BAB 2
LANDASAN TEORI

2.1 Kecerdasan Buatan

Kecerdasan Buatan atau *artificial intelligence* merupakan salah satu bagian ilmu komputer yang membuat agar mesin (komputer) dapat melakukan pekerjaan seperti dan sebaik yang dilakukan manusia. Pada awal diciptakannya, komputer hanya difungsikan sebagai alat hitung saja. Namun seiring dengan perkembangan jaman, maka peran komputer semakin mendominasi kehidupan umat manusia. Komputer tidak lagi hanya digunakan sebagai alat hitung, lebih dari itu, komputer diharapkan untuk dapat diberdayakan untuk mengerjakan segala sesuatu yang bisa dikerjakan oleh manusia [5].

Manusia bisa menjadi pandai dalam menyelesaikan segala permasalahan di dunia ini karena manusia mempunyai pengetahuan dan pengalaman. Pengetahuan diperoleh dari cara mempelajarinya. Semakin banyak bekal pengetahuan yang dimiliki oleh seseorang tentu saja diharapkan akan lebih mampu dalam menyelesaikan permasalahan. Namun bekal pengetahuan saja tidak cukup, manusia juga diberi akal untuk melakukan penalaran, mengambil kesimpulan berdasarkan pengetahuan dan pengalaman yang mereka miliki. Tanpa memiliki kemampuan untuk menalar dengan baik, manusia dengan segudang pengalaman dan pengetahuan tidak akan dapat menyelesaikan masalah dengan baik. Demikian juga dengan kemampuan menalar yang sangat baik, namun tanpa dibekali pengetahuan dan pengalaman yang memadai, manusia juga tidak akan bisa menyelesaikan masalah dengan baik [5].

Berikut ini beberapa pengertian kecerdasan buatan dari berbagai sudut pandang antara lain [5]:

1. Sudut pandang kecerdasan

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

2. Sudut pandang penelitian

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

   Domain yang sering dibahas para peneliti meliputi:

   a. Mundane task

      1. persepsi (*vision & speech*).
      2. bahasa alami (*understanding, generation & translation*).
      3. pemikiran yang bersifat *common sense* dan robot kontrol.

   b. Formal task

      1. permainan/*games.*
      2. matematika (geometri, logika, kalkulus integral, pembuktian).

   c. Expert task

      1. analisis *financial*.
      2. analisis *medical.*
      3. analisis ilmu pengetahuan.
      4. rekayasa (desain, pencarian kegagalan, perencanaan manufaktur).

3. Sudut pandang bisnis

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

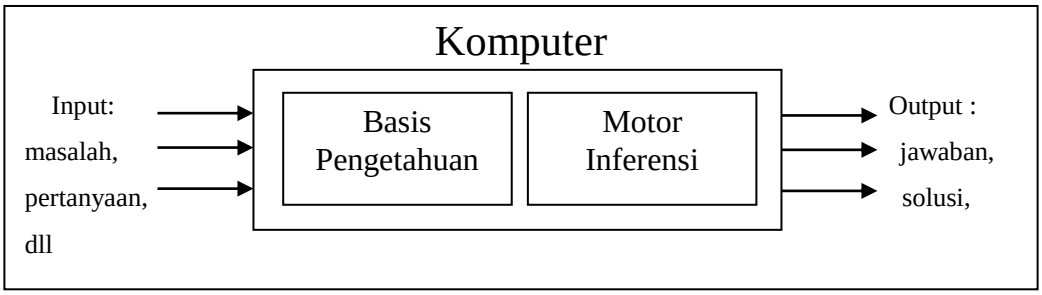
4. Sudut pandang pemrograman

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

Untuk melakukan aplikasi kecerdasan buatan ada 2 bagian utama yang sangat dibutuhkan (Gambar 2.1), yaitu:

a. Basis pengetahuan (*Knowledge Base*), berisi fakta-fakta, teori, pemikiran dan hubungan antara satu dengan yang lainnya.

b. Motor Inferensi (*Inference Engine*), yaitu kemampuan menarik kesimpulan berdasarkan pengalaman.



Gambar 2.1 Penerapan Konsep Kecerdasan Buatan di Komputer [5]

2.1.1 Sejarah Kecerdasan Buatan

Istilah AI (*Artificial Intelligence*) yang dapat diartikan sebagai kecerdasan buatan, pertama kali dikemukakan pada tahun1956 di konferensi Darthmouth. Sejak saat itu, AI terus dikembangkan sebab berbagain penelitian mengenai teori-teori dan prinsip-prinsipnya juga terus berkembang. Meskipun istilah AI baru muncul tahun 1956, tetapi teori-teori yang mengarah ke AI sudah muncul sejak tahun 1941.[6]

Kecerdasan buatan atau *Artificial Intelligence*dimunculkan oleh seorang profesor dari *Massachusetth Institute of Technology* yang bernama John McCharty pada tahun 1956 pada *Dartmouth Conference* yang dihadiri oleh para peneliti AI. Pada konferensi tersebut juga didefinisikan tujuan utama dari kecerdasan buatan, yaitu mengetahui dan memodelkan proses-proses berfikir manusia dan mendesain mesin agar dapat menirukan kelakuan manusia tersebut [5].

Beberapa program AI yang mulai dibuat pada periode 1956-1966 antara lain [5]:

1. *Logic Theorist* diperkenalkan pada *Dartmouth Conference*, program ini untuk membuktikan teorema matematik

2. Sad Sam diprogram oleh Robert K. Lindsay (1960) merupakan program yang dapat mengetahui kalimat sederhana dalam bahasa Inggris dan memberikan jawaban dari fakta yang didengar dalam sebuah percakapan
3. ELIZA diprogram oleh Joseph Weisenaum(1967) adalah program untuk terapi pasien dengan memberikan beberapa pertanyaan

2.1.2 Jaringan Syaraf Tiruan

Jaringan syaraf tiruan merupakan salah satu cabang dari AI (*Artificial Intellegence*), adalah salah satu representasi buatan dari otak manusia yang selalu mencoba untuk mensimulasikan proses pembelajaran pada otak manusia. Istilah buatan ini diimplementasikan dengan menggunakan program komputer yang mampu menyelesaikan sejumlah proses perhitungan selama proses pembelajaran [1].

Jaringan syaraf tiruan merupakan suatu metode komputasi yang meniru sistem jaringan syaraf biologis. Metode ini menggunakan elemen perhitungan *non linier* dasar yang disebut *neuron* yang diorganisasikan sebagai jaringan yang saling berhubungan, sehingga mirip dengan jaringan syaraf manusia. *Neuron* merupakan unit pemroses informasi yang menjadi dasar dalam pengoperasian jaringan syaraf tiruan [7].

Neuron terdiri dari tiga elemen pembentuk [7] , yaitu :

1. Himpunan unit-unit yang dihubungkan dengan jalur koneksi.
2. Suatu unit penjumlah yang akan menjumlahkan sinyal-sinyal *input* yang sudah dikalikan dengan bobotnya.
3. Fungsi aktifasi yang menentukan apakah sinyal dari *input neuron* akan diteruskan ke *neuron* lain atau tidak.

Jaringan syaraf tiruan memiliki beberapa arsitektur jaringan, antara lain [7]:

1. Jaringan Layar Tunggal

   Jaringan dengan lapisan tunggal terdiri dari satu layar *input* dan satu layar *output*. Setiap *neuron* yang terdapat pada layar *input* selalu terhubung dengan *neuron* yang terdapat pada layar *output*.

2. Jaringan Layar Jamak

   Jaringan layar jamak merupakan perluasan dari jaringan layak tunggal. Jaringan dengan lapisan jamak memiliki ciri khas tertentu, yaitu memiliki tiga jenis layar yakni layar *input*, layar *output*, dan layar tersembunyi.

Cara berlangsungnya pembelajaran atau pelatihan jaringan syaraf tiruan dapat dibedakan menjadi dua, yaitu [6]:

1. *Supervised Learning* (pembelajaran terawasi)

   Pada metode *supervised learning*, setiap pola yang diberikan ke dalam suatu jaringan syaraf tiruan telah diketahui *output*-nya. Sejumlah pasangan data (masukan – target keluaran) dipakai untuk melatih jaringan hingga diperoleh bobot yang diinginkan. Pada setiap kali pelatihan, suatu *input* diberikan ke dalam jaringan. Jaringan akan memproses dan mengeluarkan keluaran. Selisih antara keluaran jaringan dengan target (keluaran yang diinginkan) merupakan kesalahan yang terjadi. Kemudian selisih tersebut akan digunakan untuk memodifikasi bobot jaringan syaraf tiruan sehingga mampu menghasilkan *output* sedekat mungkin dengan pola target yang diketahui oleh jaringan.

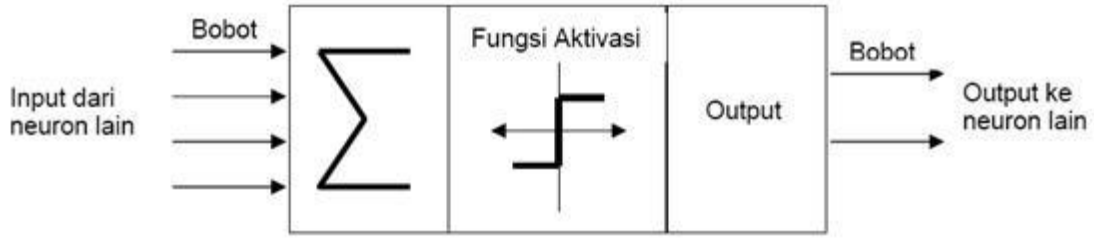
2. *Unsupervised Learning* (pembelajaran tak terawasi)

   Pada metode ini, tidak diperlukan target *output*. Pada metode ini tidak dapat hasil seperti apakah yang diharapkan selama proses pelatihan. Selama proses

pelatihan, perubahan bobot jaringan dilakukan berdasarkan parameter tertentu dan jaringan dimodifikasi menurut parameter tersebut

2.1.3 Komponen Jaringan Syaraf Tiruan

Jaringan syaraf tiruan (JST) terdiri dari beberapa *neuron* yang terhubung satu sama lainnya. *Neuron-neuron* tersebut akan mentransformasikan informasi yang diterima melalui sambungan keluarnya menuju ke *neuron-neuron* yang lain. Pada jaringan syaraf tiruan , hubungan ini dikenal dengan nama bobot. Informasi tersebut disimpan pada suatu nilai tertentu pada bobot tersebut. Gambar 2.2 menunjukan struktur *neuron* pada jaringan syaraf tiruan [5].



Gambar 2.2 Struktur neuron jaringan syaraf tiruan [5]

Pada gambar tersebut dapat dilihat, informasi (disebut dengan : *input*) akan dikirim ke *neuron* dengan bobot kedatangan tertentu. *Input* ini akan diproses oleh suatu fungsi perambatan yang akan menjumlahkan nilai-nilai semua bobot yang datang. Hasil penjumlahan ini kemudian akan dibandingkan dengan suatu nilai ambang (*treshold)* tertentu melalui fungsi aktivasi setiap *neuron*. Apabila *input* tersebut melewati suatu nilai ambang tertentu, maka *neuron* tersebut akan diaktifkan, tapi jika tidak, maka *neuron* tersebut tidak akan diaktifkan. Apabila *neuron* tersebut diaktifkan, maka *neuron* tersebut akan mengirimkan *output* melalui bobot-bobot *output*-nya ke semua *neuron* yang berhubungan dengannya. Demikian seterusnya [5].

2.1.4 Fungsi Aktivasi pada Jaringan Syaraf Tiruan

Ada beberapa fungsi aktivasi yang sering digunakan dalam jaringan syaraf tiruan, antara lain [5] :

1. Fungsi Undak Biner (*Hard Limit*)

   Jaringan dengan lapisan tunggal sering menggunakan fungsi undak (*step function)* untuk mengkonversi *input* dari suatu *variable* yang bernilai kontinu ke suatu *output biner*.

   Fungsi undak biner (*hard limit*) dirumuskan sebagai:

$$Y = \begin{cases} 0, jika\ x \leq 0 \\ 1, jika\ x > 0 \end{cases} \quad (2.1)$$

2. Fungsi Undak Biner (Threshold)

   Fungsi undak biner dengan menggunakan nilai ambang sering juga disebut dengan nama fungsi nilai ambang atau fungsi *Heaviside*.

   Fungsi undak biner (dengan nilai ambang 0) dirumuskan sebagai:

$$Y = \begin{cases} 0, jika\ x < \theta \\ 1, jika\ x \geq \theta \end{cases} \quad (2.2)$$

3. Fungsi Bipolar (*Symetric Hard Limit*)

   Fungsi bipolar sebenarnya hampir sama dengan fungsi undak biner, hanya *output* yang dihasilkan berupa 1, 0 atau -1.

   Fungsi *Symetric Hard Limit* dirumuskan sebagai:

$$Y = \begin{cases} 1, jika\ x > 0 \\ 0, jika\ x = 0 \\ -1, jika\ x < 0 \end{cases} \quad (2.3)$$

4. Fungsi Bipolar (dengan Threshold)

   Fungsi bipolar sebenarnya hampir sama dengan fungsi undak biner dengan *treshold*,hanya saja *output* yang dihasilkan berupa 1, 0 atau -1.

Fungsi bipolar (dengan nilai ambang 0) dirumuskan sebagai:

$$Y = \begin{cases} 1, jika\ x\ \geq \theta \\ -1, jika\ x < \theta \end{cases} \qquad (2.4)$$

5. Fungsi Linear (identitas)

   Fungsi linear memiliki nilai *output* yang sama dengan nilai *input*-nya.

   Fungsi linear dirumuskan sebagai : y = x

6. Fungsi Saturating Linear

   Fungsi ini akan bernilai 0 jika *input*-nya kurang dari -$^1/_2$ dan akan bernilai 1 jika *input*-nya lebih dari $^1/_2$. Sedangkan jika nilai *input* terletak antara -$^1/_2$ dan $^1/_2$, maka *output*-nya akan bernilai sama dengan nilai *input* ditambah $^1/_2$.

   Fungsi *Saturating Linear* dirumuskan sebagai:

$$Y = \begin{cases} 1; & jika\ x \geq 0 \\ x + 0{,}5\ ; & jika - o, 5 \leq x \leq 0{,}5 \\ 0; & jika\ x \leq 0 \end{cases} \qquad (2.5)$$

7. Fungsi Symetric Saturating Linear

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

   Fungsi *Symetric Saturating Linear* dirumuskan sebagai:

$$Y = \begin{cases} 1; & jika\ x \geq 0 \\ x\ ; & jika - 1 \leq x \leq 1 \\ -1; & jika\ x \leq -1 \end{cases} \qquad (2.6)$$

8. Fungsi Sigmoid Biner

   Fungsi ini digunakan untuk jaringan syaraf yang dilatih dengan menggunakan metode *backpropagation*. Fungsi sigmoid biner memiliki

nilai pada range 0 sampai 1. Fungsi ini sering digunakan untuk jaringan syaraf yang membutuhkan nilai *output* yang terletak pada interval 0 sampai 1. Namun, fungsi ini bisa juga digunakan oleh jaringan syaraf yang nilai *output*-nya 0 atau 1.

Fungsi sigmoid biner dirumuskan sebagai berikut:

$$y = f(x) = \frac{1}{(1+e^{-x})} \quad (2.7)$$

Dengan turunan : f'(x) = f(x) (1- f(x))

9. 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:

$$y = f(x) = \frac{1-e^{-x}}{(1+e^{-x})} \quad (2.8)$$

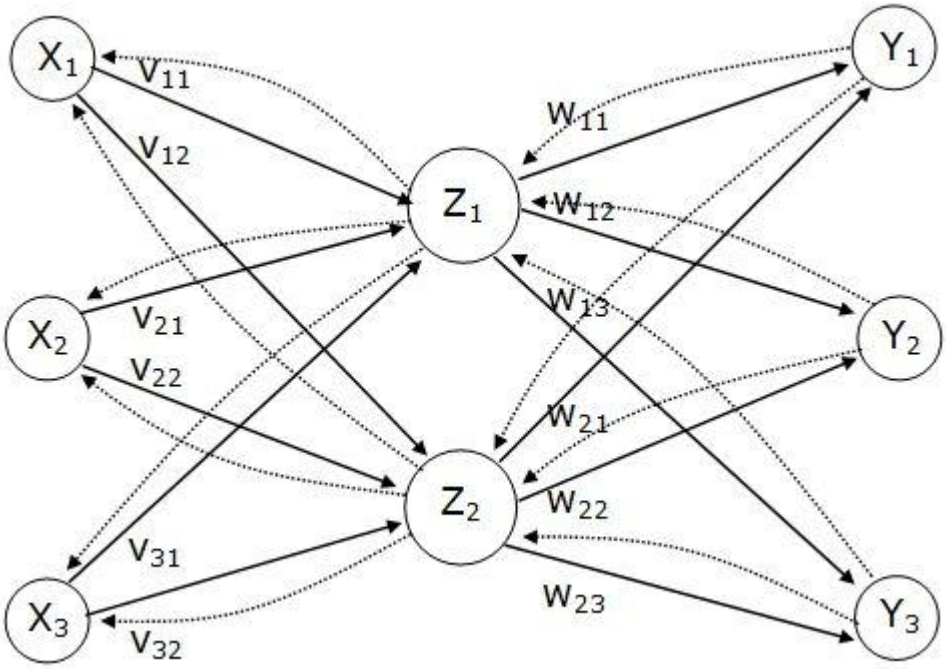
   Dengan turunan : f'(x) = ½ (1+ f(x)) (1- f(x))

2.2 JST Backpropagation

Perambatan galat mundur (*Backpropagation*) merupakan algoritma pembelajaran yang terawasi dan biasanya digunakan oleh perceptron dengan banyak lapisan untuk mengubah bobot-bobot yang terhubung dengan *neuron-neuron* yang ada pada lapisan tersembunyinya. Algoritma *bacpropagation* menggunakan *error output* untuk mengubah nilai bobot-bobotnya dalam arah mundur (*backward).* Untuk mendapatkan *error* ini, tahap perambatan maju (*forward propagation*) harus dikerjakan terlebih dahulu. Pada saat perambatan maju, *neuron-neuron* diaktifkan dengan menggunakan fungsi aktivasi sigmoid, yaitu [5]:

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

Arsitektur jaringan *backpropagation* dapat dilihat pada gambar berikut



Gambar 2.3 Arsitektur Jaringan Backpropagation [5]

2.2.1 Algoritma Pelatihan *Backpropagation*

Pelatihan suatu jaringan dengan algoritma *backpropagation* meliputi dua tahap yaitu perambatan maju dan perambatan mundur.

Selama perambatan maju, tiap unit masukan (x_i) menerima sebuah masukan sinyal ini ke tiap-tiap lapisan tersembunyi $z_1,\ldots..,z_p$. Tiap unit tersembunyi ini kemudian menghitung aktivasinya dan mengirimkan sinyalnya (z_j) ke tiap unit keluaran. Tiap unit keluaran (y_k) menghitung aktivasinya (y_k) untuk membentuk respon pada jaringan untuk memberikan pola masukan [1].

Selama pelatihan, tiap unit keluaran membandingkan perhitungan aktivasinya y_k dengan nilai targetnya t_k untuk menentukan kesalahan pola tersebut dengan unit itu. Berdasarkan kesalahan ini, faktor δ_k *(k = 1,..,m)* dihitung. δ_k digunakan untuk

menyebarkan kesalahan pada unit keluaran y_k kembali ke semua unit pada lapisan sebelumnya (unit-unit tersembunyi yang dihubungkan ke y_k). Juga digunakan (nantinya) untuk meng-*update* bobot-bobot antara keluaran dan lapisan tersembunyi. Dengan cara yang sama, faktor ($j = 1,...,p$) dihitung untuk tiap unit tersembunyi z_j. Tidak perlu untuk menyebarkan kesalahan kembali ke lapisan masukan, tetapi δ_j digunakan untuk meng-*update* bobot-bobot antara lapisan tersembunyi dan lapisan masukan [1].

Setelah seluruh faktor δ ditentukan, bobot untuk semua lapisan diatur secara serentak. Pengaturan bobot w_{jk} (dari unit tersembunyi z_j ke unit keluaran y_k) didasarkan pada faktor δ_k dan aktivasi z_j dari unit tersembunyi z_j. didasarkan pada faktor δ_j dan dan aktivasi x_i unit masukan [1].

2.2.2 Prosedur Pelatihan Algoritma *Backpropagation*

Berikut ini adalah langkah-langkah pelatihan algoritma *backpropagation* [1].

Langkah 0 :

Inisialisasi semua bobot dengan bilangan acak kecil

Langkah 1 :

Ulangi langkah 2 hingga 9 sampai kondisi akhir iterasi dipenuhi

Langkah 2 :

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

Perambatan maju

Langkah 3 :

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

Langkah 4 :

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

$$Z_{in_j} = V_{oj} + \sum_{l=1}^{n} X_i V_{ij} \qquad (2.10)$$

Kemudian menghitung sesuai dengan fungsi aktifasi yang digunakan :

$$Z_j = f(Z_{in_j}) \qquad (2.11)$$

Bila yang digunakan adalah fungsi sigmoid biner maka bentuk fungsi tersebut adalah :

$$Z_j = \frac{1}{1+exp^{-z_in_j}} \qquad (2.12)$$

Langkah 5 :

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

$$Y_{in_k} = W_{ok} + \sum_{j=1}^{p} Z_j W_{jk} \qquad (2.13)$$

Menghitung kembali sesuai dengan fungsi aktifasi

$$Y_k = f(y_{in_k}) \qquad (2.14)$$

Bila yang digunakan adalah fungsi sigmoid biner maka bentuk fungsi tersebut adalah :

$$Y_k = \frac{1}{1+exp^{-y_in_k}} \qquad (2.15)$$

Perambatan mundur

Langkah 6 :

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

$$\delta_k = (t_k - y_k) f'(y_in_k) \qquad (2.16)$$

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

$$f'(y_in_k) = f(y_in_k)(1 - f(y_in_k)) \quad (2.17)$$

$$= y_k(1 - y_k) \quad (2.18)$$

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

$$\Delta W_{kj} = \alpha . \partial_k . Z_j \quad (2.19)$$

Menghitung perbaikan koreksi :

$$\Delta W_{0j} = \alpha . \partial_k \quad (2.20)$$

Dan menggunakan nilai δ_k pada semua unit lapisan sebelumnya.

Langkah 7 :

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.

$$\delta_in_j = \sum_{k=1}^{n} \delta_k W_{jk} \quad (2.21)$$

Selanjutnya dikalikan dengan turunan dari fungsi aktifasinya untuk menghitung galat.

$$\delta_j = \delta_in_j f'(z_in_j) \quad (2.22)$$

Kemudian menghitung perbaikan penimbang (digunakan untuk memperbaiki V_{ij})

$$\Delta V_{ij} = \alpha \delta_j X_i \quad (2.23)$$

Kemudian menghitung perbaikan bias (untuk memperbaiki V_{oj})

$$\Delta V_{oj} = \alpha \delta_j \quad (2.24)$$

Memperbaiki penimbang dan bias

Langkah 8 :

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

$$W_{jk}(baru) = W_{jk}(lama) + \Delta W_{jk} \quad (2.25)$$

$$W_{0j}(baru) = W_{0j}(lama) + \Delta W_{0j} \quad (2.26)$$

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

$$V_{jk}(baru) = V_{jk}(lama) + \Delta V_{jk} \quad (2.27)$$

$$V_{0j}(baru) = V_{0j}(lama) + \Delta V_{0j} \quad (2.28)$$

Langkah 9 :

Uji kondisi pemberhentian (akhir iterasi)

Daftar Notasi

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
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 niloai penimbang sambungan pada lapisan tersembunyi
α	=	Konstanta laju pelatihan (learning rate) 0<α<1
E	=	Total galat

2.2.3 Prosedur Pengujian Algoritma *Backpropagation*

Setelah pelatihan, jaringan saraf *backpropagation* diaplikasikan dengan hanya menggunakan tahap perambatan maju dari algoritma pelatihan. Prosedur aplikasinya adalah sebagai berikut [1]:

Langkah 0: Inisialisasi bobot (dari algoritma pelatihan).

Langkah 1: Untuk tiap vektor masukan, lakukan langkah 2-4.

Langkah 2: for i = 1,…, n : atur aktivasi unit masukan x_i .

Langkah 3: for j = 1,…, p :

$$Z_{in_j} = V_{oj} + \sum_{I=1}^{n} X_i V_{ij} \quad (2.10)$$

$$Z_j = f(Z_{in_j}) \quad (2.11)$$

Langkah 4: for k = 1,…, m :

$$Y_{in_k} = W_{ok} + \sum_{j=1}^{p} Z_j W_{jk} \quad (2.13)$$

$$y_k = f(y_{in_k}) \quad (2.14)$$

Langkah 5: Jika $y_k \geq 0{,}5$ maka $y_k = 1$, else $y_k = 0$.

2.3 Penyakit Mata Menular

Penyakit mata menular adalah penyakit mata yang biasanya timbul akibat infeksi virus dan bakteri sehingga penyakit tersebut bisa menular kepada orang lain yang berhubungan dengan orang yang sedang menderita penyakit mata menular ini. Terdapat banyak jenis dari penyakit mata menular yang di derita oleh sebagian masyarakat Indonesia. Beberapa diantaranya adalah jenis penyakit *Konjungtivitis, Keratokonjutivitis Vernalis, Endoftalmitis, Selulitis Orbitalis, Trakoma, Blefaritis,* dan *Ulkus Kornea.* Penyakit tersebut memiliki gejala dan ciri-ciri yang hampir mirip.

Pembagian jenis penyakit mata menular beserta gejala yang dialami oleh penderita dapat dilihat pada tabel 2.1 berikut.

Tabel 2.1 Penyakit Mata Menular dan Gejalanya

Jenis Penyakit mata menular	Gejala
Konjungtivitis	Mata merah, mata terasa perih, mata terasa gatal, mata berair, mata mengeluarkan kotoran(belekan), penglihatan kabur
Keratokonjungtivitis vernalis	Sering mengalami mata merah, mata berair, mata terasa gatal, kelopak mata membengkak, mata mengeluarkan kotoran(belekan)
Endoftalmitis	Mata merah, mata terasa perih, mengalami gangguan pada penglihatan(penglihatan berkurang atau penglihatan kabur), demam, sakit kepala, mata menjadi sensitif terhadap cahaya
Selulitis Orbitalis	Mata merah, mata terasa perih, kelopak mata membengkak, bola mata menonjol dan membengkak, demam
Trakoma	Mata merah, mata mengeluarkan kotoran(belekan), kelopak mata membengkak, kelenjar getah bening membengkak, kornea terlihat keruh
Blefaritis	Mata merah, mata terasa panas, mata terasa perih, mata terasa gatal, mata berair, terdapat luka pada kelopak mata, kelopak mata membengkak,bulu mata rontok
Ulkus Kornea	Mata merah, mata terasa perih, mata terasa gatal, mata berair, mata mengeluarkan kotoran(belekan), terdapat bintik nanah berwarna kuning keputihan pada kornea, mengalami gangguan pada penglihatan(penglihatan kabur), sensitif terhadap cahaya

2.4 Algoritma

Algoritma adalah urutan langkah-langkah untuk menyelesaikan suatu masalah. Terdapat beberapa definisi lain dari algoritma, tetapi pada prinsipnya senada dengan definisi yang diungkapkan. Definisi lainnya antara lain [12] :

1. Algoritma adalah deretan langkah-langkah komputasi yang mentransformasikan data masukan menjadi keluaran.
2. Algoritma adalah deretan instruksi yang jelas untuk memecahkan masalah, yaitu memperoleh keluaran yang diinginkan dari suatu masukan dalam jumlah waktu yang terbatas.
3. Algoritma adalah prosedur komputasi yang terdefinisi dengan baik yang menggunakan beberapa nilai sebagai masukan dan menghasilkan beberapa nilai yang disebut keluaran

Terdapat 3 macam cara penulisan algoritma atau disebut notasi algoritmik. Berikut ini adalah ketiga notasi algoritmik [12]:

1. Notasi I : menyatakan langkah-langkah algoritma dengan untaian kalimat deskriptif.
2. Notasi II : menggunakan bagan-alir (*flowchart*).
3. Notasi III : menggunakan *pseudo-code.*

Sebuah algoritma tidak hanya harus benar, tetapi juga harus efisien. Ukuran sebuah algoritma dikatakan efisien yaitu melalui waktu dan ruang memori (*space*). Algoritma yang efisien adalah algoritma yang dapat meminimumkan kebutuhan waktu dan ruang. Untuk itu, diperlukan analisis algoritma untuk dapat mengetahui performansi, keakuratan, optimasi serta efisiensi sebuah algoritma. Setiap algoritma memiliki nilai performansi yang berbeda-beda. Performansi suatu algoritma dapat diukur dengan menghitung nilai kompleksitas waktunya dimana setiap langkahnya akan dihitung.

2.4.1 Kompleksitas Algoritma

Umumnya Algoritma mempunyai parameter utama yang paling menentukan waktu proses. Parameter ini berkaitan dengan jumlah data yang diproses algoritma tersebut. Parameter ini dapat berupa pangkat suatu persamaan polinom atau jumlah *record* yang akan diurutkan atau dicari. Kompleksitas algoritma menyatakan kelajuan perkembangan waktu proses dibandingkan dengan kelajuan perkembangan data yang diproses [13]. Salah satu notasi kompleksitas algoritma adalah *Big-Oh*. Definisi dari *Big-O* Adalah fungsi yang lebih berkaitan dengan kelajuan proses daripada kelajuan pertambahan data yang dirumuskan dengan T(n) = O(f(n)). Jika ada konstan c dan n_o sedemikian rupa sehingga $T(n) \leq c.\ f(n)$ untuk $n \geq n_o$. Secara sederhana dikatakan bahwa O(f(n)) dapat dianggap sebagai nilai maksimum dari c.f(n).

Contoh :

Jika $f(n) = 2n^2$ maka $T(n) = O(n^2)$

Jika $f(n) = 4n^2 + 7n + 3$ maka $T(n) = O(n^2)$

Contoh perhitungan 1

```
Sum = 0;
For (i=0; i<n; i++)
        Sum = sum + a[i];
```

Perhitungan eksekusi :

sum = 0	dieksekusi 1 kali
i = 0	dieksekusi 1 kali
i < n	dieksekusi n kali
i++	dieksekusi n kali
sum = sum +a[i]	dieksekusi n kali
	2 + 3n kali

Jadi **T(n) = *O*(n)**

Contoh perhitungan 2

```
Sum = 0;
For (i=0; i<n; i++)
        For (j=0; j<n; j++)
                Sum = sum + a[i];
```

Perhitungan eksekusi :

sum = 0	dieksekusi 1	kali
i = 0	dieksekusi 1	kali
j = 0	dieksekusi n	kali
i < n	dieksekusi n	kali
i++	dieksekusi n	kali
j < n	dieksekusi n x n	kali
j++	dieksekusi n x n	kali
sum = sum +a[i]	dieksekusi n x n	kali
	$2 + 3n + 3n^2$	kali

Jadi **$T(n) = O(n^2)$**

Tiap-tiap algoritma mempunyai kompleksitas waktu asimptotik masing-masing. Kompleksitas waktu asimptotik ini dapat digunakan untuk mengelompokan algoritma [14].

Urutan spektrum kompleksitas waktu algoritma adalah [14]:

$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2)\ O(n^3) < \ldots < O(2^n) < O(n!)$

algoritma polinomial — algortima eksponensial

Pembagian kelompok algoritma berdasarkan kompleksitas waktu asimptotik dapat dilihat pada tabel 2.2 berikut.

Tabel 2.2 Kelompok Algoritma Berdasarkan Kompleksitas Waktu Asimptotik [14]

Kelompok Algoritma	Nama
O(1)	Konstan
O(log n)	Logaritmik
O(n)	Lanjar
O(n log n)	n log n
$O(n^2)$	Kuadratik
$O(n^3)$	Kubik
$O(2^n)$	Eksponensial
O(n!)	Faktorial

Penjelasan masing-masing kelompok algoritma adalah sebagai berikut [14]:

O(1) Kompleksitas *O*(1) berarti waktu pelaksanaan algoritma adalah tetap, tidak bergantung pada ukuran masukan. Contohnya prosedur tukar di bawah ini:

```
procedure Tukar(input/output a, b : integer)
{ Mempertukarkan nilai a dan b. Setelah pertukaran, a berisi nilai b dan b
  berisi nilai a semula.
  Masukan : a dan b
  Keluaran : a dan b
}
Deklarasi
   temp: integer
Algoritma
   temp ← a
   a ← b
```

b ← temp

Di sini jumlah operasi penugasan (*assignment*) ada tiga buah dan tiap operasi dilakukan satu kali. Jadi, $T(n) = 3 = O(1)$.

O(log n) Kompleksitas waktu logaritmik berarti laju pertumbuhan waktunya berjalan lebih lambat daripada pertumbuhan *n*. Algoritma yang termasuk kelompok ini adalah algoritma yang memecahkan persoalan besar dengan mentransformasikannya menjadi beberapa persoalan yang lebih kecil yang berukuran sama (misalnya algoritma pencarian_biner). Di sini basis algoritma tidak terlalu penting sebab bila *n* dinaikkan dua kali semula, misalnya, log *n* meningkat sebesar sejumlah konstanta. Fungsi log *n* hanya meningkat menjadi dua kali semula jika *n* dinaikan sebesar n^2 kali semula(basis dua).

O(n) Algoritma yang waktu pelaksanaannya lanjar umumnya terdapat pada kasus yang setiap elemen masukannya dikenai proses yang sama, misalnya algoritma pencarian_beruntun. Bila *n* dijadikan dua kali semula, maka waktu pelaksanaan algoritma juga dua kali semula.

O(n log n) Waktu pelaksanaan yang *n* log *n* terdapat pada algoritma yang memecahkan persoalan menjadi beberapa persoalan yang lebih kecil, menyelesaikan tiap persoalan secara independen, dan menggabung solusi masing-masing persoalan. Algoritma yang diselesaikan dengan teknik bagi dan gabung mempunyai kompleksitas asimptotik jenis ini. Bila *n* = 1000, maka *n* log *n* mungkin 20.000. Bila *n* dijadikan dua kali semual, maka *n* log *n* menjadi dua kali semula (tetapi tidak terlalu banyak).

$O(n^2)$ Algoritma yang waktu pelaksanaannya kuadratik hanya praktis digunakan untuk persoalan yang berukuran kecil. Umumnya algoritma yang termasuk kelompok ini memproses setiap masukan dalam dua buah kalang bersarang, misalnya pada algoritma urut_maks. Bila *n* = 1000, maka waktu pelaksanaan algoritma adalah 1.000.000. Bila *n* dinaikkan menjadi dua kali semula, maka waktu pelaksanaan algoritma meningkat menjadi empat kali semula.

$O(n^3)$ Seperti halnya algoritma kuadratik, algoritma kubik memproses setiap masukan dalam tiga buah kalang bersarang, misalnya algoritma perkalian matriks. Bila *n* = 100, maka waktu pelaksanaan algoritma adalah 1.000.000. Bila *n* dinaikkan menjadi dua kali semula, waktu pelaksanan algoritma meningkat menjadi delapan kali semula

$O(2^n)$ Algoritma yang tergolong kelompok ini mencari solusi persoalan secara "*brute force*", misalnya pada algoritma mencari sirkuit Hamilton. Bila *n* = 20, waktu pelaksanaan algoritma adalah 1.000.000. Bila *n* dijadikan dua kali semula, waktu pelaksanaan menjadi kuadrat kali semula!

O(n!) Seperti halnya pada algoritma eksponensial, algoritma jenis ini memproses setiap masukan dan menghubungkannya dengan *n* - 1 masukan lainnya, misalnya algoritma Persoalan Pedagang Keliling (*Travelling Salesperson Problem*). Bila *n* = 5, maka waktu pelaksanaan algoritma adalah 120. Bila *n* dijadikan dua kali semula, maka waktu pelaksanaan algoritma menjadi faktorial dari 2*n*.

2.5 Cross Validation

Untuk tujuan replikasi, yang dibutuhkan adalah mendapatkan jumlah dari data latih dan pasangan data tes dari *dataset* X (setelah lebih dahulu memisahkan sebagian

data sebagai data uji). Untuk mendapatkannya, jika sampel X cukup besar, kita dapat membaginya secara acak kedalam K bagian, lalu secara acak memisahkan masing-masing bagian ke dalam 2 kelompok dan menggunakan setengah data pertama untuk data latih dan setengah data lainnya untuk data uji. Pengujian ini dilakukan berulang dengan menggunakan data *split* yang berbeda, dinamakan *cross validation.*

K-fold cross-validation adalah sebuah pengujian dimana *dataset* X dibagi secara acak kedalam K bagian dengan ukuran yang sama, Xi, i=1,...,K. Untuk memproses tiap pasangnya, diambil terlebih dahulu salah satu bagian dari K bagian sebagai data tes, dan mengkombinasikan sisanya sebanyak K-1 bagian sebagai data latih. Proses pengujian dilakukan sebanyak K kali [17].

Dalam contoh kasus yang memiliki banyak prediksi, hasil pada data uji sering ditampilkan sebagai *confusion matrix* dua dimensi dengan baris dan kolom untuk setiap kelas. setiap elemen *matrix* menunjukkan jumlah data uji. Pada bagian baris adalah kelas data uji yang sebenarnya dan pada bagian kolom adalah kelas hasil prediksi [15]. Contoh *confusion matrix* pada dapat dilihat pada tabel 2.3.

Tabel 2.3 Contoh Confusion Matrix

	Hasil Identifikasi			
Target Identifikasi	a	b	c	d
a	4	0	0	0
b	1	3	0	0
c	1	0	2	1
d	0	0	0	4

Berikut ini adalah penjelasan pada tabel 2.3

Target a teridentifikasi menjadi a sebanyak 4
teridentifikasi menjadi b sebanyak 0
teridentifikasi menjadi c sebanyak 0

	teridentifikasi menjadi d sebanyak 0
Target b	teridentifikasi menjadi a sebanyak 1
	teridentifikasi menjadi b sebanyak 3
	teridentifikasi menjadi c sebanyak 0
	teridentifikasi menjadi d sebanyak 0
Target c	teridentifikasi menjadi a sebanyak 1
	teridentifikasi menjadi b sebanyak 0
	teridentifikasi menjadi c sebanyak 2
	teridentifikasi menjadi d sebanyak 1
Target d	teridentifikasi menjadi a sebanyak 0
	teridentifikasi menjadi b sebanyak 0
	teridentifikasi menjadi c sebanyak 0
	teridentifikasi menjadi d sebanyak 4