Bab 5
Penerapan Neural Network
Dalam Klasifikasi Citra Penginderaan Jauh

Klasifikasi citra penginderaan jarak jauh (inderaja) merupakan proses penentuan piksel-piksel masuk ke dalam suatu kelas obyek tertentu. Pendekatan *klasifikasi citra* yang dapat digunakan antara lain *Neural Network* (NN). Pendekatan *Neural Network* mulai tahun 1960 dikembangkan *Multilayer Perceptron Neural Network* (MLPNN) dan menunjukkan kemampuannya dalam menyelesaikan masalah yang lebih kompleks, maka minat menggunakan *neural network* untuk klasifikasi citra inderaja cukup besar dan hasilnya cukup memuaskan [2]. Masalah multivariasi cukup menyulitkan pendekatan statistik karena perlu model yang berbeda untuk citra sensor yang berbeda. *Neural network* dapat mengatasi masalah tersebut dan selanjutnya dikenal sebagai pengklasifikasi non-parametrik yang tidak bergantung model.

Neural network (NN) adalah suatu model matematis yang meniru sistem kerja dari jaringan *neural* otak manusia. *NN* telah dikenal sukses untuk solusi dari suatu problem dengan data yang bersifat kompleks (*uncertainty*), memiliki *noise atau incomplete* [2, 14]. Berbeda dengan metode statistik, *NN* tidak bergantung pada model sebaran data sehingga dapat juga dikatakan bebas parametrik dan bebas informasi *prior*. Hal inilah yang merupakan kelebihan NN untuk klasifikasi data inderaja, mengingat bahwa data inderaja tidak selalu memiliki model sebaran normal, dan belum tentu memiliki informasi *prior* terhadap data tersebut. Bila sebaran data tidak diketahui dan informasi *prior* tidak dimiliki, metode klasifikasi statistik akan memberikan hasil dengan akurasi yang buruk [5]. Dalam kondisi yang

demikian, maka dapat memilih *NN* sebagai alternatif terbaik untuk memperoleh hasil klasifikasi citra dengan akurasi yang tinggi.

Dalam melakukan klasifikasi citra inderaja dengan *NN*, langkah-langkah penting yang dilakukan sebagai berikut :

1. Menentukan fitur-fitur atau ciri yang dapat memudahkan klasifikasi kelas-kelas, yang akan dijadikan input. Pengetahuan akan fitur kelas-kelas, akan membantu dalam klasifikasi.
2. Mengumpulkan data pelatihan yang representatif dengan jumlah yang memadai bila memungkinkan. Data dapat diperoleh dari data lapangan, atau dari peta-peta yang akurat.
3. Pemilihan model jaringan, dan *algoritme* pelatihan. Setiap model arsitektur dan *algoritme* memiliki kelebihan dan kekurangan dalam hal waktu dan akurasi, sehingga pemilihan model *neural network* bergantung pada faktor mana yang lebih diprioritaskan [10]. Penentuan model arsitektur dan *algoritme* pelatihan jaringan, bergantung pada pendekatan yang akan digunakan, dengan pengarahan atau tanpa pengarahan. Bila memiliki himpunan data pelatihan dengan jumlah yang memadai, pendekatan dengan pengarahan dapat digunakan. Bila sulit memperoleh himpunan data pelatihan, pendekatan tanpa pengarahan dapat digunakan.

Dalam menentukan fitur yang menjadi input, dapat digunakan fitur spektral *(gray level)*, atau fitur *tekstural* obyek yang penggunaannya disesuaikan dengan tujuan klasifikasi. Bila kelas-kelas yang akan dibentuk dapat dibedakan dengan fitur spektral, maka fitur spektral dapat digunakan, demikian pula halnya dengan pemilihan fitur tekstural [2, 20, 24].

Dalam merepresentasikan *gray level* sebagai vektor input, dapat menggunakan 3 cara sebagai berikut :

1. setiap nilai *gray level* suatu citra dijadikan satu elemen vektor input dengan range 0 - 127 (untuk 7 bit) atau 0 - 255 (untuk 8 bit). Cara merepresentasikan nilai input ini diistilahkan dengan *temperatur code*.
2. Setiap nilai *gray level* direpresentasikan dalam bentuk biner dengan jumlah bit 7 atau 8 sesuai dengan resolusi citra. Sehingga jumlah elemen vektor input adalah 7 atau 8. Pengkodean ini dinamakan *binary code*.
3. Setiap nilai *gray level* direpresentasikan dalam bentuk gray code, yang diperoleh dari aturan berikut :

$g_1 = b_1$

$g_k = b_k \ XOR \ b_{k-1} \ untuk \ k >= 2$

dimana : $b_1, b_2, b_3,..,.b_k$ *adalah n digit binary code,*

$g_1, g_2, g_3...,g_n$ *adalah n digit gray code.*

Keuntungan *gray code* dibandingkan dengan *binary code* adalah jumlah bit yang berbeda antara 2 bilangan integer yang berdekatan adalah 1 bit. Contohnya dalam *binary code* 11=1011, 12=1100, dalam *gray code* 11=1010, 12=1011. Maka jarak piksel yang bernilai 11 dan bernilai 12 lebih dekat dengan representasi *gray code*. Jarak yang dekat akan cenderung berada dalam kelas yang sama.

Penentuan jumlah elemen vektor output, atau jumlah unit output, dapat ditempuh dengan 2 cara yaitu :

1. Pengkodean output dengan *temperatur code*, yaitu jumlah elemen vektor output sama dengan jumlah kelas.
2. Pengkodean dengan *binary code*, yaitu jumlah kelas direpresentasikan dalam *binary code*. Maka jumlah elemen vektor output adalah jumlah bit-nya.

3. Misalnya terdapat 8 kelas maka dengan temperatur code, diperoleh 8 elemen vektor yaitu (1,0,0,0,0,0,0,0). Sedangkan dengan *binary code* jumlah elemen vektor dapat direduksi menjadi 4. Contoh, untuk kelas ke-7 dapat direpresentasikan sebagai vektor (0,1,1,1).

5.1 Back Propagation Neural Network (BPNN)

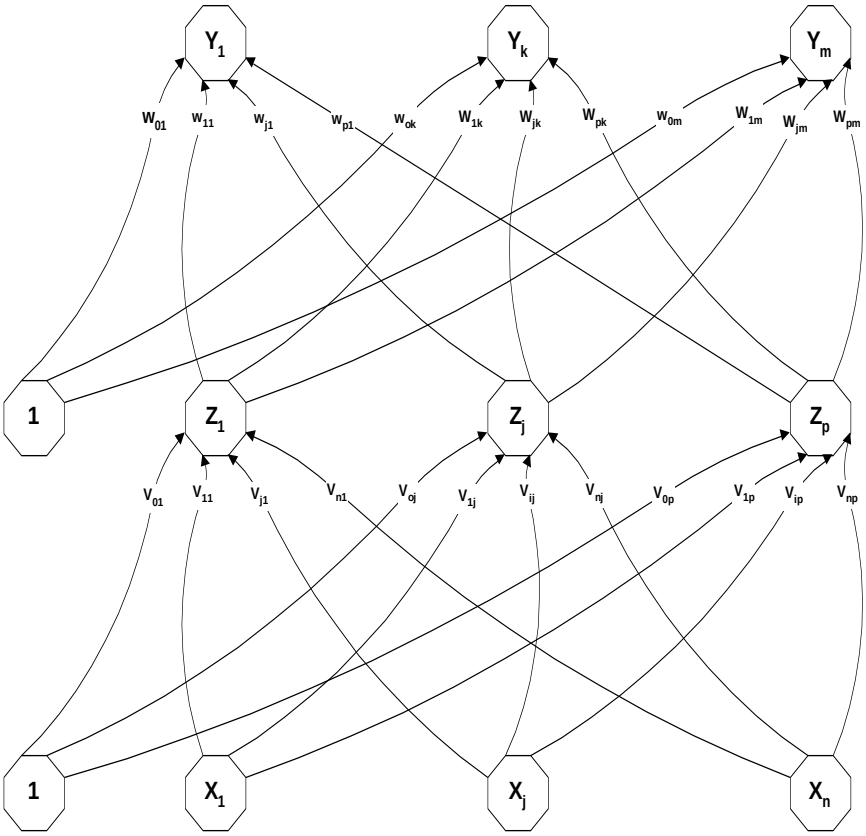
Tahun 1970-an telah ditemukan bahwa *neural network* lapis tunggal memiliki keterbatasan dimana tidak dapat memecahkan masalah, salah satunya adalah fungsi *XOR* [2, 10]. Kemudian muncul ide untuk mengembangkan *neural network* lapis jamak (*Multi Layer Perceptron/MLP*). Namun perkembangan *neural network* terhambat dikarenakan tidak adanya aturan pelatihan dan tahun 1986 ditemukan aturan pelatihan *back propagation*. Tujuan pelatihan adalah untuk adaptasi bobot sehingga input tertentu dapat menghasilkan keluaran yang diinginkan. Arsitektur *Back Propagation Neural Network* (BPNN) terdiri dari satu lapis masukan, satu atau lebih lapis tersembunyi dan satu lapis keluaran. Lapis masukan bertugas meneruskan input dan tidak melakukan komputasi, sementara lapis tersembunyi dan lapis keluaran melakukan komputasi. Jumlah neuron pada lapis masukan sama dengan jumlah fitur atau atribut pada pola yang akan dikenali, sedang jumlah neuron pada lapis keluaran sama dengan jumlah kelas pola. Lapis masukan menjadi input bagi lapis tersembunyi dan output dari lapis tersembunyi menjadi input bagi lapis keluaran.

Notasi vektor lapis masukan, output dari lapis tersembunyi dan output dari lapis keluaran dinyatakan sebagai:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ .... \\ x_I \end{bmatrix} \qquad y = \begin{bmatrix} y_1 \\ y_2 \\ .... \\ y_J \end{bmatrix} \qquad o = \begin{bmatrix} o_1 \\ o_2 \\ .... \\ o_K \end{bmatrix}$$

di mana I, J dan K menyatakan jumlah neuron pada tiap lapis .

Jumlah lapis tersembunyi yang digunakan dan jumlah neuron pada tiap lapis tergantung pada tingkat kompleksitas masalah. Biasanya satu lapis tersembunyi sudah memadai untuk beberapa aplikasi [10, 14]. Sementara cara terbaik untuk menentukan jumlah neuron pada lapis tersembunyi adalah dengan metode *trial and error*.



Gambar-5.1 : Arsitektur *Back Propagation Neural Network*

Mekanisme komputasi jaringan dalam *algoritme back propagation*, meliputi dua jenis komputasi, yaitu alur maju (*forward pass*) dan alur mundur (*backward pass*). Dalam komputasi maju, bobot sinapsis dalam jaringan tidak diubah. Untuk *neural network* dengan satu lapis tersembunyi, perhitungan yang terjadi dapat dijabarkan sebagai berikut:

1. Neuron lapis tersembunyi menghitung nilai aktivasi dengan cara menjumlahkan perkalian sinyal masukan dari lapis masukan dengan bobot antara lapis masukan dan lapis tersembunyi, dan bias tertentu.

2. Neuron lapis tersembunyi lalu menghitung sinyal keluaran dengan menerapkan fungsi aktivasi untuk mendapatkan nilai aktivasi .
3. Sinyal keluaran dari lapis tersembunyi menjadi masukan bagi lapis keluaran. Neuron pada lapis keluaran menghitung nilai aktivasi dengan cara menjumlahkan perkalian sinyal masukan dengan bobot antara lapis tersembunyi dan lapis keluaran, dan bias tertentu.
4. Neuron lapis keluaran menghitung sinyal keluaran dengan menerapkan fungsi aktivasi untuk mendapatkan nilai aktivasi .

Berbeda dengan alur maju yang dimulai dari lapis masukan, alur mundur dimulai dari lapis keluaran. Alur mundur bertujuan untuk menyesuaikan bobot sinapsis jaringan. Secara singkat, langkah-langkah yang dilakukan adalah:

1. Setiap neuron di lapis keluaran menghitung selisih target keluaran dan keluaran aktual.
2. Setiap neuron di lapis keluaran menyesuaikan bobot hubungan dari dirinya ke semua neuron di lapis tersembunyi.
3. Lapis keluaran mempropagasikan sinyal kesalahan ke lapis tersembunyi sehingga setiap neuron di lapis tersembunyi menyesuaikan bobot hubungan dari dirinya ke semua neuron di lapis masukan.

Perseptron lapis jamak memberikan kemampuan yang lebih daripada perseptron lapis tunggal karena peranan fungsi aktivasi nonlinier antar lapis [10]. Perhitungan output dari lapis keluaran terdiri atas perkalian input dengan bobot pertama (antara lapis masukan dan lapis tersembunyi), dan dikalikan lagi dengan bobot kedua (lapis tersembunyi dan lapis keluaran). Hal ini dapat dinyatakan dengan:

$$out = (x.w_1).w_2 \text{ . ..... } (5.1)$$

Karena perkalian matriks adalah asosiatif, persamaan di atas menjadi:

$$out = x.(w_1.w_2) \quad \text{.....(5.2)}$$

Hal tersebut menunjukkan bahwa jaringan dengan 2 lapis linier ekivalen dengan jaringan lapis tunggal. Oleh karena itu, jaringan lapis jamak linier dapat digantikan oleh jaringan lapis tunggal.

Perhitungan *sinyal error* pada tiap neuron membutuhkan turunan pertama dari fungsi aktivasi . Karena itu, fungsi aktivasi harus non linier dan kontinu [14]. Fungsi aktivasi yang paling umum dan dipakai dalam tugas akhir ini ialah fungsi *sigmoid unipolar* :

$$out = f(net)$$
$$= \frac{1}{1 + e^{-net}} \quad \text{.....(5.3)}$$

Keterangan :

net : tingkat aktivasi dari suatu neuron.

Algoritme BP menggunakan prosedur *gradient descent* atau menuruni lembah permukaan *error*, untuk meminimisasi suatu fungsi kesalahan. Fungsi kesalahan yang umum dipakai ialah *fungsi error kuadratis*:

$$E = \frac{1}{2} \sum_k (d_k - o_k)^2 \quad \text{.....(5.4)}$$

Dengan *fungsi error kuadratis*, *sinyal error* pada lapis keluaran yang dipropagasikan ke lapis tersembunyi adalah:

$$\delta o_k = o_k.(1-o_k).(d_k-o_k) \quad \text{.....(5.5)}$$

Sinyal error mengandung suku $o_k.(1-o_k)$. Untuk keluaran o_k yang mendekati nilai 0 atau 1, sinyal error menjadi sangat kecil menyebabkan proses pembelajaran

menjadi lambat [10]. Fungsi kesalahan yang juga banyak dipakai sebagai alternatif *fungsi error kuadratis* adalah *cross-entropy function*. Persamaan fungsi ini untuk *sigmoid unipolar* adalah [14]:

$$E = \sum_k -d_k.\ln(o_k)-(1-d_k).\ln(1-o_k) \quad \text{.....(5.6)}$$

Dengan *cross-entropy function*, *sinyal error* menjadi tidak mengandung suku $o_k.(1-o_k)$, sehingga proses pembelajaran dapat dilakukan dalam waktu yang lebih singkat dari kuadratik.

Algoritme pelatihan dengan pengarahan melibatkan pengurangan nilai kesalahan. Untuk tujuan penyesuaian bobot jaringan, kesalahan yang diminimisasi adalah kesalahan yang dihitung untuk pola yang sedang dikomputasi dalam jaringan (fungsi kesalahan). Untuk tujuan penilaian kualitas dan keberhasilan pelatihan, kesalahan harus dihitung untuk semua pola yang dilibatkan dalam pelatihan. Jenis kesalahan ini disebut kesalahan pelatihan [14].

Untuk mengevaluasi proses pelatihan, dapat digunakan berbagai kriteria kesalahan, antara lain:

1. *Cumulative Error*

$$E = \frac{1}{2}\sum_{p=1}^{P}\sum_{k=1}^{K}(d_{pk}-o_{pk})^2 \quad \text{.....(5.7)}$$

di mana P adalah jumlah pola, K adalah jumlah neuron pada lapis keluaran, d_{pk} adalah target keluaran neuron k untuk pola k dan o_{pk} adalah keluaran neuron k untuk pola k.

Persamaan di atas kurang cocok untuk membandingkan jaringan dengan jumlah pola pelatihan yang berbeda atau jumlah neuron keluaran yang berbeda.

2. *Root-Mean-Square Normalized Error*

$$E_{rms} = \frac{1}{PK}\sqrt{\sum_{p=1}^{P}\sum_{k=1}^{K}(d_{pk}-o_{pk})^2} \qquad \text{.....(5.8)}$$

Nilai E_{rms} sudah dinormalisasi terhadap jumlah pola dan jumlah neuron pada lapis keluaran. Nilai ini lebih deskriptif daripada E untuk membandingkan hasil pelatihan jaringan yang berbeda, dan kriteria ini digunakan dalam tugas akhir ini.

3. *Decision Error*

$$E_d = \frac{N_{err}}{PK} \qquad \text{.....(5.9)}$$

N_{err} adalah total kesalahan bit (bit error) pada K buah neuron keluaran yang di-threshold-kan selama siklus pelatihan. Meskipun E_d=0, jaringan dapat menghasikan E dan E_{rms} yang besar.

Faktor-faktor yang mempengaruhi keberhasilan *algoritme back propagation*, antara lain:

1. Inisialisasi bobot

Bobot awal akan menentukan apakah jaringan akan mencapai *global minima* atau *local minima* kesalahan, dan seberapa cepat jaringan akan konvergen [10]. Inisialisasi acak merupakan cara yang paling mudah dan sering digunakan dalam inisialisasi bobot. Nilai awal bobot biasanya berkisar antara -0,5 dan 0,5. Untuk mempercepat proses pelatihan, suatu modifikasi dari inisialisasi acak diperkenalkan oleh Nguyen dan Widdrow. Inisialisasi *Nguyen-Widdrow* didefinisikan sebagai berikut [10]:

1. Untuk bias dan bobot hubungan antara semua neuron di lapis tersembunyi dan semua neuron di lapis keluaran, lakukan inisialisasi acak dengan jangkauan -0,5 dan 0,5.
2. Untuk bias bobot hubungan antara lapis tersembunyi dan lapis masukan, lakukan sebagai berikut:

a. Hitung $\beta = 0.7\ (H)^{1/N}$ di mana adalah faktor skala, H adalah ukuran lapis tersembunyi dan N adalah ukuran lapis masukan.
b. Untuk setiap neuron j di lapis tersembunyi, lakukan sebagai berikut:
c. Inisialisasi secara acak bobot antara neuron bersangkutan dengan semua neuron di lapis masukan: w_{ij}=bilangan acak antara -0,5 dan 0,5.
d. Hitung norma wj:

$$||w_j|| = \sum_i w_{ij}$$

e. Inisialisasi kembali wij:

$$w_{ij} = \frac{b\ w_{ij}}{||w_j||} \quad \text{.....(5.10)}$$

f. Inisialisasi bias: θj = bilangan acak antara -β dan β.

Sebagai pengganti inisialisasi acak, selanjutnya sering digunakan inisialisasi *Nguyen-Widdrow*.

2. Laju pembelajaran

Merupakan parameter jaringan dalam mengendalikan proses penyesuaian bobot. Nilai laju pembelajaran yang optimal bergantung pada kasus yang dihadapi. Laju pembelajaran yang terlalu kecil menyebabkan konvergensi jaringan menjadi lebih lambat, sedang laju pembelajaran yang terlalu besar dapat menyebabkan ketidakstabilan pada jaringan [14].

3. Momentum

Momentum digunakan untuk mempercepat pelatihan jaringan. Metode momentum melibatkan penyesuaian bobot ditambah dengan faktor tertentu dari penyesuaian sebelumnya. Penyesuaian ini dinyatakan sebagai berikut:

$$\Delta w(t) = -\eta.\nabla E(t) + \alpha.\Delta w(t-1) \qquad \text{……..(5.11)}$$

BPNN merupakan andalan pendekatan *neural network* dan menunjukkan hasil yang lebih baik dari statistik, namun lambat dalam pembelajarannya dan berpotensi terjebak lokal minima [10].

5.2 Probabilistik Neural Network (PNN)

Donald Specht (1990) mengembangkan bentuk *neural network* dengan algoritme yang dapat beroperasi secara paralel berdasarkan teori probabilitas dan selanjutnya disebut *Probabilistic Neural Network* (PNN) [9]. Sebagai sebuah algoritme, PNN dapat dilatih pada beberapa kelas yang digunakan untuk menguji data yang belum diketahui dan untuk memutuskan kelas yang dimilikinya menggunakan teori *probabilita*s dan *probabilitas bersyarat*. Untuk melakukan klasifikasi suatu pola, digunakan atribut-atribut untuk mengukur probabilitas suatu pola termasuk ke dalam kelas tertentu. Jika C_i adalah kelas yang mungkin dimana i = 1, 2, 3, …, maka dapat didefinisikan probabilitas kelas i sebagai $P(C_i)$ dimana $0 \leq P(C_i) \leq 1$. Selanjutnya digunakan probabilitas bersyarat yang memungkinkan untuk memasukkan informasi *prior* dari pola yang akan diklasifikasi untuk meningkatkan kemampuan estimasi terhadap keanggotaan suatu pola pada kelas yang ada. Dengan memasukkan informasi *prior* ke dalam estimasi, akan memberikan pengaruh besar terhadap tingkat keberhasilan estimasi. Dengan mengkombinasi informasi *prior* melalui estimasi keanggotaan pola terhadap suatu kelas, maka permasalahan klasifikasi dapat dinyatakan dengan $P(C_i|X)$ yaitu

probabilitas pola X masuk ke dalam kelas C_i. Jika pola X adalah pola yang ingin di klasifikasi ke dalam kelas C_1, C_2, …, C_3, maka aturan keputusan akan memasukkan pola X ke dalam kelas C_i jika :

$$P(C_i \mid X) > P(C_j \mid X) \text{ untuk i=1, 2, …, n } i \neq j \quad \text{..... (5.12)}$$

Aturan tersebut menyatakan juga, bahwa suatu pola di masukkan ke dalam suatu kelas tertentu apabila pola tersebut memiliki probabilitas bersyarat (*posterior*) tertinggi.

Permasalahan selanjutnya bagaimana menentukan nilai probabilitas bersyarat $P(C_i|X)$ yang diperlukan untuk melakukan klasifikasi. Probabilitas bersyarat harus diestimasi secara akurat untuk menentukan performansi sistem klasifikasi yaitu dengan menggunakan sebuah model distribusi probabilitas dan mengasumsikan bahwa pola tersebut mengikuti kecenderungan yang sama. Dengan menggunakan teori probabilitas, dapat dinyatakan hubungan sebagai berikut [9]:

$$P(C_i \mid X) = \frac{P(X \mid C_i)P(C_i)}{P(x)} \quad \text{……. (5.13)}$$

dan

$$P(x) = \sum_{j=1}^{n} P(X \mid C_j)P(C_j) \quad \text{…….. (5.14)}$$

Keterangan :

$P(X|C_i)$: *probabilitas posterior X untuk kelas Ci*

$P(C_i)$: *probabilitas prior kelas Ci*

P(x) : *probabilitas kumulatif*

n : *jumlah atribut masukan*

i : *nomor kelas*

j : *nomor atribut*

Karena komponen P(x) bersifat *independent* terhadap setiap kelas, maka komponen ini dapat ditiadakan tanpa memperngaruhi proses pengambilan keputusan. Sedangkan komponen $P(C_i)$ dapat diketahui berdasarkan data pelatihan, $P(X|C_i)$ dapat ditentukan menggunakan *Probability Density Function* (pdf) dari pola X terhadap kelas C_i. Bila diasumsikan $P(C_i)$ sama untuk setiap kelas, maka untuk menentukan pola X masuk pada kelas C_i ditentukan oleh nilai pdf tertinggi dari seluruh kelas. Menentukan pdf suatu pola dapat dipecahkan dengan menggunakan model distribusi pola (data).

Untuk distribusi Gaussian, estimasi pdf suatu kelas yang didasarkan pada data pelatihan dapat dinyatakan sebagai berikut :

$$P_i(x) = \frac{1}{(2\pi)^{d/2}\sigma^d}.\sum_{j=1}^{m} exp[-\frac{(x-x_{ij})^T(x-x_{ij})}{2\sigma^2}] \quad \text{..... (5.14)}$$

atau dapat dinyatakan pula dengan :

$$P_i(x) = \frac{1}{(2\pi)^{d/2}\sigma^d}.\sum_{j=1}^{m} exp[-\frac{(V_{ij}.X-1)}{\sigma^2}] \quad \text{..... (5.15)}$$

Keterangan :

$P_l(X)$	:	*pdf dari kelas i untuk vektor x*
d	:	*dimenasi dari pattern vektor x, yaitu jumlah atribut data*
i	:	*nomor kelas*
j	:	*pola pelatihan*
σ	:	*parameter penghalus*
x	:	*vektor uji*
x_{ij}	:	*vektor pelatihan j dari kelas i*

V_{ij} : *bobot neuron input dengan neuron pola, sama dengan vektor pelatihan* x_{ij}

Parameter σ merupakan parameter penting karena mempengaruhi fungsi probabilitas tersebut, nilai σ yang terlalu kecil akan membuat setiap kasus sampel akan sangat banyak berpengaruh, sehingga akan kehilangan manfaat dari informasi kelompok. Sementara jika σ terlalu besar akan kabur, dimana informasi detil dari kerapatan akan hilang dan mengganggu estimasi dari kerapatan.

Arsitektur :

1. Terdiri atas 4 lapis : unit masukan, unit pola, unit jumlah, dan unit keputusan.
2. Terdapat korespondensi satu-satu antara unit pola dan sampel pelatihan sehingga unit pola paling banyak sama dengan jumlah sampel pelatihan.
3. Unit pola terhubung ke unit jumlah jika dan hanya jika data yang bersangkutan merupakan kelas yang sama.
4. Setiap unit keputusan menyatakan satu kelas (kategori obyek).
5. Penentuan Bobot :
    a. Bobot dari unit masukan ke unit pola diinisialisasi dengan sampel pelatihan.
    b. Bobot dari unit jumlah diinisialisasi dengan vektor bernilai 1.
    c. Perhitungan Aktivasi:
    d. Aktivasi dari unit masukan ditentukan oleh data masukan.
    e. Aktivasi Z_j unit pola j dinyatakan dengan :

$$Z_j = \frac{1}{(2\pi)^{d/2}\sigma^d}.f\left(\sum_{i=1}^{m} exp\left[-\frac{(x-x_{ij})^T(x-x_{ij})}{\sigma^2}\right]\right)$$

dimana x_{ij} adalah sampel pelatihan ke-i dan pola ke-j dari kelas pelatihan.

f. Aktivasi P_k dari unit jumlah k dinyatakan dengan :

$$P_k = \frac{1}{m}P(\omega_k).f\left(\sum_j \omega_{jk}.Z_j\right)$$

dimana W_{jk} adalah bobot antara unit pola ke-j dengan unit keluaran ke-k, m adalah jumlah sampel, dan $P(\omega_k)$ adalah probabilitas *prior* kelas (random).

g. Keputusan klasifikasi adalah kelas yang mempunyai aktivasi unit jumlah maksimum.

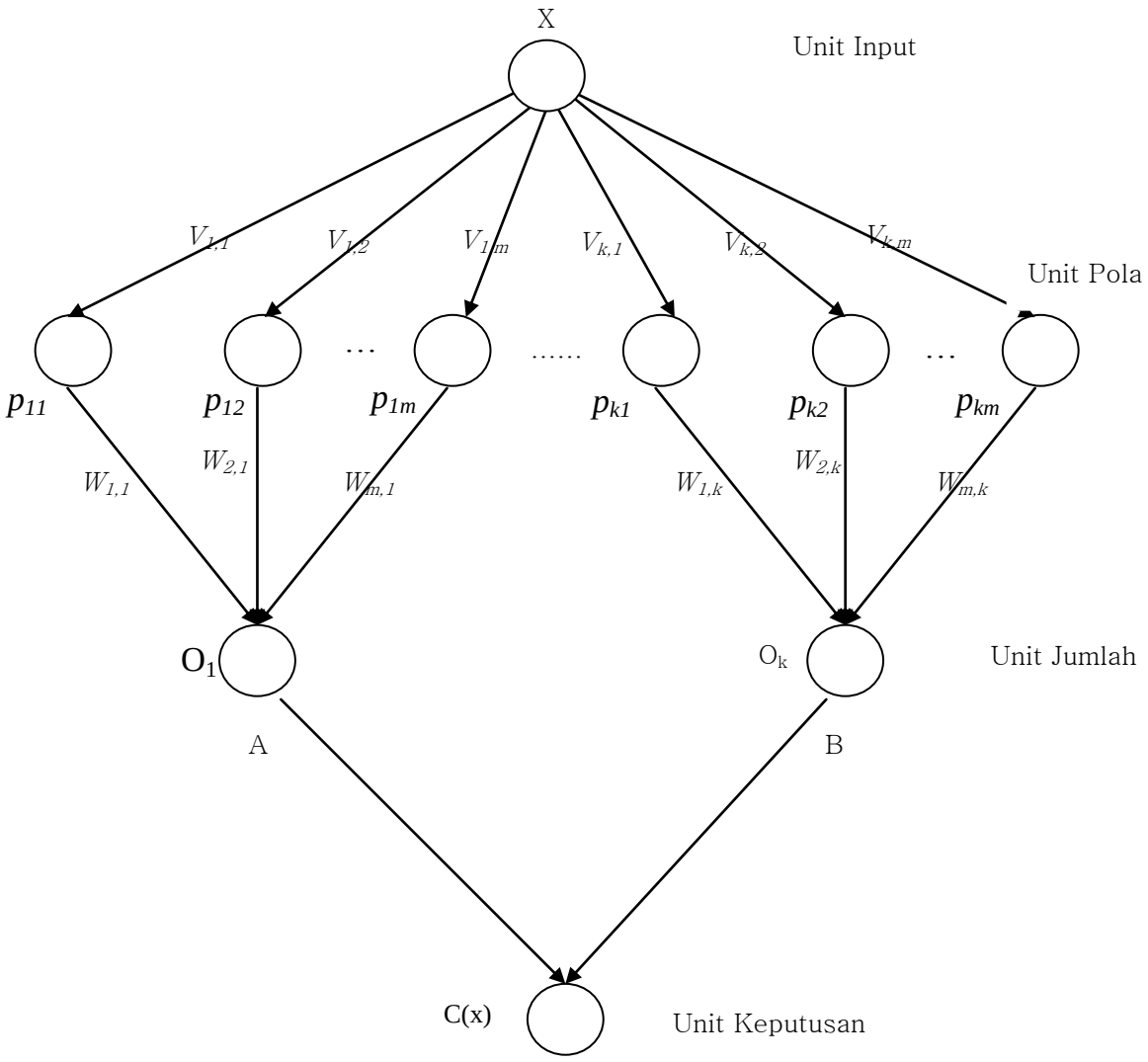
$C(x) = argmax\ P_k$

Pelatihan Jaringan

1. Selama pelatihan bobot tidak dirubah.
2. Ketika masuk data baru, sebuah unit pola baru ditambahkan dan bobot yang berhubungan diinisialisasi dengan cara di atas.
3. Pelatihan jaringan sifatnya inisialisasi semua variabel, dan keputusan jaringan adalah kelas yang mempunyai unit keluaran dengan aktivasi maksimum.

Beberapa penelitian menunjukkan bahwa PNN mempunyai kinerja lebih baik dari pada BPNN dalam akurasi klasifikasi dan lebih cepat dalam pembelajarannya [4, 9]. PNN masih mendapat kritikan dengan jumlah neuron pola yang cenderung

bertambah seiring jumlah pola pelatihan sehingga waktu komputasi pengujian menjadi naik. Namun setiap pengklasifikasi *neural network* yang digunakan untuk citra inderaja umumnya perlu menggunakan *neuron hidden* (pola) yang lebih besar dari masalah biasa demikian pula untuk BPNN [2, 4]. Dengan demikian PNN masih relatif singkat dalam tingkat komputasi dan sebaliknya untuk BPNN, neuron *hidden* yang besar dapat mengakibatkan terjadinya osilasi [14]. Dilain pihak pertambahan neuron pola pada PNN penting karena merupakan tingkat akomodasi pengklasifikasi terhadap data masukan [9].



Gambar-5.2: Arsitektur *Probabilistic Neural Network (PNN)*