# LAPORAN PENGANTAR KECERDASAN BUATAN
# PENERAPAN ALGORITMA A* DAN JARINGAN SYARAF TIRUAN
# (HEBB & MCCULLOCH-PITTS)

**OLEH :**
**MOHAMMAD FIRYANUL RIZKY (1708561006 )**

**PROGRAM STUDI TEKNIK INFORMATIKA**
**FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM**
**2018**

## 1. Algoritma A*

Algoritma ini pertama kali ditemukan pada tahun 1968 oleh Peter Hart, Nils Nilsson dan Bertram Raphael. A* (dibaca "A bintang"/"A star") adalah algoritma pencarian graf/pohon yang mencari jalur dari satu titik awal ke sebuah titik akhir yang telah ditentukan. Algoritma A* menggunakan pendekatan heuristik h(x) yang memberikan peringkat ke tiap-tiap titik x dengan cara memperkirakan rute terbaik yang dapat dilalui dari titik tersebut. Setelah itu, tiap-tiap titk x tersebut dicek satu-persatu berdasarkan urutan yang dibuat dengan pendekatan heuristik tersebut. Beberapa terminologi dasar yang terdapat pada algoritma ini adalah *starting point*, simpul (nodes), A, *open list*, *closed list*, harga (cost), halangan (*unwalkable*).

- *Starting point* adalah sebuah terminologi posisi awal sebuah benda.
- A adalah simpul yang sedang dijalankan algortima pencarian jalan terpendek.
- Simpul adalah petak-petak kecil sebagai representasi dari area*path finding*. Bentuknya dapat berupa persegi, lingkaran, maupun segitiga.
- *Open list* adalah tempat menyimpan data simpul yang mungkin diakses dari *starting point* maupun simpul yang sedang dijalankan.
- *Closed list* adalah tempat menyimpan data simpul sebelum A yang juga merupakan bagian dari jalur terpendek yang telah berhasil didapatkan.
- Harga (F) adalah nilai yang diperoleh dari penjumlahan nilai G, jumlah nilai tiap simpul dalam jalur terpendek dari starting point ke A, dan H, jumlah nilai perkiraan dari sebuah simpul ke simpul tujuan.
- Simpul tujuan yaitu simpul yang dituju.
- Rintangan adalah sebuah atribut yang menyatakan bahwa sebuah simpul tidak dapat dilalui oleh A.

Prinsip algoritma ini adalah mencari jalur terpendek dari sebuah simpul awal (*starting point*) menuju simpul tujuan dengan memperhatikan harga (F) terkecil.A* memperhitungkan cost dari *current state* ke tujuan denga fungsi heuristic. Algoritma ini juga mempertimbangkan cost yang telah ditempuh selama ini dari *initial state* ke *current state*. Jadi, jika ada jalan yang telah ditempuh sudah terlalu panjang dan ada jalan lain yang cost-nya lebih kecil tetapi memberikan posisi yang sama dilihat dari *goal*, jalan yang lebih pendek yang akan dipilih.

### a. Karakteristik Algoritma A*

Karakteristik yang menjelaskan algoritma A* adalah pengembangan dari “daftar tertutup” untuk merekam area yang dievaluasi. Daftar tertutup ini adalah sebuah daftar untuk merekam area berdekatan yang sudah dievaluasi, kemudian melakukan perhitungan jarak yang dikunjungi dari “titik awal” dengan jarak diperkirakan ke “titik tujuan” (Reddy,2013).

Algoritma A* menggunakan path dengan costpaling rendah ke node yang membuatnya sebagaialgoritma pencarian nilai pertama yang terbaik ataubest first search. Menggunakan rumus

$f(x) = g(x) + h(x)$ ……..(1)

dimana:

- $g(x)$ adalah jarak total dari posisi asal ke lokasisekarang.
- $h(x)$ adalah fungsi heuristik yang digunakan untuk memperkirakan jarak dari lokasi sekarang ke lokasi tujuan. Fungsi ini jelas berbeda karena ini adalah perkiraan semata dibandingkan dengan nilai aslinya. Semakin tinggi keakuratan heuristik, semakin cepat dan bagus lokasi tujuan ditemukan dan dengan tingkat keakuratan yang lebih baik. Fungi $f(x) = g(x) + h(x)$ ini adalah perkiraan saat ini dari jarak terdekat ke tujuan (Lubis, 2016).

Algoritm A* juga menggunakan 2 (dua) senarai Open List dan Closed List sama seperti algoritma dasar Best First Search. Terdapat 3 (tiga) kondisibagi setiap suksesor yang dibangkitkan, yaitu sudahberada pada di Open, sudah berada di Closed, dantidak berada di Open maupun Closed. Pada ketigakondisi tersebut diberikan penanganan yang berbeda-beda(Suyanto, 2014).

Jika suksesor sudah pernah berada di Open,maka dilakukan pengecekan apakah perlupengubahan parent atau tidak tergantung pada nilai $g$-nya melalui parent lama atau parent baru. Jika melalui parent baru memberikan nilai $g$ yang lebih kecil, maka dilakukan pengubahan parent. Jika pengubahan parent dilakukan, maka dilakukan pula update nilai $g$ dan $f$ pada suksesor tersebut. Dengan perbaharuan ini, suksesor tersebut memiliki kesempatan yang lebih besar untuk terpilih sebagai simpul terbaik (best node) (Suyanto, 2014).

Jika suksesor sudah pernah berada di Closed, maka dilakukan pengecekan apakah perlu pengubahan parent atau tidak, jika ya, maka dilakukan perbaharuan nilai $g$ dan $f$ pada suksesor tersebut serta pada semua "anak cucunya" yang sudah pernah berada di Open. Dengan perbaharuan ini, maka semua anak cucunya tersebut memiliki kesempatan lebih besar untuk terpilih sebagai simpul terbaik (best node) (Suyanto, 2014).

Jika suksesor tidak berada di Open maupun Closed, maka suksesor tersebut dimasukkan kedalam Open. Tambahkan suksesor tersebut sebagai suksesornya best node. Hitung cost suksesor tersebut dengan menggunakan persamaan 1 (Suyanto, 2014).

**b. Contoh Penerapan Algoritma A***

```
#include <stdio.h>
#include <math.h>
#include <conio.h>
#include <stdlib.h>

int main(){
      int n,i,j;
```

```
	printf("====Algoritma A*====");
	printf("\nMasukan Jumlah vertek :
");scanf("%d",&n);
	int
cost[n],h[n],m[n][n],x[n],y[n],visited[n],prev[n];
	printf("===Hubungan vertek===\n");
	for(i=0;i<n;i++){
		for(j=0;j<n;j++){
			scanf("%d",&m[i][j]);
			if(m[i][j]==0){
				m[i][j]=9999;
			}
		}
	}
	printf("==Nilai Heuristik setiap vertek ke vertek
lain ==\n");
	for(i=0;i<n;i++){
		printf("Vertek %d = ",i);scanf("%d",&h[i]);
	}

	//algoritma a*
	int start,end,temp,min,temp2;
	printf("Masukan vertek asal =
");scanf("%d",&start);

	printf("Masukan vertek tujuan =
");scanf("%d",&end);
	for(i=0;i<n;i++){
			cost[i]=999;
			visited[i]=0;
	}
	cost[start]=0;
	temp=start;
	while(temp!=end){
		min=999;
		for(i=0;i<n;i++){
			if(min > cost[i] && visited[i]==0){
				min=cost[i];
				temp2=i;
			}
		}
		temp=temp2;
		visited[temp]=1;
		for(i=0;i<n;i++){
			if(m[temp][i]!=9999 && visited[i]==0
&&temp!=end){
				if(min+h[i]+m[temp][i] <
cost[i]){
					cost[i]=min+m[temp][i];
					prev[i]=temp;
				}
```

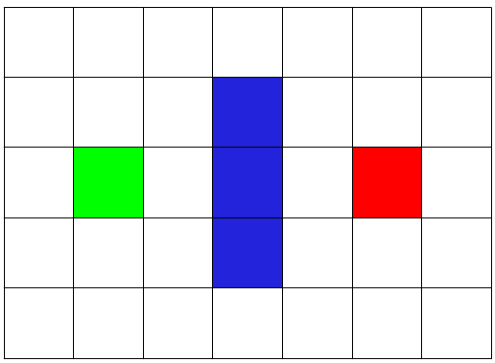
```
                        }
                    }
                }
                printf("\n\n Cost terpendek dari %d ke %d adalah =
        %d dengan rute : \n",start,end,cost[end]);
                i=end;
                while(i!=start){
                    printf("%d<--",i);
                    i=prev[i];
                }
                printf("%d",start);



                return 0;
}
```

Prinsip algoritma ini adalah mencari jalur terpendek dari sebuah simpul awal (starting point) menuju simpul tujuan dengan memperhatikan harga (F) terkecil.

A* memperhitungkan cost dari current state ke tujuan denga fungsi heuristic, Algoritma ini juga mempertimbangkan cost yang telah ditempuh selama ini dari initial state ke current state. Jadi jika ada jalan yang telah ditempuh sudah terlalu panjang dan ada jalan lain yang cost-nya lebih kecil tetapi memberikan posisi yang sama dilihat dari goal, jalan yang lebih pendek yang akan dipilih.

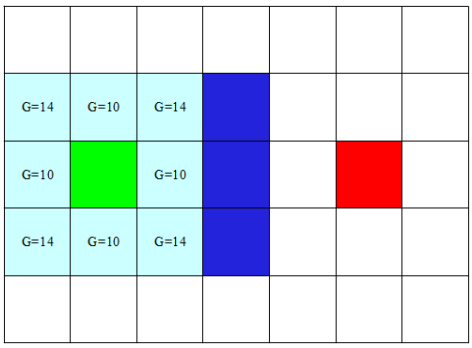
Langkah 1 : Arena

Berikut adalah contoh simple arena yang akan kita gunakan. Warna hijau adalah starting point, warna merah adalah goal/end point, dan biru adalah penghalang. Goal dari aplikasi ini adalah mencari rute dari titik hijau ke merah tanpa melewati penghalang biru



Langkah 2 : Movement Cost / Biaya Pergerakan

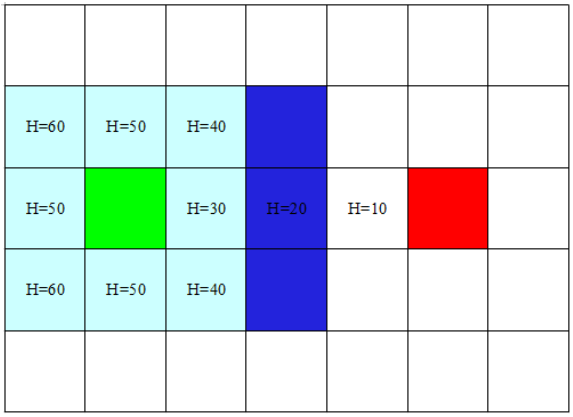
Kita asumsikan setiap langkah dari hijau adalah legal baik vertikal, horizontal, maupun diagonal dengan catatan tidak membentur tembok. Setiap langkah yang diizinkan kita berikan nilai G dimana G adalah cost atau biaya dalam setiap langkah. Dalam kasus ini kita akan berikan nilai 10 untuk setiap langkah vertikal maupun horizontal, dan 14 untuk diagonal. Nilai 14 kita dapatkan dari perhitungan pitagoras dimana 14,1421 = sqrt(sqr(10)+sqr(10)). Hasil data nilai G ini selanjutnya kita gambarkan sbb :




Selain dari perhitungan tersebut, kita dapat mengalikan dengan konstanta tertentu untuk memanipulasi biaya, misal : ketika melewati sungai maka G = G * 2.

Langkah 3 : Estimated Movement / Estimasi gerakan

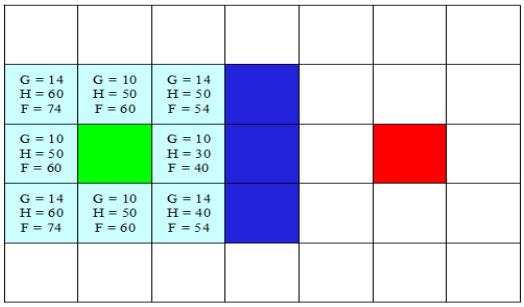
Langkah selanjutnya kita hitung biaya estimasi pergerakan dan kita simbolkan dengan H. Nilai H ini secara singkat adalah nilai jarak / estimasi biaya dari pergerakan dari suatu titik terhadap titik finish dengan mengabaikan penghalang yang ada. Untuk lebih jelasnya silahkan lihat gambar di bawah :



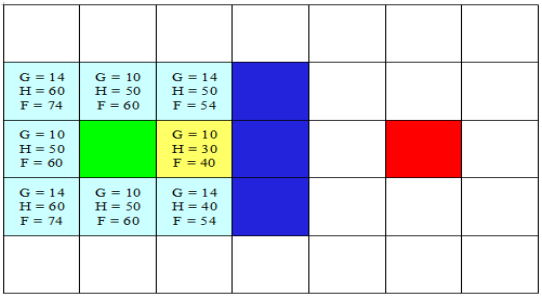

Langkah 4 : Scoring / Penilaian

Setelah nilai G dan H kita dapatkan, maka kita berikan skor dari masing-masing titik yang akan dilalui. Skor kita lambangkan misalnya dengan F dimana nilai F = G + H. Nilai F selanjutnya kita masukkan dalam setiap titik

dari setiap langkah yang akan dilalui. Untuk lebih jelasnya lihat gambar di bawah :

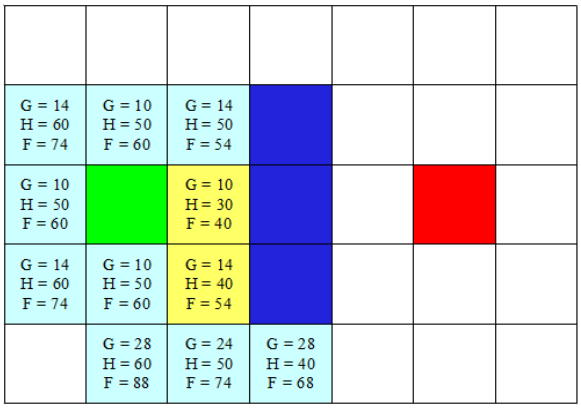


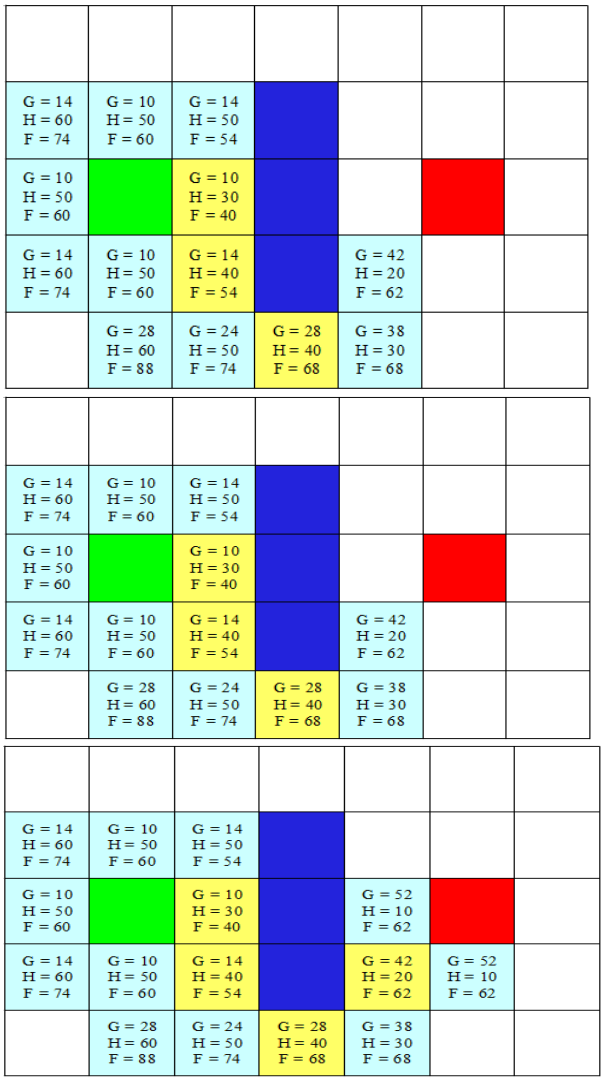

Dari setiap nilai tersebut kita ambil keputusan dengan mengambil langkah dengan nilai F terkecil.




Langkah 5 : Looping / Perulangan

Setelah pergerakan pertama selesai selanjutnya lakukan perulangan dari dari langkah 1 sampai 4. Untuk lebih jelasnya setiap pergerakan akan digambarkan di bawah :



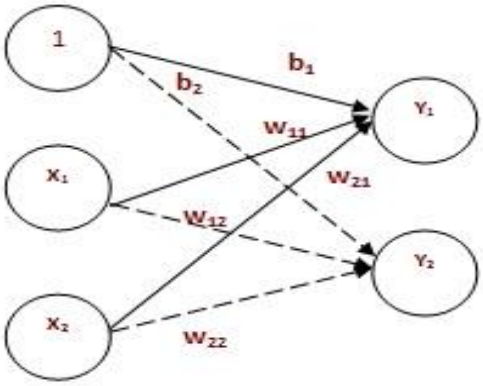



## 2. Algoritma JST (Metode Hebb & Mcculloch-Pitts)

### Metode HEBB

Hebb merupakan metoda pembelajaran sederhana jaringan saraf tiruan yang pertama kali digunakan untuk pengenalan pola. Bobot antar neuron akan meningkat bila 2 neuron menembak pada saat yang sama atau bila 2 neuron tidak menembak pada saat yang sama.

Jaringan Hebb menggunakan arsitektur lapisan tunggal dengan 1 lapisan input dan 1 lapisan output. Setiap neuron pada lapisan input berhubungan dengan lapisan output dengan suatu bobot keterhubungan.




Nilai bobot yang baru ($w_i$ (baru)) didapat dari penambahan nilai bobot yang lama ($w_i$ (lama)) dengan nilai perubahan bobotnya (Dw), atau dapat dituliskan sebagai:

$$w_i(baru) = w_i(lama) + \Delta w_i$$

sedangkan nilai perubahan bobot didapat dari:

$$\Delta w_i = x_i y_i$$

dan untuk perubahan biasnya adalah:

$$\Delta b = y$$

Proses pembelajaran akan berhenti jika syarat hentinya dipenuhi. Hebb menggunakan separabilitas linier untuk menghentikan proses pembelajarannya, jika separabilitas liniernya belum terpenuhi maka proses pembelajaran akan berjalan terus. Selain itu separabilitas linier digunakan juga dalam proses pengenalan.

**Algoritma Pembelajaran**

**Langkah 0:** Inisialisasi semua bobot

$w_i = 0$ (i = 1..n);

b = 0;

**Langkah 1:** Untuk setiap pasangan vektor (s & t), kerjakan langkah 2-5.

**Langkah 2:** Tentukan nilai aktifasi unit input

$x_i = s_i$

**Langkah 3:** Tentukan nilai aktifasi unit output

$y_i = t_i$.

**Langkah 4:** Update nilai bobot dan bias

$w_i$(baru) = $w_i$(lama) + $x_i y_i$ .

b(baru) = b(lama) + $y_i$ .

**Langkah 5:** Periksa syarat henti dengan menggunakan separabilitas linier.

**Contoh Penerapan Metode Hebb :**

```
#include<iostream>
```

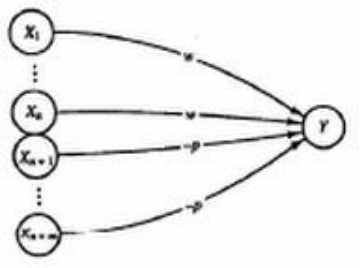
```
using namespace std;

int main(int argc, char *argv[])
{
    int n[8];
     int x1[8] = {-1,-1,-1,-1,1,1,1,1};
    int x2[8] = {-1,-1,1,1,-1,-1,1,1};
    int x3[8] = {-1,1,-1,1,-1,1,-1,1};
    int t[8] = {-1,-1,-1,-1,-1,-1,-1,1};
    int w=0;
    int w1[8],w2[8],w3[8],wb[8];
    int nw1[8],nw2[8],nw3[8],nwb[8];
    int lw1, lw2, lw3, lwb;
    int b=1;
    int a[8];

    cout<<"\t\t\tModel Hebbian";
    cout<<"\n\n\t\tuntuk penyelesaian gerbang AND\n\n";
    cout<<"[x1]\t[x2]\t[x3]\t[b]\t[t]\n\n";

for(int i=0; i<8; i++){
        cout<<"  "<<x1[i]<<"\t  "<<x2[i]<<"\t  "<<x3[i]<<"\t
"<<b<<"\t "<<t[i]<<"\n";
    }


cout<<"\n\n[x1*t]\t[x2*t]\t[x3*t]\t[b*t]\t[nw1]\t[nw2]\t[nw3
]\t[nwb]\n\n";
for(int i=0; i<8; i++){
        w1[i]= t[i]*x1[i];
        w2[i]= t[i]*x2[i];
        w3[i]= t[i]*x3[i];
        wb[i]= t[i]*1;

        nw1[0]=(t[0]*x1[0])+0;
        nw2[0]=(t[0]*x2[0])+0;
        nw3[0]=(t[0]*x3[0])+0;
        nwb[0]=(t[0]*1)+0;

        nw1[i+1]= (t[i+1]*x1[i+1])+nw1[i];
        nw2[i+1]= (t[i+1]*x2[i+1])+nw2[i];
        nw3[i+1]= (t[i+1]*x3[i+1])+nw3[i];
        nwb[i+1]= (t[i+1]*1)+nwb[i];

        lw1=nw1[7];
        lw2=nw2[7];
```

```
        lw3=nw3[7];
        lwb=nwb[7];

        cout<<"  "<<w1[i]<<"\t  "<<w2[i]<<"\t  "<<w3[i]<<"\t
"<<wb[i]<<"\t  "<<nw1[i]<<"\t  "<<nw2[i]<<"\t  "<<nw3[i]<<"\t
"<<nwb[i]<<"\n";
    }

    cout<<"\nbobot baru jaringan hasil training\n";

cout<<"w1="<<lw1<<"\nw2="<<lw2<<"\nw3="<<lw3<<"\nwb="<<lwb<<
"\n";

    cout<<"\nSimulasinya\n";
    cout<<"[w1]\t[w2]\t[w3]\t[net=total(xi*wi)+wb]
[Y]\n\n";
for(int i=0; i<8; i++){
        n[i]= (x1[i]*lw1)+(x2[i]*lw2)+(x3[i]*lw3)+lwb;
        if(n[i]<0){
            a[i]=-1;
        }
else{
            a[i]=1;
        }
        cout<<"    "<<x1[i]<<"\t  "<<x2[i]<<"\t  "<<x3[i]<<"\t
"<<n[i]<<"                  "<<a[i]<<"\n";
    }

    system("pause");
    return 0;
}
```

**Metode MCCULLOCH-PITTS**

Model JST yang digunakan oleh McCulloch-Pitts (McP) merupakan model yang pertama ditemukan. Model neuron McP memiliki karakteristik sebagai berikut.

a. Fungsi aktivasinya biner
b. Semua garis yang memperkuat sinyal (bobot positif) ke arah suatu neuron memiliki kekuatan (besar bobot) yang sama. Hal yang sama untuk garis yang memperlemah sinyal (bobot negatif) ke arah neuron tertentu
c. Setiap neuron memiliki batas ambang (threshold) yang sama. Apabila total input ke neuron tersebut melebihi threshold, maka neuron akan meneruskan sinyal



*model neuron McP*

- Neuron Y menerima sinyal dari (n+m) buah neuron x1 x2, …..xn, xn+1, ….xn+m
- n buah penghubung dengan dari x1 x2, …..xn ke Y merupakan garis yang memperkuat sinyal (bobot positif), sedangkan m buah penghubung dari xn+1,….xn+m ke Y merupakan garis yang memperlemah sinyal (bobot negatif).
- Semua penghubung dari x1 x2, …..xn ke Y memiliki bobot yang sama. Hal yang sama dengan penghubung dari xn+1,….xn+m ke Y memiliki bobot yang sama. Namun jika ada neuron lain katakan Y2, maka bobot x1 ke Y1 boleh berbeda dengan bobot dari x2 ke Y2.
- Fungsi aktivasi neuron Y adalah

$$f(net) = \begin{cases} 1 & jika\ net \geq a \\ 0 & jika\ net < a \end{cases}$$

- Bobot tiap garis tidak ditentukan dengan proses pelatihan, tetapi dengan metode analitik. Beberapa contoh berikut memaparkan bagaimana neuron McP digunakan untuk memodelkan fungsi logika sederhana.

**Contoh Penerapan Metode Mcculloch-Pitts**

```
#include<iostream>

using namespace std;

int main(in
t argc, char *argv[])
{
    int n[4],a[4];
    int x1[4] = {0, 0, 1, 1};
    int x2[4] = {0, 1, 0, 1};
    int target[4] = {0, 0, 0, 1};
    int target2[4] = {0, 1, 1, 1};
    int target3[4] = {0, 1, 1, 0};
    int w=1;

    cout<<"JST-Model Neuron McCulloch-Pitts";

    cout<<"\n\n#gerbang AND\n\n";
```

```
    cout<<"[x1]\t[x2]\t[N]\t[target]\t[Y]\n\n";
for(int i=0; i<4; i++){
        n[i]= w*x1[i]+w*x2[i];
        if(n[i]<2){
            a[i]=0;
        }
else{
            a[i]=1;
        }
        cout<<"   "<<x1[i]<<"\t   "<<x2[i]<<"\t   "<<n[i]<<"\t
"<<target[i]<<"\t\t "<<a[i]<<"\n";
    }

    cout<<"\n\n#gerbang OR\n\n";
    cout<<"[x1]\t[x2]\t[N]\t[target]\t[Y]\n\n";
for(int i=0; i<4; i++){
        n[i]= w*x1[i]+w*x2[i];
        if(n[i]<1){
            a[i]=0;
        }
else{
            a[i]=1;
        }
        cout<<"   "<<x1[i]<<"\t   "<<x2[i]<<"\t   "<<n[i]<<"\t
"<<target2[i]<<"\t\t "<<a[i]<<"\n";
    }

    cout<<"\n\n#gerbang XOR\n\n";
    int w0[2]={-1, 2};
    int z1[4],z2[4],a1[4],n1[4],n2[4];
    cout<<"[x1]\t[x2]\t[z1]\t[z2]\t[N]\t[target]\t[Y]\n\n";

for(int i=0; i<4; i++){
        n1[i]= w0[1]*x1[i]+w0[0]*x2[i];
        if(n1[i]<2){
            z1[i]=0;
        }
else{
            z1[i]=1;
        }

        n2[i]= w0[0]*x1[i]+w0[1]*x2[i];
        if(n2[i]<2){
            z2[i]=0;
        }
else{
            z2[i]=1;
```

```
        }
        n[i]= w*z1[i]+w*z2[i];
        if(n[i]<1){
            a1[i]=0;
        }
else{
            a1[i]=1;
        }
        cout<<"                                          "<<x1[i]<<"\t
"<<x2[i]<<"\t"<<z1[i]<<"\t"<<z2[i]<<"\t          "<<n[i]<<"\t
"<<target3[i]<<"\t\t "<<a1[i]<<"\n";
    }
    system("pause");
    return 0;
}
```

**Contoh Penerapan Kasus**

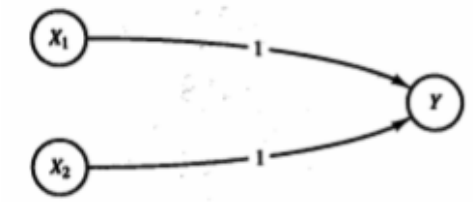
Fungsi logika ”AND” dengan 2 masukan x1 dan x2 akan memiliki keluaran Y =1 jika dan hanya jika kedua masukan bernilai 1.

Tabel kebenaran

| x1 | x2 | y |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

Buatlah model neuron Mcp untuk menyatakan fungsi logika AND.

Model neuron fungsi AND tampak pada gambar di bawah ini. Robot tiap garis adalah = 1 dan fungsi aktivasi memiliki nilai threshold = 2



Untuk semua kemungkinan masukan, nilai aktivasi tampak pada tabel berikut:

| $x_1$ | $x_2$ | $net = \sum_{i=1}^{2} x_i w$ | $f(net) = \begin{cases} 1 & jika\ net \geq 2 \\ 0 & jika\ net < 2 \end{cases}$ |
|---|---|---|---|
| 1 | 1 | 1*1 + 1*1=2 | 1 |
| 1 | 0 | 1*1 + 0*1=1 | 0 |
| 0 | 1 | 0*1 + 1*0=1 | 0 |
| 0 | 0 | 0*1 + 0*1=0 | 0 |

Tampak bahwa keluaran jaringan tepat sama dengan tabel logika AND. Berarti jaringan dapat dengan tepat merepresentasika fungsi AND Besarnya nilai threshold dapat diganti menjadi suati bias dengan nilai yang sama. Dengan menggunakan nilai bias, batas garis pemisah ditentukan dari persamaan

$$\text{net} = b + \sum_i x_i w_i = 0$$

b + x1w1 + x2w2 = 0 atau x2 = - w1x1/w2 - b/w2 Apabila garis pemisahnya diambil dengan persamaan x1 + x2 = 2, maka berarti - w1/w2 = -1 dan – b/w2 = 2. Ada banyak w1, w2 dan b yang memenuhi persamaan tersebut, salah satunya adalah w1= w2 = 1 dan b = -2, seperti penyelesaian contoh di atas.