BAB 2
LANDASAN TEORI

2.1 Pengertian *Game*

Game adalah permainan yang menggunakan interaksi dengan antarmuka pengguna melalui gambar yang dihasilkan oleh piranti *video*. Permainan *video* umumnya menyediakan sistem penghargaan, misalnya skor yang dihitung berdasarkan tingkat keberhasilan yang dicapai dalam menyelesaikantugas-tugas yang ada di dalam permainan. Sistem elektronik yang digunakan untuk menjalankan *game* disebut *platform*, seperti komputer pribadi ataupun konsol permainan[6]. *Game* bertujuan untuk menghibur, biasanya game banyak disukai oleh anak-anak hingga orang dewasa. *Games* sebenarnya penting dalam perkembangan otak, meningkatkan konsentrasi dan melatih untuk memecahkan masalah dengan tepat dan cepat. Karena dalam *game* terdapat berbagai konflik atau masalah yang menuntut untuk diselesaikan dengan cepat dan tepat[7].

2.1.1 Jenis *Game*

Berdasarkan jenis permainanya, *game* dibagi menjadi[8]:

1. *Action games* memerlukan pemain untuk menggunakan refleks, akurasi, dan waktu yang tepat untuk menyelesaikan sebuah tantangan. Ini mungkin merupakan genre dasar dari sebuah permainan, dan yang memiliki permainan terbanyak. Dalam permainan aksi, biasanya terdapat pertempuran.
2. *Fighting games* adalah jenis permainan dimana terjadi pertarungan satu lawan satu antara dua karakter, salah satunya dapat merupakan komputer. Jenis permainan ini biasanya dimainkan dengan menghubungkan kombinasi-kombinasi tombol untuk bertarung. Rata-rata gerakan yang dihasilkan oleh karakter dramatis dan tidak mungkin dilakukan di kehidupan nyata.
3. *Shooter games* adalah *sub-genre* dari permainan aksi, meskipun banyak pemain yang menganggap bahwa jenis ini merupakan jenis yang berdiri sendiri. Jenis ini menguji kecepatan dan reaksi dari pemainnya. Tembak-menembak memiliki *sub-genre* tersendiri, yang semuanya terfokus pada

mengontrol seseorang yang menggunakan sebuah senjata, biasanya pistol, senapan, atau senjata jarak jauh lainnya. Hal yang biasanya ada pada permainan jenis ini adalah amunisi. Biasanya tujuan dari permainan ini adalah menembak lawan dan melanjutkan misi tanpa karakter yang dipakai mati.

4. *First-person shooter* (FPS), menekankan ke menembak dengan perspektif orang-pertama yang dikontrol oleh pemain. Perspektif ini bertujuan untuk memberikan pemain rasa seakan benar-benar berada disana, dan dapat membuat pemain fokus membidik sasaran. Kebanyakan FPS memerlukan refleks yang cepat dalam tingkat kesulitan yang tinggi. Unsur 3D diperlukan untuk membuat FPS yang baik. Sejak kemunculannya, FPS sekarang memiliki fitur *multiplayer* yang memungkinkan berkompetisi antar pemain.
5. *Third-person shooter* (TPS/3PS), menekankan ke menembak dari persperktif orang-ketiga atau karakter pemainnya dapat terlihat sepenuhnya. Persperktif ini memberikan pemain pandangan yang lebih luas akan lingkungan sekitar dari FPS. Juga, TPS memungkinkan pergerakan yang lebih rumit seperti bergulir atau menyelam, dibandingkan dengan melompat dan merunduk sederhana yang biasa terlhat di FPS. Meskipun begitu, kebanyakan jenis permainan ini memiliki elemen *puzzle* dari FPS.
6. *Massively multiplayer online first-person shooter* (MMOFPS), merupakan permainan *online* yang menggabungkan pemain FPS pada sebuah dunia *virtual* dimana para pemain dapat berinteraksi satu dengan lainnya.
7. *Racing game* juga termasuk dalam kategori ini, dimana jenis permainan *video* yang menuntut keterampilan pemain untuk mengemudi dalam sebuah kompetisi balap-membalap.
8. *Sports games* yaitu jenis permainan *video* yang menuntut keterampilan pemain untuk melakukan pertandingan olahraga secara *virtual*, seperti pertandingan sepak bola, basket, dan sebagainya.
9. *Role-playing games* yaitu jenis permainan *video* yang menempatkan pemain sebagai tokoh dalam permainan tersebut untuk memecahkan suatu misteri dengan menyelesaikan berbagai macam *puzzle* dan *quest*. Jenis ini biasanya memiliki alur cerita yang kompleks.

10. *Action-adventure games* yaitu jenis permainan *video* yang menempatkan pemain sebagai tokoh dalam permainan tersebut, mirip dengan permainan peran, tetapi ditambah dengan unsur-unsur aksi, misalnya perkelahian dan tembak-menembak.
11. *Real-time strategy* (RTS), yaitu jenis permainan *video* yang mengandung unsur strategi. Permainan *video* RTS memerlukan kecakapan pemain untuk memimpin suatu bangsa, mengelola sumber daya, dan melancarkan perang untuk memperluas wilayah kekuasaan.
12. *Turn-based strategy* (TBS), yaitu jenis permainan *video* strategi yang pemainnya saling menunggu giliran untuk bergerak.

2.1.2 *Real-Time Strategy* (RTS)

Real Time Strategy merupakan game yang memerlukan kemampuan berpikir dan memutuskan strategi dengan tepat. Pemain akan mengontrol sebuah kelompok untuk mencapai tujuan, contohnya adalah memenangkan perang melawankelompok lain. Kebalikan dari *video game* jenis aksi-menembak yang berjalan cepat, *video game* jenis strategi memberikan waktu lebih untuk memutuskan setiap gerakan secara hati-hati dan terencana[9].

2.2 *Tower Defense*

Tower defense adalah game yang bertipe *real time strategy* yang berfokus pada pengaturan sumber daya yang dimiliki pemain dan penempatan *tower*. dalam bentuk yang paling sederhana, sebuah *game tower defense* terdiri dari pemain yang bertugas membeli dan mengorganisir *tower* yang akan menembak berbagai jenis musuh yang menyerang pertahanan pemain.

Inti dari permainan *Tower Defense* ini adalah *gamer* ditugaskan untuk melindungi *goal point* dari serbuan musuh yang dikendalikan oleh *artificial intelligence*. Musuh ini akan datang secara bergelombang dan dalam jumlah yang banyak. *Gamer* melindungi target dengan cara mendirikan *tower-tower* atau menara-menara pertahanan yang akan menembaki setiap musuh yang mendekati target.

Permainan dibagi dalam tingkatan-tingkatan, dimulai dari tingkatan rendah, menegah, hingga tingkatan tinggi sesuai dengan tingkat keberhasilan gamer dalam mempertahankan goal point setiap tingkatannya. Dan semakin tinggi tingkatan yang dicapai, musuh yang menyerbu semakin kuat dan banyak. Serta uang yang didapatkan dari membunuh musuh-musuh ini pun semakin meningkat untuk digunakan mendirikan menara-menara pertahanan yang lebih canggih dan kuat.

Permainan ini menuntut ketangkasan, perhitungan, strategi, menempatkan berbagai macam *tower* dengan harga yang beragam. Adapun uang yang digunakan untuk membeli *tower* didapatkan dari hasil membunuh setiap musuh yang ada. Permaianan dinyatakan sukses apabila pemain dapat mempertahankan darah atau nyawanya sampai waktu yang ditentukan habis atau berdasarkan jumlah musuh yang dibunuh[10].

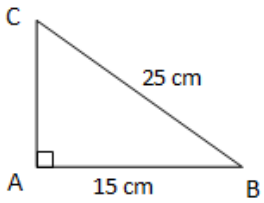
2.3 Teorema Pythagoras

Teorema Pythagoras dinamakan oleh ahli matematika Yunani kuno yaitu Pythagoras, yang dianggap sebagai orang yang pertama kali memberikan bukti teorema ini. Akan tetapi, banyak orang yang percaya bahwa terdapat hubungan khusus antara sisi dari sebuah segi tiga siku-siku jauh sebelum Pythagoras menemukannya.

Teorema Pythagoras memainkan peran yang sangat signifikan dalam berbagai bidang yang berkaitan dengan matematika. Misalnya, untuk membentuk dasar trigonometri dan bentuk aritmatika, di mana bentuk ini menggabungkan geometri dan aljabar. Teorema ini adalah sebuah hubungan dalam geometri Euclides di antara tiga sisi dari segi tiga siku-siku. Hal ini menyatakan bahwa jumlah dari persegi yang dibentuk dari panjang dua sisi siku-sikunya akan sama dengan jumlah persegi yang dibentuk dari panjang hipotenusanya.

Secara matematis, teorema ini biasanya ditulis sebagai a dan b mewakili panjang dari dua sisi lain dari segitiga siku-siku dan c mewakili panjang dari hipotenusanya [11].

$$a^2 + b^2 = c^2 \qquad (2.1)$$



$$AC^2 = BC^2 - AB^2$$
$$AC = \sqrt{BC^2 - AB^2}$$
$$AC = \sqrt{(25\ \text{cm})^2 - (15\ \text{cm})^2}$$
$$AC = \sqrt{625\ \text{cm}^2 - 225\ \text{cm}^2}$$
$$AC = \sqrt{400\ \text{cm}^2}$$
$$AC = 20\ \text{cm}$$

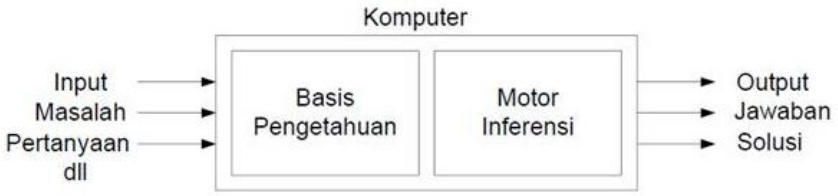
Gambar 2.1 Contoh Teorema Pythagoras

2.3 Kecerdasan Buatan

Kecerdasan buatan berasal dari bahasa Inggris *Artificial Intelligence* atau disingkat AI, yaitu *intelligence* adalah kata sifat yang berarti cerdas, sedangkan *artificial* artinya buatan. Kecerdasan yang dimaksud disini merajuk pada mesin yang mampu berpikir seperti manusia, berfikir secara rasional, dan mampu mengambil keputusan seperti yang dilakukan oleh manusia[12]. Kecerdasan buatan telah banyak digunakan dan diaplikasikan kedalam kehidupan sehari-hari.

Kecerdasan buatan memiliki beberapa kelebihan dibandingkan dengan kecerdasan buatan yang dimiliki oleh manusia yaitu

a. Bersifat lebih permanen: Selama sistem dan program tidak berubah, maka kecerdasan buatan tersebut tidak akan berubah.
b. Lebih mudah untuk diduplikasikan dan disebarkan: Akan lebih mudah memindahkan data dari satu komputer ke komputer lain dibandingkan dengan memindahkan pengetahuan dari satu manusia ke manusia yang lain.
c. Lebih murah dibandingkan dengan mendatangkan seorang ahli.
d. Konsisten: Kecerdasan buatan merupakan sebuah teknologi komputer sedangkan kecerdasan buatan memiliki kecendrungan untuk berubah.
e. Bisa didokumentasi: Tiap aktifitas yang dilakukan oleh kecerdasan buatan dapat dilacak dengan mudah sedangkan kecerdasan alami termasuk sulit untuk direproduksi.
f. Mengerjakan pekerjaan dengan waktu lebih cepat dan lebih baik.



Gambar 2.2 Penerapan Kecerdasan Buatan

Kecerdasan buatan menyediakan dasar-dasar ilmu pengetahuan pada beberapa bidang teknologi yang dapat digunakan secara komersial diantaranya:

a. Sistem pakar: Sebuah sistem komputer yang digunakan sebagai sarana untuk menyimpan pengetahuan yang dimiliki oleh seorang atau lebih pakar dengan tujuan agar komputer memilki keahlian untuk menyelesaikan permasalahan dengan meniru keahlian yang dimilki pakar tersebut.
b. Pengolahan bahasa alami: Pemrograman sistem komputer dimana pengguna dapat berkomunikasi dengan komputer menggunakan bahasa sehari-hari
c. Pengenalan suara: Pengguna dapat berkomunikasi dengan komputer untuk memberi perintah kepada untuk melakukan sebuah pekerjaan.
d. Robotika dan sistem sensor: Sebuah lengan robotik yang dilengkapi oleh berbagai sensor yang diprogram untuk mampu mendeteksi jenis pekerjaan yang perlu dilakukan oleh lengan tersebut.
e. *Computer vision*: Pemrograman sistem yang bertujuan untuk menginterpretasikan gambar dan objek tampak melalui komputer untuk proses lebih jauh.
f. *Game playing*: *Game playing* merupakan implementasi *artificial intelegence* yang berguna untuk meniru cara berfikir seorang manusia dalam bermain *game*.

Adapun teknik pemecahan masalah yang ada didalam kecerdasan buatan terbagi menjadi empat yaitu[13]:

a. *Searching*: Merupakan teknik menyelesaikan masalah dengan cara merepresentasikan masalah ke dalam state dan ruang masalah serta menggunakan strategi pencarian untuk menemukan solusi.

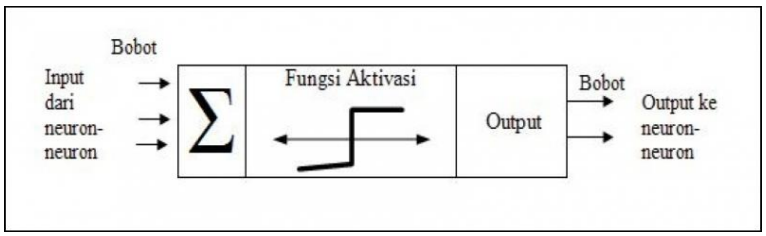
b. *Reasoning*: Merupakan teknik menyelesaikan masalah dengan cara merepresentasikan masalah ke dalam basis pengetahuan menggunakan *logic* atau bahasa formal.
c. *Planing*: Merupakan teknik menyelesaikan masalah dengan cara memecah masalah ke dalam sub-sub masalah dan menyelesaikan sub-sub masalah satu demi satu.
d. *Learning*: Merupakan teknik menyelesaikan masalah dengan mengotomatisasi mesin untuk menemukan aturan yang diharapkan bisa berlaku umum untuk data-data yang belum pernah diketahui.

2.4 Jaringan Saraf Tiruan (JST)

Merupakan simulasi dari otak biologis. Tujuan dari JST adalah untuk belajar mengenali pola-pola pada data dan mensimulasikan proses belajar adaptif biologis, walau dalam skala yang sangat sederhana. Sekali JST telah dilatih terhadap data, akan dapat membuat prediksi dengan melakukan deteksi kemiripan/kesamaan pola-pola data masukan.

JST bukanlah duplikasi persis dari sistem biologis otak manusia, tetapi jaringan saraf tiruan ini dapat melakukan kemampuan seperti generalisasi, belajar, abstraksi, dan bahkan intuisi. Mudahnya, merupakan suatu model dari sistem saraf biologis yang disederhanakan sebagai suatu alternatif sistem komputer. Kenyataan menunujukan bahwa banyak masalah dalam kehidupan manusia yang sulit dipecahkan dengan komputer konvensional yang paling canggih sekalipun, namun manusia dapat menyelesaikannya dengan baik. Dengan kemampuannya untuk belajar, jaringan saraf tiruan ini diharapkan dapat menyelesaikan masalah yang tidak dapat diselesaikan oleh komputer konvensional.

Jaringan saraf tidak diprogram dalam arti tradisional. Sebaliknya dilatih dengan contoh. Latihan itu terdiri dari banyak pengulangan *input* yang mengungkapkan berbagai hubungan. Dengan memperhalus bobot node sistem secara progresif, jaringa saraf tiruan ini menemukan hubungan antar *input*. Proses penemuan ini menandakan belajar.



Gambar 2.3 Jaringan Saraf Tiruan

JST merupakan salah satu bentuk dari Kecerdasan Buatan. JST dipandang sebagai suatu *Black Box* yang dapat melakukan prediksi keluaran dari suatu pola masukan yang dia kenali. Untuk itu JST harus dilatih terlebih dahulu terhadap sejumlah pola masukan dan target yang diharapkan dari tiap pola masukan tersebut (*Supervised Learning*). Sekali dilatihkan, JST akan dapat mengenali kesamaan ketika dihadapkan terhadap pola masukan baru, dan menghasilkan prediksi pola keluarannya.

JST dapat mendeteksi kesamaan masukan, bahkan sebagian masukan yang mungkin belum pernah dilatihkan atau diberikan sebelumnya. Karena JST mempunyai kemampuan interpolasi yang hebat, terutama bila data masukan tidak eksak, banyak gangguan didalamnya. Sehingga memungkinkan JST untuk digunakan sebagai subtitusi langsung bagi auto korelasi, regresi multivariabel, regresi linier, trigonometri, dan teknik regresi lainnya. Ketika data dianalisa menggunakan JST, akan memungkinkan untuk melakukan prediksi pola yang penting sebagaimana bila seorang ahli menganalisa data tersebut, karena JST dapat beraksi seperti selayaknya seorang yang ahli di bidangnya[14].

2.5 *Backpropagation Neural network* (BPNN)

Merupakan salah satu terapan dari *Multi Layered Network*. BPNN sendiri sebenarnya lebih merupakan algoritma pelatihan daripada jaringan itu sendiri. Metode pelatihan BPNN adalah *supervised training algorithm* untuk jaringan layar jamak. Dikarenakan metode yang digunakan adalah pelatihan terpantau, maka baik *input* maupun target *output* telah disediakan untuk melatih jaringan. Kesalahan pada data di layer *output* dihitung menggunakan *network output* dan target *output*. Kemudian kesalahan itu di propagasikan kembali ke *hidden layer*, memberikan perubahan *weight* pada *synapses* yang menuju ke layer tersebut.

Backpropagation atau propagasi balik suatu algoritma pembelajaran dan digunakan oleh perceptron dengan banyak lapisan untuk mengubah bobot - bobot yang terhubung dengan *neuron* - *neuron* yang terdapat pada lapisan tersembunyi. Algoritma propagasi balik menggunakan *error output* agar dapat mengubah nilai bobot - bobotnya dalam proses mundur untuk mendapatkan *error* ini, tahap perambatan maju, *neuron* - *neuron* diaktifkan dengan menggunakan fungsi aktivasi sigmoid, sedangkan dalam melakukan sebuah proses perambatan maju dan mundur disebut epoch atau iterasi[15].

2.5.1 Fungsi Aktifasi

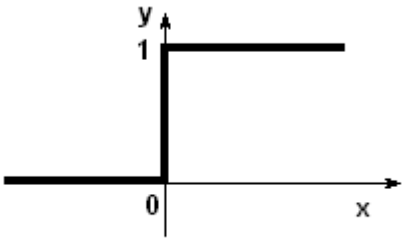
Fungsi aktivasi merupakan fungsi matematis yang berguna untuk membatasi dan menentukan jangkauan *output* suatu *neuron*. Fungsi aktivasi untuk JST *Backpropagation* harus memiliki beberapa karakteristik penting, yaitu kontinyu, dapat dideferensialkan, dan merupakan fungsi yang tidak turun. Fungsi aktivasi biasanya digunakan untuk mencari nilai asimtot maksimum dan minimum.

Ada beberapa fungsi aktivasi yang sering digunakan dalam jaringan syaraf tiruan, antara lain[16]:

a. Fungsi Undak *Biner Hard Limit*

Jaringan dengan lapisan tunggal sering menggunakan fungsi undak (*step function*) untuk mengkonversikan *input* dari suatu variabel yang bernilai kontinu ke suatu *output* biner (0 atau 1). Fungsi undak *biner* dirumuskan sebagai berikut:

$$y = \begin{cases} 0, jika\ x\ \leq 0 \\ 1, jika\ x\ > 0 \end{cases} \quad (2.2)$$

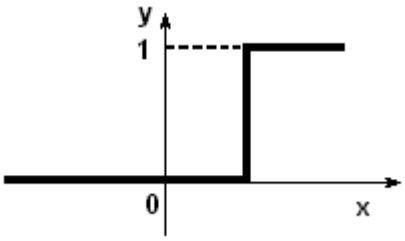


Gambar 2.4 Fungsi Aktivasi Undak Biner *Hard Limit*

b. Fungsi Undak *Biner Threshold*

Fungsi undak *biner* dengan menggunakan nilai ambang sering juga disebut dengan nama fungsi nilai ambang (*threshold*) atau fungsi *Heaviside*. Fungsi undak biner (dengan nilai ambang 0) dirumuskan sebagai:

$$y = \begin{cases} 0, jika\ x < 0 \\ 1, jika\ x \geq 0 \end{cases} \quad (2.3)$$

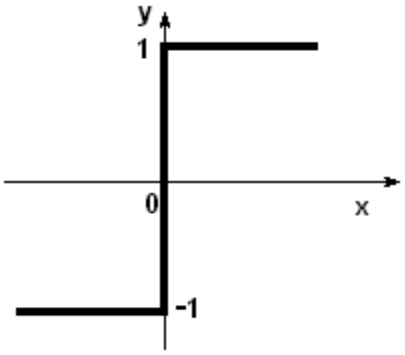


Gambar 2.5 Fungsi Aktivasi Undak *Biner Treshold*

c. Fungsi *Bipolar Symetric Hard Limit*

Fungsi *bipolar* sebenarnya hampir sama dengan fungsi undak biner, hanya saja *output* yang dihasilkan berupa 1, 0 atau -1. Fungsi *bipolar Symetric Hard Limit* dirumuskan sebagai:

$$y = \begin{cases} 1\ jika\ x > 0 \\ 0\ jika\ x = 0 \\ -1\ jika\ x < 0 \end{cases} \quad (2.4)$$

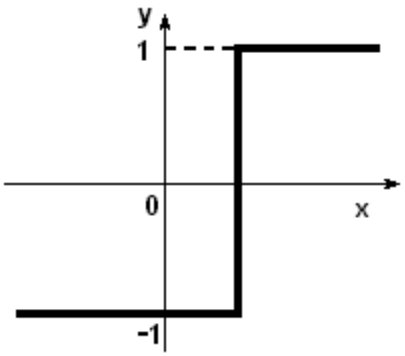


Gambar 2.6 Fungsi Aktivasi *Bipolar Symetric Hard Limit*

d. Fungsi *Bipolar Threshold*

Fungsi *bipolar* sebenarnya hampir sama dengan fungsi undak *biner threshold*, hanya saja *output* yang dihasilkan berupa 1, atau -1. Fungsi *bipolar threshold* dirumuskan sebagai:

$$y = \begin{cases} 1, jika\ x \geq 0 \\ -1, jika\ x < 0 \end{cases} \quad (2.5)$$

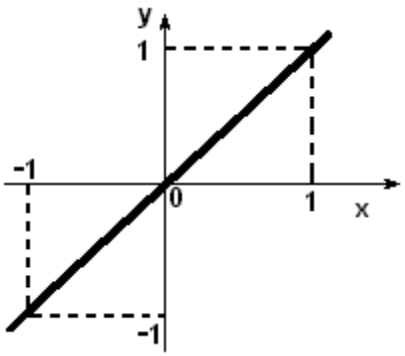


Gambar 2.7 Fungsi Aktivasi *Bipolar Treshold*

e. Fungsi *Linear* (Identitas)

Fungsi *linear* memiliki nilai *output* yang sama dengan nilai *input*nya. Fungsi *linear* dirumuskan sebagai:

$$y = x \tag{2.6}$$



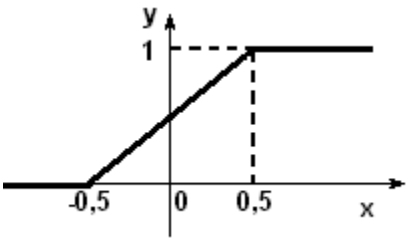
Gambar 2.8 Fungsi Aktivasi *Linear*

f. Fungsi *Saturating Linear*

Fungsi ini akan bernilai 0 jika *input*nya kurang dari -½, dan bernilai 1 jika *input*nya lebih dari ½. Sedangkan jika nilai *input* terletak antara -½ dan½, maka *output* akan bernilai sama dengan *input* ditambah ½. Fungsi *Saturating linear* dirumuskan sebagai:

$$y = \begin{cases} 1 \; jika \; x \geq 0{,}5 \\ x + 0{,}5 \; jika - 0{,}5 \leq x \leq 0{,}5 \\ 0 \; jika \; x \leq -1 \end{cases}$$

(2.7)

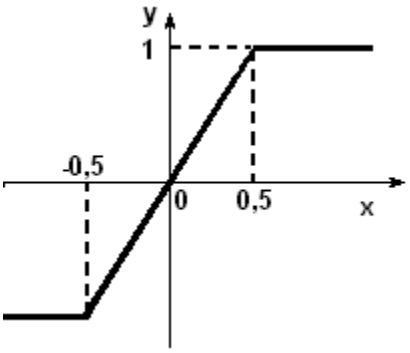


Gambar 2.9 Fungsi Aktivasi *Saturating Linear*

g. Fungsi *Symetric Saturating Linear*

Fungsi ini akan bernilai -1 jika *input*nya kurang dari -1, dan akan bernilai 1 jika *input* lebih dari 1. Sedangkan jika nilai *input* terletak antara -1 dan 1, maka *output*nya akan bernilai sama dengan *input*nya. Fungsi *symetric saturating linear* dirumuskan sebagai:

$$y = \begin{cases} 1 \; jika \; x \geq 1 \\ x \; jika -1 \leq x \leq 1 \\ 1 \; jika \; x \leq -1 \end{cases} \tag{2.8}$$



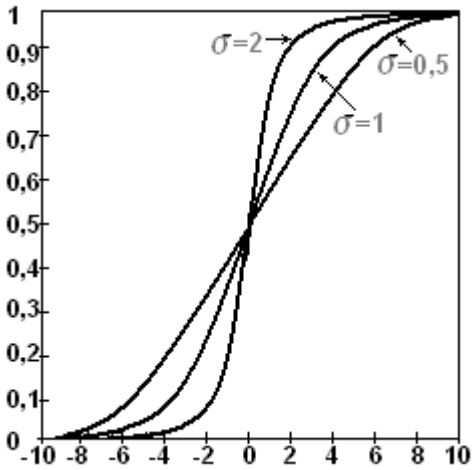
Gambar 2.10 Fungsi Aktivasi *Symetric Saturating Linear*

h. Fungsi *Sigmoid Biner*

Fungsi ini digunakan untuk jaringan saraf yang dilatih dengan menggunakan metode *backpropagation*. Fungsi *sigmoid biner* memiliki nilai *range* 0 sampai 1. Oleh karena itu, fungsi ini sering digunakan untuk jaringan saraf yang membutuhkan nilai *output* yang terletak pada interval 0 sampai 1. Namun, fungsi ini bisa juga digunakan oleh jaringan saraf yang nilai *output*nya 0 atau 1 (Gambar 2.9). Fungsi *sigmoid biner* dirumuskan sebagai:

$$y = f(x) = \frac{1}{1+e^{-\sigma x}} \tag{2.9}$$

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



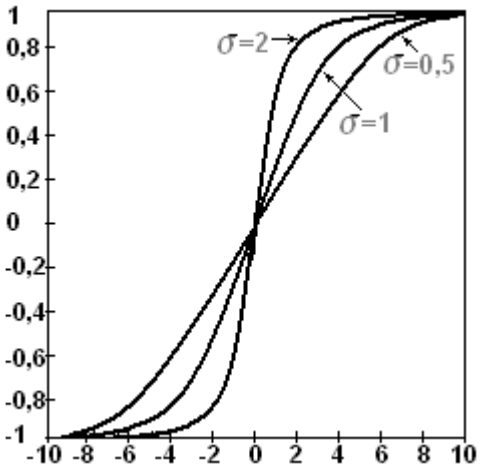
Gambar 2.11 Fungsi Aktivasi *Sigmoid Biner*

i. Fungsi *Sigmoid Bipolar*

Fungsi *sigmoid bipolar* hampir sama dengan fungsi *sigmoid biner*, hanya saja *output* dari fungsi ini memiliki *range* antara 1 sampai -1. Fungsi *sigmoid bipolar* dirumuskan sebagai:

$$y = f(x) = \frac{1-e^{-\sigma x}}{1+e^{-\sigma x}} \tag{2.10}$$

$$dengan\ f'(x) = \frac{\sigma}{2}[1 + f(x)][1 - f(x)] \tag{2.11}$$



Gambar 2.12 Fungsi Aktivasi *Sigmoid Bipolar*

2.5.2 Proses Pelatihan JST *Backpropagation*

Inti dari algoritma pembelajaran dengan metode *Backpropagation* terletak pada kemampuannya untuk mengubah nilai-nilai bobotnya untuk menanggapi adanya kesalahan. Untuk dapat menghitung kesalahan, pada proses pembelajaran perlu adanya pola-pola keluaran yang dijadikan target oleh jaringan, sehingga setiap keluaran yang dihasilkan oleh jaringan akan dibandingkan dengan targetnya. Hasil dari perbandingan ini berupa *error* atau kesalahan. Oleh karena itu, JST *Backpropagation* merupakan jaringan dengan proses pembelajaran secara terbimbing.

Setelah kesalahan diperoleh, selanjutnya jaringan melewatkan turunan-turunan dari kesalahan ke lapisan tersembunyi menggunakan sambungan terbobot yang masih belum diubah nilainya. Setiap simpul pada lapisan tersembunyi menghitung jumlah terbobot dari kesalahan yang telah dipropagasikan balik untuk menghitung sumbangan tidak langsungnya kepada kesalahan keluaran yang telah diketahui. Setelah masing-masing simpul pada lapisan tersembunyi dan lapisan keluaran menemukan besarnya kesalahan, simpul-simpul tersebut akan mengubah bobot-bobotnya untuk mengurangi kesalahan tersebut. Perubahan bobot ditujukan untuk meminimalkan jumlah kesalahan kuadrat jaringan. Oleh sebab inilah, algoritma *Backpropagation* dikatakan sebagai suatu prosedur untuk mendapatkan paket bobot yang meminimalkan jumlah kuadrat kesalahan. Jumlah kuadrat kesalahan akan semakin mengecil dengan berjalannya waktu dan iterasi yang dilakukan oleh jaringan tersebut.

Pelatihan dalam *Backpropagation* meliputi 3 fase. Fase pertama adalah fase maju, yaitu pola masukan dihitung sampai layer keluaran dengan fungsi aktivasi yang ditentukan. Fase selanjutnya adalah propagasi mundur, yaitu fase pengoreksian antara nilai keluaran terhadap targetnya yang kemudian dipropagasi mundur. Fase yang terakhir adalah fase perubahan bobot[17].

Langkah 0 : Inisialisasi bobot

Langkah 1 : Sementara kondisi berhenti salah, lakukan langkah 2-9

Langkah 2 : Untuk setiap pasang data belajar, lakukan langkah 3-8

Fase I : Propagasi Maju

Langkah 3 : Masing-masing unit masukan (Xi, i= 1,2,…,n) menerima sinyal masukan Xi dan sinyal tersebut disebarkan ke unit-unit bagian berikutnya (unit-unit lapisan tersembunyi)

Langkah 4 : Masing-masing unit dilapisan tersembunyi (Zj, j= 1,2,…,p) dikalikan dengan faktor penimbang dan dijumlahkan serta ditambahkan dengan biasnya:

$$Z_in_j = V_{0j} + \sum_{i=1}^{n} X_i V_{ij} \quad (2.12)$$

Kemudian menghitung sesuai dengan fungsi aktivasi yang digunakan.

Langkah 5 : Masing-masing unit keluaran (Yk, k=1,2,3,…m) dikalikan dengan faktor penimbang dan dijumlahkan:

$$Y_in_k = W_{0k} + \sum_{j=1}^{p} Z_j W_{ij} \quad (2.13)$$

Menghitung kembali sesuai dengan fungsi aktivasi yang digunakan:

Fase II : Propagasi Mundur

Langkah 6 : Masing-masing unit keluaran (Yk, k=1,…,m) menerima pola target sesuai dengan pola masukan saat pelatihan dan dihitung galatnya:

$$\delta_k = (t_k - y_k) f'(y_in_k) \quad (2.14)$$

Menghitung dengan menggunakan fungsi aktivasi yang digunakan sehingga menghasilkan suatu persamaan untuk menghitung nilai nya.

Menghitung nilai koreksi bobot yang digunakan untuk memperbarui nilai bobot $w_j k$:

$$\Delta w_{jk} = \alpha \delta_k z_j \quad (2.15)$$

Menghitung nilai koreksi bias yang digunakan untuk memperbarui nilai $w_0 k$:

$$\Delta w_{0k} = \alpha \delta_k \quad (2.16)$$

Dan mengirim nilai *error* δ_k ke lapisan dibawahnya.

Langkah 7 : Setiap *neuron* tersembunyi (Zj, j=1,..,m) menghitung masukan deltadari *neuron* di atasnya:

$$\delta_in_j = \sum_{k=1}^{m} \delta_k w_{jk} \quad (2.17)$$

Dikalikan dengan turunan dari fungsi aktivasinya untuk menghasilkan nilai *error*:

$$\delta_j = \delta_in_j f'(z_in_j) \quad (2.18)$$

Menghitung nilai koreksi bobot yang digunakan untuk memperbarui nilai bobot $v_j k$:

$$\Delta v_{ij} = \alpha \delta_j x_i \quad (2.19)$$

Menghitung nilai koreksi bias yang digunakan untuk memperbarui nilai $v_0 k$

$$\Delta v_{0j} = \alpha \delta_j \quad (2.20)$$

Memperbarui nilai bobot dan bias:

Fase III : Perubahan Bobot

Langkah 8 : Hitung semua perubahan bobot

Masing-masing keluaran unit (yk, k=1,2…,m) diperbaiki bias dan penimbangnya (j=0,1…,p)

$$w_{jk}(new) = w_{jk}(old) + "w_{jk} \quad (2.21)$$

Masing-masing unit tersembunyi (Zj, j:1,2…,p) diperbaiki bias dan penimbangnya (i=0,1…,n)

$$v_{ij}(new) = v_{ij}(old) + "w_{ij} \quad (2.22)$$

Langkah 9 : Uji kondisi pemberhentian (akhir iterasi).

2.5.3 Proses Pengujian JST *Backpropagation*

Proses Pengujian arsitektur jaringan syaraf tiruan (ANN) digunakan untuk mengetahui apakah arsitektur jaringan yang telah dilatih telah dapat mengenali pola-pola data, selain data latih, dengan baik atau tidak. Untuk menilai kedekatan atau kecocokan data hasil pemodelan dengan data hasil pengamatan, dilakukan uji kecocokan dengan menggunakan fungsi objektif atau fungsi kesalahan yang merupakan persamaan dari perhitungan dan pengamatan.

a. *Mean Square Error* (MSE)
b. *Root Mean Square Error* (RMSE)
c. Kesalahan *Absolute* Rata-Rata (KAR)

2.6 *Object Oriented Programming* (OOP)

Objek adalah kesatuan entitas yang memiliki sifat dan tingkah laku. Dalam kehidupan sehari-hari, objek adalah benda, baik benda berwujud nyata seperti manusia, hewan, mobil, komputer, handphone, pena, ataupun benda yang tidak nyata atau konsep, seperti halnya tabungan bank, sistem antrian, sistem internet banking, dan sebagainya. Jadi pengertian OOP adalah konsep yang membagi program menjadi objek-objek yang saling berinteraksi satu sama lain. Objek adalah benda, baik benda yang berwujud nyata maupun benda yang tidak nyata (konsep). Jika menggunakan OOP maka akan ada enam keuntungan yang dapat diperoleh, yaitu[18]:

a. Alami
b. Dapat diandalkan
c. Dapat digunakan kembali
d. Mudah untuk dalam perawatan
e. Dapat diperluas
f. Efisiensi waktu

2.6.1 Analisis Berorientasi Objek

Analisis berorientasi objek atau *Object Oriented Analysis* (OOA) adalah tahap untuk menganalisis spesifikasi atau kebutuhan akan sistem yang akan dibangun dengan konsep berorientasi objek, apakah benar kebutuhan yang ada dapat diimplementasikan menjadi sebuah sistem berorientasi objek.

OOA biasanya menggunakan kartu CRC (*Component, Responsibility, Collaborator*) untuk membangun kelas-kelas yang akan digunakan atau menggunakan UML (*Unifed Modeling Language*) pada bagian diagram use case, diagram kelas, dan diagram objek[20].

2.6.2 Desain Berorientasi Objek

Desain berorientasi objek atau *Object Oriented Design* (OOD) adalah tahapan perantara untuk spesifikasi atau kebutuhan sistem yang akan dibangun

dengan konsep berorientasi objek ke desain pemodelan agar lebih mudah diimplementasikan dengan pemograman berorientasikan objek.

Pemodelan berorientasi objek biasanya dituangkan dalam dokumentasi perangkat lunak dengan menggunakan perangkat lunak dengan menggunakan perangkat pemodelan berorientasi objek, diantaranya adalah UML (*Unified Modeling Language*). Kendala dan permasalahan pembangunan sistem berorientasi objek biasanya dapat dikenali dalam tahap ini[19].

2.6.3 *Unified Modeling Language* (UML)

UML adalah bahasa spesifikasi standar untuk mendokumentasikan, menspesifikasikan, dan membangun sebuah sistem. UML adalah himpunan struktur dan teknik untuk pemodelan desain program berorientasi objek (OOP) serta aplikasinya. UML adalah metodologi untuk mengembangkan sistem OOP dan sekelompok perangkat *tool* untuk mendukung pengembangan sistem tersebut. UML mulai diperkenalkan oleh *Object Management Group*, sebuah organisasi yang telah mengembangkan model, teknologi, dan standar OOP sejak tahun 1980-an. Sekarang UML sudah mulai banyak digunakan oleh para praktisi OOP. UML merupakan dasar bagi perangkat (*tool*) desain berorientasi objek dari IBM. UML mendefinisikan diagram-diagram sebagai berikut[20].

1. *Use case diagram*
2. *Activity diagram*
3. *Sequence diagram*
4. *Class diagram*

UML dapat digunakan untuk membuat model untuk semua jenis perangkat lunak, dimana perangkat lunak tersebut dapat berjalan pada piranti keras, sistem operasi dan jaringan apapun, serta ditulis dalam bahasa pemrograman apapun. Tetapi karena UML juga menggunakan class dan operation dalam konsep dasarnya, maka akan lebih cocok untuk penulisan piranti lunak dalam bahasa berorientasi objek.

2.6.4 *Use Case Diagram*

Use case diagram adalah suatu model untuk menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah "apa" yang diperbuat sistem, dan bukan "bagaimana". Sebuah *use case* merepresentasikan sebuah interaksi antara aktor dengan sistem. *Use case* merupakan sebuah pekerjaan tertentu, misalnya *login* ke sistem, meng-*create* sebuah daftar belanja, dan sebagainya. Seorang atau sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu.

Use case diagram dapat sangat membantu menyusun *requirement* sebuah sistem, mengkomunikasikan rancangan dengan klien, dan merancang *test case* untuk semua *feature* yang ada pada sistem. Sebuah *use case* dapat meng-*include* fungsionalitas *use case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa *use case* yang di-*include* akan dipanggil setiap kali *use case* yang meng-*include* dieksekusi secara normal.

Sebuah *use case* dapat di-*include* oleh lebih dari satu *use case* lain, sehingga duplikasi fungsionalitas dapat dihindari dengan cara menarik keluar fungsionalitas yang *common*. Sebuah *use case* juga dapat meng-*extend use case* lain dengan *behaviour*-nya sendiri. Sementara hubungan generalisasi antar *use case* menunjukkan bahwa *use case* yang satu merupakan spesialisasi dari yang lain.

Syarat penamaan pada *use case* adalah nama didefinisikan sesimpel mungkin dan dapat dipahami. Ada dua hal utama pada *use case* yaitu pendefinisian apa yang disebut aktor dan *use case*[20]:

1. Aktor merupakan orang, proses atau sistem lain yang berinteraksi dengan sistem yang akan dibuat. Jadi walaupun simbol aktor dalam diagram *use case* berbentuk orang, namun aktor belum tentu orang.
2. *Use case* merupakan fungsionalitas yang disediakan sistem sebagai unit-unit yang slaing berinteraksi atau bertukar pesan antar unit maupun aktor.

3. Relasi merupakan hubungan yang terjadi pada sistem baik antar aktor maupun antar *use case* maupun anatara usecase dan aktor. Relasi yang digunakan dalam diagram *use case* antara lain.

Berikut merupakan komponen-komponen lain yang terdapat pada *use case:*

a. *Assosiation* merupakan rlasi yang digunakan untuk menggambarkan interaksi antara usecase dan aktor. Asosiasi juga menggambarkan berapa banyak objek lain yang bisa berinteraksi dengan suatu objek atau disebut *multiplicity*.
b. *Generalization* merupakan relasi yang menggambarkan inheritance baik aktor maupun usecase.
c. *Dependency* merupakan relasi yang menggambarkan ketergantungan antara usecase yang satu dengan *usecase* yang lain. Ada dua macam *dependency* yaitu *include* dan *extends*.
d. *Include* menggambarkan bahwa jalannya suatu usecase memicu jalannya usecase lain. Misalnya *usecase* login di *include* oleh *usecase* memilih menu, artinya *usecase* memilih menu akan memicu dijalankannya *usecase login*. Sebelum aktor menjalankan *usecase* memilih menu, aktor harus menjalankan *usecase login* dulu. Dalam penggambaran diagram *usecase*, parah mengarah kepada *usecase* yang di *include*.
e. *Extends* menggambarkan bahwa suatu *usecase* dijalankan karena ada persyaratan tertentu dari *usecase* lain. Misal, dalam sebuah sistem *user* tidak bisa menjalankan *login* sebelum dia mendaftar akun. Dalam diagram *usecase*, *usecase* daftar akun meng *extends usecase login*. Artinya aktor harus menjalankan *usecase* daftar akun dulu sebelum menjalankan *usecase login* karena *usecase login* memiliki syarat aktor yang melakukan *login* harus sudah melakukan pendaftaran akun. Arah panah *dependency* mengarah pada *usecase* yang memiliki syarat.

2.6.5 *Activity Diagram*

Activity diagram menggambarkan berbagai alur aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alur berawal, *decision* yang

mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi.

Activity diagram merupakan *state diagram* khusus, dimana sebagian besar *state* adalah *action* dan sebagian besar transisi di-*trigger* oleh selesainya *state* sebelumnya (*internal processing*). Oleh karena itu *activity diagram* tidak menggambarkan *behaviour* internal sebuah sistem (dan interaksi antar subsistem) secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari *level* atas secara umum[20].

Berikut merupakan komponen-komponen lain yang terdapat pada *activity diagram*:

1. *Activity* atau *state*: Menunjukan aktivitas yang dilakukan.
2. *Initial activity* atau *initial state*: Menunjukan awal aktivitas dimulai.
3. *Final Activity* atau *final state*: Menunjukan bagian akhir dari aktivitas.
4. *Decission*: Digunakan untuk menggambarkan test kondisi untuk memastikan bahwa *control flow* atau *object flow* mengalir lebih ke satu jalur. Jumlah jalur sesuai yang diinginkan.
5. *Merge*: Berfungsi menggabungkan flow yang dipecah oleh decission.
6. *Synchronization*: Dibagi menjadi 2 yaitu *fork* dan *join*. *Fork* digunakan untuk memecah behaviour menjadi activity atau action yang paralel, sedangkan *join* untuk menggabungkan kembali *activity* atau *action* yang paralel.
7. *Swimlanes*: Memecah *activity diagram* menjadi baris dan kolom untuk membagi tangung jawab obyek-obyek yang melakukan aktivitas.
8. *Transition*: Menunjukan aktivitas selanjutnya setelah aktivitas sebelumnya.

2.6.6 *Sequence Diagram*

Sequence diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pemain, *display*, dan sebagainya) berupa *message* yang digambarkan terhadap waktu. *Sequence diagram* terdiri atar dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait).

Sequence diagram biasa digunakan untuk menggambarkan skenario 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[20].

2.6.7 *Class Diagram*

Class diagram merupakan diagram statis. *Class diagram* mewakili pandangan statis dari aplikasi. *Class diagram* tidak hanya digunakan untuk memvisualisasikan, menggambarkan, dan mendokumentasikan perbedaan aspek dari sistem, tetapi juga untuk membangun kode *executable* dari aplikasi *software*.

Class diagram menampilkan koleksi dari *class*, *interface*, *associations*, *collaborations*, dan *constraints*. Hal ini juga dikenal sebagai diagram struktural. *Class diagram* memberikan pandangan dari kelas desain dari sistem. *Class diagram* mengandung representasi dari kelas dan paket dan bagaimana mereka berhubungan[20]. Berikut merupakan komponen-komponen lain yang terdapat pada *class diagram:*

1. *Main class* yaitu kelas yang memiliki fungsi awal dieksekusi ketika sistem dijalankan.
2. *Interface class* merupakan kelas yang mendefinisikan dan mengatur tampilan ke pemakai. Biasanya juga disebut kelas *boundaries*.
3. Kelas yang diambil dari pendefinisian *usecase* merupakan kelas yang menangani fungsi-fungsi yang harus ada dan diambil dari pendefinisian *usecase*.
4. Kelas Entitas merupakan kelas yang digunakan untuk memegang atau membungkus data menjadi sebuah kesatuan yang diambil maupun akan disimpan ke basis data.

Dalam suatu diagram kelas *atribut* dan *method* dapat memiliki salah satu sifat berikut:

a. *Private*: Tidak dapat dipanggil dari luar kelas yang bersangkutan
b. *Protected*: Hanya dapat dipanggil oleh kelas yang bersangkutan dan anak-anak kelas yang mewarisinya.
c. *Public*: Dapat dipanggil oleh siapa saja.

2.7 C# (C *Sharp*)

C# (C *Sharp*) merupakan sebuah bahasa pemrograman yang berorientasi objek yang dikembangkan oleh *Microsoft* sebagai bagian dari inisiatif kerangka .NET Framework. Bahasa pemrograman ini dibuat berbasiskan bahasa C++ yang telah dipengaruhi oleh aspek-aspek ataupun fitur bahasa yang terdapat pada bahasa-bahasa pemrograman lainnya seperti Java, Delphi, Visual Basic, dan lain-lain) dengan beberapa penyederhanaan. Menurut standar ECMA-334 C# *Language Specification*, nama C# terdiri atas sebuah huruf Latin C (U+0043) yang diikuti oleh tanda pagar yang menandakan angka # (U+0023). Tanda pagar # yang digunakan memang bukan tanda kres dalam seni musik (U+266F), dan tanda pagar # (U+0023) tersebut digunakan karena karakter kres dalam seni musik tidak terdapat di dalam *keyboard* standar[21].

2.8 *Black Box*

Metode ujicoba *blackbox* memfokuskan pada keperluan fungsional dari software. Karna itu ujicoba *blackbox* memungkinkan pengembang *software* untuk membuat himpunan kondisi input yang akan melatih seluruh syarat-syarat fungsional suatu program. Ujicoba *blackbox* bukan merupakan alternatif dari ujicoba *whitebox*, tetapi merupakan pendekatan yang melengkapi untuk menemukan kesalahan lainnya, selain menggunakan metode *whitebox*[22].

Uji coba *blackbox* berusaha untuk menemukan kesalahan dalam beberapa kategori, diantaranya:

1. Fungsi-fungsi yang salah atau hilang
2. Kesalahan interface
3. Kesalahan dalam struktur data atau akses database eksternal
4. Kesalahan performa
5. kesalahan inisialisasi dan terminasi

2.9 Pengujian *Confusion Matrix*

Confusion matrix memberikan keputusan yang diperoleh dalam *traning* dan *testing*, *confusion matrix* memberikan penilaian *performance* klasifikasi berdasarkan objek dengan benar atau salah. *Confusion matrix* berisi informasi aktual(*actual*) dan prediksi(*predicted*) pada sistem klasifikasi[23].

Tabel 2.1 *Confusion Matrix* 2 Kelas

Classification	*Predicted Class*		
Observed Class		*Class = Yes*	*Class = No*
	Class = Yes	*A (true positif - tp)*	*B (false negative - fn)*
	Class = No	*C (false positif - fp)*	*D (true negative - tn)*

Keterangan:

a. *True Positive* (tp) = proporsi positif dalam data set yang diklasifikasikan positif.

b. *True Negative* (tn) = proporsi negatif dalam data set yang diklasifikasikan negatif.

c. *False Positive* (fp) = proporsi negatif dalam data set yang diklasifikasikan positif.

d. *False Negative* (fn) = proporsi negatif dalam data set yang diklasifikasikan negatif.

Tabel 2.2 *Confusion Matrix* 3 Kelas

		Predicted Class		
		Class 1	*Class 2*	*Class 3*
Actual Class	*Class 1*	*Class 11*	*Class 12*	*Class 13*
	Class 2	*Class 21*	*Class 22*	*Class 23*
	Class 3	*Class 31*	*Class 32*	*Class 33*

Berikut adalah persamaan model *confusion matrix* untuk 3 kelas:

a. Nilai Akurasi

$$accuracy = \frac{(a+d)}{(a+b+c+d)} \tag{2.22}$$

b. *Error Rate*

$$error\ rate = \frac{(b+c)}{(a+b+c+d)} \tag{2.23}$$