Bab 8

Jaringan Syaraf Tiruan (*Neural Network*)

POKOK BAHASAN:

- ✓ Struktur Dasar Jaringan Biologi
- ✓ Konsep Dasar Pemodelan Jaringan Syaraf Tiruan
- ✓ Mengaktifkan Jaringan Syaraf Tiruan
- ✓ Jaringan Single Perceptron
- ✓ Jaringan Multi Perceptron
- ✓ Metode Backpropagasi

TUJUAN BELAJAR:

Setelah mempelajari bab ini, mahasiswa diharapkan mampu:

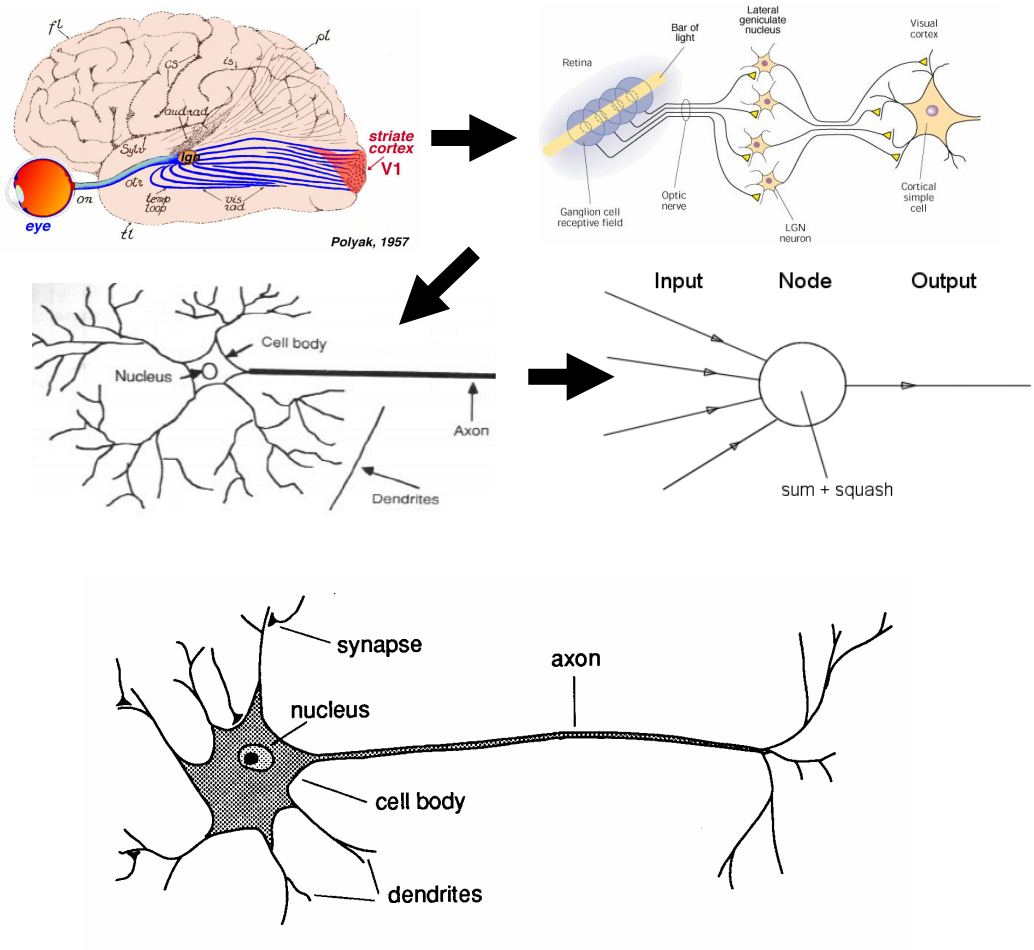
- ✓ Mahasiswa mengerti konsep dasar Jaringan saraf tiruan dan pemodelannya
- ✓ Mahasiswa mengerti metode pelatihan dan model jaringan saraf tiruan
- ✓ Mahasiswa dapat membuat program jaringan saraf tiruan

8.1 KONSEP DASAR JARINGAN SARAF TIRUAN DAN PEMODELANNYA

8.1.1 Struktur Dasar Jaringan Biologi

Pembuatan struktur jaringan saraf tiruan diilhami oleh struktur jaringan biologi, khususnya jaringan otak manusia. Untuk lebih mengenal asal-usul serta bagaimana suatu struktur jaringan saraf tiruan dibuat dan dapat dipakai sebagai suatu alat penghitung, berikut ini akan diulas sedikit istilah yang secara umum digunakan.

Neuron adalah satuan unit pemroses terkecil pada otak, bentuk sederhana sebuah neuron yang oleh para ahli dianggap sebagai satuan unit pemroses tersebut di gambarkan sebagai berikut:



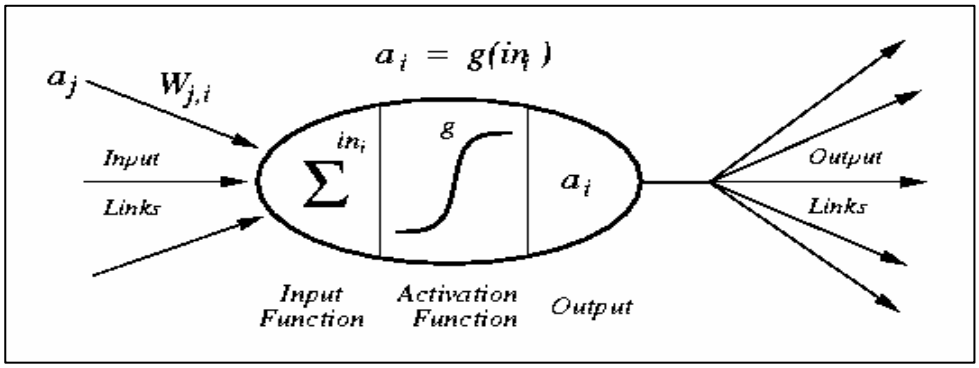
Gambar 8.1 Struktur dasar jaringan syaraf tiruan dan Struktur sederhana sebuah neuron

Struktur pada gambar 8.1 tersebut adalah bentuk standard dasar satuan unit jaringan otak manusia yang telah disederhanakan. Bentuk standard ini mungkin dikemudian hari akan berubah bila ada ilmuwan yang dapat menciptakan bentuk standard yang lebih baik ataupun memperbaiki bentuk standard yang digunakan saat ini. Jaringan otak manusia tersusun tidak kurang dari 10^{13} buah neuron yang masing-masing terhubung oleh sekitar 10^{15} buah dendrite. Fungsi dendrite adalah sebagai penyampai sinyal dari neuron tersebut ke neuron yang terhubung dengannya. Sebagai keluaran, setiap neuron memiliki axon, sedangkan bagian penerima sinyal disebut synapse. Penjelasan lebih rinci tentang hal ini dapat diperoleh pada disiplin ilmu biology *molecular*.

Secara umum jaringan saraf terbentuk dari jutaan (bahkan lebih) struktur dasar neuron yang terinterkoneksi dan terintegrasi antara satu dengan yang lain sehingga dapat melaksanakan aktifitas secara teratur dan terus menerus sesuai dengan kebutuhan.

8.1.2 Konsep Dasar Pemodelan Jaringan Syaraf Tiruan

Tiruan neuron dalam struktur jaringan saraf tiruan adalah sebagai elemen pemroses seperti pada gambar 8.2 yang dapat berfungsi seperti halnya sebuah neuron. Sejumlah sinyal masukan a dikalikan dengan masing-masing penimbang yang bersesuaian w. Kemudian dilakukan penjumlahan dari seluruh hasil perkalian tersebut dan keluaran yang dihasilkan dilalukan kedalam fungsi pengaktip untuk mendapatkan tingkatan derajad sinyal keluarannya F(a,w). Walaupun masih jauh dari sempurna, namun kinerja dari tiruan neuron ini identik dengan kinerja dari sel biologi yang kita kenal saat ini.



Gambar 8.2. Model tiruan sebuah neuron

- aj :Nilai aktivasi dari unit j
- wj,i :Bobot dari unit j ke unit i
- ini :Penjumlahan bobot dan masukan ke unit i
- g :Fungsi aktivasi
- ai :Nilai aktivasi dari unit i

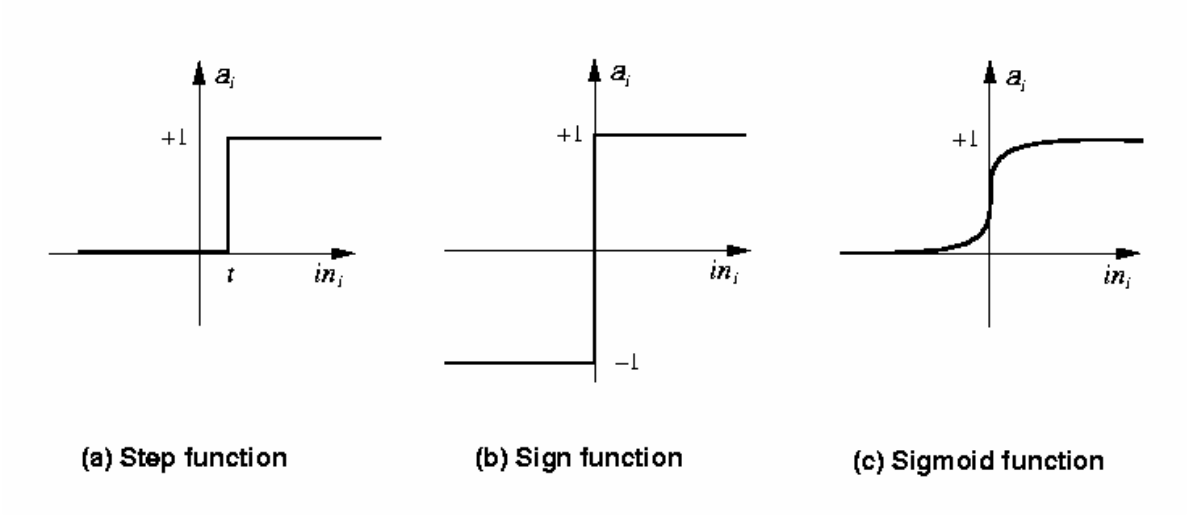
Misalkan ada n buah sinyal masukan dan n buah penimbang, fungsi keluaran dari neuron adalah seperti persamaan (8.1) berikut,

$$in_i = \sum_j W_{ji} * a_j \qquad (8.1)$$

Kumpulan dari neuron dibuat menjadi sebuah jaringan yang akan berfungsi sebagai alat komputasi. Jumlah neuron dan struktur jaringan untuk setiap problema yang akan diselesaikan adalah berbeda.

8.1.3 Mengaktifkan Jaringan Saraf Tiruan

Mengaktipkan jaringan saraf tiruan berarti mengkatipkan setiap neuron yang dipakai pada jaringan tersebut. Banyak fungsi yang dapat dipakai sebagai pengaktip, seperti fungsi-fungsi goniometri dan hiperboliknya, fungsi unit step, impulse, sigmoid, dan lain sebagainya seperti pada gambar 8.3, tetapi yang lazim digunakan adalah fungsi sigmoid, karena dianggap lebih mendekati kinerja sinyal pada otak

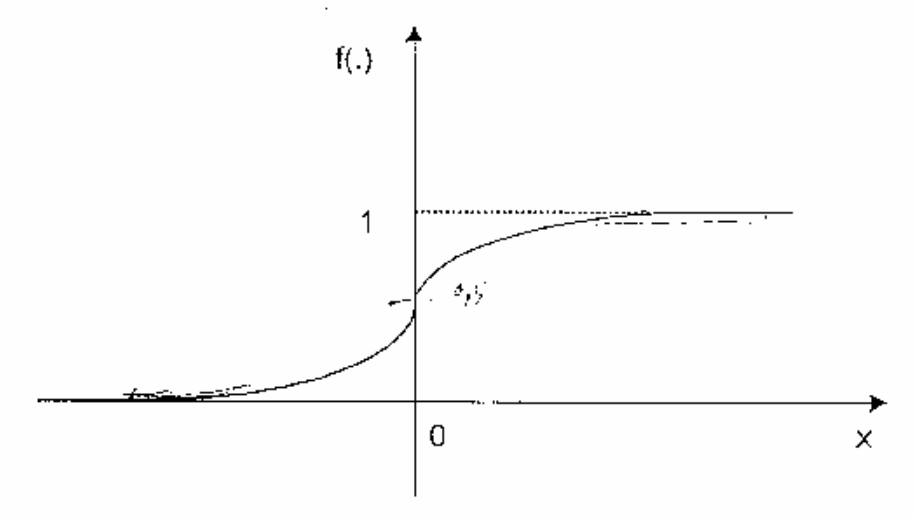


Gambar 8.3 Fungsi pengaktif

- Stept(x) = 1 if x >= t else 0
- Sign(x) = +1 if x >= 0 else –1
- Sigmoid(x) = 1/(1+e-x)
- Fungsi Identitas

Ada dua jenis fungsi sigmoid, unipolar dan bipolar. Fungsi sigmoid unipolar dituliskan pada persamaan (8.2) dan ditunjukkan pada gambar 8.4

$$y = \left(\frac{1}{1 + e^{-f(x)}} \right) \tag{8.2}$$

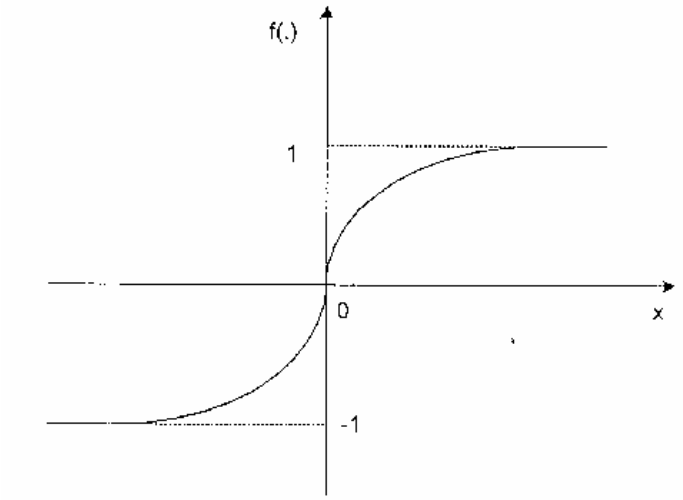


Gambar 8.4. Fungsi sigmoid unipolar

Sedangkan fungsi pengaktip bipolar adalah persamaan (8.3) atau (8.4). Persamaan (8.4) disebut juga sebagai persamaan tangen hiperbolik dan bentuk fungsi seperti pada gambar 8.5.

$$y = \left(\frac{1 - e^{-f(x)}}{1 + e^{-f(x)}} \right) \tag{8.3}$$

$$y = \left(\frac{e(x) - e(-x)}{e(x) + e(-x)} \right) = \frac{1 - e(-2x)}{1 + e(-2x)} \tag{8.4}$$



Gambar 8.5. Fungsi sigmoid bipolar

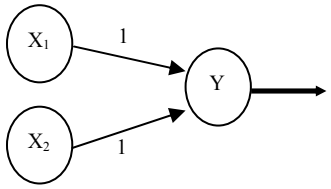
8.1.4 Contoh dan Program Jaringan Sederhana

Beberapa contoh jaringan sederhana AND, OR dan NOT dapat dijelaskan seperti dibawah ini. Tabel 8.1 adalah tabel hasil operasi AND

Tabel 8.1 Operasi AND

X1	X2	Y
1	1	1
1	0	0
0	1	0
0	0	0

Dari table diatas terlihat ada dua input (X1 dan X2) dan satu output (Y) dengan bobot yang sesuai untuk w1 adalah 1 dan w2 adalah 1 sehingga diperoleh nilai threshold yang sesuai dengan tabel adalah 2. Arsitektur jaringan sederhana untuk kasus AND seperti pada gambar 8.6



Threshold(Y) = 2

Gambar 8.6 Arsitektur jaringan sederhana operasi AND

Contoh program jaringan sederhana operasi AND adalah:

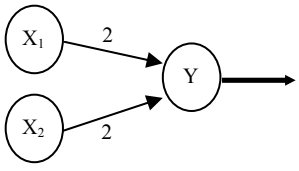
```
#include<iostream.h>
main()
{
        int w1,w2,x1,x2,O;
        w1=1;w2=1;
        cout<<"X1=";
        cin>>x1;
        cout<<"X2=";
        cin>>x2;
        O=x1*w1+x2*w2;
        if(O>=2)
                cout<<"Output AND=1"<<endl;
        else
                cout<<"Output AND=0"<<endl;
}
```

Tabel 8.2 adalah tabel hasil operasi OR.

Tabel 8.2 Operasi OR

X1	X2	Y
1	1	1
1	0	1
0	1	1
0	0	0

Dari table diatas terlihat ada dua input (X1 dan X2) dan satu output (Y) dengan bobot yang sesuai untuk w1 adalah 2 dan w2 adalah 2 sehingga diperoleh nilai threshold yang sesuai dengan tabel adalah 2. Arsitektur jaringan sederhana untuk kasusOR seperti pada gambar 8.7



Threshold(*Y*) = 2

Gambar 8.7 Arsitektur jaringan sederhana operasi OR

Contoh program jaringan sederhana operasi OR adalah:

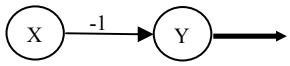
```
#include<iostream.h>
main()
{
        int w1,w2,x1,x2,O;
        w1=2;w2=2;
        cout<<"X1=";
        cin>>x1;
        cout<<"X2=";
        cin>>x2;
        O=x1*w1+x2*w2;
        if(O>=2)
                cout<<"Output OR=1"<<endl;
        else
                cout<<"Output OR=0"<<endl;
}
```

Tabel 8.3 adalah tabel hasil operasi NOT.

Tabel 8.3 Operasi NOT

X	**Y**
0	1
1	0

Dari table diatas terlihat ada satu input (X) dan satu output (Y) dengan bobot yang sesuai untuk w adalah -1 sehingga diperoleh nilai threshold yang sesuai dengan tabel adalah -0.5. Arsitektur jaringan sederhana untuk kasus OR seperti pada gambar 8.8



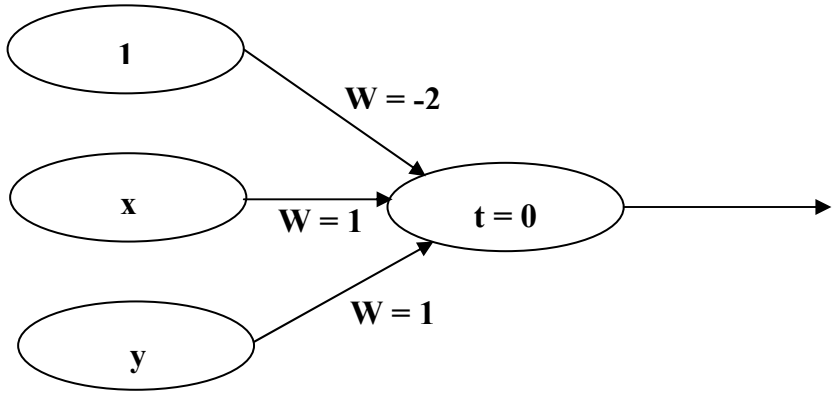
Threshold(Y) = -0.5

Gambar 8.8 Arsitektur jaringan sederhana operasi NOT

Contoh program jaringan sederhana operasi OR adalah:

```
#include<iostream.h>
main()
{
        int w1,x1,O;
        w1=-1;
        cout<<"X1=";
        cin>>x1;
        O=x1*w1;
        if(O>=-0.5)
                cout<<"Output NOT=1"<<endl;
        else
                cout<<"Output NOT=0"<<endl;
}
```

Dari Arsitektur jaringan sederhana operasi And diatas dapat disederhanakan lagi dengan membuat threshold = 0 sehingga diperoleh arsitektur jaringan dengan bias = 1 seperti pada gambar 8.9.



Gambar 8.9 Arsitektur jaringan sederhana operasi AND dengan Bias

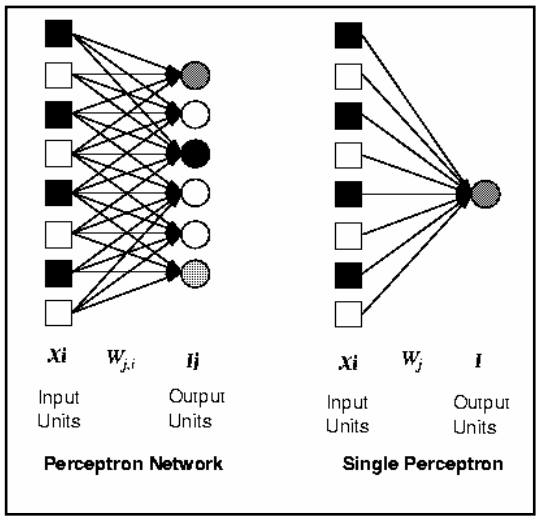
8.2 METODE PELATIHAN DAN MODEL JARINGAN SARAF TIRUAN

8.2.1 Metode Pelatihan Terbimbing

Metoda pelatihan terbimbing adalah metoda pelatihan yang memasukkan target keluaran dalam data untuk proses pelatihannya. Ada beberapa metoda pelatihan terbimbing yang telah diciptakan oleh para peneliti, diantaranya yang sering diaplikasikan adalah Single Perseptron, Multi Perseptron dan Back Propagation (BP). Metoda BP tersebut sampai saat ini masih sangat banyak yang menggunakan, begitu juga yang telah dimodifikasi sehingga menjadi lebih efektip kinerjanya.

8.2.1.1 Jaringan Single Perseptron

Jaringan lapis tunggal Perseptron (single layer perceptron) terdiri dari beberapa unit pemroses (neuron) seperti gambar 8.10, yang terhubung dan mempunyai beberapa masukan serta sebuah atau beberapa keluaran. Single Perceptron sering disebut juga dengan Perceptron. Perseptron menghitung jumlah nilai perkalian penimbang dan masukan dari parameter permasalahan yang kemudian dibandingkan dengan nilai treshold. Bila nilai keluaran lebih besar dari treshold maka keluarannya adalah satu, sebaliknya adalah nol.



Gambar 8.10 Single Perseptron

Pernyataan ini merupakan hasil proses pelatihan yang dalam bentuk bahasanya adalah pernyataan ya dan tidak atau bukan. Secara matematis dapat dituliskan sebagai berikut:
Jumlah perkalian penimbang dan parameter masukan adalah,

$$I = w_{ji} x_i \tag{8.5}$$

dimana x_i, adalah sinyal masukan dan w_{ji} adalah penimbangnya. Bila I>T maka keluaran O = 1 sebaliknya O = 0 dimana T adalah treshold.

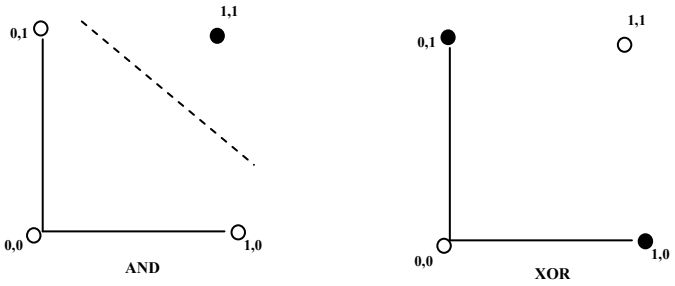
Pelatihan pada perseptron dilakukan dengan merubah nilai penimbangnya sehingga sesuai dengan kebutuhan yang dilakukan dengan membandingkan keluaran dari jaringan dengan targetnya dan proses tersebut dituliskan seperti berikut ini:

$$wbaru_{ji} = wlama_{ji} + \alpha(t_j - O_j)x_i \tag{8.6}$$

t_j adalah target, dana adalah bilangan konstan bernilai kecil antara 0,1 sampai 0,9 yang disebut sebagai laju pelatihan (learning rate).

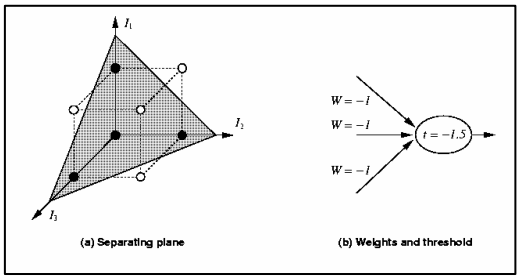
Proses ini dijalankan pada setiap neuron yang ada pada setiap lapisan (layer) dari jaringan sampai penimbang tersebut sesuai dengan yang diinginkan. Nilai awal penimbang adalah bilangan kecil antara 0-1 yang dibangkitkan secara acak.

Perseptron dapat dipresentasikan dan hanya fungsi pemisah linear yang dapat dipisahkan dengan perseptron seperti pada gambar 8.11.



Gambar 8.11 Fungsi pemisah linear untuk perseptron

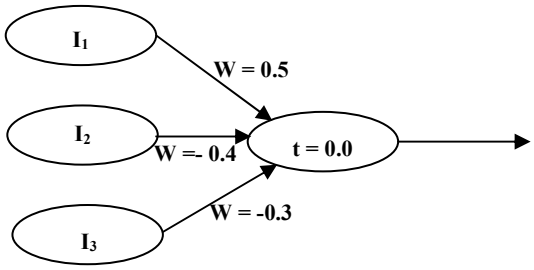
Fungsi pemisah linear dapat digunakan untuk lebih dari tiga variabel tapi sangat sulit untuk digambarkan seperti pada gambar 8.12



Gambar 8.12 Fungsi pemisah linear untuk tiga variabel

8.2.1.2 Pelatihan Sebuah Single Perseptron

Pelatihan pada perseptron dilakukan dengan merubah nilai penimbangnya sehingga sesuai dengan kebutuhan yang dilakukan dengan membandingkan keluaran dari jaringan dengan targetnya. Contoh kasus operasi And seperti pada tabel 8.1. Mula-mula dilakukan random pada bobot (W) kemudian dilakukan perhitungan perkalian antara input (I) dan bobot (W) seperti pada gambar 8.13.



I_1	I_2	I_3	Penjumlahan	Output
0	0	1	(0*0.5) + (0*-0.4) + (1*-0.3) = -0.3	0
0	1	1	(0*0.5) + (1*-0.4) + (1*-0.3) = -0.7	0
1	0	1	(1*0.5) + (0*-0.4) + (1*-0.3) = 0.2	1
1	1	1	(1*0.5) + (1*-0.4) + (1*-0.3) = -0.2	0

Gambar 8.13 Perhitungan pelatihan perseptron

Output diperoleh dengan melakukan fungsi aktifasi dalam kasus AND dilakukan dengan unit step yaitu bila kurang dari nol output = 0, bila lebih output = 1. Error diperoleh apabila terjadi perbedaan antara target dengan output jaringan. Jika error tidak sama dengan nol maka bobot (W) diperbaiki.

```
While(epoch menghasilkan error)
        …..
        Err = T – O
        If Err <> 0 then
                Wj = Wj + LR * Ij * Err
        End If
End
```

Keterangan:

Epoch : Set training pada jaringan

KasusAND epoch berisi 4 set dari input dipresentasikan pada jaringan([0,0], [0,1], [1,0], [1,1])

Nilai Training, T : Training sebuah jaringan

Kita tidak hanya memperhatikan input tetapi juga nilai yang dihasilkan jaringan. Contoh jika kita mempresentasikan jaringan dengan[1,1] untuk fungsi AND nilai training 1

Error, Err : Nilai error

adalah perbedaan nilai output dari jaringan dengan nilai training. Contoh jika kita membutuhkan jaringan untuk

training 0 dan output 1, maka Err = -1

Output dari Neuron, O :

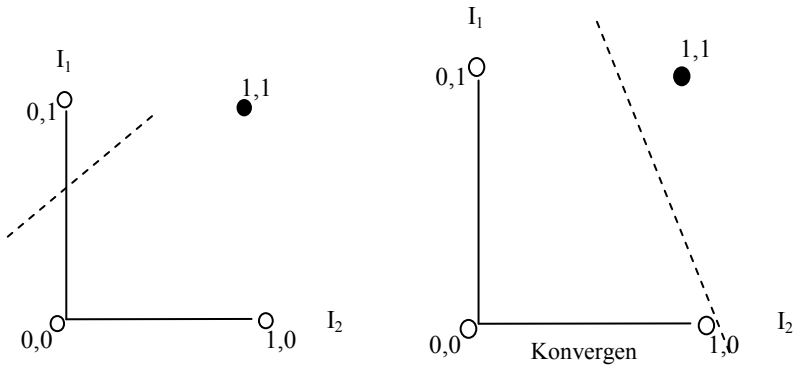
Ij : Input dari neuron

Wj : Bobot dari input neuron (I**j**) ke output neuron

LR : Learning Rate.

Bagaimana membuat jaringan konvergen? Dari gambar 8.14 mula-mula bobot dirandom misalnya diperoleh (0.1) kemudian dilakukan perbaikan bobot terus menerus sehingga diperoleh persamaan garis yang konvergen artinya dapat membagi dua kelompok dengan baik.

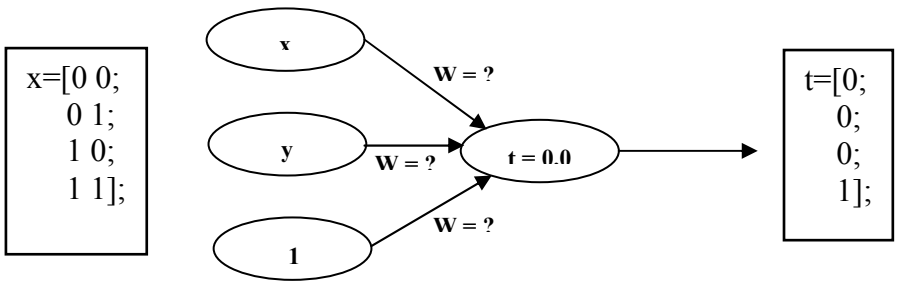
Catatan
I_1 point = W_0/W_1
I_2 point = W_0/W_2



Gambar 8.14 Proses jaringan konvergen

8.2.1.3 Contoh dan Program Jaringan Single Perseptron

Pada kasus operasi AND diatas dapat dibuat suatu disain arsitektur jaringan seperti pada gambar 8.15.



Gambar 8.15. Arsitektur jaringan Operasi AND

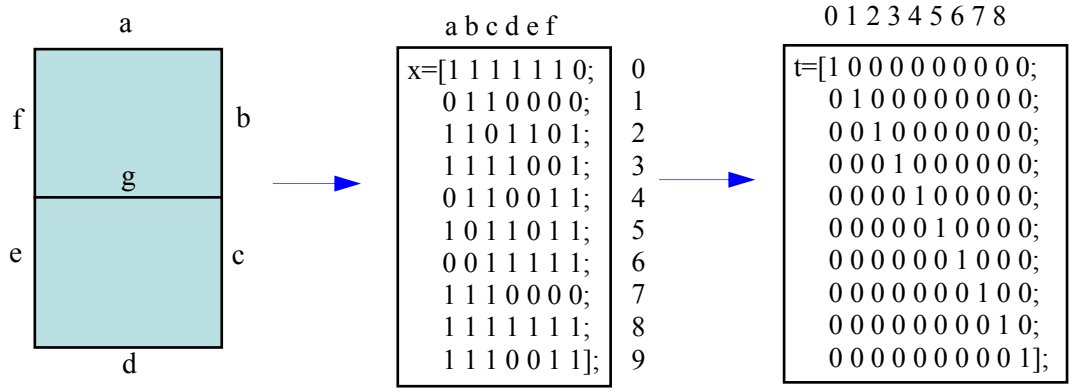
- Langkah penyelesaian:
    - Struktur JST yang digunakan adalah 2-lapis yaitu 2-unit masukan kode biner (00-01-10-11) ditambah 1-bias dan 1-unit keluaran kode biner (0 atau 1).
    - Inisialisasi penimbang diambil bilangan acak antara -0.15 sampai 0.15 sebanyak 3 penimbang yang menghubungkan antara unit pada lapisan masukan dengan unit pada lapisan keluaran.

- Besarnya nilai laju pelatihan ditentukan:0,1

- Program:

```
#include<iostream.h>
#include<stdlib.h>

/*---------------------- random function -----------------------*/
float  d_rand( void )
{
        return ( ( (float)(rand() % 32767) / 32767.0 - 0.5 ) * 2.0 ) ;
}

main()
{
        int i,j,out,ERR;
        int x[3][4]={{0,0,1,1},
                     {0,1,0,1},
                     {1,1,1,1}};
        int T[4]={0,0,0,1};
        float w[3],O,LR=0.1,init=0.15;
        //inisialisasi bobot
        for(i=0;i<3;i++)
        {
                w[i]=init*d_rand();
        }
        //training
        for(i=0;i<10;i++)
        {
        for(j=0;j<4;j++)
        {
                O=x[0][j]*w[0]+x[1][j]*w[1]+x[2][j]*w[2];
                if(O>0.0)
                        out=1;
                else
                        out=0;
                ERR=T[j]-out;
                if(ERR!=0)
                {
                        w[0]=w[0]+LR*x[0][j]*ERR;
                        w[1]=w[1]+LR*x[1][j]*ERR;
                        w[2]=w[2]+LR*x[2][j]*ERR;
                }
        }
        cout<<i<<":"<<ERR<<endl;
        }
        //running
        cout<<"X1=";
        cin>>x[0][0];
        cout<<"X2=";
        cin>>x[1][0];
        O=x[0][0]*w[0]+x[1][0]*w[1]+x[2][0]*w[2];
        cout<<O<<endl;
        if(O>0.0)
                cout<<"Output AND=1"<<endl;
        else
                cout<<"Output AND=0"<<endl;
}
```

Pada kasus aplikasi perseptron multi output untuk mengenali angka desimal yang terbentuk dari BCD 7-segment. Sebagai contoh bagaimana proses perseptron mengenali angka dari 0-9 terlihat seperti pada gambar 8.16.



Gambar 8.16. Pengenalan angka desimal dari BCD 7-Segment

- Langkah penyelesaian:
  - Struktur JST yang digunakan adalah 7-lapis yaitu 7-unit masukan kode biner (0000000-1111111) ditambah 1-bias dan 10-unit keluaran kode biner (1000000000 - 0000000001).
  - Inisialisasi penimbang diambil bilangan acak antara -0.15 sampai 0.15 sebanyak 7 unit pada lapisan masukkan dikali 10 unit pada lapisan keluaran (7*10).
  - Besarnya nilai laju pelatihan ditentukan:0,1
- Program:

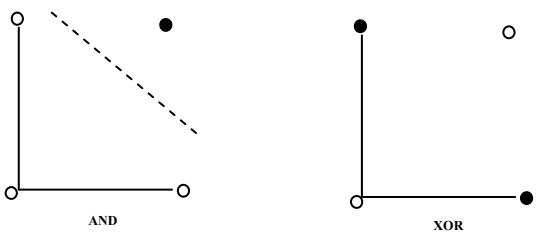
```
#include<iostream.h>
#include<stdlib.h>

/*---------------------- random function -----------------------*/
float  d_rand( void )
{
  return ( ( (float)(rand() % 32767) / 32767.0 - 0.5 ) * 2.0 ) ;
}

void main()
{
        int i,j,p,l,out[10],ERR[10],jum;
        int x[10][8]={{1,1,1,1,1,1,0,1},
                      {0,1,1,0,0,0,0,1},
                      {1,1,0,1,1,0,1,1},
                      {1,1,1,1,0,0,1,1},
                      {0,1,1,0,0,1,1,1},
                      {1,0,1,1,0,1,1,1},
                      {0,0,1,1,1,1,1,1},
                      {1,1,1,0,0,0,0,1},
                      {1,1,1,1,1,1,1,1},
                      {1,1,1,0,0,1,1,1}};
```

```
int T[10][10]={{1,0,0,0,0,0,0,0,0,0},
               {0,1,0,0,0,0,0,0,0,0},
               {0,0,1,0,0,0,0,0,0,0},
               {0,0,0,1,0,0,0,0,0,0},
               {0,0,0,0,1,0,0,0,0,0},
               {0,0,0,0,0,1,0,0,0,0},
               {0,0,0,0,0,0,1,0,0,0},
               {0,0,0,0,0,0,0,1,0,0},
               {0,0,0,0,0,0,0,0,1,0},
               {0,0,0,0,0,0,0,0,0,1}};
float w[10][8],O[10],LR=0.1f,init=0.15f,error;
//inisialisasi bobot
for(j=0;j<10;j++)
{
        for(i=0;i<8;i++)
        {
                w[j][i]=init*d_rand();
        }
}
//training
for(l=0;l<100;l++)
{
error=0.0;jum=1;
for(p=0;p<10;p++)
{
        for(j=0;j<10;j++)
        {
                O[j]=0.0;
                for(i=0;i<8;i++)
                {
                        O[j]=O[j]+x[p][i]*w[j][i];
                }
                if(O[j]>0.0)
                        out[j]=1;
                else
                        out[j]=0;
                ERR[j]=T[p][j]-out[j];
                error=error+abs(ERR[j]);
                jum++;
                if(ERR[j]!=0)
                {
                        for(i=0;i<8;i++)
                        {
                                w[j][i]=w[j][i]+LR*x[p][i]*ERR[j];
                        }
                }
        }
}
error=error/jum;
cout<<l<<":"<<error<<endl;
}
//running
for(i=0;i<7;i++)
{
        cout<<"X"<<i+1<<":";
        cin>>x[0][i];
}
```

```
        for(j=0;j<10;j++)
        {
                O[j]=0.0;
                for(i=0;i<8;i++)
                {
                        O[j]=O[j]+x[0][i]*w[j][i];
                }
                if(O[j]>0.0)
                        out[j]=1;
                else
                        out[j]=0;
                cout<<O[j]<<endl;
        }
        if((out[0]==1)&&(out[1]==0)&&(out[2]==0)&&(out[3]==0)&&(out[4]==0)&&(out[5]==0)&&
(out[6]==0)&&(out[7]==0)&&(out[8]==0)&&(out[9]==0))cout<<"Output=0"<<endl;
        if((out[0]==0)&&(out[1]==1)&&(out[2]==0)&&(out[3]==0)&&(out[4]==0)&&(out[5]==0)&&
(out[6]==0)&&(out[7]==0)&&(out[8]==0)&&(out[9]==0))cout<<"Output=1"<<endl;
        if((out[0]==0)&&(out[1]==0)&&(out[2]==1)&&(out[3]==0)&&(out[4]==0)&&(out[5]==0)&&
(out[6]==0)&&(out[7]==0)&&(out[8]==0)&&(out[9]==0))cout<<"Output=2"<<endl;
        if((out[0]==0)&&(out[1]==0)&&(out[2]==0)&&(out[3]==1)&&(out[4]==0)&&(out[5]==0)&&
(out[6]==0)&&(out[7]==0)&&(out[8]==0)&&(out[9]==0))cout<<"Output=3"<<endl;
        if((out[0]==0)&&(out[1]==0)&&(out[2]==0)&&(out[3]==0)&&(out[4]==1)&&(out[5]==0)&&
(out[6]==0)&&(out[7]==0)&&(out[8]==0)&&(out[9]==0))cout<<"Output=4"<<endl;
        if((out[0]==0)&&(out[1]==0)&&(out[2]==0)&&(out[3]==0)&&(out[4]==0)&&(out[5]==1)&&
(out[6]==0)&&(out[7]==0)&&(out[8]==0)&&(out[9]==0))cout<<"Output=5"<<endl;
        if((out[0]==0)&&(out[1]==0)&&(out[2]==0)&&(out[3]==0)&&(out[4]==0)&&(out[5]==0)&&
(out[6]==1)&&(out[7]==0)&&(out[8]==0)&&(out[9]==0))cout<<"Output=6"<<endl;
        if((out[0]==0)&&(out[1]==0)&&(out[2]==0)&&(out[3]==0)&&(out[4]==0)&&(out[5]==0)&&
(out[6]==0)&&(out[7]==1)&&(out[8]==0)&&(out[9]==0))cout<<"Output=7"<<endl;
        if((out[0]==0)&&(out[1]==0)&&(out[2]==0)&&(out[3]==0)&&(out[4]==0)&&(out[5]==0)&&
(out[6]==0)&&(out[7]==0)&&(out[8]==1)&&(out[9]==0))cout<<"Output=8"<<endl;
        if((out[0]==0)&&(out[1]==0)&&(out[2]==0)&&(out[3]==0)&&(out[4]==0)&&(out[5]==0)&&
(out[6]==0)&&(out[7]==0)&&(out[8]==0)&&(out[9]==1))cout<<"Output=9"<<endl;
}
```

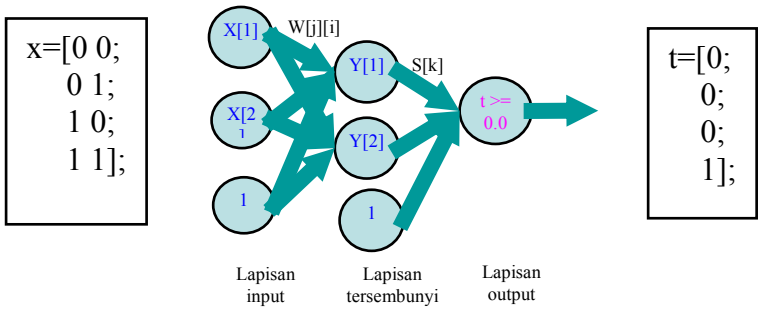
8.2.1.4 Multi Perceptron

Single perceptron membagi input dalam dua class dan vektor bobot dari single perceptron terbagi dalam persamaan hyperplane. Persamaan harus linear seperable. Untuk mengatasi permasahan yang tidak linear seperable seperti gambar 8.17 maka harus mengatur input menjadi linear seperable.



Gambar 8.17. Persamaan multi perceptron yang tidak linear seperable

Pada kasus operasi XOR tidak bisa diselesaikan dengan single perceptron tetapi harus dengan multi perceptron. Disain arsitektur jaringan untuk operasi XOR seperti pada gambar 8.18.



Gambar 8.18. Arsitektur jaringan Operasi XOR

- Langkah penyelesaian:

  Lapisan Input: N input {Xi}, i=0,1,…,N-1

  Lapisan Tersembunyi: M neuron $Y_i = \sum w_{ji} x_i$

  Lapisan Output: 1 neuron $z = \sum s_k y_k$

  Proses Pembelajaran:

  Input: $\bar{x}$

  Minimum Square Error

$$E = \frac{1}{2}(T_{\bar{x}} - O_{\bar{x}})^2$$

  Pembelajaran dengan merubah parameter w

$$w \leftarrow w - \eta \frac{\partial E}{\partial w}$$

  Pembelajaran untuk lapisan output

$$\frac{\partial E}{\partial s_k} = (T_{\bar{x}} - O_{\bar{x}}) \frac{\partial z}{\partial s_k} = (T_{\bar{x}} - O_{\bar{x}}) g' s_k$$

$$g' = \frac{dg(x)}{dx}, x = \sum_k s_k y_k$$

$$s_k \leftarrow s_k - \eta (T_{\bar{x}} - O_{\bar{x}}) g' y_k$$

  Pembelajaran pada lapisan tersembunyi

$$\frac{\partial E}{\partial w_{ji}} = (T_{\bar{x}} - O_{\bar{x}}) \sum_k \frac{\partial z}{\partial y_k} \frac{\partial y_k}{\partial w_{ji}} = (T_{\bar{x}} - O_{\bar{x}}) g' s_k$$

$$= (T_{\bar{x}} - O_{\bar{x}}) \sum_k g' s_k f' \delta_{ki} x_j = (T_{\bar{x}} - O_{\bar{x}}) g' s_k f' x_j$$

$$f' = \frac{df(x)}{dx}, x = \sum_j w_{ji} x_j$$

$$w_{ji} \leftarrow w_{ji} - \eta(T_{\bar{x}} - O_{\bar{x}}) g' s_i f' x_j$$

- Program:

```
#include<iostream.h>
#include<stdlib.h>
#include<math.h>

/*---------------------- random function ----------------------*/
float  d_rand( void )
{
  return ((float)(((rand() % 32767) / 32767.0 - 0.5 ) * 2.0)) ;
}
/*------------------------- sigmoid -----------------------------*/
float  sigmoid(float u)
{
  return ((float)(1.0/(1.0 + exp(-u))));
}

void main()
{
        int i,j,p,l;
        float z,delta_o,delta_h[2],g1,f1[2];
        float y[3]={0.0,0.0,1.0};
        int x[4][3]={{0,0,1},
                                {0,1,1},
                                {1,0,1},
                                {1,1,1}};
        float t[4]={0.0,1.0,1.0,0.0};
        float w[2][3],O[2],s[3],LR=0.1f,init=0.15f,error;
        //inisialisasi bobot
        for(j=0;j<2;j++)
        {
                for(i=0;i<3;i++)
                {
                        w[j][i]=init*d_rand();
                }
        }
        for(j=0;j<3;j++)
        {
                s[j]=init*d_rand();
        }
        //training
        for(l=0;l<15000;l++)
        {
        error=0.0;
        for(p=0;p<4;p++)
        {
                for(j=0;j<2;j++)
                {
                        O[j]=0.0;
                        for(i=0;i<3;i++)
                        {
                                O[j]=O[j]+x[p][i]*w[j][i];
                        }
```

```
                        y[j]=sigmoid(O[j]);
                }
                O[0]=0.0;
                for(i=0;i<3;i++)
                {
                        O[0]=O[0]+y[i]*s[i];
                }
                z=sigmoid(O[0]);
                g1=z*(1-z);
                delta_o=(t[p]-z)*g1;
                for(j=0;j<2;j++)
                {
                        f1[j]=y[j]*(1-y[j]);
                }
                for(j=0;j<2;j++)
                {
                        delta_h[j]=f1[j]*delta_o*s[j];
                }
                for(i=0;i<3;i++)
                {
                        s[i]=s[i]+LR*delta_o*y[i];
                }
                for(j=0;j<2;j++)
                {
                        for(i=0;i<3;i++)
                        {
                                w[j][i]=w[j][i]+LR*delta_h[j]*x[p][i];
                        }
                }
                error=error+((t[p]-z)*(t[p]-z))/2;
        }
        error=error/4;
        cout<<l<<":"<<error<<endl;
        }
        //running
        for(i=0;i<2;i++)
        {
                cout<<"X"<<i+1<<":";
                cin>>x[0][i];
        }
        for(j=0;j<2;j++)
        {
                O[j]=0.0;
                for(i=0;i<3;i++)
                {
                        O[j]=O[j]+x[0][i]*w[j][i];
                }
                y[j]=sigmoid(O[j]);
        }
        O[0]=0.0;
        for(i=0;i<3;i++)
        {
                O[0]=O[0]+y[i]*s[i];
        }
        z=sigmoid(O[0]);
        cout<<z<<endl;
        if(z<0.5)
                cout<<"Output=0"<<endl;
        else
                cout<<"Output=1"<<endl;
}
```

8.2.1.5 Metode Backpropagation

Algoritma pelatihan Backpropagasi (Back Propagation) atau ada yang menterjemahkannya menjadi propagasi balik, pertama kali dirumuskan oleh Werbos dan dipopulerkan oleh Rumelhart dan McClelland untuk dipakai pada JST, dan selanjutnya algoritma ini biasa disingkat dengan BP. Algoritma ini termasuk metoda pelatihan supervised dan didesain untuk operasi pada jaringan feed forward multi lapis.

Metoda BP ini banyak diaplikasikan secara luas. Sekitar 90 %, bahkan lebih BP telah berhasil diaplikasikan di berbagai bidang, diantaranya diterapkan di bidang finansial, pengenalan pola tulisan tangan, pengenalan pola suara, sistem kendali, pengolah citra medika dan masih banyak lagi keberhasilan BP sebagai salah satu metoda komputasi yang handal.

Algoritma ini juga banyak dipakai pada aplikasi pengaturan karena proses pelatihannya didasarkan pada hubungan yang sederhana, yaitu : Jika keluaran memberikan hasil yang salah, maka penimbang (Weight) dikoreksi supaya galatnya dapat diperkecil dan respon jaringan selanjutnya diharapkan akan lebih mendekati harga yang benar. BP juga berkemampuan untuk memperbaiki penimbang pada lapisan tersembunyi (hidden layer).

Secara garis besar, mengapa algoritma ini disebut sebagai propagasi balik, dapat dideskripsikan sebagai berikut: Ketika Jaringan diberikan pola masukan sebagai pola pelatihan maka pola tersebut menuju ke unit-unit pada lapisan tersembunyi untuk diteruskan ke unit-unit lapisan keluaran. Kemudian unit-unit lapisan keluaran memberikan tanggapan yang disebut sebagai keluaran jaringan. Saat keluaran jaringan tidak sama dengan keluaran yang diharapkan maka keluaran akan menyebar mundur (backward) pada lapisan tersembunyi diteruskan ke unit pada lapisan masukan. Oleh karenanya maka mekanisme pelatihan tersebut dinamakan backpropagation/propagasi balik.

Tahap pelatihan ini merupakan langkah bagaimana suatu jaringan saraf itu berlatih, yaitu dengan cara melakukan perubahan penimbang (sambungan antar lapisan yang membentuk jaringan melalui masing-masing unitnya). Sedangkan

pemecahan masalah baru akan dilakukan jika proses pelatihan tersebut selesai , fase tersebut adalah fase mapping atau proses pengujian/testing.

Algoritma Pelatihan Back Propagasi terdiri dari dua proses, feed forward dan backpropagation dari galatnya. Untuk jelasnya dapat dijelaskan rinciannya sebagai berikut :

Langkah 0:

- Pemberian inisialisasi penimbang (diberi nilai kecil secara acak)

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

Propagasi maju (Feedforward)

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 ditambah dengan biasnya:

$$Z\_in_j = V_{0j} + \sum_{i=1}^{n} X_i V_{ij} \tag{8.7}$$

Kemudian menghitung sesuai dengan fungsi aktifasi yang digunakan:

$$Z_j = f(Z\_in_j) \tag{8.8}$$

bila yang digunakan adalah fungsi sigmoid maka bentuk fungsi tersebut adalah:

$$Z_j = \frac{1}{1+\exp^{(-z\_in_j)}} \tag{8.9}$$

Kemudian mengirim sinyal tersebut ke semua unit keluaran (unit keluaran).

Langkah 5 :

- Masing-masing unit keluaran (y_k., k=l,2,3...m) dikalikan dengan faktor penimbang dan dijumlahkan:

$$Y\_in_k = W_{ok} + \sum_{j=1}^{p} ZjW_{jk} \tag{8.10}$$

Menghitung kembali sesuai dengan fungsi aktifasi

$$y_k = f(y\_in_k) \tag{8.11}$$

BackPropagasi dan Galatnya

Langkah 6:

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

$$\delta_k = (t_k - y_k) f'(y\_in_k) \tag{8.12}$$

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

$$\begin{aligned} f'(y\_in_k) &= f(y\_in_k)(1 - f(y\_in_k)) \\ &= y_k (1 - y_k) \end{aligned} \tag{8.13}$$

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

$$\Delta W_{kj} = \alpha.\partial_k.Z_j \tag{8.14}$$

Menghitung perbaikan koreksi:

$$\Delta W_{0k} = \alpha.\partial_k \tag{8.15}$$

dan menggunakan nilai δ_k pada semua unit lapisan sebelumnya.

Langkah 7 :

- masing-masing penimbang yang menghubungkan unit-unit lapisan keluaran dengan unit-unit pada lapisan tersembunyi (Z_j ,j=1..,p) dikalikan delta dan dijumlahkan sebagai masukan ke unit-unit lapisan berikutnya.

$$\delta\_in_j = \sum_{k=1}^{m} \delta_k W_{jk} \tag{8.16}$$

Selanjutnya dikalikan dengan turunan dari fungsi aktifasinya untuk menghitung galat.

$$\delta_j = \delta\_in_j f'(y\_in_j) \tag{8.17}$$

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

$$\Delta V_{ij} = \alpha \delta_j X_i \tag{8.18}$$

Kemudian menghitung perbaikan bias (untuk memperbaiki v_{0j})

$$\Delta V_{0j} = \alpha \delta_j \tag{8.19}$$

Memperbaiki penimbang dan bias

Langkah 8:

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

$$W_{jk}(baru) = W_{jk}(lama) + \Delta W_{jk} \tag{8.20}$$

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

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

Langkah 9 :

Uji kondisi pemberhentian (akhir iterasi).

Daftar Notasi

X^p = Pola masukan pelatihan ke-p, p=1,2,. . ..,p<=1

$X^p = (X_1, X_z, X_3,...,X_n)$

t^p = pola keluaran target dari pelatihan

$t^p = (t_1, t_2, t_3, . . . ,t_n)$

xi = Unit ke-I pada lapisan masukan

Xi = nilai aktifasi dari unit X_i

Z_j = Unit ke-j pada lapisan tersembunyi

Z_in_j = keluaran untuk unitZ_j

z_j = nilai aktifasi dafi unit Z_j

Y_k = unit ke -k pada lapisan keluaran

Y_in_k = net masukan untuk unit Y_k

Y_k = nilai aktifasi dari 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 antara $W_{kj}(t)$ dengan$W_{kj}(t+1)$

Vi_0 = nilai penimbang sambungan pada bias untuk unit Z_i

V_{ij}: nilai penimbang sambungan dari unit X_i ke unit Z_i

∇W_{ij} = selisih antara V_{ij} (t) dengan$V_{ij}(t+1)$

δ_k = faktor pengaturan nilai penimbang sambungan pada lapisan keluaran

δ_j = faktor pengaturan nilai penimbang sambungan pada lapisan tersembunyi

α = konstanta laju pelatihan (leaming rate) $0<\alpha<1$

E = Total galat

8.2.2 Metode Pelatihan Tak Terbimbing/Mandiri

Metoda pelatihan tak terbimbing adalah pelatihan tanpa memerlukan target pada keluarannya. Proses pelatihan berdasarkan proses transformasi dari bentuk variabel kontinyu menjadi variabel diskrit yang dikenal dengan kuantisasi vektor. Jaringan yang digunakan untuk proses pelatihan tak terbimbing ini adalah jaringan umpan balik (feedback network).

8.2.2.1 Metoda Kohonen / Self-Organizing Map (SOM)

Teknik self-organizing map (SOM) dikenalkan pertama kali oleh Teuvo Kohonen, merupakan proses unsupervised learning yang mempelajari distribusi himpunan pola-pola tanpa informasi kelas.

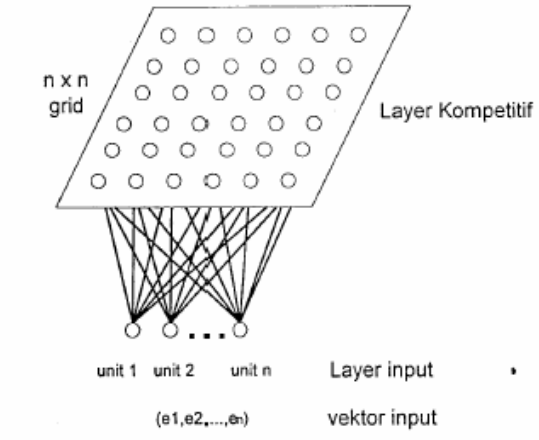
Ide dasar teknik ini diilhami dari bagaimana proses otak manusia menyimpan gambar/pola yang telah dikenalinya melalui mata, kemudian mampu mengungkapkan kembali gambar/pola tersebut. Pada mata kita proses tersebut adalah realisasi pemetaan lmapping dari retina menuju cortex. oleh karenanya aplikasi model JST ini banyak digunakan pada pengenalan obyek/citra visual (visual image).

Proses pemetaan terjadi bila sebuah pola berdimensi bebas diproyeksikan dari ruang masukan ke posisi pada array berdimensi satu atau dua. Metoda ekstraksi informasi tersebut dapat dinyatakan sebagai observasi terhadap sebuah pola melalui

jendela yang terbentuk oleh lokasi unit-unit luasan pola. pola yang dikenali hanya pola yang batasan unit lokasinya jelas berbeda, biasanya observasi hanya dapat dilakukan bila lokasi pola tersebut mendapat iluminasi/pencahayaan yang cukup/normal.

Meskipun SOM adalah proses klasifikasi, namun tidak seperti teknik klasifikasi atau pengelompokan yang umum digunakan, yang hanya menyediakan penataan kelas-kelas berdasarkan topologinya.Kemiripan pada pola masukan dipertahankan agar tidak berubah sampai pada keluaran proses. Topologi untuk mempertahankan pola kemiripan pada proses SOM membuatnya berguna sekali, khususnya pada klasifikasi data yang memiliki jumlah kelas yang besar. Pada klasifikasi sampel subcitra, sebagai contoh, mungkin ada sejumlah besar kelas yang perubahannya dari satu kelas ke kelas selanjutnya tidak begitu jauh (membuatnya sulit untuk mendefiniskan batas kelas yang jelas).

Pada jaringan saraf tiruan ini lapisan masukan (pertama) terhubung secara penuh dengan lapisan kompetitif (kedua). Jadi setiap unit masukan terhubung ke semua unit keluaran dan pada hubungan tersebut terdapat nilai penimbang (weight) tertentu.



Gambar 8.19. Struktur Dasar Pelatihan Mandiri Kohonen

8.2.2.2 Algoritma Pelatihan Mandiri (SOM)

Berikut ini adalah tahapan/argoritma dalam pelatihan mandiri kohonen :

- Menentukan unit keluaran pemenang yaitu yang memiliki derajat kemiripan terbesar diantara semua vektor penimbang w_i dan vektor masukan e. Metoda jarak Euclidean digunakan untuk menentukan unit pemenangnya.

$$\|x - w_c\| = \min_i \|x - w_i\| \tag{8.21}$$

dimana c adalah indek pemenang.

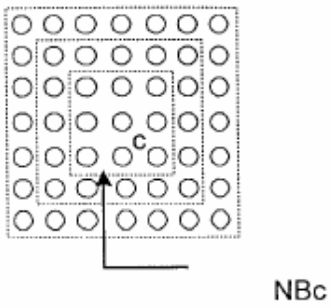
- Menentukan tetangga disekitar pemenang. Memperbarui penimbang pemenang dan tetangganya.

$$\Delta w_i = \eta(x - w_i), \qquad i \in NB_c \tag{8.22}$$

dimana NB_c adalah tetangga disekitar pemenang c, dan η adalah laju pelatihan. Untuk menentukan tetangga dapat menggunakan fungsi Gaussian:

$$\Omega_c(i) = \exp\left(\frac{-\|p_i - p_c\|^2}{2\sigma^2}\right) \tag{8.23}$$

dimana p_i dan p_c adalah posisi keluaran dan σ adalah cakupan tetangga

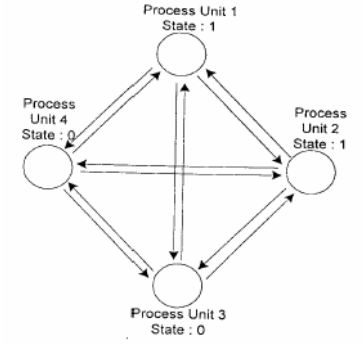


Gambar 8.20. Tetangga di sekitar pemenang

8.2.2.3 Metoda Hopfield

Metoda ini dikembangkan oleh John Hopfield pada tahun 1980. Cara pemahaman paling mudah JST Hopfield bila dinyatakan sebagai sebuah memori asosiatip (associative memory) yang gambaran umumnya adalah sebagai berikut:

Bila ada sebuah JST yang terbentuk dari N X N neuron dimana N adalah jumlah variabel dari obyek yang akan dioptimasi. Setiap neuron terhubung penuh satu sama lainnya. Ilustrasi pernyataan tersebut adalah gambar x.9.



Gambar 8.21. Layout dari JST Hopfield

JST Hopfield merupakan salah satu metoda optimasi untuk pencarian nilai minimum dari kombinasi fungsi obyektif. Sebagai contoh untuk menjelaskan metoda ini secara rinci digambarkan/dianalogikan sebagai suatu problema penyelesaian permasalahan rute perjalanan salesman/pramuniaga. permasalahan ini dituntut untuk menentukan rute/jalur terpendek yang bisa ditempuh oleh seorang sales dengan beberapa ketentuan antara lain :

l. Setiap kota harus dikunjungi satu kali saja.

2. Setiap singgah hanya mungkin untuk mengunjungi satu kota saia.

3. Dicari rute dengan total jarak minimum.

Jika terdapat 4 kota maka dapat didefinisikan matrik 4 x 4 berisikan neuron / elemen biner sebagai berikut :

Tabel 8.4 Contoh neuron permasalahan rute sales

		Kunjungan ke :			
		1	2	3	4
kota	1	1	0	0	0
	2	0	0	1	0
	3	0	1	0	0
	4	0	0	0	1

Terlihat bahwa setiap kolom hanya ada 1 neuron yang bernilai "l" dan setiap baris hanya terdapat neuron yang bernilai "l,'. Dengan maksud bahwa hanya

diperbolehkan mengunjungi satu kota saja dalam tiap kunjungan, dan setiap kota hanya diperbolehkan dikunjugi satu kali saja. contoh kota ke-3 dikunjugi pada urutan ke - 2 setelah kota pertama. Berikutnya kota ke tiga dan terakhir kota ke - 4.

Dari tetapan tetapan tersebut dapat disusun sebuah persamaan energi yang merupakan gabungan dari beberapa fungsi obyektif sebagai berikut :

$$E = A \sum_i \sum_k \sum_{j \neq k} V_{ki} V_{ji} + B \sum_i \sum_k \sum_{j \neq k} V_{ki} V_{ji} + \sum_i \sum_k \sum_{j \neq k} C \left[\left(\sum_{i,k} V_{ik} \right) - n' \right]^2$$

$$+ D \sum_k \sum_{j \neq k} \sum_i d_{kj} V_{ki} \left(V_{j,i+1} + V_{j,i-1} \right) \qquad (8.24)$$

Dimana:

A,B,C,D,n’ = Konstanta

V_{xy} = Aktifasi antara neuron x dan neuron y

d_{xy} = Parameter jarak neuron x dan neuron y

Pada setiap notasi penjumlahan tersebut, dimaksudkan mempunyai batas (range) dari 1 ke n dimana n mengacu pada indek yang berhubungan dengan permasalahan, untuk problema perjalanan pramuniaga tersebut adalah jumlah kota.

Persamaan energi tersebut merupakan implementasi dari tetapan - tetapan yang digunakan dalam masalah ini. pada suku pertama persamaan (3.59), jika merupakan rute yang valid menyatakan bahwa tidak boleh lebih dari satu elemen yang bernilai "1" pada setiap barisnya, ini berarti bahwa tidak ada satu kota yang dikunjungi lebih dari satu kali.

Suku kedua jika merupakan rute yang valid menyatakan bahwa tidak boleh lebih dari satu elemen yang bernilai "1" pada setiap kolomnya, ini berarti bahwa tidak mungkin dua kota dikunjungi dalam waktu yang sama.

Pada suku ketiga terdapat variabel dpi ini menunjukkan jarak kota k ke kota j, atau parameter lainnya sesuai dengan permasalahan yang dioptimasi.

Sebagai contoh perhitungan berikut matrik jarak antara empat kota yang ada :

Tabel 8.5 Matrik jarak empat kota

	1	2	3	4
1	0	10	14	7
2	10	0	6	12
3	14	6	0	9
4	7	12	9	0

Seperti telah dijelaskan bahwa masing-masing suku pada persamaan energi menggambarkan keadaan dari suatu rute yang didapat. Kita tetapkan suatu rute perjalanan sebagai berikut, kota pertama kota 1, diteruskan ke kota 2 kemudian kota 3 dan terakhir 4

sebelum kembali ke kota 1. Secara jelas ditulis kembali rute perjalanan adalah : l-2-3-4-1.

Jika rute ini merupakan sebuah rute yang benar maka suku pertama dan kedua dari persamaan energi akan bernilai nol (0). Suku ketiga dengan sendirinya akan bernilai nol (0). Sehingga berapa jarak total yang ditempuh pada rute tersebut dapat dihitung dengan suku ketiga dari persamaan energi yang dituliskan kembali berikut ini.

$$D\sum_{k}\sum_{j\neq k}\sum_{i} d_{kj} V_{ki}\left(X_{j,i+1} + X_{j,i-1}\right) \qquad (8.25)$$

Diketahui bahwa D adalah konstanta energi dan diambil 0.5, maka persamaan (x.25) dapat dikembangkan sebagai berikut :

$$
\begin{aligned}
&E = D\{ \\
&d12[x12(x23+x21)+ x13(x24+x22)+ x14(x21+x23)]+ \\
&d13[x12(x33+x31)+ x13(x34+x32)+ x14(x31+x33)]+ \\
&d14[x12(x43+x41)+ x13(x44+x42)+ x14(x41+x43)]+ \\
&d21[x21(x12+x14)+ x23(x14+x12)+ x24(x11+x13)]+ \\
&d23[x21(x32+x34)+ x23(x34+x32)+ x24(x31+x33)]+ \\
&d24[x21(x42+x44)+ x23(x44+x42)+ x24(x41+x43)]+ \\
&d31[x31(x12+x14)+ x32(x13+x12)+ x34(x11+x13)]+ \\
&d33[x31(x22+x34)+ x32(x23+x32)+ x34(x21+x23)]+ \\
&d34[x31(x42+x44)+ x32(x43+x42)+ x34(x41+x43)]+ \\
&d41[x41(x12+x14)+ x42(x13+x12)+ x43(x14+x13)]+ \\
&d42[x41(x22+x24)+ x42(x23+x32)+ x43(x24+x23)]+ \\
&d44[x41(x32+x34)+ x42(x33+x42)+ x43(x34+x33)]\}
\end{aligned}
\tag{8.26}
$$

Untuk data jarak antar kota seperti tabel x.2 dapat dibuat matrik jarak sebagai berikut :

$$
D = \begin{bmatrix} d_{11} & d_{12} & d_{13} & d_{14} \\ d_{21} & d_{22} & d_{23} & d_{24} \\ d_{31} & d_{32} & d_{33} & d_{34} \\ d_{41} & d_{42} & d_{43} & d_{44} \end{bmatrix} = \begin{bmatrix} 0 & 10 & 14 & 7 \\ 10 & 0 & 6 & 12 \\ 14 & 6 & 0 & 9 \\ 7 & 12 & 9 & 0 \end{bmatrix}
$$

Sedangkan untuk rute 1 - 2 - 3 - 4 - 1 matrik neuronnya adalah sebagai berikut :

$$
X = \begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \\ x_{41} & x_{42} & x_{43} & x_{44} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Sehingga dengan memasukkan nilai ke persamaan (8.26) didapatkan perhitungan sebagai berikut :

$$
\begin{aligned}
E &= 0.5 \{ \\
&10[0(0+1)+ 0(0+0) + 1(1+0)] + 14[0(0+0)+ 0(0+1)+ 1(0+0)] \\
&7[0(1+0)+ 0(0+0)+ 1(0+1)] + 10[1(0+1)+ 0(1+0)+ 0(0+0)] \\
&6[1(1+0)+ 0(0+1)+ 0(0+0)] + 12[1(1+0)+ 0(0+0)+ 0(0+1)] \\
&14[0(0+1)+ 1(0+0)+ 0(0+0)] + 6[0(0+0)+ 1(0+1)+ 0(1+0)] \\
&9[0(0+0)+ 1(1+0)+ 0(0+1)] + 7[0(0+1)+ 0(0+0)+ 1(1+0)] \\
&12[0(0+0)+ 0(0+1)+ 1(0+0)] + 9[0(1+0)+ 0(0+0)+ 1(0+1)]\} \\
&= 0.5 *(10+0+7+10+6+0+0+6+9+7+0+9) \\
&= 0.5 * 64 \\
&= 32
\end{aligned}
$$

Untuk beberapa kombinasi rute yang bisa dicapai tabel 8.3 adalah beberapa rute dengan jarak total yang ditempuh :

Tabel 8.6 Kombinasi rute 4 kota

No.	$X_{ij} \neq 0$	Total jarak	Level energi	Kombinasi rute
1	$X_{ij}, X_{ij}, X_{ij}, X_{ij}$	32	32	1 – 2 – 3 – 4 - 1
2	$X_{ij}, X_{ij}, X_{ij}, X_{ij}$	45	45	1 – 3 – 4 – 2 - 1
3	$X_{ij}, X_{ij}, X_{ij}, X_{ij}$	39	39	1 – 4 – 2 – 3 - 1
4	$X_{ij}, X_{ij}, X_{ij}, X_{ij}$	45	45	1 – 2 – 4 – 3 - 1
5	$X_{ij}, X_{ij}, X_{ij}, X_{ij}$	39	39	3 – 2 – 4 – 1 – 3
6	$X_{ij}, X_{ij}, X_{ij}, X_{ij}$	32	32	2 – 1 – 4 – 3 – 2

8.2.2.4 Komputasi Pada Metoda Hopfiled

Contoh kasus yang telah diuraikan tersebut adalah jarak tempuh minimal untuk mencapai titik optimal. Dalam ukuran matrik yang sangat besar, komputasi diselesaikan oleh program komputer.

Untuk membawa analogi perhitungan tersebut kedalam JST, kita lihat kembali bahasan didepan yang mana disebutkan terdapat sejumlah N X N neuron yang didefinisikan sebagai U_{xi} ymg berarti sebuah neuron itu menghubungkan elemen permasalahan x dengan elemen i. Atau, kota x ditempuh pada kunjungan ke-i. Sebagai contoh U_{xi} menunjukkan bahwa kota ke - x dilewati oleh sales pada kunjungan ke - i. Sehingga terdapat n^2 neuron, dengan nilai "0" dan "1". Dengan nilai "0" menunjukkan tidak ada hubungan antara x dan i, dan nilai “1” berarti kota ke - x disinggahi pada urutan ke - i.

Matrik penimbang (Weight Matrix)

Langkah yang paling penting adalah bagaimana menemukan matrik penimbang. Elemen penimbang diperlukan untuk memisahkan jalur - jalur yang valid dengan jalur - jalur yang tidak valid. Artinya dengan matrik penimbang akan diketahui neuron mana yang valid dan neuron mana yang tidak bisa digunakan.

Jika sebuah neuron yang menggambarkan kota dan nomor kunjungan menggunakan dua index maka elemen penimbang yang menghubungkan dua elemen neuron memerlukan empat (4) index. Contoh, $W_{ik,lj}$ mengacu pada nilai penimbang pada hubungan antaru neuron V_{ik} dan neuron V_{lj} Persamaan dari penimbangnya adalah sebagai berikut :

$$W_{ik,lj} = -A\delta_{il}\left(1-\delta_{kj}\right) - B\delta_{kj}\left(1-\delta_{il}\right) \quad -C - Dd_{il}\left(\delta_{j,k+1}+\delta_{j,k-1}\right) \tag{8.27}$$

Dimana:

A, B, C, D = Konstanta positif.

$W_{ik,lj}$ = Nilai penimbang neuron U_{ik} dan neuron U_{lj}.

Dan, didefinisikan δ_{xy}, sebagai fungsi Kroneckers Delta sebasai berikut :

$\delta_{xy} = 0$ jika $x \neq y$ dan

$\delta_{xy} = 1$ jika $x=y$

Maka hanya ada nilai yang berarti untuk sepasang neuron dari neuron-neuron yang berurutan, dengan kata lain untuk kota yang bisa ditempuh secara berurutan atau berhubungan langsung akan bemilai sebanding dengan jaraknya. Dan untuk kota dimana tidak bisa ditempuh secara berurutan atau dipisahkan oleh kota lain mempunyai nilai konstan.

Musukan

Sebagai nilai masukan dari metoda JST ini bisa dipilih masukan yang berubah- ubah, namun demikian pemilihan masukan ini bisa menyebabkan kondisi minimal dari perhitungan tidak tercapai. Dan untuk menghindari hal tersebut diberikan sinyal tambahan ke neuron sebagai sinyal gangguan berupa angka acak yang berbeda setiap neuron. Banyak dipakai sinyal masukan yang digunakan adalah perkalian sebuah angka konstan dengan jumlah kota ditambah dengan angka acak kecil yang berbeda pada setiap neuronnya.

Aktifasi, Keluaran dan Pembaharuan Neuron

Telah kita definisikan sebelumnya bahwa U_{ij} sebagai aktifasi dari neuron pada baris ke - i dan kolom ke - j dan mempunyai nilai keluaran yang didefinisikan sebagai X_{ij}, sebuah konstanta waktu τ dan sebuah konstanta penguatan λ juga parameter n. Dari konstanta tersebut selanjutnya didefinisikan ∇T sebagai kenaikan nilai aktifasi setiap waktu, dari satu pengulangan ke pengulangan berikutnya.

Sebagai nilai masukan dari metoda JST ini bisa dipilih masukan yang berubah- ubah, namun demikian pemilihan masukan ini bisa menyebabkan kondisi minimal dari perhitungan tidak tercapai. Dan untuk menghindari hal tersebut diberikan sinyal tambahan ke neuron sebagai sinyal gangguan berupa angka acak yang berbeda setiap neuron. Banyak dipakai sinyal masukan yang digunakan adalah perkalian sebuah angka konstan dengan jumlah kota ditambah dengan angka acak kecil yang berbeda pada setiap neuronnya.

Fungsi dari dilakukannya iterasi ini adalah untuk mendapatkan nilai aktifasi sedemikian hingga keluaran yang dicap ai dapatmemenuhi tetapan yang ada.

persamaan yang digunakan dalam memperbarui (updating) nilai aktifasi adalah sebasai berikut :

$$\Delta U_{ij} = \Delta t(\text{term 1} + \text{term 2} + \text{term 3} + \text{term 4} + \text{term 5}) \quad (8.28)$$

dimana:

$$\text{term 1} = -\frac{U_{ij}}{\tau} \qquad \text{term 4} = -C\sum_i\sum_k\left(X_{kj} - n\right)^2$$

$$\text{term 2} = -A\sum_{k\neq j} X_{ik} \qquad \text{term 5} = -D\sum_{k\neq i} d_{ik}\left(X_{k,j+1} + X_{k,j-1}\right)$$

$$\text{term 3} = -B\sum_{k\neq i} X_{kj}$$

U_{ij} = aktifasi dari setiap neuron

X_{ij} = nilai keluaran dari aktifasi yang dimaksud.

d_{ik} = Nilai parameter jarak

Maka perubahan nilai dari aktifasi neuron untuk setiap kali dilakukan pembaharuan diperoleh dengan persamaan sebagai berikut :

$$U_{ij(\text{baru})} = U_{ij(\text{lama})} + \Delta U_{ij} \quad (8.29)$$

Dan keluaran masing - masing aktifasi diperoleh dengan persamaan sebagai berikut :

$$X_{ij(\text{baru})} = \left(1 + \tanh\left(\lambda U_{ij}\right)\right)/2 \quad (8.30)$$

Disini dipakai persamaan hiperbolik tangen dengan 1 sebagai konstanta penguatan. Nilai keluaran dicari pada setiap sekali iterasi dilakukan. Idealnya nilai keluaran yang dihasilkan adalah "0" atau "1", maka diperlukan pendekatan terhadap nilai keluaran yang real ke nilai terdekatnya "0" atau "1".

8.2.3 Metode Pelatihan Hibrida

Semakin tinggi tingkat ketidaklinieran suatu sistem, sering kali sulit di selesaikan dengan kedua metoda yang telah diuraikan sebelumnya yaitu metoda pelatihan terbimbing dan tak terbimbing. Untuk mengatasi problema tersebut, banyak periset yang mencoba menggabungkannya dan diperoleh hasil yang lebih baik. Salah

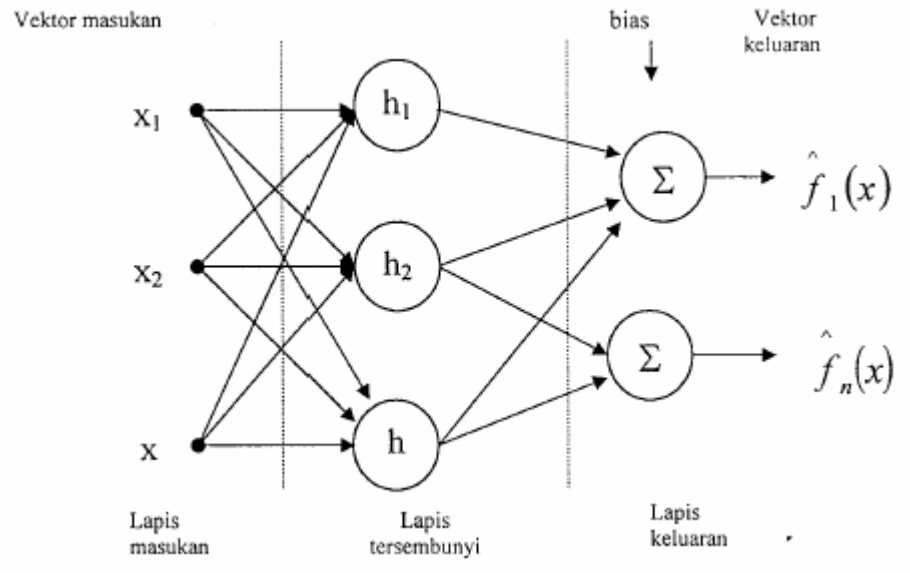
satu metoda hibrida yang berhasil dan banyak digunakan adalah metoda radial basis function network (RBFN)

8.2.3.1 Jaringan Berbasis Fungsi Radial (Radial Basis Function Networks)

Model jaringan ini, neuron-neuron keluarannya adalah hasil kombinasi linier fungsi basis neuron-neuron pada lapisan tersembunyi. Sebagai fungsi basis yang umum digunakan adalah Gaussian. Perbedaan utama antara jaringan multi lapis perceptron (MLP) dengan jaringan berbasis fungsi radial yaitu penggunaan gaussian pada lapisan tersembunyi jaringan RBF, sedangkan jaringan MLP menggunakan fungsi sigmoid.

Pada prinsipnya RBF adalah emulasi sifat jaringan biologi yang umumnya sel/neuron yang paling aktip adalah sel/neuron yang paling sensitip menerima rangsangan sinyal masukan. Sehingga orientasi sensitivitas respon tersebut hanya terhadap beberapa daerah (local response) dalam wilayah masukan. JST dengan lapisan tersembunyi tunggal, pada dasarnya lapisan tersebut berisi neuron-neuron (unit-unit) yang sensitip atau aktip secara lokal. Sedangkan keluarannya terdiri dari unit-unit linear.

Pada unit-unit dalam lapisan tersembunyi, respon unitnya bersifat lokal dan berkurang sebagai fungsi jarak masukan dari pusat unit penerima rangsangan. Metoda ini menjadi terkenal sejak Broomhead dan Lowe's pada tahun 1988 menyampaikan makalahnya yang berjudul "Multivariate functional interpolation and adaptive network ". JST-RBF mempunyai kesamaan dasar dengan JST-MLP yang struktur dasarnya ditunjukkan pada gambar x.10



Gambar 8.22 JST -RBF

Unit-unit pada lapisan tersembunyi menggunakan fungsiaktivasi Ga uss (8.31).

Fungsi Radial

Fungsi Radial adalah suatu fungsi yang mempunyai karakteristik merespon pengurangan ataupun penambahan secara monoton dengan jarak yang berasal dari nilai tengahnya. Jenis fungsi radial yang banyak digunakan adalah fungsi gaussian seperti pada persamaan berikut:

$$h(x) = \phi\left((x-c)^T R^{-1}(x-c)\right) \tag{8.31}$$

dimana ϕ adalah jenis fungsi aktivasi yang digunakan dalam RBF

c adalah pusat (nilai tengah).

R adalah matrik dengan R:r2 dan r merupakan jari-jari skalar

$(x - c)^T R^{-1}(x - c)$ adalah jarak antara input x dan pusat c

dalam matriks yang ditetapkan oleh R.

Beberapa tipe fungsi aktivasi RBF adalah sebagai berikut:

- Fungsi Thinplate-Splane

$$\phi(z,1) = z^2 \log(z) \tag{8.32}$$

- Fungsi Multiquadratic

$$\phi(z,\sigma) = \left(z^2 + \sigma^2\right)^{1/2} \tag{8.33}$$

- Fungsi Inverse Multiquadratic

$$\phi(z,\sigma) = 1/\left(z^2+\sigma^2\right)^{1/2} \tag{8.34}$$

- Fungsi Gauss

$$\phi(z,\sigma) = \exp\left(-z^2/\sigma^2\right) \tag{8.35}$$

Hasil penelitian menyatakan bahwa seleksi dari keempat fungsi nonlinier tersebut tidak dominan menentukan kinerja RBF. Bila jarak Euclidian antara vektor masukan dan unit-unit dalam lapis tersembunyi mempunyai nilai yang berbeda, maka jarak.yang sama untuk setiap unitnya cukup untuk pendekatan secara universal. Ini berarti bahwa semua jarak dapat disesuaikan pada sebuah nilai σ untuk menyederhanakan strategi pelatihannya.

Optimalisasi Vektor Penimbang

Pada metoda Least Square diterapkan pelatihan terbimbing (supervised learning) dengan perhitungan galatnya sebagai fungsi Sum Square Error (SSE)

$$S = \sum_{i=1}^{p}\left(y_i - \hat{f}(x_i)\right)^2 \tag{8.36}$$

dimana

$$f(x) = \sum_{j=1}^{m} w_j h_j(x) \tag{8.37}$$

Jika satu syarat penimbang akhir yang dijumlahkan ke fungsi SSE, seperti pada kasus dengan menggunakan ridge regression maka nilai fungsi mengikuti persamaan berikut:

$$C = \sum_{i=1}^{p}\left(y_i - \hat{f}(x_i)\right)^2 + \sum_{j=1}^{m}\lambda_j w_j^2 \tag{8.38}$$

dimana $\{\lambda_j\}_{j=1}^{m}$ adalah parameter regulasi

y = f(x) sebagai keluaran.

Nilai Pruning rate fungsi disesuaikan dengan m buah persamaan linier secara simultan dengan penimbang dan persamaan linier tersebut dinyatakan sebagai persamaan matriks berikut:

$$A\hat{w} = H^T\hat{y} \tag{8.39}$$

dimana matrik H adalah;

$$H = \begin{bmatrix} h_1(x_1) & h_2(x_1) & \cdots & h_m(x_1) \\ h_1(x_2) & h_2(x_2) & \cdots & h_m(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ h_1(x_p) & h_2(x_p) & \cdots & h_m(x_p) \end{bmatrix} \quad (8.40)$$

A-l adalah matriks varian;

$$\mathbf{A}^{-1} = (\mathbf{H}^T\mathbf{H} + \Lambda)^{-1} \quad (8.41)$$

Elemen matriks A adalah semua yang bernilai nol kecuali diagonalnya sebagai parameter regulasi dan y = [y_1, y_2,… y_p] adalah vektor pelatihan data keluaran. Hasil solusinya dinyatakan dengan persamaan normal,

$$\hat{\mathbf{w}} = \mathbf{A}^{-1}\mathbf{H}^T\hat{\mathbf{y}} \quad (8.42)$$

dan w =[w_1, w_2,... w_m] adalah vektor penimbang yang memperkecil nilai fungsi.

Metoda Pelatihan RBF

Proses pelatihan RBF dilakukan untuk perbaikan penimbang yang menghubungkan unit-unit masukan dengan lapis tersembunyi dan unit-unit lapis tersembunyi dengan unit-unit keluaran dari JST. Lapis yang berbeda dari suatu RBF mengerjakan tugas yang berbeda dan oleh karenanya optimasi lapis tersembunyi dan lapis keluaran dari jaringan dipisahkan dengan memakai teknik yang tidak sama.

Ada perbedaan strategi pelatihan yang akan dijelaskan dalam merancang suatu RBF, dan tergantung pada bagaimana pusat-pusat RBF dari jaringan dispesifikasi. Tiga macam pendekatan yang akan dijelaskan sebagai dasar teori untuk melakukan pelatihan adalah sebagai berikut:

- Seleksi Titik Pusat Secara Random

Pendekatan yang pertama dengan mengasumsikan fungsi-fungsi aktivasi dari unit-unit lapis tersembunyi adalah tetap/fixed. Secara khusus lokasi-lokasi dari pusat RBF dipilih secara random dari himpunan data pelatihan. untuk itu digunakan suatu fungsi Gauss yang sama sebagai standart deviasinya yang tepat dalam mengikuti penyebaran dari titik-titik pusatnya. Secara spesifik suatu RBF temormalisasi dengan titik pusat di c dan didefinisikan sebasai berikut:

$$H\left(\|x-c_i\|^2\right)=\exp\left(-\frac{M}{d^2}\|x-c\|^2\right) \tag{8.43}$$

M = jumlah dari titik-titik pusat

d = jarak maximum diantara titik pusat yang dipilih

i = 1 , 2 . . . M

Standar deviasi (width dari RBF Gauss fixed)

$$\sigma=\frac{d}{\sqrt{2M}} \tag{8.44}$$

sebagai suatu pilihan untuk standard deviasi o adalah menyatakan bahwa fungsi-fungsi Gauss tidak begitu tajam atau datar (kedua ekstrim ini dihindari).

Parameter yang dibutuhkan untuk dilatihkan dalam pendekatan ini hanya penimbang linier pada lapis keluaran dari jaringan. Suatu metoda yang secara langsung untuk melakukan ini adalah menggunakan metoda psedoinvers. secara khusus memiliki persamaan seperti pada penimbang optimal

dimana D adalah vektor tanggapan yang dikehendaki dalam himpunan data ($A.(A^{-1}y)$ pelatihan, H^{-1} adalah psedoinvers dari matriks H yang mana didefinisikan sebagai

Dengan

dimana xl adalah vektor input ke j dari himpunan pelatihan j:l,2,...,N dan I:1,2,...,M.

Dasar dari semua algoritma untuk komputasi dari psedoinvers dari suatu matriks adalah singular value decomposition.

- Titik Pusat Diseleksi dengan Metoda pelatihan Mandiri

Pendekatan yang kedua, fungsi-fungsi radial-basis yang diijinkan untuk memindahkan lokasi-lokasi dari titik pusat yang terorganisasi mandiri, dimana penimbang-penimbang linier dari lapis keluaran dihitung menggunakan aturan pelatihan terbimbing. Dengan kata lain, jaringan mengalami proses pelatihan hibrida (Moody and Darken, 1989; Lippmann, l9S9). proses pelatihan pada komponen

terorganisasi mandiri menempatkan titik-titik pusat dari RBF hanya dalam daerah-daerah dari ruang masukan /input space dimana data yang signifikan muncul.

RBFN adalah jaringan dan pelatihan hibrida yang mengkombinasi paradigma tak-terbimbing dan skema pelatihan terbimbing, (Moody dan Darken,lggg). Dimana sebagian dari penimbang ditentukan dengan pelatihan terbimbing dan sebagian lain diperoleh dari pelatihan tak-terbimbing. RBFN dilatih dengan aturan tak-terbimbing pada lapis masukan, dan aturan pelatihan terbimbing pada lapis keluaran.

- Titik Pusat Diseleksi Secara terbimbing

Pendekatan ketiga, pusat-pusat dari RBF dan semua parameter-parameter bebas dari jaringan mengalami suatu proses pelatihan terbimbing. Langkah pertama dalam pengembangan suatu proses pelatihan adalah mendefenisikan nilai fungsi cost:

dimana N adalah jumlah sample pelatihan yang digunakan dalam proses pelatihan, dan e, adalah sinyal error, yang didefinisikan dengan:

$$E = \frac{1}{2}\sum_{j=1}^{N} e_j^2 \tag{8.45}$$

Parameter yang dibutuhkan adalah menemukan parameter-parameter bebas, w_i, t_i, dan (dihubungkan ke normalisasi matriks-penimbang), yang meminimisasi E.

$$e_j = d_j - y(x_j)$$
$$e_j = d_j - \sum_{i=1}^{M} w_i H(\|x_j - t_i\|) \tag{8.46}$$

8.3 RINGKASAN

1. Pembuatan struktur jaringan araf tiruan diilhami oleh struktur jaringan biologi otak manusia.
2. Metoda pelatihan terbimbing adalah metoda pelatihan yang memasukkan target keluaran dalam data untuk proses pelatihan.
3. Metoda pelatihan tak terbimbing adalah pelatihan tanpa memerlukan target pada keluarannya.
4. Metoda pelatihan Hibrida adalah metoda yang menggabungkan pelatihan terbimbing dan tak terbimbing untuk mengatasi ketidaklinearan suatu sistem.

8.4 LATIHAN

1. Sebuah jaringan saraf tiruan dengan algoritma backpropagasi digunakan untuk mengkonversi bilangan biner ke kode gray. Topologi jaringan terdiri dari 1 lapisan tersembunyi dengan 4 unit sedangkan fungsi aktifasi menggunakan fungsi sigmoid. Learning rate dan momentum dipilih sendiri yang terbaik. Tentukan nilai bobot pada iterasi pertama dengan training data sbb:

Binary input	Gray output
000	000
001	001
010	011
011	010
100	110
101	111
110	101
111	100

2. Sebuah jaringan saraf tiruan dengan algoritma preseptron digunakan untuk operasi fungsi logika x1 XOR x2. Topologi jaringan terdiri dari 1 lapisan tersembunyi dengan 2 unit sedangkan fungsi aktifasi menggunakan fungsi sigmoid. Learning rate dan momentum dipilih sendiri yang terbaik. Tentukan nilai bobot pada iterasi pertama dengan training data sbb: