir dengan simpul mati, maka proses pencarian diteruskan dengan membangkitkan simpul anak yang lainnya. Bila tidak ada lagi simpul anak yang dapat dibangkitkan, maka pencarian solusi dilanjutkan dengan melakukan *Backtracking* ke simpul hidup terdekat (simpul orang tua). Selanjutnya simpul ini menjadi simpul-E yang baru. Lintasan baru dibangun kembali sampai lintasan tersebut membentuk solusi.
4. Pencarian dihentikan bila kita telah menemukan solusi atau tidak ada lagi simpul hidup untuk *Backtracking*

2.2.8.3. Kegunaan Metode Backtracking

Penggunaan terbesar *backtracking* adalah untuk membuat *Artificial Intelegence* pada *board games*. Dengan algoritma ini program dapat menghasilkan pohon sampai dengan kedalaman tertentu dari *current status* dan memilih solusi yang akan membuat langkah-langkah yang dapat dilakukan oleh *user* akan menghasilkan pohon solusi baru dengan jumlah pilihan langkah terbanyak. Cara ini dipakai sebagai *Artificial Intelegence* yang digunakan untuk *dynamic problem solving*.

Beberapa kegunaan yang cukup terkenal dari algoritma *backtracking* dari suatu masalah "statik" adalah untuk memecahkan masalah *N-Queen problem* dan *maze solver*. *N-Queen problem* adalah bagaimana cara meletakkan bidak *Queen* catur sebanyak N buah pada papan catur atau pada papan ukuran NxN sedemikian rupa sehingga tidak ada satu bidak pun yang dapat memangsa bidak lainnya dengan 1 gerakan. Meskipun mungkin terdapat lebih dari satu solusi untuk masalah ini, tetapi pencarian semua solusi biasanya tidak terlalu diperlukan, tetapi untuk beberapa kasus tertentu diperlukan pencarian semua solusi sehingga didapatkan solusi yang optimal. *Maze solver* adalah bagaimana cara mencari jalan keluar dari suatu *maze* (labirin) yang diberikan. Pada *maze* yang sederhana dimana *field* yang dibentuk dapat direpresentasikan dalam bentuk biner dan pada setiap petak maksimal terdapat 4 kemungkinan: atas, kanan, bawah, dan kiri.

Untuk masalah ini biasanya solusi pertama yang ditemukan bukanlah solusi yang paling optimal sehingga untuk mendapatkan hasil yang optimal dibutuhkan pencarian terhadap seluruh kemungkinan solusi. Hal ini disebabkan oleh urutan pencarian yang telah ditetapkan dalam program (apakah menyelidiki kemungkinan ke arah atas dahulu atau ke arah lainnya dahulu).

2.2.8.4. Implementasi Algoritma *Backtracking*

Karena algoritma *backtracking* akan mencoba menelusuri semua kemungkinan solusi yang mungkin, maka hal pertama yang harus dilakukan adalah membuat algoritma dasar yang akan menjelajahi semua kemungkinan solusi. Kemudian algoritma ini diperbaiki sehingga cara pencarian solusinya dibuat lebih sistematis. Lebih tepatnya jika algoritma itu dibuat sehingga akan menelusuri kemungkinan solusi pada suatu pohon solusi abstrak. Algoritma ini memperbaiki teknik *brute-force* dengan cara menghentikan penelusuran cabang jira jika pada suatu saat sudah dipastikan tidak akan mengarah ke solusi. Dengan demikian jalan-jalan yang harus ditempuh yang ada dibawah *node* pada pohon tersebut tidak akan ditelusuri lagi sehingga kompleksitas program akan berkurang.

Kembali pada pohon solusi yang sebelumnya (Gambar 2.15 Tree) :

Jika pada pengecekan status di *node* (2) sudah dapat dipastikan bahwa jalan ini tidak akan menghasilkan solusi yang layak maka penelusuran jalan langsung dibatalkan dan dalam kasus ini langsung menelusuri *node* (3). Pembatalan penelusuran pda *node* (2) secara otomatis akan menghilangkan pengecekan jalan (1,2,5) dan (1,2,6) yang merupakan factor untuk mengurangi kompleksitas waktu yang diperlukan. Semakin cepat terdeteksi bahwa jalan yang ditempuh tidak akan mengarah ke suatu solusi yang layak maka program akan bekerja dengan lebih efisien.

Untuk kembali pada *state* sebelumnya (dalam kondisi *backtracking*) maka agar hasil perhitungan sampai dengan *state* tersebut harus disimpan dalam memori. Penyimpanan ini paling mudah dilakukan pada bahasa pemograman yang telah bisa menangani fungsi-fungsi atau prosedur-prosedur secara rekursif, sehingga manajemen

memori akan dilakukan sepenuhnya oleh *compiler*. Pada bahasa pemograman yang lain, algoritma *backtracking* masih dapat diimplementasikan meskipun manajemen memori harus dilakukan oleh *programmer*. Cara menajemen memori yang baik adalah menggunakan *pointer* atau *dynamic array* karena kedalaman pohon solusi yang harus ditelusuri biasanya bervariasi dan tidak dapat ditentukan. Algoritma *backtracking* dapat diimplementasikan dengan mudah pada bahasa-bahasa pemograman yang telah men-*support* pemograman rekursif. Bahasa pemograman yang nyaman digunakan adalah Pascal atau Java. Bahasa Pascal dipilih karena bisa diprogram secara rekursif dan mendukung penggunaan *pointer*. Sedangkan bahasa Java meskipun lebih rumit tetapi dapat bekerja secara rekursif dan sangat mudah dalam membuat *dynamic array*. Skema yang umum digunakan pada pemograman dengan fungsi rekursif adalah telusuri solusi yang ada kemudian cek *state* program apakah sedang menuju ke suatu solusi. Jika ya maka panggil kembali fungsi itu secara rekursif. Kemudian cek apakah solusi sudah ditemukan (jika hanya perlu mencari sebuah solusi) atau semua kemungkinan solusi sudah diperiksa (jika ingin mengecek semua kemungkinan solusi). Jika ya maka langsung keluar dari prosedur atau fungsi tersebut. Kemudian pada akhir fungsi kembalikan semua perubahan yang dilakukan pada awal fungsi. Pada bahasa pemograman yang telah mendukung pemanggilan secara rekursif semua *state* setiap fungsi akan diatur oleh *compiler*. Dengan skema ini maka jika program tidak memiliki solusi maka *state* akhir program akan sama dengan *state* awal program.

2.3 Game

Pengertian *game* (Roger Caollois, 1957) adalah suatu aktifitas yang memiliki karateristik sebagai berikut :

- Menyenangkan.
- Terpisah waktu dan tempat.
- Akhir dari permainan tidak dapat ditebak.
- Memiliki peraturan.
- Fiktif.

Video game adalah *game* yang diatur oleh komputer atau *microprocessor*. Komputer dapat membuat *virtual tool* untuk membuat sebuah *game* dimana benda-benda dapat dimanipulasi melalui *gameplay*.

Komponen pada *game* komputer:

2.3.1 *Graphics*

Graphics adalah nama yang umum digunakan untuk presentasi visual pada lingkungan *game*. Pembuatan lingkungan ini dimulai dengan konsep artis untuk mendesain karakter, objek, dan lingkungan yang seperti apa yang akan dibuat.

2.3.2 *Sound*

Sound memainkan peranan penting pada *game* dimana *sound* akan mempengaruhi *mood* dari tingkat kesadaran pemain. *Sound* dalam *games* pada umumnya terdiri dari *background music, event sound effects* (seperti klakson, suara tembakan) dan *environmental sound effects* (suara langkah kaki, suara

angin berhembus, suara burung, suara gelombang laut, suara serangga, suara gema).

2.3.3 *User Interface*

Sebuah *user interface* terdiri dari

- *Graphics* seperti *Buttons, info panels, maps*.
- Tata letak *(Layout).*
- Interaksi.

User interface merupakan salah satu komponen penting pada setiap *game,* karena hal ini adalah hal pertama yang dilihat pemain saat memulai suatu permainan.

2.3.4 *Game Playing*

Seiring dengan perkembangan industri *game* di dunia yang semakin pesat, negara kita tidak boleh kalah bersaing dengan industri-industri *game* di dunia. Dinegara kita sendiri *game* PC sudah banyak dikenal di lingkungan masyarakat, tetapi masih sedikit para *programer* yang membuat atau mengembangkan sebuah *game*. Dari beberapa macam *game* yang ada, ada berbagai macam tipe permainan *game* yang ada, antara lain permainan *game* komputer dan permainan *game online*.

Game playing adalah permainan video yang dimainkan pada *personal computer. Game playing* telah berevolusi dari sistem grafis sederhana sampai menjadi kompleks dan mutakhir.

Beberapa macam karateristik dan batasan *game* yang ada didalam *game playing* adalah :

- Dimainkan oleh 2 (dua) pemain: manusia dan komputer. Para pemain saling bergantian melangkah.
- *Perfect Information Game*: kedua pemain sama-sama memiliki akses pada informasi yang lengkap tentang keadaan permainan, sehingga tidak ada informasi yang tertutup bagi lawan mainnya.
- *No Determined by Chances.* Tidak melibatkan faktor probabilitas, misalnya dengan menggunakan dadu.
- *No Phsychological Factors.* Tidak melibatkan faktor psikologi, seperti "gertakan" (misalnya Poker).
- *No Oversight Errors. Smart Opponent.* Lawan diasumsikan pintar juga, jadi jangan mengharap lawan *khilaf,* sehingga terjadi salah langkah.

2.3.5 *Input*

Game biasanya memberikan banyak pilihan terhadap *input*. Yang termasuk didalam *input* antara lain *mouse, keyboard, joysticks,* dan *gamepads*. Idealnya suatu *game engine* harus dapat mengabstraksikan *input* sehingga user dapat memilih *input* yang akan digunakan.

2.3.6 *Game* Tool atau *Game* Engine

Game engine merupakan komponen inti piranti lunak dari sebuah *game* komputer atau aplikasi interaktif lain dengan grafik *real-time*. *Game engine* menyediakan teknologi untuk mempermudah pengembangan dan seringkali

memungkinkan *game* untuk dijalankan di berbagai *platform* seperti *console game* dan sistem operasi seperti Linux, Mac Os X, dan Windows. Fungsi pokok yang disediakan *game engine* meliputi *rendering engine* untuk grafik 2D atau 3D, *physic engine,* suara, *script*, animasi, kepandaian buatan, *networking*, *streaming*, pengaturan memori, *threading*, dan grafik tampilan. Sebagian besar modul tersebut berisi algoritma tingkat tinggi yang digunakan di beberapa kasus yang telah dikembangkan untuk militer, pengetahuan, pengobatan, spesial efek dari industri film.

2.4 *Path planning*

Bagi agen, pergerakan yang telah dipertimbangkan biasanya dapat dipisah menjadi dua bagian, yaitu fase penentuan dan fase pelaksanaan. Meskipun pemisahan ini merupakan kunci dari berbagai pendekatan berbasis robot, namun sering dibuat secara explisit dalam *Artificial Intelligence* pada *game*. Kesederhaan alami pada lingkungan saat ini adalah salah satu penyebab utama hal tersebut. Sekali direncanakan, pergerakan biasanya di asumsikan untuk mungkin dilaksanakan dan rencana apriori jarang dipertimbangkan.

Adapun definisi dari *path planning*:

- *Path planning* merupakan seni untuk menentukan rute mana yang akan diambil, berdasar pada aturan dalam representasi internal pada dataran yang ada saat ini.

- *Path Finding* adalah suatu pelaksanaan dari rute yang secara teoris ini, dengan menerjemahkan rencana berdasar representasi internal dalam aturan pada pergerakan dalam lingkungan.

Meskipun beberapa aplikasi membutuhkan penggabungan dari dua konsep berikut, perbedaan ini memiliki banyak keunggulan termasuk kesederhanaan, sebuah implementasi modular, dan perngintegrasian yang mudah dari berbagai teknologi.

Dan juga, meskipun dua hal ini berbeda dalam hal konsep, mereka berinteraksi dalam banyak yang terlalu penting untuk dilewatkan. *Path Planner* harus secara alami membiarkan *path finder* mengetahui *path* yang sering terpilih. Selanjutnya, harus ada *feedback* dari *path finder* kepada *path planner* sehingga *path* yang tidak mungkin dicapai dapat dicatat dan dipertimbangkan.

2.5 *Obstacle avoidance*

Obstacle avoidance merupakan suatu kegiatan untuk menyelesaikan subyek obyektif kontrol dimana letaknya bukan pada persimpangan atau terletak di posisi bukan tabrakan. *Obstacle avoidance* termasuk di dalam *path planning*. Dimana biasanya terimplementasi sebagai aturan kontrol yang reaktif selama yang lain memasukkan perhitungan terlebih dahulu terhadap path bebas obstacle, dimana pengontrol kemudian akan dibimbing selama perjalanan.

Obstacle avoidance merupakan suatu atuan robotic dengan tujuan untuk memindahkan kendaraan dengan informasi berdasarkan sensor. Penggunaan metode dihadapkan dengan *path planning* yang menjadi alternative alami saat skenario yang digunakan dinamis dengan kebiasaan yang tidak dapat ditebak. Dalam kasus ini,

keadaan sekitar tidak selalu berubah, dan informasi sensoris digunakan untuk mendeteksi perubahan gerak yang adaptif secara bertahap.