BAB 2

LANDASAN TEORI

2.1 Teori-teori Dasar/Umum

2.1.1 *Neural Network*

Neural Network merupakan suatu metode *Artificial Intelligence* yang konsepnya meniru sistem jaringan syaraf yang ada pada tubuh manusia, dimana dibangun *node – node* yang saling berhubungan satu dengan yang lainnya. *Node – node* tersebut terhubung melalui suatu *link* yang biasa disebut dengan istilah *weight*. Ide dasarnya adalah mengadopsi cara kerja otak manusia yang memiliki ciri – ciri *pararel processing*, *processing element* dalam jumlah besar dan *fault tolerance*.

Menurut *Haykin (1999,p2)*, jaringan syaraf tiruan (*Artificial Neural Network*) adalah sejumlah besar prosesor yang terdistribusi secara pararel dan terdiri dari unit pemrosesan sederhana, dimana masing – masing unit memiliki kecenderungan untuk menyimpan pengetahuan yang dialami dan dapat digunakan kembali.

Jaringan syaraf tiruan menyerupai otak manusia dengan dua cara :

a) Pengetahuan yang diperoleh jaringan dari lingkungannya melalui proses pembelajaran.

b) Kekuatan hubungan antar neuron, dikenal dengan istilah *synaptic weights*, dan digunakan untuk menyimpan pengetahuan yang diperoleh.

Neural Network sendiri pun dibagi – bagi kembali menjadi beberapa bagian yang lebih kecil, dimana masing – masing metode mempunyai karakteristiknya sendiri – sendiri, serta memiliki keunggulan dan kelemahan dalam mengenali suatu pola. Metode – metode tersebut diantaranya adalah : *Back Propagation* (yang dibahas dalam skripsi ini), *Bidirectional Assosiate Memory* atau lebih dikenal dengan istilah BAM, *Hopfield Network*, *Counter Propagation Network* dan masih banyak metode – metode lainnya yang sudah atau sedang dikembangkan oleh para ahli.

Pada umumnya *neural network* dibagi berdasarkan *layer – layer* yaitu *input layer*, *hidden layer* dan *output layer*. Setiap *node* pada masing – masing *layer* memiliki suatu *error rate*, yang akan digunakan untuk proses *training*. N*eural network* dengan *layer – layer* memiliki konsep kerja sebagai berikut : *input layer* menunggu user memasukan input ke masing – masing *node* nya, setelah masing – masing *node* di *input layer* memperoleh data yang dibutuhkan maka akan dikalikan dengan *weight*-nya menghasilkan *sum* (jumlah) atau yang lebih dikenal dengan akumulator dengan rumus $NET = O_1 W_1 + O_2 W_2 + ... + O_n W_n = \Sigma O_i W_{i,}$ lalu akumulator tersebut akan dimasukan kedalam Fungsi Aktivasi yang digunakan, rumusnya adalah $OUT = F (NET)$, untuk *Back Propagation* umumnya menggunakan fungsi *sigmoid biner* dan fungsi *sigmoid bipolar*.

Pada kenyataannya (kebiasaannya), kebanyakan *neural system* harus diajari (*training*). Mereka akan mempelajari asosiasi, *patterns*, dan fungsi yang baru. Pemakai-pemakai *neural network* tidak menspesifikasikan sebuah algoritma untuk dieksekusi dalam setiap perhitungan. Mereka akan memilih arsitektur tertentu dengan pandangan mereka, dengan karakteristik *neuron*, *weight*, dan memilih model *training* sendiri. Sehingga dari hasil tersebut, informasi *network* dapat diubah oleh para pemakai. *Artificial Neural System* juga dapat mengkalkulasi teknik matematik, seperti minimalisasi kesalahan suatu perhitungan.

Neural network sangat berperan dalam teknologi dan beberapa disiplin ilmu, yang membantu dalam menentukan model-model *neural network* dan *system non-linear dynamic*. Matematika adalah model neural yang paling berpotensi karena kekompleksan-nya. Elektronika dan ilmu komputer juga menggunakan metode ini, karena berperan dalam pengiriman sinyal data.

2.1.2 *Artifical Neuron*

Neuron adalah unit pemrosesan informasi yang merupakan dasar dari operasi jaringan syaraf tiruan. Sel – sel syaraf tiruan ini dirancang berdasarkan sifat – sifat dari neuron biologis. Sel syaraf tiruan ini biasa disebut juga sebagai *processing elements,* unit atau *node*.

2.1.3 *Weight*, *Output* dan *Error*

Hubungan antar *node* diasosiasikan dengan suatu nilai yang disebut dengan bobot atau *weight*. Setiap *node* pasti memiliki output, *error* dan *weight*-nya masing - masing.

Output merupakan keluaran dari suatu *node*. *Error* merupakan tingkat kesalahan yang terdapat dalam suatu *node* dari proses yang dilakukan. *Weight* merupakan bobot dari *node* tersebut ke *node* yang lain pada *layer* yang berbeda. Nilai *weight* berkisar antara -1 dan 1.

Bobot – bobot atau *weight* yang tersimpan di dalam jaringan syaraf tiruan ini disebut sebagai bobot interkoneksi. Nilai bobot yang baik akan memberikan keluaran yang sesuai, dalam arti mendekati keluaran yang diharapkan (*target output*) untuk suatu input yang diberikan.

Bobot awal dalam suatu jaringan syaraf tiruan biasanya diperoleh secara *random* dan sebaiknya di inisialisasi dengan nilai yang relatif kecil, yaitu berkisar antara -0,1 sampai 0,1 *(Mitchell, 1997, p108)*. Baru dalam tahap pelatihan, bobot tersebut akan mengalami penyesuaian melalui suatu proses perhitungan matematik agar tercapai nilai bobot yang sesuai.

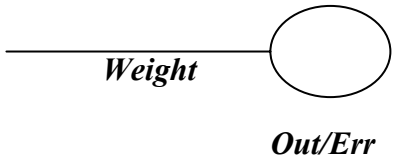
2.1.4 *Neural System*

Pada dasarnya untuk membentuk suatu sistem neural, hanya diperlukan 3 tahap, yaitu *forward phase*, *backward propagation*, dan *update weight*.

a. *Node*

Node adalah sebuah sel neuron yang di setiap *node*nya memiliki output, *error*, dan *weight*. Jadi di setiap *node*, dimanapun itu pasti memiliki ketiga unsur tersebut.

Output merupakan keluaran (hasil) dari suatu *node*. *Error* merupakan tingkat kesalahan yang terdapat dalam suatu *node* dari proses yang ia lakukan. Sedangkan *weight* merupakan berat dari *node* tersebut ke *node* yang lain (beda *layer*), besarnya *weight* adalah berkisar antara –1 sampai dengan 1.



Gambar 2.1 *Node*

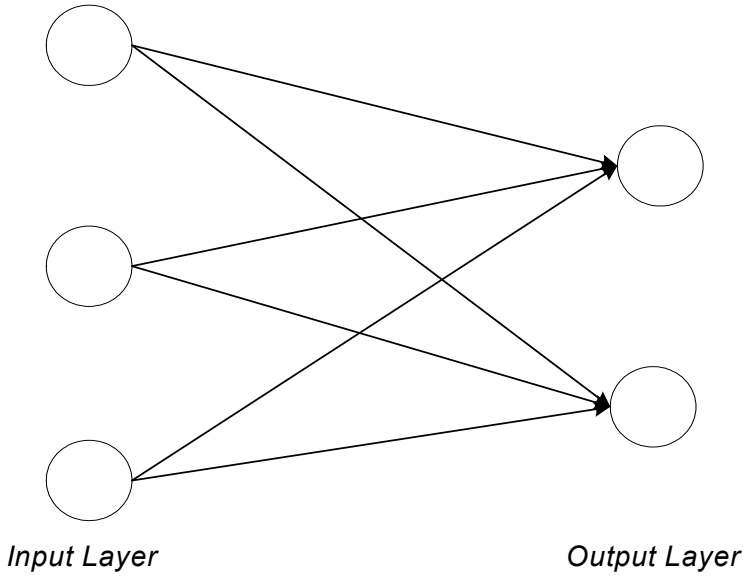
b. *Input* dan *Hidden Layer*

Input layer merupakan *layer* tempat sebuah input dimasukkan (inisialisasi input), dan dari *layer* ini dilakukan proses-proses selanjutnya.

Hidden layer disini berfungsi untuk membantu proses, semakin banyak *hidden layer* yang digunakan maka semakin bagus dan semakin cepat pula didapat output yang diinginkan, tetapi waktu *training* akan semakin lama *(Mitchell, 1997, p115)*.

c. *Forward Propagation*

Forward propagation bertujuan untuk menentukan output dari suatu *node*. Output yang dimaksud di sini adalah output dari *output layer*. Karena masing-masing *node* tersebut memiliki output.



Gambar 2.2 ***Neural Network***

d. *Output Layer*

Output layer adalah *layer* yang menampung hasil proses dari suatu *neural network*. *Forward propagation* dilakukan untuk mencari *error* di *output layer*, seperti yang terlihat dalam **Gambar 2.1**.

e. *Training*

Proses belajar suatu *neural network* terdiri dari proses *Forward*, *Backward*, dan *Update Weight*. Sekali melewati 3 tahap itu disebut dengan 1 kali *training* (1 *cycle*).

Semakin banyak *training* yang dilakukan maka akan semakin kecil pula tingkat *erro*r yang dihasilkan di *output layer*-nya, dan dengan demikian semakin kecil juga *error* suatu sistem.

Menurut *Rao (1995, p5)*, ada dua metode *learning* dalam *neural network*, yaitu :

1. *Supervised Learning*

Supervised Learning adalah suatu metode dimana *neural network* belajar dari pasangan data *input* dan *target*, pasangan ini disebut *training pair*. Biasanya jaringan dilatih dengan sejumlah *training pair*, dimana suatu input vektor diaplikasikan, menghasilkan nilai di output, lalu hasil di output tersebut akan dibandingkan dengan target output. Selisihnya akan dikembalikan ke jaringan, dihitung *error*-nya, melalui *error* ini akan didapatkan selisih yang terdapat di *weight*-nya. Maka itu terdapat *weight* baru yang cenderung memiliki *error* yang lebih kecil, jadi akan didapat *error* yang lebih minimum dari *error* yang pertama. Vektor – vektor dalam *training set* diaplikasikan seluruhnya secara berurutan, *error* dihitung, *weight* disesuaikan sampai seluruh *training set* menghasilkan *error* yang sekecil – kecilnya. Sebenarnya konsep ini belajar dengan menggunakan konsep *human brain*.

Model *Neural Network* yang menggunakan metode *supervised learning* diantaranya adalah sebagai berikut :

a. Model *Back Propagation*

b. Model *Biderectional Associative Memory*

c. *Hopfield Network*

2. *Non-Supervised (Unsupervised) Learning*

Unsupervised Learning dianggap sebagai model dalam konsep sistem biologis. Teori ini dikembangankan oleh *Kohonen (1984)* dan beberapa ilmuwan lainnya. Dalam *unsupervised learning* tidak diperlukan target output, *training* hanya terdiri dari vektor – vektor input, tanpa pasangan target. Algoritma *training* merubah *weight* jaringan untuk menghasilkan output yang konsisten. Aplikasi dari vektor – vektor yang cukup serupa akan menghasilkan pola output yang sama. Dengan demikian proses *training* akan menghasilkan sifat – sifat statistik dalam bentuk pengelompokan vektor – vektor dalam beberapa kelas. Dengan mengaplikasikan suatu vektor dari suatu kelas sebagai input akan menghasilkan vektor output yang spesifik.

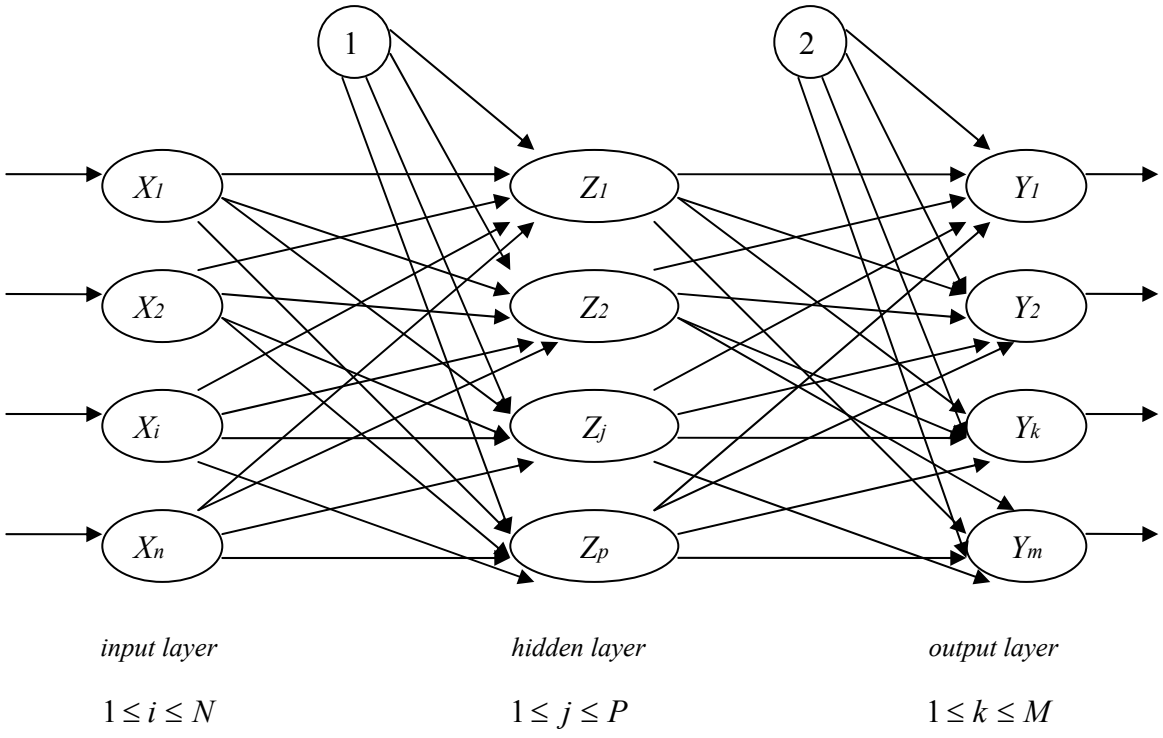
2.1.5 *Multi Layer Neural Network*

a. Pengertian *Multi Layer Neural Network*

Multi Layer Neural Network adalah *neural network* yang memiliki karakteristik *multi layer* dimana setiap *node* pada suatu *layer*

terhubung dengan setiap *node* pada *layer* di depannya. Ber-arsitektur umpan maju atau (*feed forward network*), menggunakan metode *supervised learning*.

Model ini merupakan model yang paling sering dipakai dalam pengembangan sistem neural dan memiliki kinerja yang sangat baik dalam sisi keakuratan. Model ini mempunyai dua fase dalam pelatihannya yaitu fase *forward* dan *backward*. Cara kerja jaringan ini adalah, setelah input masuk ke *input layer* maka data akan diolah dan diteruskan ke masing – masing bagian di depannya sampai ke *output layer*. Nilai di *output layer* akan dibandingkan dengan nilai target, lalu akan dihasilkan sinyal *error* bagi masing *node* di *output layer*. Sinyal ini ditransmisikan balik (*back propagation*) dari lapisan keluaran ke masing – masing sel pada lapisan sebelumnya.



Gambar 2.3 *Multi Layer Network*

Umumnya operasi model jaringan ini terdapat dua mekanisme kerja yaitu :

1) Mekanisme latihan atau belajar (*Training mode* / *Learning Mode*)

   Pada mekanisme ini, jaringan akan dilatih untuk dapat menghasilkan data sesuai dengan target yang diharapkan melalui satu atau lebih pasangan pasangan data (data input dan data target). Semakin lama waktu latihan maka kinerja jaringan akan semakin baik. Demikian juga dengan semakin banyak pasangan data yang digunakan dalam pelatihan maka kinerja akan semakin baik.

2) Mekanisme produksi (*Production Mode*) atau biasa disebut juga dengan mekanisme pengujian (*Try Out Mode*)

   Pada mekanisme ini, jaringan diuji apakah dapat mengenali sesuai dengan yang diharapkan setelah melalui proses pelatihan terlebih dahulu.

b. *Back Propagation*

Inti dari *back propagation* adalah untuk mencari *error* suatu *node***.** Dari hasil *forward phase* akan dihasilkan suatu output, dari output tersebut, pastilah tidak sesuai target yang diinginkan. Perbandingan kesalahan dari target yang diinginkan dengan output yang dihasilkan disebut dengan *error*.

Dalam *Back Propagation* juga dikenal istilah yang disebut inisialisasi output**.** Inisialisasi output pada dasarnya adalah menentukan

error di suatu *node* dengan sebuah target yang diinginkan. Karakteristik *Back Propagation* dapat diuraikan sebagai berikut :

1. *Node* / *processing element* dan fungsi aktivasi

   a) Kontinu.

   b) Dapat dideferensiasikan / diteruskan.

   c) Turunan fungsi mudah dihitung.

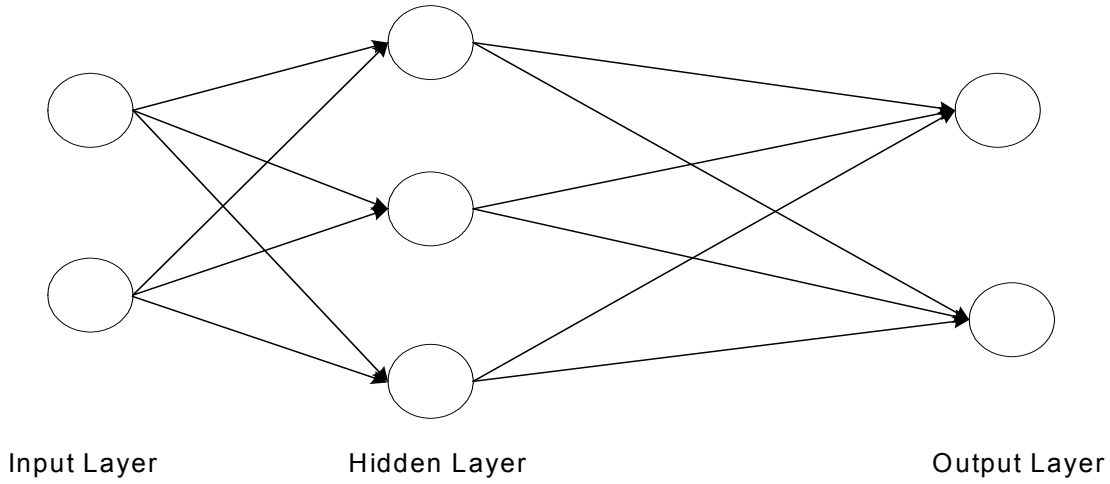
   d) Fungsi aktivasi yang biasa digunakan adalah fungsi sigmoid.

2. *Topology*

   Terdiri dari satu lapisan masukan (*input layer*), satu atau lebih lapisan tersembunyi (*hidden layer*), dan satu lapisan keluaran (*output layer*). Setiap neuron / *processing element* pada suatu lapisan mendapat sinyal masukkan dari semua neuron pada lapisan sebelumnya (beserta sinyal bias).

3) *Learning Rule*

   Menggunakan *delta rule* atau *error connection learning rule*.



Gambar 2.4 *Multi Layer Network* dengan 1 *Hidden Layer*

Contoh Perhitungan dengan metode *Back Propagation*

Weight yang terdapat pada gambar merupakan *weight* dari *output layer* Untuk mencari output yang terdapat pada *output layer* dapat dicari dengan rumus :

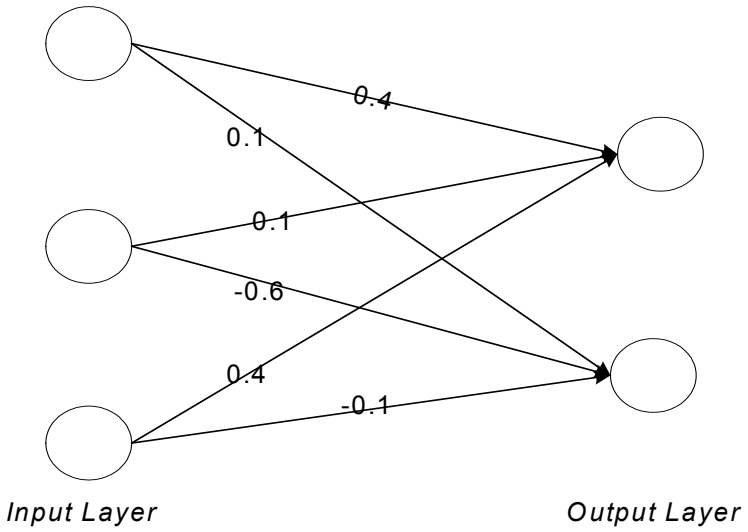
$$Output = \frac{1}{1 + e^{-acc}} \quad \text{..... (2.1)}$$

Sedangkan *Acc* merupakan bilangan *Accumulator*, yaitu jumlah perkalian dari *Weight Output Layer* dengan *Output Input Layer*.

$$Acc = \sum Weight_{ij} \times Out_i \quad \text{.....(2.2)}$$

Contoh penerapannya adalah sebagai berikut :

Input layer ada 3 *node*, *output layer* ada 2 *node*.



Gambar 2.5 Contoh *Neural Network*

Output 1 = 0.1 Output 1 = ?

Output 2 = 0.3 Output 2 = ?

Output 3 = 0.2

Maka langkah pertama yang dilakukan adalah mencari Bilangan *Accumulator*-nya.

$$Acc = \sum Weight_{ij} \times Out_{i} \quad \text{..... (2.3)}$$

Acc Output 1 = (0.1 x 0.4) + (0.3 x 0.1) + (0.2 x 0.4) = 0.15

Acc Output 2 = (0.1 x –0.3) + (0.3 x –0.6) + (0.2 x –0.1) = -0.23

Setelah mendapatkan bilangan *Accumulator*-nya, maka mencari output-nya, dengan rumus :

$$Output = \frac{1}{1 + e^{-acc}} \quad \text{..... (2.4)}$$

Output 1 = 1 / (1 + $e^{-0.15}$) = 0.537

Output 2 = 1 / (1 + $e^{0.23}$) = 0.443

Untuk mencari *error* di *output layer* :

$$Error = Output \times (1 - Output) \times (T\arg et - Output)) \quad \text{..... (2.5)}$$

Untuk mencari *error* di tahap selanjutnya :

$$Error = Output \times (1 - Output) * Acc \quad \text{..... (2.6)}$$

Sedangkan untuk mencari *Accumulator*-nya :

$$Acc = \sum Weight_{ij} \times Error_{j} \quad \text{..... (2.7)}$$

c. *Update Weight* di *Multi Layer Network*

Inti dari *neural network* adalah mencari *weight* yang sesuai dari input yang dimasukkan menjadi output yang diinginkan. *Weight* ini nantinya didapat setelah mendapatkan *Output* dan *Error* dari setiap *Node*.

Untuk mencari *weight* baru :

$$Weight = Weight + (\alpha \times Output_{sebelum} \times Error_{sesudah}) \quad \text{..... (2.8)}$$

Alpha(α) di atas dapat diartikan sebagai ketelitian suatu sistem dalam belajar, semakin tinggi nilainya, maka semakin tinggi kecepatan belajarnya, namun dengan demikian akan berdampak kurang baik, karena akan mendapatkan *error* yang tidak merata.

Sehingga dengan demikian, lebih baik digunakan *Alpha* yang kecil, walaupun tahap belajarnya lambat, tetapi hasil keakuratannya tinggi.

d. Algoritma Proses *Training* di *Multi Layer Network* / *Back Propagation*

Langkah 0 : Inisialisasi bobot (gunakan nilai acak kecil -0.5 s/d 0.5).

Langkah 1 : Selama syarat henti salah, lakukan langkah 2-9.

Langkah 2 : Untuk setiap pasang pelatihan (masukan dan target), lakukan langkah 3-8.

FEED FORWARD (fase maju)

Langkah 3 : Setiap masukan (X_i , i = 1, 2, …n) menerima sinyal masukan x_i dan meneruskannya ke seluruh unit pada lapisan di atasnya (*hidden units*).

Langkah 4 : Setiap unit tersembunyi (Z_j , j = 1, 2, …n) menghitung total sinyal masukan terbobot sebagai berikut :

$$Z_{\_in\_j} = V_{0j} + \sum_{j=1}^{p} X_i V_{ij} \quad \text{..... (2.9)}$$

lalu menghitung sinyal keluarannya dengan aktivasi :

$$Z_j = f(Z_{\_in\_j}) \quad \text{..... (2.10)}$$

lalu sinyal ini dikirimkan ke seluruh unit pada lapisan berikutnya (*output layer*).

Langkah 5 : Setiap unit output $(Y_k$, k = 1, 2 , …m) menghitung total sinyal masukan berbobot.

$$Y_{\_in\_k} = W_{0k} + \sum_{j=1}^{p} Z_j W_{jk} \quad \text{.....} \quad \textbf{(2.11)}$$

lalu dihitung sinyal keluarannya dengan fungsi aktivasi :

$$Y_k = f(Y_{\_in\_k}) \quad \text{.....} \quad \textbf{(2.12)}$$

Back Propagation Error **(fase mundur)**

Langkah 6 : Setiap unit output (Y_k , k = 1, 2, … m) menerima sebuah pola target yang sesuai dengan pola masukan pelatihannya. Unit tersebut menghitung informasi kesalahan (*error*) :

$$\delta_k = (t_k - y_k)\, f'(Y_{\_in\_k}) \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad \textbf{(2.13)}$$

kemudian dihitung kondisi bobot (digunakan untuk merubah W_{jk} nantinya).

$$\Delta W_{jk} = \alpha\ \delta_k Z_j \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad \textbf{(2.14)}$$

$$\Delta W_{0k} = \alpha\ \delta_k \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad \textbf{(2.15)}$$

Langkah 7 : setiap unit tersembunyi (Z_j , j = 1, 2, … p) menghitung selisih input

$$\delta_{\_in\_j} = \sum_{k=1}^{m} \delta_k W_{jk} \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad \textbf{(2.16)}$$

$$\delta_j = \delta_{\_in\_j}\ f'(Z_{\_in\_j}) \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad \textbf{(2.17)}$$

$$\Delta V_{ij} = \alpha\ \delta_j X_i \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad \textbf{(2.18)}$$

$$\Delta V_{0j} = \alpha\ \delta_j \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad \textbf{(2.19)}$$

Langkah 8 : (meng-*update weights* dan *biases*);

Setiap unit output (Y_k , k = 1, 2, … m ; j = 0, 1, 2, … p)

$$W_{jk(new)} = W_{jk(old)} + \Delta W_{jk} \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad \textbf{(2.20)}$$

Setiap unit tersembunyi (Z_j , j = 1, 2, … p ; i = 0, 1, 2, … n)

$$V_{ij(new)} = V_{ij(old)} + \Delta V_{ij} \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad \textbf{(2.21)}$$

Langkah 9 : uji kondisi henti.

e. Jenis Perhitungan *Error* dalam *Back Propagation*

Error disuatu node dapat dihitung dengan metode yang berbeda – beda. Metode *error* yang umum digunakan adalah :

1. *Generalizes Data Rule* (GDR) *(Freeman,1991,p96)*

$$\delta_k = (y_k - o_k) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \textbf{(2.22)}$$

dimana :

y_k : target output pada *neuron* ke – k

o_k : output pada *neuron* ke – k

2. *City Block Distance* (CBD)

$$E_p = \sum_{k=1}^{M} \delta_k \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \textbf{(2.23)}$$

$$E_p = \sum_{k=1}^{M} |y_k - o_k| \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \textbf{(2.24)}$$

dimana :

δ_k : *Error Generalizes Data Rule* (GDR) di *node* ke - *k*

y_k : target output pada *neuron* ke – *k*

o_k : output pada *neuron* ke – *k*

E_p : *Error*

3. *Least Mean Square* (LMS)

$$E_p = \tfrac{1}{2} \sum_{k=1}^{M} \delta_k^{\,2} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \textbf{(2.25)}$$

$$E_p = \tfrac{1}{2} \sum_{k=1}^{M} (y_k - o_k)^2 \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \textbf{(2.26)}$$

dimana :

δ_k : *Error Generalizes Data Rule* (GDR) di *node* ke - k

y_k : target output pada *neuron* ke – k

o_k : output pada *neuron* ke – k

E_p : *Error*

f. Kecepatan *Training*

Kecepatan *Training* didalam metode *Back Propagation* dipengaruhi oleh karakter α, semakin kecil α maka semakin lama pula waktu *training*-nya. Menurut *Freeman (1991,p105)*, sebaiknya harga α berikisar antara 0.05 – 0.25 sedangkan menurut *Rich (1991,p505)*, harga yang baik untuk α adalah 0.35.

Proses belajar dapat dipercepat dengan menggunakan suatu teknik yang disebut momentum, yaitu dengan menambahkan sebagian perbaikan bobot sebelumnya $(\eta.\Delta W_{t-1})$, dimana η adalah parameter momentum. Menurut *Freeman (1991,p105)*, harga η diisi dengan nilai positif yang lebih kecil dari 1, sedangkan menurut *Rich (1991,p506)*, harga η sebaiknya diisi dengan nilai 0.9. Penggunaan momentum bersifat optional.

Dengan penambahan momentum, persamaan untuk perbaikan bobot pada *output layer* menjadi :

$$w_{kj}^{\ o}(t+1) = w_{kj}^{\ o}(t) + \alpha \delta_k^{\ o} i_{pj} + \eta \Delta_p w_{kj}^{\ o}(t-1) \quad \ldots\ldots \quad \textbf{(2.27)}$$

Proses perhitungan tersebut diatas akan dilakukan berulang – ulang melalui suatu iterasi untuk mendapatkan harga E_p yang akan cenderung turun.

2.1.6 Inisialisasi *Nguyen – Widrow*

Menurut *Fausett (1994,p.297)*, inisialisasi *Nguyen – Widrow* merupakan modifikasi sederhana berdasarkan analisa geometrid dan pendekatan transformasi *Fourier* yang mampu meningkatkan kecepatan belajar jaringan.

Inisialisasi *Nguyen – Widrow* didefinisikan dengan rumus :

$$\beta = 0.7(p)^{1/n} = 0.7\sqrt[n]{p} \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad \textbf{(2.28)}$$

dimana :

n : jumlah *node* pada lapisan masukan (*input layer*)

p : jumlah *node* pada lapisan tersembunyi (*hidden layer*) pertama

β : *factor scale*

Langkah – langkah penerapan inisialisasi *Nguyen – Widrow* pada bobot antara lapisan input dan lapisan tersembunyi (*hidden layer* pertama) adalah :

1) Nilai bobot diacak antara -0.5 sampai 0.5
2) Jumlahkan semua nilai bobot tersebut
3) Nilai bobot diinisialisasi ulang dengan rumus :

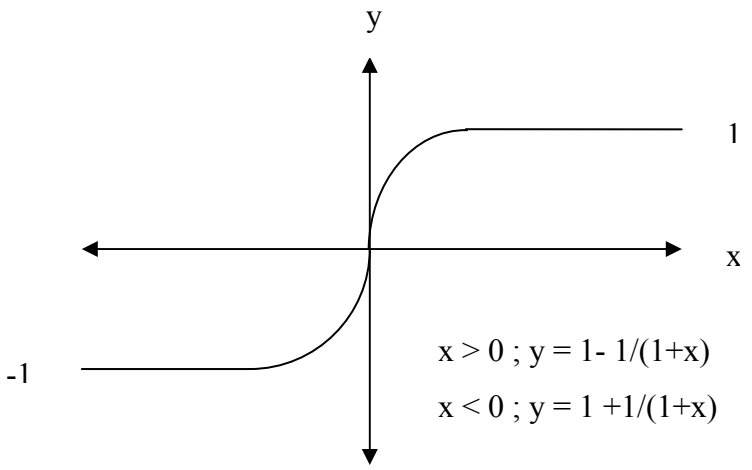
Bobot baru = Bobot lama x β / *Total Bobot Lama* …….. **(2.29)**

2.1.7 Fungsi *Sigmoid*

a. Kegunaan Fungsi *Sigmoid Biner*

Menurut (*Anderson, 1995 p413), (Nelson, 1990 p108)*. Fungsi *Sigmoid* bertujuan untuk menolong sistem untuk mendapatkan output yang diinginkan. Input yang masuk kedalam *neuron* tidak hanya dikalikan dengan *weigh*tnya, akan tetapi mereka juga dikalikan dengan ekuisasi karakter dari suatu *neuron* atau lebih dikenal dengan istilah *transfer function*.

Fungsi *Sigmoid* adalah suatu *non-linear transfer function* yang membantu menyesuaikan output yang diinginkan. Sebuah sifat *non-linear* adalah sesuatu yang signifikan, karena jika suatu *transfer function* bersifat *linear*, maka setiap input akan dikalikan dengan proporsi yang sama setiap kali *training*, ini akan mengakibatkan seluruh sistem akan meleset dalam proses pelatihannya (*training*-nya). Karena itu sistem dapat tidak mendapatkan output yang sudah ada, selagi menyimpan output yang baru. Karena itu sifat *non – linear* dalam suatu sistem membantu mengisolasi *path – path input* yang spesifik.



Gambar 2.6 ***Non-Linear Sigmoid Function***

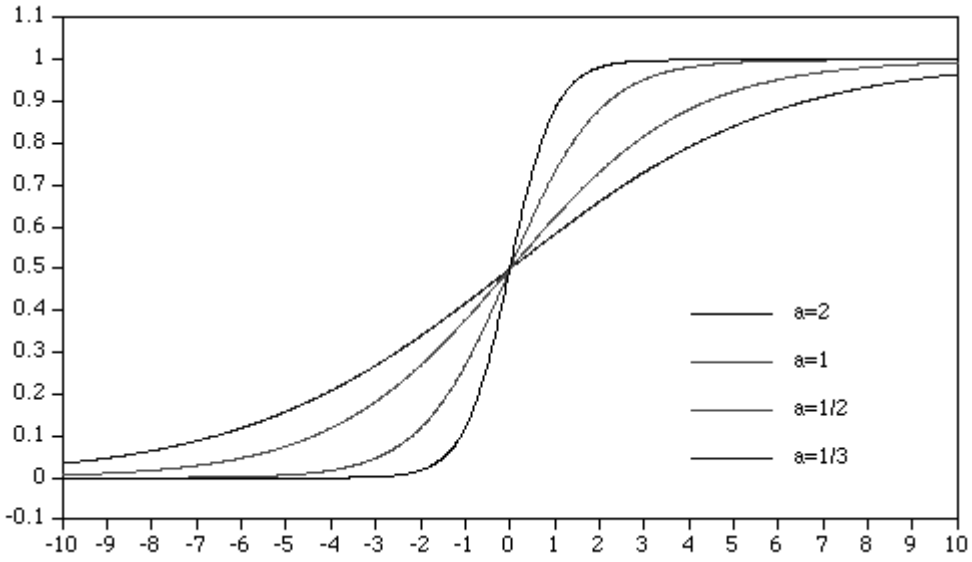
b. Fungsi *Sigmoid Biner*

Yang dimaksud di sini mencakup fungsi – fungsi berbentuk kurva *S*. Sebagai contoh yang sering digunakan adalah fungsi logistik. Fungsi ini memiliki kelebihan dalam melatih *neural network* terutama yang menggunakan algoritma *Back Propagation*, karena hubungan yang sederhana antara nilai fungsi pada suatu titik dengan nilai turunannya, sehingga mengurangi biaya komputasi selama pembelajaran sehingga kompleksitas waktu dapat dipercepat.

Fungsi logistik *sigmoid* :

$$f(x) = \frac{1}{1 + e^{-ax}} \qquad \text{.....} \qquad \textbf{(2.30)}$$

dimana *a* merupakan parameter kecuaraman yang diberikan. Umumnya nilai *a* dipilih 1.



Gambar 2.7 Grafik Perbedaan Nilai *α*

2.1.8 Pengertian Citra

Citra adalah suatu fungsi intensitas warna dua dimensi *f(x,y),* dimana *x* & *y* mewakili lokasi koordinat suatu titik & nilai dari fungsi yang merupakan tingkat itensitas warna / tingkat keabu – abuan dari titik tersebut. *(Schalkoff (1989))*.

2.1.9 *Computer Vision*

Menurut *Fairhurst (1988), Computer Vision* merupakan ilmu yang mempelajari bagaimana komputer dapat mengenali objek yang diamati.

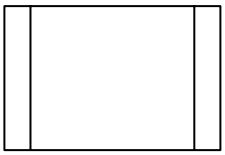
Computer Vision adalah kombinasi antara pengolahan citra dan pengenalan pola, bersama dengan *Artificial Intelligence* akan mampu menghasilkan *Visual Intelligent System*.

2.1.10 *Flowchart*

Menurut www.*thefreedictionary.com*, *flowchart* adalah suatu diagram urutan operasi dalam suatu program komputer / suatu proses sistem. Simbol – simbol yang sering digunakan untuk flowchart adalah sebagai berikut :

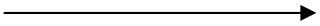
1. Proses

Berupa proses

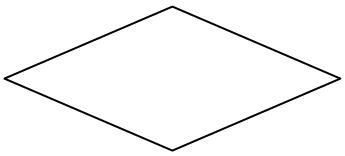


Untuk *predefined process*

2. Panah, menghubungkan antar komponen dan menunjukan arah



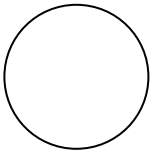
3. *Decision*, berupa pertanyan / penentuan suatu keputusan



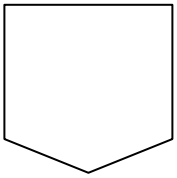
4. *Termination*, sebagai awal atau akhir dari suatu program

5. *Preparation*, untuk inisialisasi awal

6. *Connector*, sebagai penghubung dalam satu halaman



7. *Off Page Connector*, sebagai penghubung antar halaman



2.2 Teori-teori Khusus

2.2.1 Relasi Antar *Pixel*

Menurut *Parker (1993)*, hubungan antar *pixel* berperan dalam menentukan karakteristik suatu citra, hubungan yang paling mendasar ini disebut dengan *connectivity*.

Hubungan antar *pixel* tersebut dapat digambarkan sebagai berikut :

0 (*i*-1,*j*-1)	1 (i,*j*-1)	2 (*i*+1,*j*-1)
3 (*i*-1,j)	4 (*i*,*j*)	5 (*i*+1,*j*)
6 (i-1,*j*+1)	7 (*i*,*j*+1)	8 (*i*+1,*j*+1)

Gambar 2.8 Relasi Antar *Pixel*

Ada dua aturan utama dalam relasi antar *pixel*, yaitu :

a) Dua *pixel* dikatakan *4-adjacent* jika bersebelahan secara horisontal atau vertikal saja. Contoh untuk *pixel* 4 maka 4-*adjacent*-nya adalah 1,3,5,7.

b) Dua *pixel* dikatakan *8 – adjacent* jika bersebelahan secara diagonal atau *pixel – pixel* tersebut juga bersebelahan secara 4-*adjacent*. Contoh untuk *pixel* 4 maka 8 - *adjacent* -nya adalah 0,1,2,3,5,6,7,8.

2.2.2 Tipe Citra

Menurut *Chastain et al (2000)*, ada dua tipe dasar citra berdasarkan cara penyimpanannya, yaitu :

a) *Raster* atau *Bitmap Format*

Citra yang disimpan dengan *format bitmap* atau *raster* memiliki karakteristik menyimpan setiap *pixel* dari sebuah citra. Format ini biasa digambarkan dengan bentuk matriks berukuran panjang *pixel x* lebar *pixel*, yang masing masing dari *pixel* tersebut berisi nilai *rgb* dari *pixel* tersebut. Contoh : BMP, JPG, GIF, dll.

b) Vector Format

Citra yang disimpan dengan format vektor memiliki karakteristik menyimpan konstruksi dari suatu titik, garis, ukuran, warna dan sebagainya yang dapat membentuk suatu citra. Contoh penyimpanan yang menggunakan format ini adalah PS, PDF, WMF, CDM, dll.

2.3 Pengolahan Citra

2.3.1 Pengertian Pengolahan Citra

Menurut *Fairhurst (1988),* pengolahan citra atau yang lebih dikenal dengan *image processing* merupakan bidang yang berhubungan dengan proses transformasi suatu citra.

2.3.2 *Bi – Level Image*

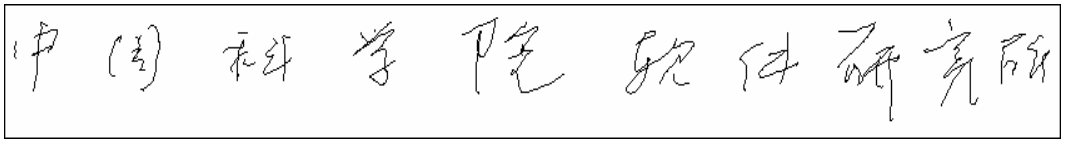
Menurut *Parker (1993)*, *bi-level image* adalah citra yang terdiri dari dua derajat keabuan, yaitu hitam dan putih. Citra ini diperoleh dari proses *thresholding*. Citra yang dihasilkan akan memiliki informasi yang lebih sedikit dan dialakukan apabila ada warna yang tidak penting untuk pengolahan selanjutnya.

Thresholding dilakukan dengan cara membandingkan nilai *pixel* dari suatu *pixel* yang telah di-*grayscale* dengan suatu nilai tetap yang disebut dengan nilai *threshold*. Tujuannya untuk menghasilkan citra yang terdiri dari dua warna *pixel* yaitu hitam dan putih, biasanya warna hitam dilambangkan dengan nilai 1 dan putih dengan nilai 0.

2.4 Segmentasi

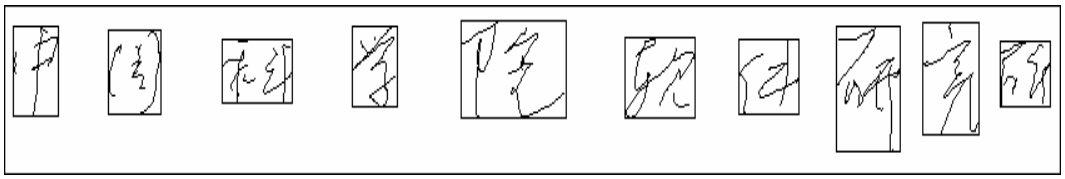
Segmentasi adalah proses mengurangi ukuran suatu citra, yaitu dengan membuang bagian – bagian yang tidak diperlukan, bagian yang dibuang tidak memiliki informasi sama sekali. Tujuan dari segmentasi adalah membuang bagian yang tidak terdapat karakter yang akan dikenali.

Contoh Segmentasi :



Gambar 2.9 Segmentasi 1

Gambar 2.10 Segmentasi 2



Gambar 2.11 Segmentasi 3

2.5 *Histogram*

Menurut *www.thefreedictionary.com*, histogram adalah suatu grafik batang yang mencerminkan suatu frekuensi, lebar masing – masing dari grafik batang tersebut mencerminkan *interval variable* yang diamati, sedangkan tinggi masing grafik tersebut menceriminkan nilai frekuensi dari variabel yang sedang diamati.

Dalam pengolahan citra histogram banyak digunakan dengan memanfaatkan fungsi – fungsi tertentu untuk menghasilkan nilai yang akan diamati, seperti mencari intensitas keabuan dari suatu citra, jumlah *pixel* dan lain – lain.

2.6 Pengenalan Pola

Men*urut Fairhurst(1988*), pengenalan pola atau yang lebih dikenal dengan *pattern recognition* merupakan bidang yang berhubungan dengan proses identifikasi objek pada citra atau interprestasi suatu citra. Proses ini bertujuan untuk mengekstrak informasi atau pesan yang disampaikan oleh citra.

2.7 Karakter Mandarin

2.7.1 Sejarah Karakter Mandarin

Hubungan antara pengucapan bahasa Mandarin dengan penulisan bahasa Mandarin sangatlah kompleks. Terlebih paling tidak sejak jaman Dinasti Han, pengucapan bahasa Mandarin telah banyak mengalami perubahan – perubahan, sedangkan dalam penulisan bahasa Mandarin, hanya sedikit perubahan yang terjadi.

Sampai abad ke-20 kebanyakan penulisan bahasa Mandarin dilakukan dalam bentuk '*wényán* (□□)' yang diterjemahkan sebagai *'Classical Chinese'* atau *'Literary Chinese'*. Sejak tahun 1919 standar penulisan bahasa Mandarin diubah menjadi 'to *báihuà* (□□/□□)' atau disebut sebagai '*Vernacular Chinese',* dimana hampir seluruh tata bahasa dan kata – kata dalam bahasa Mandarin berdasarkannya.

2.7.2 Gaya Penulisan

Penulisan bahasa Mandarin terdiri atas karakter – karakter yang masing – masing dapat berdiri sendiri sebagai sebuah kata. Tidak ada sejarah yang pasti tentang asal muasal karakter – karakter dalam bahasa Mandarin. Tapi sejarah mencatat *Changjie*, seorang birokrat pada jaman pemerintahan *Huangdi* (seorang Kaisar legendaris Cina, sekitar 2600 Sebelum Masehi) - lah yang menciptakan karakter – karakter dalam bahasa Mandarin. Tetapi bukti – bukti arkeologis menunjukkan karakter – karakter dalam bahasa Mandarin ditemukan sekitar 1700 Sebelum Masehi pada jaman pemerintahan Dinasti Shang.

Banyak bukti – bukti yang ditemukan di *Yinxu* pada masa Dinasti *Shang*, tapi juga ada beberapa bukti yang terhubung dengan Dinasti *Zhou*, walau jumlahnya lebih sedikit. Bentuk dari karakter – karakter dalam bahasa Mandarin berubah seiring dengan dua atau tiga ratus tahun penggunaannya, dan para peneliti dapat mengenali karakter – karakter dari Dinasti *Shang* berdasarkan dari isi dan penggunaannya.

Salah satu pemahaman yang salah tentang karakter – karakter dalam bahasa Mandarin adalah sebuah karakter hanyalah sebuah gambar saja, tanpa ada arti secara abstrak. Pemahaman ini salah, karakter – karakter dalam bahasa Mandarin dibuat berdasarkan artinya. Jadi ada keterkaitan antara bentuk karakter dengan arti dari karakter tersebut.

Ada 2 standar dalam penulisan karakter – karakter dalam bahasa Mandarin. Yang kesatu adalah '*traditional system'* yang masih digunakan di Hong Kong, Taiwan, dan Makau. Penulisan secara tradisional secara penulisan memang lebih rumit, tetapi lebih memiliki keterikatan bentuk dengan arti dari karakter yang dimaksud. Yang kedua adalah '*simplified system'*. Sistem ini mulai berkembang ketika tahun 1950-an Partai Komunis mengambil alih pemerintahan di daratan Cina. '*simplified system'* mengurangi jumlah garis yang dipakai dalam satu karakter dan mengurangi jumlah karakter – karakter yang memiliki kesamaan satu sama lainnya. Walaupun penulisannya lebih sederhana dari '*traditional system'*, sistem ini kurang memiliki keterkaitan yang erat dengan arti dari karakter yang dimaksud. Sistem inilah yang belakangan digunakan, sedangkan '*traditional system'* sudah jarang digunakan.

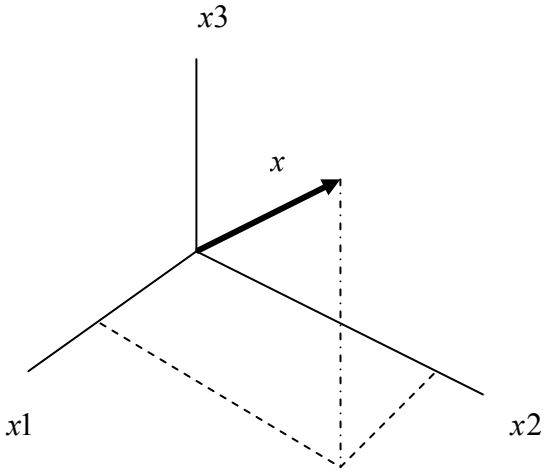
2.8 *Feature*

Menurut *Parker (1993), feature* adalah semua pengukuran yang dilakukan terhadap suatu citra. *Feature* dari suatu citra dapat berupa warna, ukuran, orientasi, panjang dan kemiringan garis, jari – jari, dll, yang diperoleh dari suatu proses ekstrasi. *Feature* harus bisa dihitung, mempunyai kemampuan pembeda yang besar, dan mencerminkan citra yang diwakili.

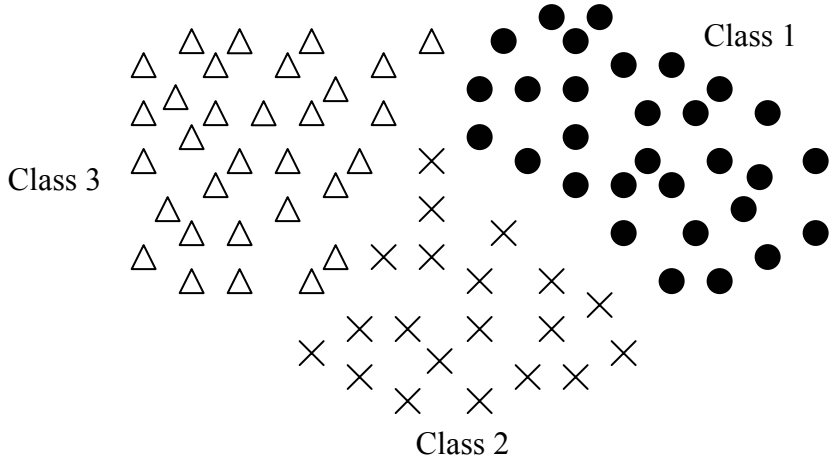
Kombinasi dari *N feature* direpresentasikan dalam bentuk matriks 1 kolom kali *N* baris yang disebut dengan *feature vector*. Ruang berdimensi *N* yang dibentuk oleh *N Feature vector* disebut *feature space*. Objek direpresentasikan sebagai titik – titik di dalam *feature space* dan representasi ini dikenal sebagai *scatter plot*.

$$X = \begin{bmatrix} X1 \\ X2 \\ \dots \\ Xn \end{bmatrix}$$

Gambar 2.12 *Feature Vector*



Gambar 2.13 *Feature Space* (3D)



Gambar 2.14 *Scatter Plot* (2D)

2.9 Normalisasi Bentuk

Menurut *Yamada et al. (1989, p1023),* normalisasi adalah suatu proses yang bertujuan untuk mengubah suatu citra menjadi citra baru yang memiliki posisi, ukuran dan atau kemiringan yang konstan berdasarkan karakteristik tertentu. Normalisasi bentuk dilakukan untuk menangani perbedaan bentuk seperti perbedaan bentuk tulisan tangan.

Lee dan Park (1994) membahas normalisasi berdasarkan dua sudut pandang, yaitu *feature projection* dan *feature density equalization*. *Feature Projection* dilakukan dengan memproyeksikan *feature* pada setiap titik ke sumbu *x* dan *y* untuk menghasilkan *feature projection* histogram. *Feature density equalization* dilakukan dengan menyamakan intensitas *feature* dari citra masukan dengan perhitungan berdasarkan *feature projection histogram*.

2.9.1 Normalisasi Bentuk Linear

Normalisasi ini dapat dipandang sebagai tranformasi *linear* dari posisi (x,y) menjadi (x',y') berdasarkan rumus berikut :

$$x' = a_1 x + a_2 y + a_3 \quad \text{..... (2.31)}$$

$$y' = a_4 x + a_5 y + a_6 \quad \text{..... (2.32)}$$

dimana :

$a_{1..6}$: konstanta

Normalisasi ukuran hanya bergantung pada ukuran citra masukan dari citra hasil normalisasi, sehingga *feature projection* dilakukan dengan menggunakan fungsi berikut :

$$H(i) = 1 \quad \text{..... (2.33)}$$

$$V(j) = 1 \quad \text{..... (2.34)}$$

dimana :

i : nilai x pada citra masukan, bernilai 1 sampai i

j : nilai y pada citra masukan bernilai 1 sampai j

$H(i)$: histogram berdasarkan proyeksi *feature* terhadap sumbu x

$V(j)$: histogram berdasarkan proyeksi *feature* terhadap sumbu y

Feature density equalization dilakukan dengan fungsi berikut :

$$m = \sum_{k=1}^{i} H(k) \times \frac{M}{\sum_{k=1}^{i} H(k)} \quad \text{..... (2.35)}$$

$$n = \sum_{l=1}^{i} V(l) \text{ x } \frac{N}{\sum_{l=1}^{i} V(l)} \quad \text{..... (2.36)}$$

dimana

m : nilai x pada citra keluaran bernilai 1 sampai M

n : nilai y pada citra keluaran bernilai 1 sampai N

Dengan substitusi persamaan **2.33** dan **2.34** ke persamaan **2.35** dan **2.36**, maka diperoleh fungsi sebagai berikut :

$$m = i\,x\,\frac{M}{I} \quad \text{..... (2.37)}$$

$$n = j\,x\,\frac{N}{J} \quad \text{..... (2.38)}$$

2.9.2 Normalisasi Bentuk Non Linear

a. Normalisasi Bentuk Non Linear berdasarkan Itensitas Titik

Normalisasi ini menggunakan jumlah titik hitam sebagai *feature*. Titik - titik pada citra diproyeksikan pada sumbu x dan y dalam proses normalisasi.

Feature projection dialakukan dengan fungsi sebagai berikut :

$$H(\text{i}) = \sum_{j=1}^{J} f(i,j) + \alpha_H \quad \text{..... (2.39)}$$

$$V(\text{j}) = \sum_{i=1}^{I} f(i,j) + \alpha_V \quad \text{..... (2.40)}$$

dimana :

$f(i,j)$: nilai *pixel* pada posisi (i,j)

$\alpha_{H,}\ \alpha_V$: konstanta

b. Normalisasi Bentuk Non Linear berdasarkan Intensitas Garis dengan Perpotongan Garis

Intensitas garis didefinisikan sebagai jumlah perpotongan garis dari citra masukan pada arah sumbu *x* dan *y*.

Feature projection dilakukan dengan fungsi berikut :

$$H(i) = \sum_{j=1}^{J} \overline{f(I,j-1)} - f(I,j) + \alpha_H \qquad \text{..... (2.41)}$$

$$V(j) = \sum_{i=1}^{I} \overline{f(I-1,j)} - f(I,j) + \alpha_v \qquad \text{..... (2.42)}$$

dimana :

$\overline{f(I,j)}$: negasi nilai pixel pada posisi I , j

Feature density equalization dilakukan dengan fungsi **2.41** dan **2.42**

c. Normalisasi Bentuk Non Linear berdasarkan Intensitas Garis dengan Perhitungan Interval Garis

Dalam normalisasi ini, intensitas garis didefinisikan dengan menggunakan jarak antar garis.

Feature projection dilakukan dengan fungsi berikut :

$$F_H(i,j) = \frac{1}{H(i,j)} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \textbf{(2.43)}$$

$$F_V(i,j) = \frac{1}{V(i,j)} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \textbf{(2.44)}$$

$$H(i) = \sum_{j=1}^{J} f_H(i,j) \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \textbf{(2.45)}$$

$$V(j) = \sum_{i=1}^{I} f_V(i,j) \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \textbf{(2.46)}$$

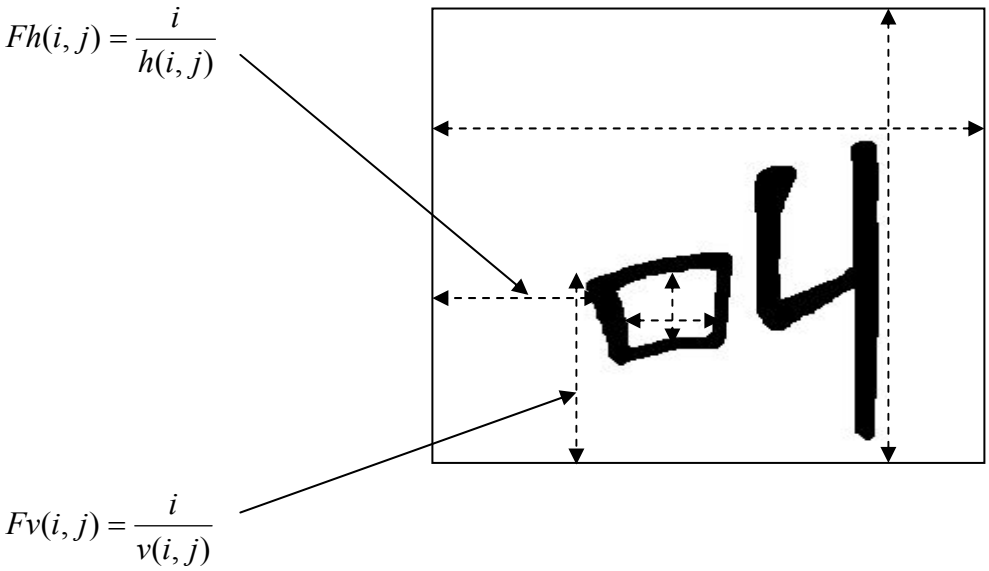
dimana :

$H(i,j)$: Jarak Horisontal antar Garis

$V(i,j)$: Jarak Vertikal antar Garis

$F_H(i,j)$: Karakteristik untuk Histogram Horisontal

$F_V(i,j)$: Karakteristik untuk Histogram Vertikal



Gambar 2.15 Normalisasi dengan Perhitungan Interval Garis

Menurut *Tsukumo dan Tanaka (1988)*, untuk *pixel* posisi *(I,j)* yang berwarna hitam, nilai *H(I,j) dan V(I,j)* diberi nilai yang sangat kecil. Sedangkan menurut Lee dan Park (1994), nilai *H(I,j)* dan *V(I,j)* diberi nilai yang sangat besar. *Feature density equalization* dilakukan dengan fungsi **(2.45)** dan **(2.46)**

d. Normalisasi Bentuk Non Linear berdasarkan Intensitas Garis dengan Perhitungan Lingkaran yang Terpahat

Dalam normalisasi ini, intensitas garis didefinisikan dengan menggunakan jarak pada lingkaran yang terpahat atau terpotong oleh garis – garis pada citra masukan.

Feature Projection dilakukan dengan fungsi berikut :

$$E_1 = \max\{i' \mid i' < i, f(i',j) - \overline{f(i'+1,j)} = 1\} \quad \text{.....} \quad \textbf{(2.47)}$$

$$E_2 = \min\{i' \mid i' < i, f(i',j) - \overline{f(i'+1,j)} = 1\} \quad \text{.....} \quad \textbf{(2.48)}$$

$$E_3 = \max\{i' \mid i' < i, \overline{f(i'-1,j)} - f(i',j) = 1\} \quad \text{.....} \quad \textbf{(2.49)}$$

$$E_4 = \min\{i' \mid i' < i, \overline{f(i'-1,j)} - f(i',j) = 1\} \quad \text{.....} \quad \textbf{(2.50)}$$

dimana :

$E_{1..4}$: Batas garis

$P(i,j)$: *feature* posisi (i,j) pada citra

L_H : Jarak horisontal antar garis

L_V : Jarak vertikal antar garis

2.9.3 Smearing

Normalisasi bentuk secara non linear akan menimbulkan jarak / ruang pada *pixel* yang berdekatan sebagai akibat dari proses *feature density equalization*. Ruang yang ditimbulkan ini dapat diperbaiki / diisi dengan menggunakan metode *smearing*. *Smearing* dapat dilakukan dengan melakukan penggandaan *pixel* pada posisi (X_i, Y_i) ke posisi (x,y) dengan fungsi :

$$f(x,y) = f(X_i, Y_i) \quad \text{……..} \quad \textbf{(2.51)}$$

$$(X_{i-1} =) X_i - \gamma < \chi \leq X_i,\ X_o = 0 \quad \text{…….} \quad \textbf{(2.52)}$$

$$(Y_{j-1} =) Y_i - \gamma < y \leq Y_j,\ Y_o = 0 \quad \text{…….} \quad \textbf{(2.53)}$$

dimana :

f(i,j) : nilai *pixel* pada posisi *(i,j)*

X, Y : Posisi *m* dan *n* hasil perhitungan dari *feature density equalization*

(a) **(b)**

Citra Asli **Hasil Normalisasi Non Linear Tanpa Smearing**

Gambar 2.16

2.10 *Feature Extraction*

Menurut *Devijver dan Kittler (1982)*, *feature extraction* adalah proses ekstraksi informasi dari suatu data sehingga relevan untuk tujuan klasifikasi dimana proses ekstraksi meminimalisasi variasi pola di dalam kelas yang sama dan memaksimalkan variasi antar kelas.

Masalah umum yang sering dihadapi dalam pengenalan tulisan tangan adalah variasi yang sangat banyak, baik variasi yang ditimbukan oleh user itu sendiri maupun variasi antar user yang satu dengan yang lain. Oleh karena itu, perlu dicari ciri – ciri khas yang mewakili setiap karakter Mandarin, yang diperoleh dari proses *feature extraction*.

Lebih baik menggunakan banyak *feature* yang mudah untuk dihitung secara akurat daripada *feature* yang sedikit, kompleks dengan tingakat keakuratan yang minim. Cara untuk menemukan *feature* dalam suatu citra juga harus mendapat perhatian yang khusus karena hal tersebut tidak mudah dilakukan. Untuk proses pengenalan karakter diperlukan *feature* yang unik, yang bisa membedakan karakter yang satu dengan karakter yang lain dan bisa diterapkan pada semua jenis karakter yang sama.

2.10.1 4 - *Directional Plane*

Menurut *Gao et al (2001),* karakter Mandarin secara umum terdiri dari 4 macam goresan dasar, yaitu horisontal, vertikal, diagonal kiri dan diagonal kanan seperti yang ditunjukkan dalam **Gambar 2.18**. Walaupun terdapat banyak bentuk variasi bentuk, relasi jarak antar goresan relatif tetap.

Menurut Chen et al. (1997), ketebalan rata – rata dari citra karakter Mandarin dapat dihitung dengan fungsi – fungsi berikut :

$$L = \sum_{i=1}^{M} t_i \qquad \text{..... (2.54)}$$

$$W << L \qquad \text{..... (2.55)}$$

dimana :

L : jumlah panjang seluruh goresan

M : jumlah goresan di dalam citra

t_i : panjang goresan

W : lebar rata – rata dari semua goresan pada citra hasil normalisasi

$$T \approx W \times L \qquad \text{..... (2.56)}$$

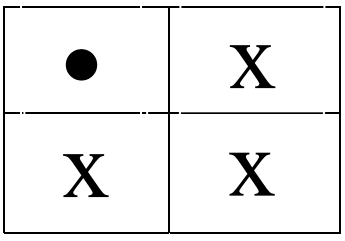
$$L \approx T - S \qquad \text{..... (2.57)}$$

$$W \approx \frac{T}{L} \approx \frac{T}{T - S} \qquad \text{..... (2.58)}$$

dimana :

T : jumlah seluruh *pixel* hitam pada citra

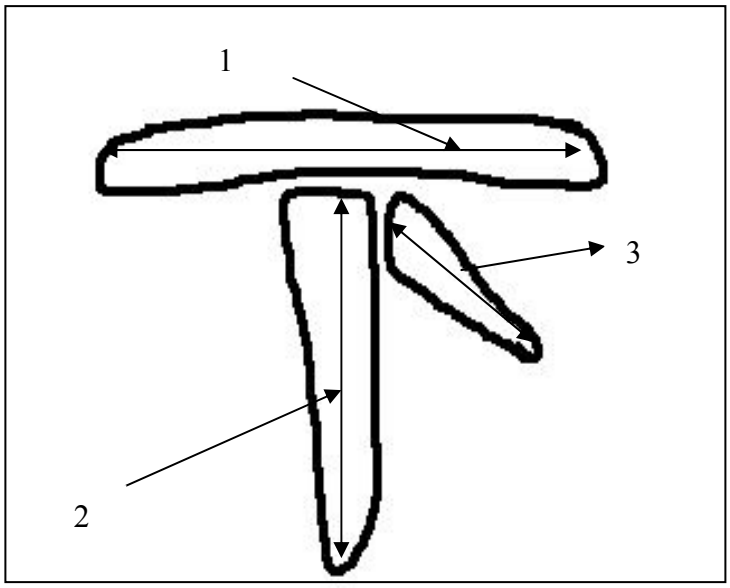
S : jumlah *pixel* pada citra yang memenuhi aturan *mask* 2 x 2 pada **Gambar 2.17**.

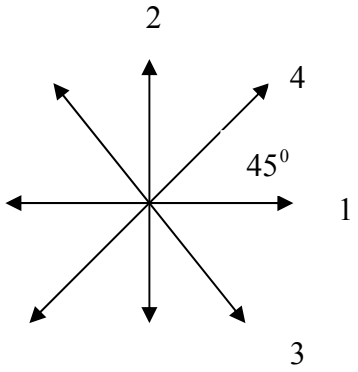


Gambar 2.17 Mask 2 x 2 Untuk perhitungan *S*

Menurut *Gao et al (2001)*, ekstraksi goresan dasar dapat dilakukan dengan memperhitungkan jarak antar kontur untuk suatu set *pixel* pada empat arah goresan dasar seperti terlihat pada **Gambar 2.18.**

Satu set *pixel* adalah *pixel P* dan seluruh *pixel* 8 – *adjacent* – nya seperti yang ditunjukan pada **Gambar 2.8** Arah suatu set *pixel* ditentukan berdasarkan jarak kontur maksimum set *pixel* tersebut untuk seluruh arah yang ada pada **Gambar 2.18** atau jika jarak kontur set *pixel* tersebut pada arah tertentu lebih dari *W*. Jarah kontur untuk *pixel* yang berwarna putih adalah 0.





Gambar 2.18 4 – *Directional Plane*

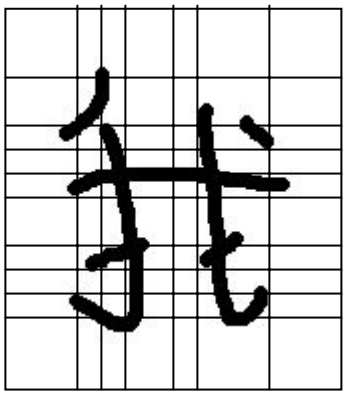
Ekstraksi goresan dasar dari suatu citra karakter Mandarin akan menghasilkan 4 citra baru dimana goresan – goresan dasar dari citra asli yang akan terdapat pada yang arahnya bersesuaian dengan arah goresan dasar yang diwakili.

2.10.2 Projection Histogram

Projection histogram adalah histogram yang dihasilkan melalui proyeksi *feature* suatu citra pada sumbu *x* maupun *y*. *Feature* yang diproyeksikan dapat berupa garis seperti yang disebutkan pada bagian **2.9**, intensitas keabuan dari citra, dan lain – lain

2.10.3 Meshing

Menurut *Gao et al (2001)*, *meshing* adalah teknik yang digunakan untuk mengekstrak *directional feature* dari suatu citra. Proses *meshing* membentuk mesh berukuran M kolom x N baris terhadap suatu citra berdasarkan *feature* tertentu.



Gambar 2.19 Meshing

Proses pembentukan posisi *mesh* ini mengikuti fungsi :

$$m = \min\left\langle i \left| \frac{s-1}{M}\sum_{k=1}^{I} H(k) \le \sum_{k=1}^{I} H(k) \le \frac{s}{M}\sum_{k=1}^{I} H(k) \right\rangle \quad \text{…….} \quad \textbf{(2.59)}$$

$$n = \min\left\langle j \left| \frac{t-1}{N} \sum_{l=1}^{j} V(j) \le \sum_{l=1}^{j} V(j) \le \frac{t}{N} \sum_{l=1}^{j} V(j) \right. \right\rangle \quad \text{…….. } \textbf{(2.60)}$$

dimana :

m : posisi x pada *mesh.*

n : posisi y pada *mesh.*

i : posisi x pada citra masukan, dari 1 sampai I.

j : posisi y pada citra masukan, dari 1 sampai J.

s : indeks *mesh* posisi horisontal, dari 1 sampai M.

t : indeks *mesh* posisi vertikal, dari 1 sampai N.

$H(i)$: *feature projection* normalisasi non linear dengan menggunakan hasil normalisasi.

$V(j)$ **:** *feature projection* normalisasi non linear dengan menggunakan hasil normalisasi