BAB II
LANDASAN TEORI

2.1 Pengenalan Suara

2.1.1 Pengertian Pengenalan Suara

Pengenalan suara (*speech recognition*) adalah suatu proses untuk mengenali huruf, kata atau kalimat yang diucapkan. Pengenalan suara lebih dikenal dengan istilah *Automatic Speech Recognition* atau *Computer Speech Recognition* dimana penggunaan sebuah mesin/komputer untuk mengenali sebuah suara atau identitas seseorang dari suara yang diucapkan. Umumnya pengucap berbicara di depan komputer/mesin kemudian komputer/mesin mengenali suara/identitas seseorang dengan tepat sesuai yang diucapkan. Pengenalan pola suara dikenali ke dalam berbagai level tugas, pengenalan dalam tingkat sinyal akustik berupa uji tingkatan dalam susunan unit sub kata berupa fonem, kata, frase dan kalimat. Pengenalan suara huruf vokal merupakan dasar dari pengenalan suara sebab susunan kata merupakan susunan dari beberapa huruf salah satunya adalah huruf vokal sehingga jika diperoleh prinsip dasar proses pengenalan dari suara huruf vokal dapat digunakan dalam penelitian lebih lanjut.
(http://elib.unikom.ac.id/download.php?id=22718)

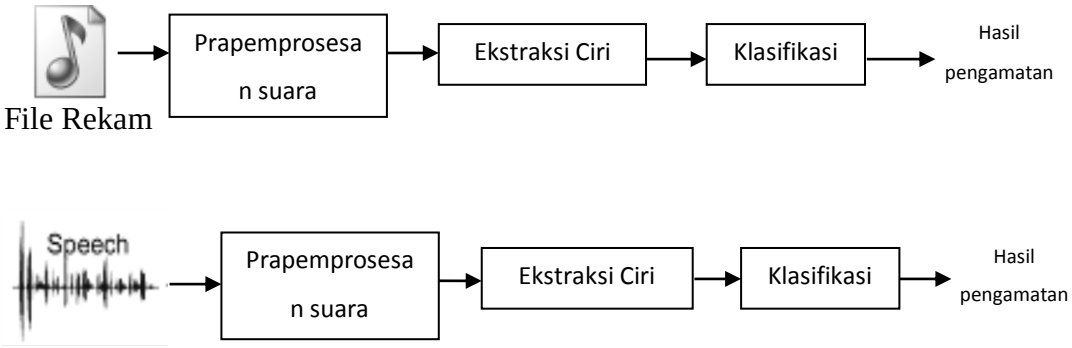
2.1.2 Proses Pengenalan Suara secara Identifikasi dan Verifikasi

Proses pengenalan suara terbagi menjadi *Verification* dan *Identification*. *Speech Identification* adalah proses pelatihan seseorang atau huruf yang diucapkan ke pengenalan suara dengan cara mendaftarkan pembicara dari ucapan yang diberikan. *Speech Verification* adalah proses penentuan identitas pembicara atau arti dari suara yang diucapkan oleh pembicara yang dibandingkan dengan data yang telah tersimpan pada sistem.

2.1.3 Proses Pengenalan Suara *Offline* dan *Online*

Sistem *offline* adalah suatu sistem yang menghasilkan output dengan bantuan proses secara manual oleh user. Sedangkan sistem *online* adalah sistem yang menghasilkan output tanpa bantuan proses secara manual oleh user.

Umumnya pengenalan suara memiliki tahap pelatihan/identifikasi dan verifikasi. Pada gambar 2.1 proses identifikasi memiliki tahap normalisasi, ekstraksi ciri, klasifikasi. Proses pengenalan suara *offline* maupun *online* terdapat proses identifikasi namun terdapat perbedaan pada proses verifikasi, pada pengenalan suara secara *offline* verifikasi dilakukan dengan cara suara yang akan dikenali direkam terlebih dahulu sebelum memulai proses pengenalan suara sedangkan jika pada proses secara *online* verifikasi dilakukan dengan dinamis yaitu menggunakan pengucapan suara langsung tanpa melalui proses perekaman terlebih dahulu.

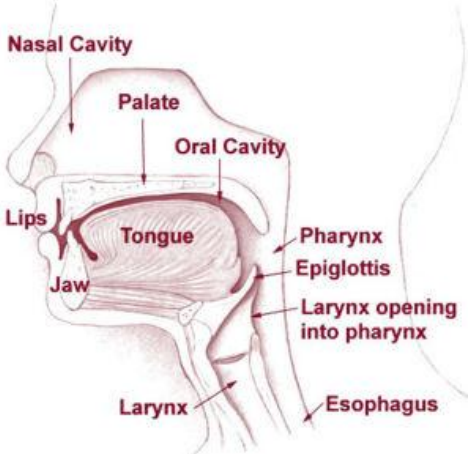


Gambar 2.1: Perbedaan proses verifikasi suara pada proses *Offline* (atas) dan *Online* (Bawah)

2.1.7 Pemrosesan Suara

Organ tubuh yang berpengaruh dalam proses produksi wicara adalah paru-paru, tenggorokan (*trachea*), larinks, farinks, rongga hidung (*nasal cavity*), dan rongga mulut (*oral cavity*). Pembangkitan sinyal suara terletak pada bentuk lintasan vokalnya (*vocal tract*). Lintasan vocal tersebut terdiri atas: dibawah katub tenggorokan (*laryngeal pharynx*), antara langit-langit lunak katub tenggorokan

(*oral pharynk*), di atas velum dan diujung depan rongga hidung (*nasal pharynx*), dan rongga hidung (*nasal cavity*), seperti ditunjukkan pada gambar 2.2**.**



Gambar 2.2 Organ suara manusia

Untuk menghasilkan sebuah *voiced* sounds (suara ucapan), paru-paru menekan udara melalui *epiglottis*, *vocal cords* bergetar, meng-*interrupt* udara melalui aliran udara dan menghasilkan sebuah gelombang tekanan *quasi-periodic*. Impuls tekanan pada umumnya disebut sebagai *pitch impulses* dan frekuensi sinyal tekanan adalah *pitch frequency* atau *fundamental frequency.*

Impuls *pitch* merangsang udara di dalam mulut dan untuk suara tertentu (*nasals*) juga merangsang *nasal cavity* (rongga hidung). Ketika rongga beresonansi, akan menimbulkan radiasi sebuah gelombang suara yang mana merupakan sinyal wicara. Kedua rongga beraksi sebagai resonator dengan karakteristik frekuensi resonansi masing-masing, yang disebut *formant frequencies*. Pada saat rongga mulut dapat mengalami perubahan besar, kita mampu untuk menghasilkan beragam pola ucapan suara yang berbeda. Sinyal suara merupakan hasil dari suara manusia, dimana sinyal suara mempunyai frekuensi kerja antara 300 sampai 3400Hz.

2.2 Sistem Pengenalan Suara

Secara garis besar, cara kerja sistem pengenalan suara ini ialah mula-mula sinyal suara manusia yang diterima dengan menggunakan *microphone* (sinyal

analog) dicuplik sehingga menjadi sinyal *digital* dengan bantuan *sound card* pada *Personal Computer*. Sinyal *digital* hasil cuplikan ini terlebih dulu dinormalisasi (disamakan panjang sinyal yang satu dengan yang lain) kemudian diproses dengan Fast Fourier Transform (FFT) untuk mendapatkan sinyal pada domain frekuensi. Hal ini bertujuan agar perbedaan antar pola kata yang satu dengan yang lain terlihat lebih jelas sehingga ekstraksi parameter sinyal memberikan hasil yang lebih baik. Hasil keluaran FFT ini merupakan masukan bagi jaringan saraf tiruan Back Propagation dimana jaringan saraf tiruan ini sebagai fungsi utama dari sistem untuk proses pengenalan suara.

2.2.1 Pencuplikan Suara

Dalam tahap ini merupakan penentuan jumlah *sampel* dalam satu detik. Jika pencuplikan dilakukan dengan frekuensi cuplik 8000Hz, maka dalam satu detik terdapat 8000 sampel. Perlu diperhatikan komponen utama frekuensi sinyal suara berada pada kisaran 300–3400 Hz. Menurut

Nyquist, frekuensi *sampling* dalam pencuplikan harus lebih besar 2 kali dari frekuensi sinyal aslinya. Sesuai dengan persamaan Nyquist, fs< 2fh dimana fh =fin. Semakin tinggi frekuensi *sampling* maka sinyal *digital* yang dihasilkan semakin bagus. Kemudian proses selanjutnya adalah *kuantisasi*, yaitu membatasi amplitude atau nilai aksis sinyal. Jika sinyal dicuplik dengan menggunakan resolusi 8 bit, maka terdapat 28 atau 256 nilai batas sinyal. Langkah terakhir adalah konversi *analog* ke *digital* adalah tahap *coding*. Karena memori computer menyimpan data berupa tipe data biner, maka nilai amplitude tiap *sampel* sinyal akan dikonversi kedalam bentuk biner. Jika sinyal dicuplik dalam resolusi 8bit, maka nilai amplitude akan disimpan pada ukuran 8bit kode biner atau 1 byte data.

2.2.2 Normalisasi

Satu masalah yang cukup rumit dalam speech recognition (pengenalan suara) adalah proses perekaman yang terjadi sering kali berbeda durasinya, biarpun kata atau kalimat yang diucapkan sama. Bahkan untuk satu suku kata atau vocal yang sama seringkali proses perekaman terjadi dalam durasi yang berbeda.

Sebagai akibatnya proses matching antara sinyal uji dengan sinyal referansi (template) seringkali tidak menghasilkan nilai yang optimal. Sebuah teknik yang cukup popular di awal perkembangan teknologi pengolahan sinyal suara adalah dengan cara memanfaatkan sebuah teknik dynamic-programing yang juga lebih dikenal sebagai dynamic time warping (DTW). Teknik ini ditujukan untuk mengakomodasi perbedaan waktu antara proses perekaman saat pengujian dengan yang tersedia pada template sinyal referensi. Prinsip dasarnya adalah dengan memberikan sebuah rentang 'step' dalam ruang (dalam hal ini sebuah frame-frame waktu dalam sampel, frame-frame waktu dalam (template) dan digunakan untuk mempertemukan lintasan yang menunjukkan local match terbesar (kemiripan) antara time frame yang lurus. Total 'similarity cost' yang diperoleh dengan algoritma ini merupakan sebuah indikasi seberapa bagus sampel dan template ini memiliki kesamaan, yang selanjutnya akan dipilih best-matching template.

(Tri Budi Santoso dan Miftahul Huda, modul pengolahan sinyal wicara: hal 54)

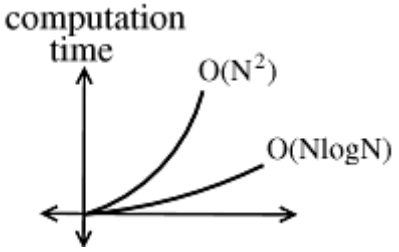
2.2.3 Fast Faurier Transform (FFT)

Transformasi Fourier merubah sebuah sinyal domain waktu untuk ditampilkan menjadi sinyal domain Frekuensi. Bagaimanapun juga, sebuah sample yang berbentuk gelombang (*waveform*) dan ditransformasikan dalam bentuk nilai diskrit. Karena transformasi ini, transformasi Fourier tidak akan bekerja dalam bentuk data diskrit. Akhirnya digunakan *Discrete Fourier Transform* (DFT), yang menghasilkan hasil akhir dalam domain frekuensi dengan nilai diskrit atau disebut *bins*. *Fast Fourier Transform* merupakan algoritma yang efisien untuk proses perhitungan DFT (*Discrete Fourier Transform*) dan inversenya dengan membutuhkan sedikit proses aritmetika. Dimana DFT didefinisikan dengan rumus 2.1:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} nk} \quad k = 0, \dots, N-1 \quad \text{..... (2.1)}$$

dimana: x_0, ..., x_{N-1} berupa angka komplek

Proses penjumlahan rumus tersebut secara langsung dapat menghabiskan $O(N^2)$ operasi aritmetika. Namun jika digunakan algoritma FFT hanya diperlukan $O(N\log N)$ operasi. Perbandingan kecepatan eksekusi DFT dan FFT dapat dilihat pada tabel 2.2:



N	10	100	1000	10^6	10^9
N^2	100	10^4	10^6	10^{12}	10^{18}
NlogN	10	200	3000	$6x10^6$	$9x10^9$

Gambar 2.2**:** Waktu komputasi yang diperlukan untuk proses $O(N\log N)$

Misalnya jika sebuah mesin memiliki 1 MFLOP (satu juta floating point per detik). Dimana N = 1 juta = 10^6. Sehingga sebuah algoritma $O(N^2)$ memerlukan 10^{12} flot atau 10^6 detik $\simeq$ 11.5 hari. Jika sebuah algoritma $O(N\log N)$ memerlukan $6x10^6$ Flot atau 6 detik. Namun N = 1 juta tidak mungkin. Contohnya, 3 mega pixel kamera digital dapat menghasilkan $3x10^6$ angka untuk tiap foto. Sehingga untuk dua titik N berulang *f[n]* dan *h[n]*. Jika menghitung (*f[n]*⊛*h[n]*) secara langsung memerlukan sejumlah $O(N^2)$ operasi.

Perhitungan FFT -- $O(N\log N)$

Perkalian FFT -- $O(N)$

Inverse FFT -- $O(N\log N)$.

Total kompleksitas adalah $O(N\log N)$.

Sedangkan Persamaan FFT adalah sebagai berikut ini:

$$X(f) = F\{x(t)\} = \int_{-\infty}^{\infty} x(t)e^{-j2\pi ft}dt \quad \text{..... (2.2)}$$

FFT Sama seperti DFT yaitu memetakan perulangan dalam bentuk nilai diskrit-waktu kemudian direpresentasikan menjadi diskrit-frekuensi.

Ketika menggunakan DFT, transformasi Fourier pada tiap perulangan *x*, dimana x berbentuk nilai real atau komplek, selalu menghasilkan dalam sebuah keluaran perulangan nilai komplek X sesuai dengan persamaan berikut:

$$F(x) = X = X_{Re} + jX_{Im} = Re\{X\} + jIm\{X\} \quad \text{..... (2.3)}$$

Suatu hal yang tidak dipisahkan dengan properti DFT berikut ini:

$$X_{n-i} = X_{-i} \quad \text{..... (2.4)}$$

Dimana elemen (n-i) dari X berisi hasil –*i* yang harmonik. Selanjutnya jika x real, maka *i* harmonik dan –*i* harmonik adalah bernilai komplek konjugate:

$$X_{n-i} = X_{-i} = X_i^* \quad \text{..... (2.5)}$$

Konsekuensinya,

$$Re(X_i) = Re(X_{i-1}) \quad \text{..... (2.6)}$$

dan

$$Im(X_i) = -Im(X_{i-1}) \quad \text{.....(2.7)}$$

Properti Fourier yang simetris ini adalah perulangan real mengacu sebagai simetri konjugate (Persamaan 2.5), simetris atau genap-simetris (persamaan 2.6), dan anti-simetris atau ganjil-simetris (persamaan 2.7).

Penggunaan FFT untuk analisa frekuensi menandakan dua relasi penting , yaitu:

1) Relasi pertama yaitu frekuensi tertinggi dapat dianalisa (F_{max}) dengan frequensi sampling (f_s).

$$F_{max} = \frac{f_s}{2} \quad \text{.....(2.8)}$$

2) Relasi kedua yaitu resolusi frekuensi (f) sampai total waktu akuisisi (T), yang berhubungan dengan frekeunsi sampling (f_s) dan ukuran blok FFT (N).

$$\Delta f = \frac{1}{T} = \frac{f_s}{N} \quad \text{.....(2.9)}$$

Keluaran spektrum FFT bernilai komplek, sehingga setiap komponen frekuensi memiliki sebuah amplitudo dan fasa. Nilai fasa tergantung waktu perekaman, atau tergantung dari gelombang cosinus yang dimulai pada waktu pertama perekaman. Pengukuran chanel satu fasa bernilai stabil hanya jika sinyal input dibangkitkan. Sedangkan pengukuran dua chanel fasa perhitungan perbedaan fasa berbeda antara chanel-chanel lain sehingga jika chanel secara simultan disample, pembangkitan biasanya tidak penting. Untuk mencari amplitudo dapat diperoleh dengan persamaan berikut:

$$A[k] = \sqrt{Re(X[k])^2 + Im(X[k])^2} \text{.....(2.10)}$$

Dan fasa dihitung dengan persamaan:

$$Fasa(X[k]) = \arctan\left(\frac{Re(X[k])}{Im(X[k])}\right) \text{.....(2.11)}$$

FFT dikembangkan oleh Cooley dan Tukey pada tahun 1965. Algoritma FFT merupakan penyederhanaan dari Diskrit Fourier Transform (DFT) yang memiliki persyaratan jumlah data harus merupakan bilangan 2n untuk n=0,1,2,3,…….. Waktu komputasi DFT memiliki kompleksitas N2 sedangkan FFT memiliki kompleksitas Np/2 dengan p= 2logN, sehingga FFT lebih cepat daripada DFT (New York: McGraw- Hill,2004,hal: 554).

$$\frac{N^2}{Np/2} + \frac{2N}{p}$$

2.3 Sejarah Jaringan Syaraf Tiruan

Jaringan syaraf tiruan sederhana pertama kali diperkenalkan oleh McCulloch dan Pitts di tahun 1943. McCulloch dan Pitts menyimpulkan bahwa kombinasi beberapa *neuron* sederhana menjadi sebuah sistem neural akan

meningkatkan kemampuan komputasinya. Bobot dalam jaringan yang diusulkan oleh McCulloch dan Pitts diatur untuk melakukan fungsi logika sederhana. Fungsi aktivasi yang dipakai adalah fungsi *threshold*.

Tahun 1958, Rosenblatt memperkenalkan dan mulai mengembangkan model jaringan yang disebut Perceptron. Metode pelatihan diperkenalkan untuk mengoptimalkan hasil iterasinya.

Widrow dan Hoff (1960) mengembangkan perceptron dengan memperkenalkan aturan pelatihan jaringan, yang dikenal sebagai aturan delta (atau sering disebut kuadrat rata-rata terkecil). Aturan ini akan mengubah bobot perceptron apabila keluaran yang dihasilkan tidak sesuai dengan target yang diinginkan.

Apa yang dilakukan peneliti terdahulu hanya menggunakan jaringan dengan *layer* tunggal (*single layer*). Rumelhart (1986) mengembangkan perceptron menjadi *Backpropagation*, yang memungkinkan jaringan diproses melalui beberapa *layer*.

Selain itu, beberapa model jaringan syaraf tiruan lain juga dikembangkan oleh Kohonen (1972), Hopfield (1982), dan sebagainya.

Pengembangan yang ramai dibicarakan sejak tahun 1990 an adalah aplikasi model-model jaringan syaraf tiruan untuk menyelesaikan berbagai masalah di dunia nyata.

2.3.1 Jaringan Syaraf Tiruan

Jaringan syaraf tiruan (*artificial neural networks*) atau disingkat JST adalah sistem pemroses informasi yang memiliki karakteristik mirip dengan jaringan syaraf biologi.

JST dibentuk sebagai generalisasi model matematika dari jaringan syaraf biologi, dengan asumsi :

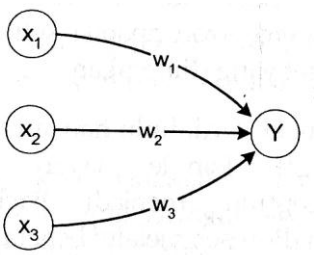
a. Pemrosesan informasi terjadi pada banyak elemen sederhana (*neuron*)
b. Sinyal dikirimkan diantara *neuron-neuron* melalui penghubung-penghubung
c. Penghubung antar *neuron* memiliki bobot yang akan memperkuat atau memperlemah sinyal

d. Untuk menentukan *output*, setiap *neuron* menggunakan fungsi aktivasi (biasanya bukan fungsi linier) yang dikenakan pada jumlahan *input* yang diterima. Besarnya *output* ini selanjutnya dibandingkan dengan suatu batas ambang.

JST ditentukan oleh tiga hal :

a. Pola hubungan antar *neuron* (disebut arsitektur jaringan)
b. Metode untuk menentukan bobot penghubung (disebut metode *training* / *learning* / algoritma)
c. Fungsi aktivasi

Sebagai contoh, perhatikan *neuron* Y pada Gambar 2.3.



Gambar 2.3. Model Sederhana Jaringan Syaraf Tiruan

Y menerima *input* dari *neuron* x_1, x_2 dan x_3 dengan bobot hubungan masing-masing adalah w_1, w_2 dan w_3. Ketiga impuls *neuron* yang ada dijumlahkan

$$net = x_1w_1 + x_2w_2 + x_3w_3$$

Besarnya impuls yang diterima oleh Y mengikuti fungsi aktivasi y = f(net). Apabila nilai fungsi aktivasi cukup kuat, maka sinyal akan diteruskan. Nilai fungsi aktivasi (keluaran model jaringan) juga dapat dipakai sebagai dasar untuk merubah bobot.

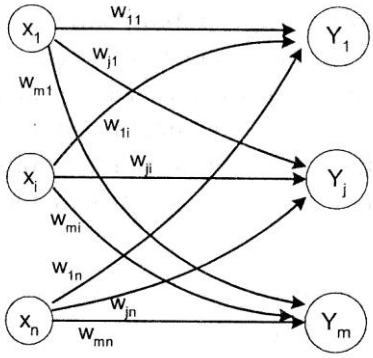
2.3.2 Arsitektur Jaringan

Beberapa arsitektur jaringan yang sering dipakai dalam jaringan syaraf tiruan antara lain :

a. Jaringan Layar Tunggal (*single layer network*)

JST dengan layar tunggal pertamakali dirancang oleh Widrow dan Hoff pada tahun 1960. Walaupun JST layar tunggal ini sangat terbatas penggunaannya, namun konsep dan gagasannya banyak dipakai oleh beberapa pakar untuk membuat model JST layar jamak.

Dalam jaringan ini, sekumpulan *input neuron* dihubungkan langsung dengan sekumpulan *output*. Dalam beberapa model (misal perceptron), hanya ada sebuah unit *neuron output*.



Gambar 2.4. Jaringan Layar Tunggal

Gambar 2.4. menunjukkan arsitektur jaringan dengan n unit *input* (x_1,x_2, ... , x_n) dan m buah unit *output* (Y_1, Y_2, ... , Y_m).

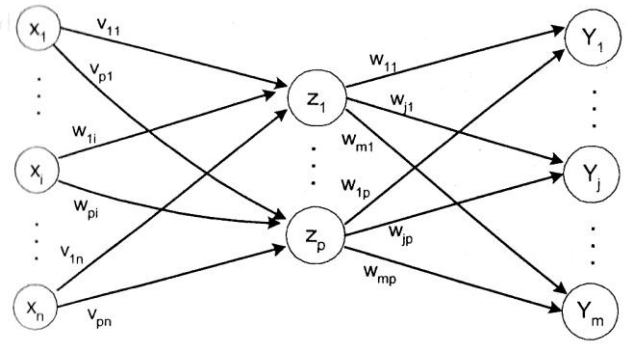
Perhatikan bahwa dalam jaringan ini, semua unit *input* dihubungkan dengan semua unit *output*, meskipun dengan bobot yang berbeda-beda. Tidak ada unit *input* yang dihubungkan dengan unit *input* lainnya. Demikian pula dengan unit *output*.

Besarnya w_{ji} menyatakan bobot hubungan antara unit ke-i dalam *input* dengan unit ke-j dalam *output*. Bobot-bobot ini saling independen. Selama proses pelatihan, bobot-bobot tersebut akan dimodifikasi untuk meningkatkan keakuratan hasil.

b. Jaringan Layar Jamak (*multi layer network*)

Jaringan layar jamak merupakan perluasan dari layar tunggal. Dalam jaringan ini, selain unit *input* dan *output*, ada unit-unit lain (sering disebut layar tersembunyi). Dimungkinkan pula ada beberapa layar tersembunyi.

Sama seperti pada unit *input* dan *output*, unit-unit dalam satu layar tidak saling berhubungan.



Gambar 2.5. Jaringan Layar Jamak

Gambar 2.5. adalah jaringan dengan n buah unit *input* (x_1, x_2, ... , x_n), sebuah layar tersembunyi yang terdiri dari p buah unit (z_1, ... , z_p) dan m buah unit *output* (Y_1, Y_2, ... , Y_m)

Jaringan layar jamak dapat menyelesaikan masalah yang lebih kompleks dibandingkan dengan layar tunggal, meskipun kadangkala proses pelatihan lebih kompleks dan lama.

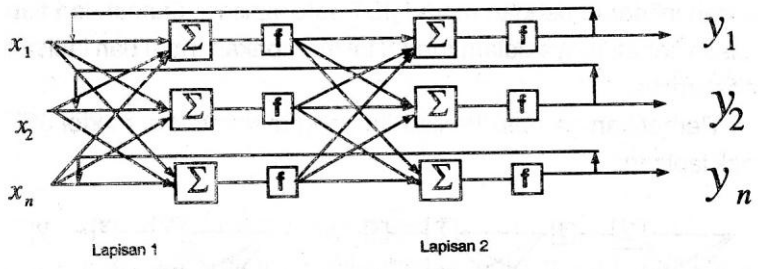
c. Model JST dua lapisan dengan umpan balik

Tokoh yang pertamakali mencetuskan ide tentang model jaringan syaraf tiruan dengan umpan balik adalah John Hopfield dari California Institute of Technology pada tahun 1982. Hopfield berpendapat bahwa kumpulan *neuron* tiruan dalam jumlah yang sangat besar dapat melakukan tugas-tugas tertentu.

Hopfield juga membandingkan antara jumlah *neuron* pada binatang dengan jumlah *neuron* diperkirakan sekitar 1000 buah dan bila dibandingkan dengan manusia, jumlah *neuron*-nya mencapai 100 trilyun buah. Sungguh jumlah yang sangat fantastis.

Dengan jumlah *neuron* yang sangat besar, JST memiliki sifat yaitu *fault tolerance*. Sifat ini mengandung maksud kerusakan sedikit atau sebagian pada sel-sel dalam jaringan tidak akan mempengaruhi *output* yang akan dikeluarkan.

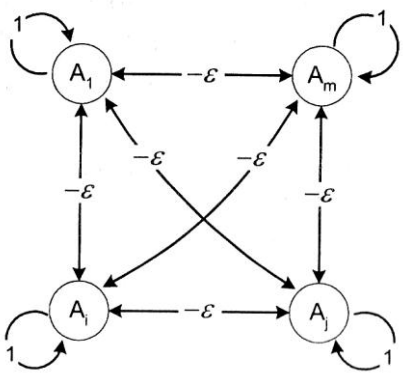
Model JST dua lapisan pada gambar 2.6 ini mempunyai sifat umpan balik, sehingga *output* yang dihasilkan akan mempengaruhi *input* yang akan masuk lagi ke dalam jaringan syaraf tersebut.



Gambar 2.6. Model JST Dua Lapisan Dengan Umpan Balik

d. Model JST lapisan kompetitif

Bentuk dari lapisan kompetitif merupakan bagian dari jumlah yang besar pada jaringan syaraf. Pada dasarnya, hubungan antara *neuron* satu dengan *neuron* yang lain pada lapisan kompetitif tidak ditunjukkan secara arsitektur pada beberapa jaringan syaraf. Contoh dari model atau arsitektur lapisan kompetitif dapat dilihat pada Gambar 2.7, dimana koneksi dari lapisan tersebut memiliki bobot –ε.



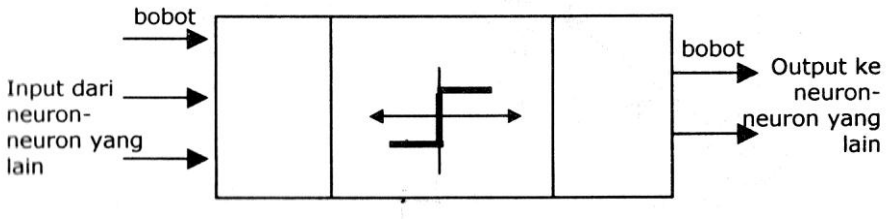
Gambar 2.7. Model JST Lapisan Kompetitif

2.4 Fungsi Aktivasi

Jaringan syaraf merupakan salah satu representasi buatan dari otak manusia yang selalu mencoba untuk mensimulasikan proses pembelajaran pada otak manusia tersebut. Istilah buatan disini digunakan karena jaringan syaraf ini

diimplementasikan dengan menggunakan program komputer yang mampu menyelesaikan sejumlah proses perhitungan selama proses pembelajaran.

Ada beberapa tipe jaringan syaraf, namun demikian, hampir semuanya memiliki komponen-komponen yang sama. Seperti halnya otak manusia, jaringan syaraf juga terdiri dari beberapa *neuron*, dan ada hubungan antara *neuron-neuron* tersebut. *Neuron-neuron* tersebut akan mentransformasikan informasi yang diterima melalui sambungan keluarnya menuju ke *neuron-neuron* yang lain. Pada jaringan syaraf, hubungan ini dikenal dengan nama bobot. Informasi tersebut disimpan pada suatu nilai tertentu pada bobot tersebut. Gambar 2.8 menunjukkan struktur *neuron* pada jaringan syaraf.

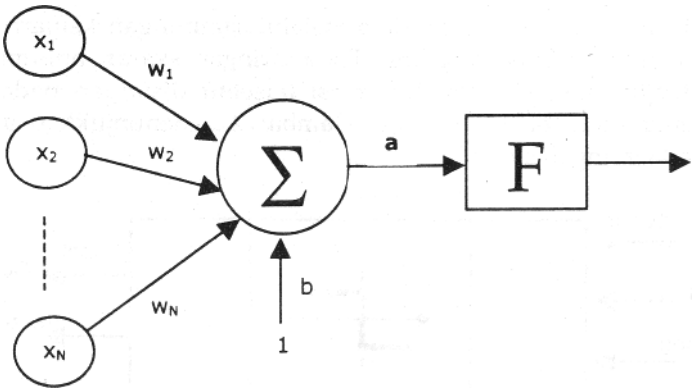


Gambar 2.8. Struktur Neuron Jaringan Syaraf

Jika dilihat, *neuron* buatan ini sebenarnya mirip dengan sel *neuron* biologis. *Neuron-neuron* buatan tersebut bekerja dengan cara yang sama pula dengan *neuron-neuron* biologis. Informasi (disebut dengan : *input*) akan dikirim ke *neuron* dengan bobot kedatangan tertentu. *Input* ini akan diproses oleh suatu fungsi perambatan yang akan menjumlahkan nilai-nilai semua bobot yang datang. Hasil penjumlahan ini kemudian akan dibandingkan dengan suatu nilai ambang (*threshold*) tertentu melalui fungsi aktivasi setiap *neuron*. Apabila *input* tersebut melewati suatu nilai ambang tertentu, maka *neuron* tersebut akan diaktifkan, tapi kalau tidak, maka *neuron* tersebut tidak akan diaktifkan. Apabila *neuron* tersebut diaktifkan, maka *neuron* tersebut akan mengirimkan *output* melalui bobot-bobot *output* nya ke semua *neuron* yang berhubungan dengannya. Demikian seterusnya.

Pada jaringan syaraf, *neuron-neuron* akan dikumpulkan dalam lapisan-lapisan (*layer*) yang disebut dengan lapisan *neuron* (*neuron layers*). Biasanya *neuron-neuron* pada satu lapisan akan dihubungkan dengan lapisan-lapisan sebelum dan sesudahnya (kecuali lapisan *input* dan lapisan *output*). Informasi

yang diberikan pada jaringan syaraf akan dirambatkan lapisan ke lapisan, mulai dari lapisan *input* sampai ke lapisan *output* melalui lapisan yang lainnya, yang sering dikenal dengan nama lapisan tersembunyi (*hidden layer*). Tergantung pada algoritma pembelajarannya, bisa jadi informasi tersebut akan dirambatkan secara mundur pada jaringan. Gambar 2.9 menunjukkan jaringan syaraf sederhana dengan fungsi aktivasi F.



Gambar 2.9. Fungsi Aktivasi Pada Jaringan Syaraf Sederhana

Pada Gambar 2.9. tersebut sebuah *neuron* akan mengolah N *input* (x_1, x_2, ... , x_N) yang masing-masing memiliki bobot w_l, w_2, ... , w_N dan bobot bias b, dengan rumus :

$$a = b + \sum_{i=1}^{N} x_i w_i$$

kemudian fungsi aktivasi F akan mengaktivasi a menjadi *output* jaringan y.

Ada beberapa fungsi aktivasi yang sering digunakan dalam jaringan syaraf tiruan. Fungsi Aktivasi yang digunakan pada *Backpropagation* antara lain :

a. Fungsi sigmoid biner
b. Fungsi sigmoid bipolar
c. Fungsi linear

a. Fungsi sigmoid biner

Dalam *backpropagation*, fungsi aktivasi yang dipakai harus memenuhi beberapa syarat yaitu : kontinu, terdiferensial dengan mudah dan merupakan fungsi yang tidak turun. Salah satu fungsi yang memenuhi ketiga syarat

tersebut sehingga sering dipakai adalah fungsi sigmoid biner yang memiliki range (0,1).

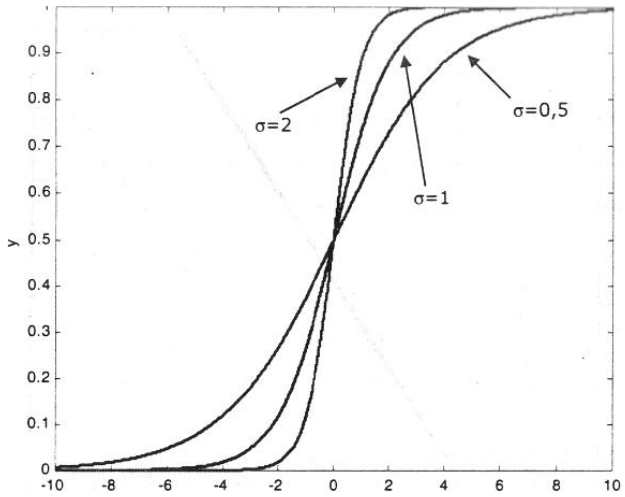
Fungsi ini digunakan untuk jaringan syaraf yang dilatih dengan menggunakan metode *backpropagation*. Fungsi sigmoid biner memiliki nilai pada *range* 0 sampai 1. Oleh karena itu, fungsi ini sering digunakan untuk jaringan syaraf yang membutuhkan nilai *output* yang terletak pada interval 0 sampai 1. Namun, fungsi ini bisa juga digunakan oleh jaringan syaraf yang nilai *output* nya 0 atau 1 (Gambar 2.10).

Fungsi sigmoid biner dirumuskan sebagai :

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

dengan :

$$f'(x) = \sigma f(x)(1 - f(x))$$



Gambar 2.10. Fungsi Aktivasi Sigmoid Biner

b. Fungsi sigmoid bipolar

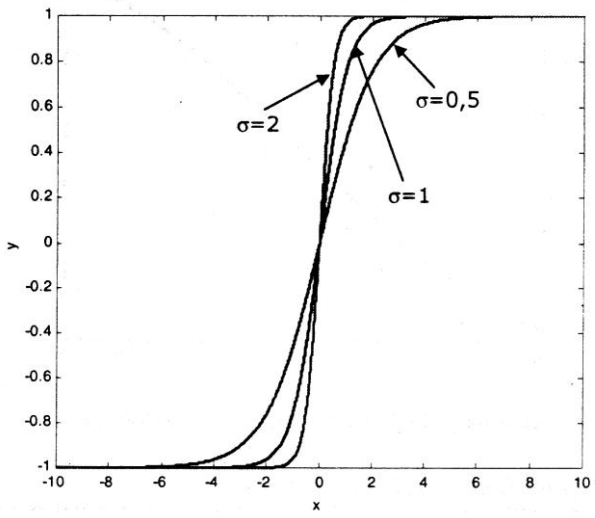
Fungsi sigmoid bipolar hampir sama dengan fungsi sigmoid biner, hanya saja *output* dari fungsi ini memiliki *range* antara 1 sampai -1 (Gambar 2.11).

Fungsi sigmoid bipolar dirumuskan sebagai :

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

dengan:

$$f'(x) = \frac{\sigma}{2}[1 + f(x)][1 - f(x)]$$



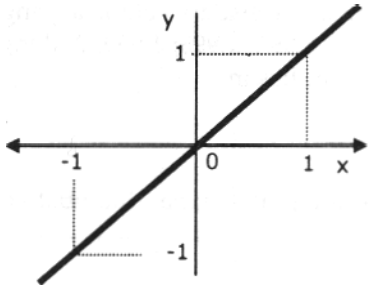
Gambar 2.11. Fungsi Aktivasi Sigmoid Bipolar

c. Fungsi linear (identitas)

Fungsi linear memiliki nilai *output* yang sama dengan nilai *input* (Gambar 2.12).

Fungsi linear dirumuskan sebagai :

y = x



Gambar 2.12. Fungsi Aktivasi Linear (Identitas)

2.5. Backpropagation

Kelemahan JST yang terdiri dari layar tunggal membuat perkembangan JST menjadi terhenti pada sekitar tahun 1970 an. Penemuan *backpropagation* yang terdiri dari beberapa layar membuka kembali cakarawala. Terlebih setelah berhasil ditemukannya berbagai aplikasi yang dapat diselesaikan dengan *backpropagation*, membuat JST semakin diminati orang.

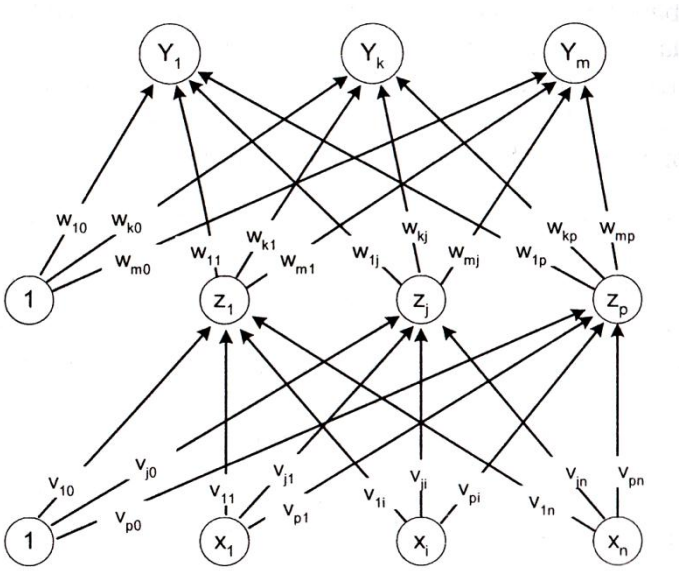
JST dengan layar tunggal memiliki keterbatasan dalam pengenalan pola. Kelemahan ini bisa ditanggulangi dengan menambahkan satu/beberapa layar tersembunyi diantara layar masukan dan keluaran. Meskipun penggunaan lebih dari satu layar tersembunyi memiliki kelebihan manfaat untuk beberapa kasus, tapi pelatihannya memerlukan waktu yang lama. Maka umumnya orang mulai mencoba dengan sebuah layar tersembunyi lebih dahulu.

Seperti halnya model JST lain, *Backpropagation* melatih jaringan untuk mendapatkan keseimbangan antara kemampuan jaringan untuk mengenali pola yang digunakan selama pelatihan serta kemampuan jaringan untuk memberikan respon yang benar terhadap pola masukan yang serupa (tapi tidak sama) dengan pola yang dipakai selama pelatihan. [JON04]

2.5.1. Arsitektur Backpropagation

Backpropagation memiliki beberapa unit yang ada dalam satu atau lebih layar tersembunyi. Gambar 2.13 adalah arsitektur *backpropagation* dengan n buah masukan (ditambah sebuah bias), sebuah layar tersembunyi yang terdiri dari p unit (ditambah sebuah bias), serta m buah unit keluaran.

v_{ji} merupakan bobot garis dari unit masukan x_i ke unit layar tersembunyi z_j (v_{j0} merupakan bobot garis yang menghubungkan bias di unit masukan ke unit layar tersembunyi z_j). w_{kj} merupakan bobot dari unit layar tersembunyi z_j ke unit keluaran y_k (w_{k0} merupakan bobot dari bias di layar tersembunyi ke unit keluaran z_k).



Gambar 2.13. Arsitektur Backpropagation

2.5.2. Neuron

Neuron adalah unit pemroses informasi yang menjadi dasar dalam pengoperasian jaringan syaraf tiruan. *Neuron* terdiri dari tiga elemen pembentuk :

a. Himpunan unit-unit yang dihubungkan dengan jalur koneksi. Jalur-jalur tersebut memiliki bobot/kekuatan yang berbeda-beda. Bobot yang bernilai positif akan memperkuat sinyal dan yang bernilai negatif akan memperlemah sinyal yang dibawanya. Jumlah, struktur dan pola hubungan antar unit-unit tersebut akan menentukan arsitektur jaringan (dan juga model jaringan yang terbentuk).
b. Suatu unit penjumlahan yang akan menjumlahkan *input-input* sinyal yang sudah dikalikan dengan bobotnya.

   Misalkan $x_1, x_2, \ldots, x_m$ adalah unit-unit *input* dan $w_{j1}, w_{j2}, \ldots, w_{jm}$ adalah bobot penghubung dari unit-unit tersebut ke unit keluaran Y_j, maka unit penjumlah akan memberikan keluaran sebesar $u_j = x_1w_{j1} + x_2w_{j2} + \ldots + x_mw_{jm}$
c. Fungsi aktivasi yang akan menentukan apakah sinyal dari *input neuron* akan diteruskan ke *neuron* lain ataukah tidak.

2.5.3. Pelatihan Standar Backpropagation

Pelatihan *Backpropagation* meliputi tiga fase. Fase pertama adalah fase maju. Pola masukan dihitung maju mulai dari layar masukan hingga layar keluaran menggunakan fungsi aktivasi yang ditentukan. Fase kedua adalah fase mundur. Selisih antara keluaran jaringan dengan target yang diinginkan merupakan kesalahan yang terjadi. Kesalahan tersebut dipropagasikan mundur, dimulai dari garis yang berhubungan langsung dengan unit-unit di layar keluaran. Fase ketiga adalah modifikasi bobot untuk menurunkan kesalahan yang terjadi.

a. Fase I : Propagasi maju

Selama propagasi maju, sinyal masukan ($=x_i$) dipropagasikan ke layar tersembunyi menggunakan fungsi aktivasi yang ditentukan. Keluaran dari setiap unit layar tersembunyi ($=z_j$) tersebut selanjutnya dipropagasikan maju lagi ke layar tersembunyi di atasnya menggunakan fungsi aktivasi yang ditentukan. Demikian seterusnya hingga menghasilkan keluaran jaringan ($=y_k$).

Berikutnya, keluaran jaringan ($=y_k$) dibandingkan dengan target yang harus dicapai ($=t_k$). Selisih $t_k - y_k$ adalah kesalahan yang terjadi. Jika kesalahan ini lebih kecil dari batas toleransi yang ditentukan, maka iterasi dihentikan. Akan tetapi apabila kesalahan masih lebih besar dari batas toleransinya, maka bobot setiap garis dalam jaringan akan dimodifikasi untuk mengurangi kesalahan yang terjadi.

b. Fase II : Propagasi mundur

Berdasarkan kesalahan $t_k - y_k$, dihitung faktor δ_k ($k = 1,2 , ... , m$) yang dipakai untuk mendistribusikan kesalahan di unit y_k ke semua unit tersembunyi yang terhubung langsung dengan y_k. δ_k juga dipakai untuk mengubah bobot garis yang berhubungan langsung dengan unit keluaran.

Dengan cara yang sama, dihitung faktor δ_j di setiap unit di layar tersembunyi sebagai dasar perubahan bobot semua garis yang berasal dari unit tersembunyi di layar di bawahnya. Demikian seterusnya hingga semua faktor δ di unit tersembunyi yang berhubungan langsung dengan unit masukan dihitung.

c. Fase III : Perubahan bobot

Setelah semua faktor δ dihitung, bobot semua garis dimodifikasi bersamaan. Perubahan bobot suatu garis didasarkan atas faktor δ *neuron* di layar atasnya. Sebagai contoh, perubahan bobot garis yang menuju ke layar keluaran didasarkan atas δ_k yang ada di unit keluaran.

Ketiga fase tersebut diulang-ulang terus hingga kondisi penghentian dipenuhi. Umumnya kondisi penghentian yang sering dipakai adalah jumlah iterasi atau kesalahan. Iterasi akan dihentikan jika jumlah iterasi yang dilakukan sudah melebihi jumlah maksimum iterasi yang ditetapkan, atau jika kesalahan yang terjadi sudah lebih kecil dari batas toleransi yang diijinkan.

Algoritma *Backpropagation*

1. Inisialisasi bobot (ambil bobot awal dengan nilai *random* yang cukup kecil).
2. Kerjakan langkah-langkah berikut selama kondisi berhenti bernilai salah.
3. Untuk tiap-tiap pasangan elemen yang akan dilakukan pembelajaran, kerjakan :

   Feedforward

   a. Tiap-tiap unit *input* (X_i, i=1,2,3,...,n) menerima sinyal x_i dan meneruskan sinyal tersebut ke semua unit pada lapisan yang ada di atasnya (lapisan tersembunyi).

   b. Tiap-tiap unit pada suatu lapisan tersembunyi (Z_j, j=1,2,3,...,p) menjumlahkan sinyal-sinyal *input* terbobot :

   $$z\_in_j = b1_j + \sum_{i=1}^{n} x_i v_{ij}$$

   gunakan fungsi aktivasi untuk menghitung sinyal *output* :

   $z_j = f(z\_in_j)$

   dan kirimkan sinyal tersebut ke semua unit di lapisan atasnya (unit-unit *output*).

   c. Tiap-tiap unit *output* (Y_k, k=1,2,3,...,m) menjumlahkan sinyal-sinyal *input* terbobot.

$$y\_in_k = b2_k + \sum_{i=1}^{n} z_i w_{jk}$$

gunakan fungsi aktivasi untuk menghitung sinyal *output* :

$y_k = f(y\_in_k)$

dan kirimkan sinyal tersebut ke semua unit di lapisan atasnya (unit-unit *output*).

Catatan :

Langkah (b) dilakukan sebanyak jumlah lapisan tersembunyi.

Backpropagation

d. Tiap-tiap unit *output* (Y_k, k=1,2,3,...,m) menerima target pola yang berhubungan dengan pola *input* pembelajaran, hitung informasi errornya :

$\delta 2_k = (t_k - y_k)\, f'(y\_in_k)$

$\varphi 2_{jk} = \delta_k z_j$

$\beta 2_k = \delta_k$

kemudian hitung koreksi bobot (yang nantinya akan digunakan untuk memperbaiki nilai w_{jk}) :

$\Delta w_{jk} = \alpha \varphi 2_{jk}$

hitung juga koreksi bias (yang nantinya akan digunakan untuk memperbaiki nilai $b2_k$) :

$\Delta b2_k = \alpha \beta 2_k$

Langkah (d) ini juga dilakukan sebanyak jumlah lapisan tersembunyi, yaitu menghitung informasi error dari suatu lapisan tersembunyi ke lapisan tersembunyi sebelumnya.

e. Tiap-tiap unit tersembunyi (Z_j, j=1,2,3,...,p) menjumlahkan delta *input* (dari unit-unit yang berada pada lapisan di atasnya) :

$$\delta\_in_j = \sum_{k=1}^{m} \delta 2_k w_{jk}$$

kalikan nilai ini dengan turunan dari fungsi aktivasinya untuk menghitung informasi *error* :

$\delta 1_j = \delta\_in_j\, f'(z\_in_j)$

$\varphi 1_{ij} = \delta 1_j x_j$

$\beta 1_j = \delta 1_j$

kemudian hitung koreksi bobot (yang nantinya akan digunakan untuk memperbaiki nilai v_{ij}) :

$\Delta v_{ij} = \alpha \varphi 1_{ij}$

hitung juga koreksi bias (yang nantinya akan digunakan untuk memperbaiki nilai $b1_j$) :

$\Delta b1_j = \alpha \beta 1_j$

f. Tiap-tiap unit *output* (Y_k, k=1,2,3,...,m) memperbaiki bias dan bobotnya (j=0,1,2,...,p) :

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

$b2_k(baru) = b2_k(lama) + \Delta b2_k$

Tiap-tiap unit tersembunyi (Z_j, j=1,2,3,...,p) memperbaiki bias dan bobotnya (i=0,1,2,...,n) :

$v_{ij}(baru) = v_{ij}(lama) + \Delta v_{ij}$

$b1_j(baru) = b1_j(lama) + \Delta b1_j$

4. Tes kondisi berhenti.

2.5.4. Optimalitas Arsitektur Backpropagation

Masalah utama yang dihadapi dalam *Backpropagation* adalah lamanya iterasi yang harus dilakukan. *Backpropagation* tidak dapat memberikan kepastian tentang berapa *epoch* yang harus dilalui untuk mencapai kondisi yang diinginkan. Oleh karena itu orang berusaha meneliti bagaimana parameter-parameter jaringan dibuat sehingga menghasilkan jumlah iterasi yang relatif lebih sedikit.

a. Inisialisasi bobot awal secara *random*

Pemilihan bobot awal sangat mempengaruhi jaringan syaraf dalam mencapai minimum global (atau mungkin hanya lokal saja) terhadap nilai *error*, serta cepat tidaknya proses pelatihan menuju kekonvergenan. Apabila nilai bobot awal terlalu besar, maka *input* ke setiap lapisan tersembunyi atau lapisan *output* akan jatuh pada daerah dimana turunan fungsi sigmoidnya akan sangat kecil. Sebaiknya, apabila nilai bobot awal terlalu kecil, maka *input* ke

setiap lapisan tersembunyi atau lapisan *output* akan sangat kecil, yang akan menyebabkan proses pelatihan akan berjalan sangat lambat. Biasanya bobot awal diinisialisasi secara *random* dengan nilai antara -0.5 sampai 0.5 (atau -1 sampai 1, atau interval yang lainnya).

b. Jumlah unit tersembunyi

Hasil teoritis yang didapat menunjukkan bahwa jaringan dengan sebuah layar tersembunyi sudah cukup bagi *Backpropagation* untuk mengenali sembarang pola antara masukan dan target dengan tingkat ketelitian yang ditentukan. Akan tetapi penambahan jumlah layar tersembunyi kadangkala membuat pelatihan lebih mudah.

Dalam propagasi maju, keluaran harus dihitung untuk tiap layar, dimulai dari layar tersembunyi paling bawah (terdekat dengan masukan). Sebaliknya, dalam propagasi mundur, faktor δ perlu dihitung untuk tiap layar tersembunyi, dimulai dari layar keluaran.

c. Jumlah pola pelatihan

Tidak ada kepastian tentang berapa banyak pola yang diperlukan agar jaringan dapat dilatih dengan sempurna. Jumlah pola yang dibutuhkan dipengaruhi oleh banyaknya bobot dalam jaringan serta tingkat akurasi yang diharapkan. Aturan kasarnya dapat ditentukan berdasarkan rumusan :

Jumlah pola = Jumlah bobot / tingkat akurasi

Untuk jaringan dengan 80 bobot dan tingkat akurasi 0.1, maka 800 pola masukan diharapkan akan mampu mengenali dengan benar 90 % pola diantaranya.

d. Lama iterasi

Tujuan utama penggunaan *Backpropagation* adalah mendapatkan keseimbangan antara pengenalan pola pelatihan secara benar dan respon yang baik untuk pola lain yang sejenis (disebut data pengujian). Jaringan dapat dilatih terus menerus hingga semua pola pelatihan dikenali dengan benar. Akan tetapi hal itu tidak menjamin jaringan akan mampu mengenali pola

pengujian dengan tepat. Jadi tidaklah bermanfaat untuk meneruskan iterasi hingga semua kesalahan pola pelatihan = 0.

Umumnya data dibagi menjadi dua bagian, yaitu pola data yang dipakai sebagai pelatihan dan data yang dipakai untuk pengujian. Perubahan bobot dilakukan berdasarkan pola pelatihan. Akan tetapi selama pelatihan (misal setiap 10 *epoch*), kesalahan yang terjadi dihitung berdasarkan semua data (pelatihan dan pengujian). Selama kesalahan ini menurun, pelatihan terus dijalankan. Akan tetapi jika kesalahannya sudah meningkat, pelatihan tidak ada gunanya untuk diteruskan lagi. Jaringan sudah mulai mengambil sifat yang hanya dimiliki secara spesifik oleh data pelatihan (tapi tidak dimiliki oleh data pengujian) dan sudah mulai kehilangan kemampuan melakukan generalisasi.

2.5.5. Variasi Backpropagation

Disamping model standar *Backpropagation*, kini sudah berkembang berbagai variasinya. Variasi tersebut bisa berupa model *Backpropagation* yang digunakan untuk keperluan khusus, atau teknik modifikasi bobot untuk mempercepat pelatihan dalam kasus tertentu.

a. Momentum

Pada standar *Backpropagation*, perubahan bobot didasarkan atas gradien yang terjadi untuk pola yang dimasukkan saat itu. Modifikasi yang dapat dilakukan adalah melakukan perubahan bobot yang didasarkan atas arah gradien pola terakhir dan pola sebelumnya (disebut momentum) yang dimasukkan. Jadi tidak hanya pola masukan terakhir saja yang diperhitungkan.

Penambahan momentum dimaksudkan untuk menghindari perubahan bobot yang mencolok akibat adanya data yang sangat bearbeda dengan yang lain (*outlier*). Apabila beberapa data terakhir yang diberikan ke jaringan memiliki pola serupa (berarti arah gradien sudah benar), maka perubahan bobot dilakukan secara cepat. Namun apabila data terakhir yang dimasukkan memiliki pola yang berbeda dengan pola sebelumnya, maka perubahan dilakukan secara lambat.

Dengan penambahan momentum, bobot baru pada waktu ke (t+1) didasarkan atas bobot pada waktu t dan (t-1). Disini harus ditambahkan 2 variabel baru yang mencatat besarnya momentum untuk 2 iterasi terakhir. Jika μ adalah konstanta $(0 \le \mu \le 1)$ yang menyatakan parameter momentum maka bobot baru dihitung berdasarkan persamaan :

$$w_{kj}(t+1) = w_{kj}(t) + \alpha\delta_k z_j + \mu\left(w_{kj}(t) - w_{kj}(t-1)\right)$$

dan

$$v_{ji}(t+1) = v_{ji}(t) + \alpha\delta_j x_i + \mu\left(v_{ji}(t) - v_{ji}(t-1)\right)$$

b. Delta - Bar - Delta

Dalam standar *Backpropagation*, laju pemahaman (α) merupakan suatu konstanta yang dipakai dalam seluruh iterasinya. Perubahan dapat dilakukan dengan memberikan laju pemahaman yang berbeda-beda untuk setiap bobotnya (atau bahkan laju pemahaman yang berbeda-beda untuk tiap bobot dalam tiap iterasinya). Apabila perubahan bobot berada dalam arah yang sama dalam beberapa pola terakhir (dapat dilihat dari tanda suku $\delta_k z_j$ yang selalu sama), maka laju pemahaman yang bersesuaian dengan bobot w_{kj} ditambah. Sebaliknya apabila arah perubahan bobot dua pola terakhir berbeda (ditandai dengan suku $\delta_k z_j$ yang berselang-seling positip - negatif) maka laju pemahaman untuk bobot tersebut harus dikurangi.

Perubahan bobot dalam aturan delta - bar - delta adalah sebagai berikut :

$$w_{kj}(t+1) = w_{kj}(t) + \alpha_{kj}(t+1)\delta_k z_j$$

2.6. Pemrograman Backpropagation Dengan Matlab

2.6.1. Membentuk Jaringan

A. Inisialisasi Jaringan

Langkah pertama yang harus dilakukan untuk memprogram *backpropagation* dengan Matlab adalah membuat inisialisasi jaringan.

Perintah yang dipakai untuk membentuk jaringan adalah newff yang formatnya adalah sebagai berikut :

net = newff(PR,[S1 S2...SN],{TF1 TF2...TFN},BTF,BLF,PF)

Dengan

net = jaringan Backpropagation yang terdiri dari n layar

PR = matriks ordo Rx2 yang berisi nilai minimum dan maksimum R buah elemen masukannya

Si (i=1,2,...,n) = jumlah unit pada layar ke-i (i=1,2,...,n)

Tfi (i=1,2,...,n) = fungsi aktivasi yang dipakai pada layar ke-i (i=1,2,...,n). Default = tansig (sigmoid bipolar)

BTF = fungsi pelatihan jaringan. Default = traingdx

BLF = fungsi perubahan bobot/bias. Default = learngdm

PF = fungsi perhitungan error. Default = mse

Beberapa fungsi aktivasi yang dipakai Matlab dalam pelatihan *backpropagation* adalah :

a. tansig (sigmoid bipolar) $f(net)=\frac{2}{1+e^{-net}}-1$. Fungsi ini adalah *default* yang dipakai. Fungsi sigmoid bipolar memiliki *range* [-1,1]
b. logsig (sigmoid biner) $f(net)=\frac{1}{1+e^{-net}}$. Fungsi sigmoid biner memiliki bentuk serupa dengan sigmoid bipolar, hanya rangenya adalah [0,1]
c. purelin (fungsi identitas) $f(net)=net$

Pelatihan yang dilakukan dalam Matlab dapat menggunakan berbagai fungsi, tujuannya adalah mempercepat pelatihan. Fungsi *default* yang dipakai oleh Matlab adalah traingdx. Dalam fungsi ini, perubahan bobot dilakukan dengan menambahkan momentum. Perubahan dilakukan dengan memperhatikan perubahan bobot pada iterasi sebelumnya. Disamping itu laju pemahaman (*learning rate* =α) bukan merupakan konstanta yang tetap, tetapi dapat berubah-ubah selama iterasi.

Umumnya, pelatihan *backpropagation* dalam Matlab dilakukan secara berkelompok (*batch training*). Semua pola dimasukkan dulu, baru kemudian bobot diubah. Dalam pelatihan berkelompok, semua data masukan harus diletakkan dalam sebuah matriks.

B. Inisialisasi Bobot

Setiap kali membentuk jaringan *backpropagation*, Matlab akan memberi nilai bobot dan bias awal dengan bilangan acak kecil. Bobot dan bias ini akan berubah setiap kali dibentuk jaringan. Akan tetapi jika diinginkan memberi bobot tertentu, dapat dilakukan dengan memberi nilai pada net.IW, net.LW dan net.b.

Perhatikan perbedaan antara net.IW dan net.LW. net.IW{j,i} digunakan sebagai variabel untuk menyimpan bobot dari unit masukan layar i ke unit tersembunyi (atau unit keluaran) layar j. Karena dalam *backpropagation*, unit masukan hanya terhubung dengan layar tersembunyi paling bawah, maka bobotnya disimpan dalam net.IW{1,1}.

Sebaliknya, net.LW{k,j} dipakai untuk menyimpan bobot dari unit di layar tersembunyi ke-j ke unit di layar tersembunyi ke-k. Sebagai contoh, net.LW{2,1} adalah penyimpan bobot dari layar tersembunyi paling bawah (layar tersembunyi ke-1) ke layar tersembunyi diatasnya (layar tersembunyi ke-2).

C. Simulasi Jaringan

Perintah sim digunakan pada *backpropagation* untuk menghitung keluaran jaringan berdasarkan arsitektur, pola masukan dan fungsi aktivasi yang dipakai.

[Y,Pf,Af,E,perf] = sim(net,P,Pi,Ai,T)

Dengan parameter masukan

net : nama jaringan dalam perintah newff

P : vektor masukan jaringan

Pi : kondisi delay awal masukan. Default = zeros

Ai : kondisi delay layar. Default = zeros

T : vektor target jaringan. Default = zeros

Dan parameter hasil

Y : keluaran jaringan

Pf : kondisi akhir delay masukan

Af : kondisi akhir delay layar

E : error jaringan = T-Y

Perf: unjuk kerja jaringan

Pi, Ai, Pf, Af hanya dipakai bagi jaringan yang memiliki *delay* masukan dan layar. Untuk sekedar menghitung keluaran jaringan, dapat dipakai statemen sederhana :

y = sim (net,p);

Perhatikan bahwa untuk menghitung keluaran jaringan, tidak perlu diketahui targetnya. Akan tetapi jika ingin dihitung *error* yang terjadi (selisih antara target dengan keluaran jaringan), maka harus diketahui targetnya.

2.6.2. Pelatihan Backpropagation

Matlab menyediakan berbagai variasi pelatihan *backpropagation*. Dalam sub bab ini akan dibahas pelatihan standar yang digunakan untuk melatih jaringan.

Pelatihan *backpropagation* menggunakan metode pencarian titik minimum untuk mencari bobot dengan *error* minimum. Dalam proses pencarian ini dikenal dua macam metode yaitu metode *incremental* dan metode kelompok (*batch*).

Dalam metode *incremental*, bobot diubah setiap kali pola masukan diberikan ke jaringan. Sebaliknya, dalam metode kelompok, bobot diubah setelah semua pola masukan diberikan ke jaringan. *Error* (dan suku perubahan bobot) yang terjadi dalam setiap pola masukan dijumlahkan untuk menghasilkan bobot baru. Matlab menggunakan metode pelatihan kelompok dalam iterasinya. Perubahan bobot dilakukan per *epoch*.

Untuk melatih jaringan digunakan perintah train yang formatnya adalah sebagai berikut:

[net,tr,Y,E,Pf,Af] = train(net,P,T,Pi,Ai,VV,TV)

Dengan

net : jaringan yang didefinisikan dalam newff

P : masukan jaringan

T : target jaringan. Default = zeros

Pi : kondisi delay awal masukan. Default = zeros

Ai : kondisi delay awal layar. Default = zeros

VV : struktur validasi vektor. Default = []

TV : struktur vektor uji. Default = []

Perintah train akan menghasilkan

net : jaringan yang baru

tr : record pelatihan (epoch dan performa)

Y : keluaran jaringan

E : error jaringan

Pf : kondisi akhir delay masukan

Af : kondisi akhir delay layar

Metode paling sederhana untuk merubah bobot adalah metode penurunan gradien (*gradient descent*). Bobot dan bias diubah pada arah dimana unjuk kerja fungsi menurun paling cepat, yaitu dalam arah negatif gradiennya.

Jika w_k adalah vektor bobot pada iterasi ke-k, g_k adalah gradien dan α_k adalah laju pemahaman, maka metode penurunan gradien memodifikasi bobot dan bias menurut persamaan $w_{k+1} = w_k - \alpha_k g_k$.

Matlab menyediakan beberapa metode pencarian titik minimumnya. Pencarian titik minimum dengan metode penurunan gradien dilakukan dengan memberikan parameter ‘traingd’ dalam parameter setelah fungsi aktivasi pada perintah newff.

Ada beberapa parameter pelatihan dapat diatur sebelum pelatihan dilakukan. Dengan memberi nilai yang diinginkan pada parameter-parameter tersebut dapat diperoleh hasil yang lebih optimal.

a. Jumlah *epoch* yang akan ditunjukkan kemajuannya.

   Menunjukkan berapa jumlah *epoch* berselang yang akan ditunjukkan kemajuannya.

   Instruksi : net.trainParam.show = EpohShow

   Nilai *default* untuk jumlah *epoch* yang akan ditunjukkan adalah 25.

b. Maksimum *epoch*

   Maksimum *epoch* adalah jumlah *epoch* maksimum yang boleh dilakukan selama proses pelatihan. Iterasi akan dihentikan apabila nilai *epoch* melebihi maksimum *epoch*.

   Instruksi : net.trainParam.epochs = MaxEpoh

   Nilai *default* untuk maksimum epoh adalah 10.

c. Kinerja tujuan

   Kinerja tujuan adalah target nilai fungsi kinerja. Iterasi akan dihentikan apabila nilai fungsi kinerja kurang dari atau sama dengan kinerja tujuan.

   Instruksi : net.trainParam.goal = TargetError

   Nilai *default* untuk kinerja tujuan adalah 0.

d. Learning rate

   Learning rate adalah laju pembelajaran. Semakin besar nilai *learning rate* akan berimplikasi pada semakin besarnya langkah pembelajaran. Jika *learning rate* diset terlalu besar, maka algoritma akan menjadi tidak stabil. Sebaliknya, jika *learning rate* diset terlalu kecil, maka algoritma akan konvergen dalam jangka waktu yang sangat lama.

   Instruksi : net.trainParam.lr = LearningRate

   Nilai *default* untuk *learning rate* adalah 0,01.

e. Waktu maksimum untuk pelatihan

   Menunjukkan waktu maksimum yang diijinkan untuk melakukan pelatihan. Iterasi akan dihentikan apabila waktu pelatihan melebihi waktu maksimum.

   Instruksi : net.trainParam.time = MaxTime

   Nilai *default* untuk waktu maksimum adalah tak terbatas (inf).

2.6.3. Mempercepat Pelatihan Backpropagation

Metode standar *backpropagation* seringkali terlalu lambat untuk keperluan praktis. Beberapa modifikasi dilakukan terhadap standar *backpropagation* dengan cara mengganti fungsi pelatihannya.

Secara umum, modifikasi dapat dikelompokkan dalam dua kategori. Kategori pertama adalah metode yang menggunakan teknik heuristik yang dikembangkan dari metode penurunan tercepat yang dipakai dalam standar *backpropagation*. Kategori kedua adalah menggunakan metode optimisasi numerik selain penurunan tercepat. Beberapa metode yang dipakai sebagai modifikasi adalah metode gradien conjugate, quasi Newton, Lavenberg Marquardt dan lain-lain. Dalam sub bab berikut ini dibicarakan dahulu tentang beberapa modifikasi yang masuk dalam kategori pertama (*backpropagation* dengan momentum, variabel laju pemahaman, dan *backpropagation resilient*). Berikutnya barulah dibahas tentang beberapa metode yang masuk dalam kategori kedua.

A. Metode Penurunan Gradien dengan Momentum (traingdm)

Meskipun metodenya paling sederhana, tapi metode penurunan gradien sangat lambat dalam kecepatan proses iterasinya. Ini terjadi karena kadang-kadang arah penurunan tercepat bukanlah arah yang tepat untuk mencapai titik minimum globalnya.

Modifikasi metode penurunan tercepat dilakukan dengan menambahkan momentum. Dengan momentum, perubahan bobot tidak hanya didasarkan atas *error* yang terjadi pada *epoch* pada waktu itu. Perubahan bobot saat ini dilakukan dengan memperhitungkan juga perubahan bobot pada *epoch* sebelumnya. Dengan demikian kemungkinan terperangkap ke titik minimum lokal dapat dihindari.

Besarnya efek perubahan bobot terdahulu (disebut faktor momentum) bisa diatur dengan suatu bilangan antara 0 dan 1. Faktor momentum = 0 berarti perubahan bobot hanya dilakukan berdasarkan *error* saat ini (penurunan gradien murni). Dalam Matlab, pelatihan *backpropagation* dengan menggunakan metode penurunan gradien dengan momentum dilakukan

dengan mendefinisikan fungsi pelatihan ‘traingdm’ dalam pembentukan jaringannya. Besarnya faktor momentum dilakukan dengan memberi nilai antara 0 – 1 pada net.trainParam.mc (*default* = 0,9). Parameter lain yang dapat diatur dalam traingdm sama dengan traingd.

B. Variabel Laju Pemahaman (traingda, traingdx)

Dalam standar *backpropagation*, laju pemahaman berupa suatu konstanta yang nilainya tetap selama iterasi. Akibatnya, unjuk kerja algoritma sangat dipengaruhi oleh besarnya laju pemahaman yang dipakai. Secara praktis, sulit untuk menentukan besarnya laju pemahaman yang paling optimal sebelum pelatihan dilakukan. Laju pemahaman yang terlalu besar maupun terlalu kecil akan menyebabkan pelatihan menjadi lambat.

Pelatihan akan lebih cepat apabila laju pemahaman dapat diubah ubah besarnya selama proses pelatihan. Jika *error* sekarang lebih besar dibandingkan *error* sebelumnya, maka laju pemahaman diturunkan. Jika sebaliknya, maka laju pemahaman diperbesar. Dengan demikian laju pemahaman dapat dibuat sebesar besarnya dengan tetap mempertahankan kestabilan proses.

Dalam Matlab, penggunaan variabel laju pemahaman dilakukan dengan menggunakan ‘traingda’ pada parameter fungsi pelatihan newff.

Penggunaan laju pemahaman juga bisa dikombinasikan dengan menambahkan faktor momentum. Fungsi pelatihan yang dipakai di Matlab adalah ‘traingdx’. Fungsi pelatihan ini memiliki kecepatan pelatihan yang tinggi sehingga dipakai sebagai *default* dalam pelatihan *backpropagation* di Matlab.

C. Resilient Backpropagation (trainrp)

Jaringan *backpropagation* umumnya menggunakan fungsi aktivasi sigmoid. Fungsi sigmoid akan menerima masukan dari range tak berhingga menjadi keluaran pada *range* [0,1]. Semakin jauh titik dari x = 0, semakin kecil gradiennya. Pada titik yang cukup jauh dari x = 0, gradiennya mendekati

0. hal ini menimbulkan masalah pada waktu menggunakan metode penurunan tercepat (yang iterasinya didasarkan atas gradien). Gradien yang kecil menyebabkan perubahan bobot juga kecil, meskipun masih jauh dari titik optimal.

Masalah ini diatasi dalam *resilient backpropagation* dengan cara membagi arah dan perubahan bobot menjadi dua bagian yang berbeda. Ketika menggunakan penurunan tercepat, yang diambil hanya arahnya saja. Besarnya perubahan bobot dilakukan dengan cara lain.

Dalam Matlab *resilient backpropagation* dilakukan dengan menuliskan 'trainrp' pada fungsi pelatihannya.

D. Algoritma Gradien Conjugate (traincgf, traincgp, traincgb)

Dalam standar *backpropagation*, bobot dimodifikasi pada arah penurunan tercepat. Meskipun penurunan fungsi berjalan cepat, tapi tidak menjamin akan konvergen dengan cepat. Dalam algoritma gradien konjugate, pencarian dilakukan sepanjang arah conjugate. Dalam banyak kasus, pencarian ini lebih cepat. Ada berbagai metode pencarian yang dilakukan berdasarkan prinsip gradien conjugate, antara lain Fletcher-Reeves ('traincgf'), Polak-Ribiere ('traincgp'), Powel Beale ('traincgb').

2.7 Tinjauan Pustaka

Utis sutisna dan Risanuri Hidayat melakukan penelitian pengenalan kata terisolasi dengan data berupa tutur "Saya"," kamu", "Dia", "Kita","Mereka". dengan Jaringan neural dilatih dengan variasi parameter Learning Rate, variasi parameter Momentum dan variasi jumlah Hidden Layer. Hasil pelatihan menunjukkan bahwa secara umum learning rate yang kecil menghasilkan perjalanan error yang lebih stabil tetapi proses pengenalannya membutuhkan waktu yang lebih lama. Sebaliknya, learning rate yang lebih besar membutuhkan waktu yang lebih cepat dalam proses pengenalan tetapi menghasilkan perjalanan error yang kurang stabil. Pelatihan dilakukan sampai *error* yang dihasilkan

mencapai nilai yang telah ditentukan, yaitu 0,00001**.**tingkat kebenaran pengenalan tertinggi yang dicapai adalah 50%.

John Adler, Muhamad Azhar, Sri Supatmi juga melakukan penilitan memanfaatkan device EEG berupa Emotiv EEG Neuroheadset yang 14 channel elektroda diletakkan pada kulit kepala dengan titik yang sudah ditentukan. Tujuannya untuk mengambil sinyal EEG (*Electroencephalograph)* untuk menjadikan pola sinyal guna mendeteksi sifat manusia. Sistem pada penelitian ini memanfaatkan metode *Backpropagation* untuk melakukan pemrosesan pola masukan, yang sebelumnya dilakukan pemrosesan terhadap data sinyal EEG dengan menggunakan Fast Fourier Transform (FFT) dan Power Spectral Density (PSD). Pemrosesan menggunakan FFT bertujuan untuk menyederhanakan banyaknya data masukan yang diperoleh, sedangkan pemrosesan menggunakan PSD bertujuan untuk meng-ekstraksi sinyal hasil pemrosesan FFT ke dalam bentuk spektrum yang spesifik. Hasil dari penelitian ini yaitu menghasilkan sistem analisis sinyal EEG untuk dapat mengetahui sifat manusia hanya dengan waktu 3 menit dengan keberhasilan 71,4 %. Sehingga diharapkan mampu digunakan sebagai pendekatan baru dalam ilmu psikologi, khususnya dalam mengetahui sifat manusia berdasarkan pola sinyal yang telah dilatih menggunakan backpropagation.