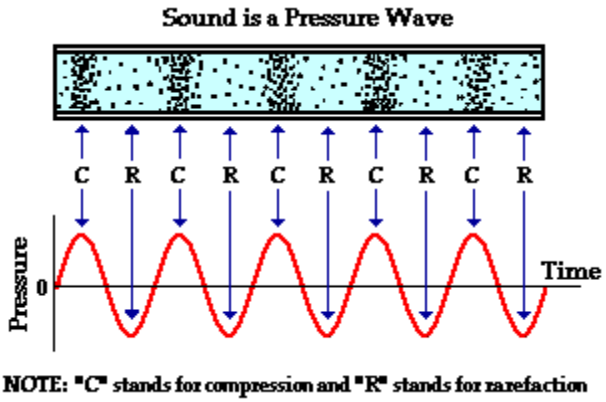
BAB 2
LANDASAN TEORI

2.1 Suara

Suara atau bunyi merupakan gelombang longitudinal yang dihasilkan oleh objek yang bergetar (karena hal ini suara termasuk kedalam energi kinetik). Gelombang yang dihasilkan oleh objek yang bergetar disebut suara atau bunyi jika gelombang tersebut bersifat *audible* yang artinya dapat didengar. Gelombang suara dapat merambat melalui beberapa medium (umumnya udara). Frekuensi merupakan banyaknya gelombang dalam satu detik. Nada dari sebuah suara ditentukan oleh frekuensi sinyal suara yang dinyatakan dalam satuan hertz (Hz). Frekuensi gelombang suara yang dapat didengar oleh manusia berada diantara rentang 20 Hz sampai 20 kHz, dibawah rentang tersebut disebut infrasonik sedangkan diatas rentang tersebut disebut ultrasonik. Selain nada yang diukur dengan frekuensi, suara juga memiliki fitur lainnya yaitu kekerasan volume yang diukur dalam satuan desibel (dB) dimana semakin besar desibelnya maka volume suara semakin keras [8]. Suara juga merupakan *pressure wave* yang artinya gelombang tekanan, hal ini dikarenakan suara merupakan gelombang yang terdiri dari gelombang dengan kompresi tinggi (keras) dan kompresi rendah (lembut) [9]. Pada Gambar 2.1 diilustrasikan perbandingan antara gelombang dan waktu pada suara.



Gambar 2.1 Suara sebagai *pressure wave*

Hal yang perlu diperhatikan dari gambar 2.1 adalah bahwa suara merupakan gelombang longitudinal dan bukan transversal. Pada gambar 2.1 dijelaskan bahwa suara terdiri dari kombinasi antara tekanan dengan kompresi tinggi yang dilambangkan dengan C dan kompresi rendah yang dilambangkan dengan R, oleh karena itu ilustrasi pada gambar 2.1 sebenarnya menyatakan grafik perbandingan antara waktu dan *pressure*, bukan bentuk dari sinyal suara.

2.1.1 Proses Produksi Suara Pada Manusia

Perlengkapan produksi suara pada manusia secara garis besar terdiri dari jalur suara (*vocal track*) dan jalur hidung (*nasal track*). Jalur suara dimulai dari pita suara (*vocal cords*), celah suara (*glottis*) dan berakhir pada bibir. Jalur hidung dimulai dari bagian belakan langit-langit (*velum*) dan berakhir pada cuping hidung (*nostrils*) [10].

Proses menghasilkan suara dimulai dari udara masuk ke paru-paru melalui pernafasan, kemudian melalui trakea, udara masuk ke batang tenggorokan, dimana pada batang tenggorokan ini terdapat pita suara. Pita suara ini kemudian bergetar dengan frekuensi tertentu karena adanya aliran udara tersebut sehingga dihasilkan suara. Suara yang dihasilkan ini berbeda-beda sesuai dengan posisi lidah, bibir, mulut dan langit-langit pada saat itu [11].

Suara yang dihasilkan terdiri dari tiga bagian yaitu *voiced sound, unvoiced sound* dan *plosive sound. Voiced sound* terjadi jika pita suara bergetar dengan frekuensi antara 50 Hz sampai 250 Hz. Contoh *voiced sound* adalah bunyi pada kata “ah”, “oh”. *Unvoiced sound* terjadi jika pita suara tidak bergetar sama sekali. Contoh *unvoiced sound* adalah bunyi “shh”. Sedangkan *plosive sound* terjadi jika pita suara tertutup sesaat kemudian tiba-tiba membuka. Contoh *plosive sound* adalah bunyi “beh” pada kata benar, “pah” pada kata pasar [11].

2.2 *Speech Recognition*

Speech recognition merupakan bagian dari penelitian di bidang biometric recognition. *Speech recognition* sebenarnya merupakan bagian dari *voice*

recognition yang memiliki beberapa bagian lainnya seperti *speaker recognition*. Meskipun *input* yang diterima oleh *speaker recognition* dan *speech recognition* adalah sama (sinyal suara), namun terdapat perbedaan pada *output* yang dihasilkan. Perbedaan yang mendasar dari *speaker recognition* dan *speech recognition* adalah pada *speaker recognition* tujuan dari proses yang dilakukan adalah untuk mengenali siapa pembicaranya seperti berapa umurnya, apa jenis kelaminnya, bagaimana emosinya, dan lain lain. Sedangkan pada *speech recognition* tujuan utamanya adalah untuk mengenali kata apa yang diucapkan oleh pembicaranya berdasarkan pengenalan sinyal suara yang dilakukan.

Pengenalan sinyal suara terdiri dari proses yang mengubah gelombang ucapan kedalam fitur-fitur yang dapat digunakan untuk dimanfaatkan selanjutnya. Terdapat banyak teknik dan algoritma yang dapat digunakan dalam melakukan pengenalan sinyal suara, hal tersebut tergantung pada kemampuan dari metode ekstraksi fitur dalam menangkap frekuensi dan energi pada waktu tertentu kedalam beberapa set koefisien untuk dapat dilakukan pengenalan suara baik untuk dikelompokkan berdasarkan artinya atau melakukan aksi tertentu. Suara manusia mengandung banyak informasi seperti jenis kelamin, emosi, dan identitas dari seorang *speaker* [12]. Berdasarkan penelitian yang dilakukan oleh B. H. Juang dan L. R. Rabiner diketahui bahwa spektrum suara manusia tidak mengalami perubahan yang berarti dalam selang waktu yang cukup singkat yaitu antara 10 – 30 ms [13]. Berdasarkan penelitian tersebut maka model digital dari sebuah ucapan dapat diperoleh jika dilakukan *framing* dalam selang waktu 10 – 30 ms.

2.3 Konversi Sinyal Analog ke Digital

Sinyal suara yang akan diproses bersifat analog sehingga jika akan dilakukan pengolahan secara digital, sinyal suara tersebut harus dikonversi menjadi sinyal digital, berupa urutan angka dengan tingkat presisi tertentu yang dinamakan *analog to digital conversion* dengan menggunakan *analog-to-digital converter* (ADC). Konsep Kerja ADC terdiri dari tiga proses :

1. Pencuplikan (*Sampling*) yaitu mengubah sinyal waktu kontinyu menjadi sinyal waktu diskrit.

2. Kuantisasi yaitu mengubah amplitudo sinyal kontinyu menjadi amplitudo diskrit, hasil proses pencuplikan dan kuantisasi adalah sinyal digital (waktu diskrit, amplitudo diskrit) dari sinyal analog (waktu kontinyu, amplitudo kontinyu).

3. Pengkodean yaitu mengkodekan sinyal digital dalam representasi binernya (digital)[14].

2.4 *End-point Detection* (*Silence Removal)*

Endpoint detection (deteksi titik akhir) merupakan sebuah metode dalam pemrosesan sinyal suara yang bertujuan untuk mengisolasi kata yang diucapkan dari *background noise*. *Endpoint detection* dibutuhkan untuk memangkas sinyal ucapan hingga merapat untuk menghindari *error* pada saat memodelkan ucapan selanjutnya dengan kata yang sama. *Endpoint detection* dapat dilakukan dengan menentukan *threshold* dengan nilai tertentu yang didapatkan dari hasil pengamatan terhadap bentuk sinyal yang didapatkan dari lingkungan tertentu [15].

2.5 MFCC

Mel Frequency Cepstral Coefficient (MFCC) adalah salah satu metode untuk melakukan ekstraksi ciri pada suara. Metode ini banyak digunakan pada penelitian di bidang pengenalan suara baik pada *speaker recognition* maupun *speech recognition*. Pada metode ini sinyal suara yang diterima akan dikonversikan menjadi beberapa parameter [14]. Metode ini banyak digunakan karena memiliki beberapa keunggulan, diantaranya adalah:

1. Mampu untuk menangkap Karakteristik suara yang sangat penting bagi pengenalan suara. Dengan kata lain, metode ini mampu menangkap informasi-informasi penting yang terkandung dalam sinyal suara.
2. Menghasilkan data seminimal mungkin tanpa menghilangkan informasi-informasi penting yang ada.
3. Mengadaptasi organ pendengaran manusia dalam melakukan persepsi terhadap sinyal suara.

Perhitungan pada metode MFCC menggunakan perhitungan *short-term analysis*. Hal ini dilakukan mengingat sinyal suara yang bersifat *quasistationary*

yang artinya hampir stasioner. Berdasarkan penelitian yang dilakukan oleh B.H. Juang dan Rabiner membuktikan bahwa semakin lama periode pengambilan suara tidak berarti informasi karakter suara yang didapatkan dari sinyal suara akan lebih akurat [13]. Pada penelitian tersebut disarankan bahwa pemotongan yang baik berkisar antara 10 sampai 30 ms.

Ektstraksi ciri pada MFCC sebenarnya merupakan adaptasi dari sistem pendengaran manusia dimana sinyal suara akan difilter secara linear untuk frekuensi rendah (frekuensi dibawah 1000 Hz) dan secara logaritmik untuk frekuensi tinggi.

2.5.1 DC Removal

DC Removal bertujuan untuk mendapatkan normalisasi dari data suara input. Proses perhitungan *DC Removal* dilakukan dengan cara menghitung rata-rata sampel suara, kemudian nilai setiap sampel suara tersebut akan dikurangi dengan nilai rata-rata yang telah didapatkan sebelumnya [10] sehingga untuk mendapatkan nilai hasil *DC Removal* dapat digunakan persamaan **(2. 1)**.

$$dc_i = s_i - \bar{x} \text{ , } 0 \leq \text{i} \leq \text{N-1} \quad \textbf{(2. 1)}$$

Dengan keterangan :

$dc_{[i]}$ = titik sinyal hasil proses *DC removal*

$s_{[i]}$ = titik sinyal ke-i

$\bar{x}$ = nilai rata-rata input titik sinyal.

i = 0,1,2,…, N-1

N = panjang sinyal (banyaknya titik sinyal)

Dimana untuk mendapatkan nilai rata-rata dari titik sinyal input dapat digunakan persamaan berikut.

$$\bar{x} = \frac{s_0 + s_1 + s_{2+} \ldots + s_{N-1}}{\text{N} - 1} \quad \textbf{(2. 2)}$$

Dengan keterangan :

$\bar{x}$ = nilai rata-rata dari titik sinyal input

s_i= input sinyal ke-i

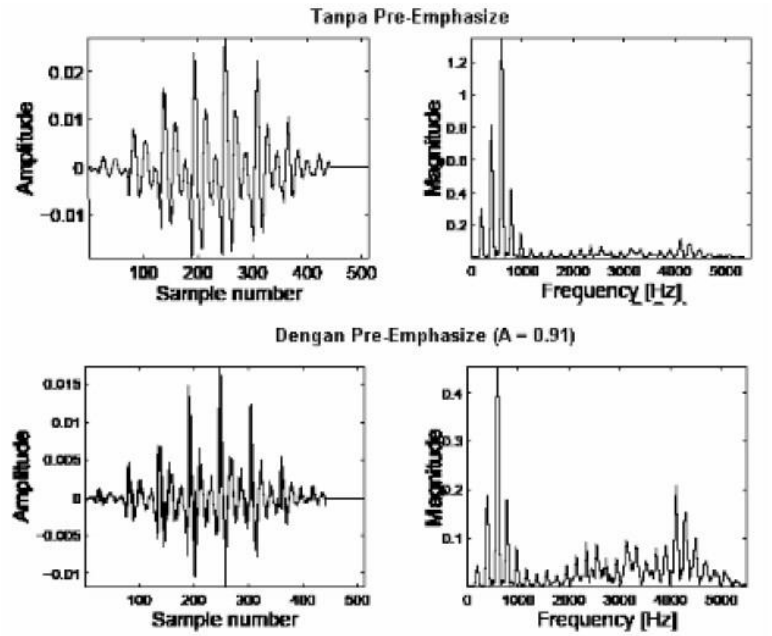
i = 0,1,2,…, N-1

N = panjang sinyal (banyaknya titik sinyal)

2.5.2 Pre Emphasis

Pre-emphasis filtering merupakan salah satu jenis filter yang dapat mempertahankan frekuensi tinggi pada sebuah spektrum yang umumnya ikut tereliminasi pada saat proses produksi suara [10]. Adapun tujuan dari pre-emphasis filter ini adalah:

1. Mengurangi *noise ratio* pada sinyal agar kualitas sinyal menjadi lebih baik.
2. Menyeimbangkan spektrum dari *voice sound*. pada saat memproduksi *voiced sound*, glotis manusia menghasilkan sekitar -12dB *octave slope*, namun ketika energi akustik tersebut dikeluarkan lewat bibir terjadi peningkatan hingga +6 dB. Sehingga sinyal yang terekam oleh mikrofon adalah sekitar 6 dB octave slope. Dampak dari efek ini dapat dilihat pada gambar berikut.



Gambar 2.2 Contoh dari *Pre-emphasize* pada sebuah *frame*

Pada Gambar 2.2 terdapat gambar tanpa *Pre-Emphasize* dan gambar dengan *Pre-Emphasize*, terlihat bahwa distribusi energi pada setiap frekuensi terlihat lebih seimbang setelah diimplementasikan *pre-emphasize filter.*

Bentuk yang paling umum digunakan dalam menghitung *pre-emphasize filter* adalah sebagai berikut :

$$pf_i = dc_i - (dc_{i-1} \times \alpha) \quad \textbf{(2. 3)}$$

Dimana :

pf_i = sinyal hasil *pre-emphasize filter* ke-i

dc_i = sinyal sebelum *pre-emphasize filter* ke-i

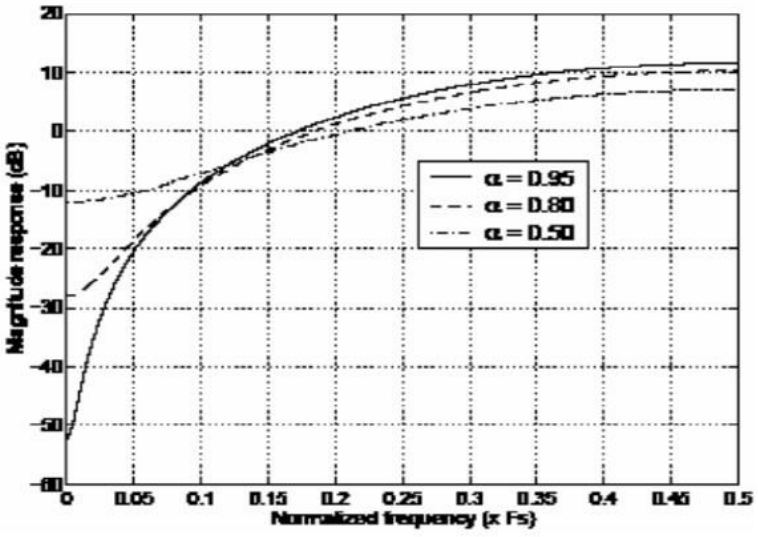
dc_{i-1} = sinyal sebelum *pre-emphasize filter* ke i-1

α = *pre-emphasize filter*

i = 0,1,2,…, N-1

N = panjang sinyal (banyaknya titik sinyal)

Pada umumnya nilai α yang paling sering digunakan adalah antara 0.9 sampai 1.0. *Magnitude response (dB scale)* untuk nilai α yang berbeda dapat dilihat pada gambar 2.7.



Gambar 2.3 *Magnitude response* dari *pre-emphasis filter*

Pada Gambar 2.3 dapat dilihat perbedaan magnitude response dengan menggunakan α yang berbeda, α yang digunakan pada gambar tersebut yaitu 0,50, 0,80 dan 0,95.

Setelah dilakukan penghitungan *pre-emphasis* filter maka sinyal suara hasil *pre-emphasis* filter dapat direpresentasikan persamaan sebagai berikut [10].

$$p_i = dc_i + pf_i \quad \textbf{(2. 4)}$$

i = 0,1,2,…, NI-1

NI = panjang sinyal (banyaknya titik sinyal)

p_i = titik sinyal baru ke-i

dc_i = sampel sebelum *pre emphasis* ke-i (hasil DC Removal)

pf_i = *pre emphasis filter* pada sampel ke-i

2.5.3 Frame Blocking

Karena sinyal suara terus mengalami perubahan akibat adanya pergeseran artikulasi dari organ produksi vokal, sinyal harus diproses secara *short segments* (*short frame*). Panjang *frame* yang biasanya digunakan untuk pemrosesan sinyal adalah antara 10-30 milidetik. Panjang *frame* yang digunakan sangat mempengaruhi keberhasilan dalam analisa spektral. Di satu sisi, ukuran dari *frame* harus sepanjang mungkin untuk dapat menunjukkan resolusi frekuensi yang baik. Tetapi di lain sisi, ukuran *frame* juga harus cukup pendek untuk dapat menunjukkan resolusi waktu yang baik [10].

Proses *frame* ini dilakukan terus sampai seluruh sinyal dapat diproses. Selain itu, proses ini umumnya dilakukan secara *overlapping* untuk setiap *frame*-nya. Panjang daerah *overlap* yang dapat digunakan berkisar antara 30% sampai 50% dari panjang *frame*, namun umumnya digunakan 50%. *Overlapping* dilakukan untuk menghindari hilangnya ciri atau karakteristik suara pada perbatasan perpotongan setiap *frame*. Adapun untuk menghitung jumlah *frame blocking* dapat dilihat pada persamaan berikut.

$$I = ((NPE-SP)/M)+1) \quad \textbf{(2. 5)}$$

I = Jumlah frame

NPE = Jumlah sampel (banyak sampel yang akan disegmentasi)

SP = *Sample point* dalam tiap *frame* (*Sample rate* × panjang *frame* dalam detik(s))

M = SP/2 (*Overlapping*)

Pada persamaan tersebut *sample rate* merupakan frekuensi sampling yang menyatakan jumlah titik sampel yang ada dalam file audio. *Sample point* didapatkan dari hasil *sample rate* dikalikan dengan panjang *frame* yang diinginkan dalam detik, misalnya *frame* yang diinginkan adalah 20 ms maka *sample point* adalah jumlah *sample rate* × 0,02. *Sample point* (SP) dapat juga disebut sebagai panjang *frame*. Adapun persamaan untuk nilai *sample point* dapat dilihat pada persamaan **(2. 6)**

SP = *Sample Rate* × durasi (s) **(2. 6)**

M pada persamaan **(2. 5)** merupakan *overlapping* yang telah dibahas pada penjelasan sebelumnya, dimana untuk *overlapping* 50% maka *overlapping* dapat dihitung sebagai *sample point* dibagi 2.

2.5.4 Time Alignment

Dalam melakukan pemrosesan terhadap sinyal suara, sinyal yang telah dimasukkan kedalam *frame-frame* tertentu akan dipilih menggunakan algoritma *time alignment* yang diperkenalkan oleh Alotaibi [15]. Metode time alignment akan membagi setiap *frame* sinyal yang akan diteliti secara proporsional untuk nanti dilakukan ekstraksi MFCC. Misalnya terdapat 15 *frame*, maka metode time-alignment akan mengambil *frame* pertama dan terakhir, kemudian *frame* lainnya yang berada diantara *frame* pertama dan terakhir yang telah dipilih sebelumnya akan diambil secara proporsional.

Proses pengambilan *frame* tersebut dilakukan dengan beberapa parameter yaitu CF (*considered frame*), SF (*start point*), dan EF (*endpoint*) dimana CF merupakan bilangan bulat positif lebih besar daripada 2, sedangkan SF dan EF merupakan bilangan dari 0 sampai 1 (dengan ketentuan SF ditambah EF sama dengan 1). CF merupakan jumlah *frame* yang ingin diambil menggunakan algoritma *time alignment*, sehingga *frame* ucapan selain CF tidak akan digunakan. SF digunakan untuk menghitung *frame* pertama yang akan diambil, sedangkan EF akan digunakan untuk mendapatkan *frame* terakhir yang akan diambil. Untuk menentukan *frame* pertama yang akan diambil kedalam CF maka dilakukan perkalian antara SF dengan jumlah *frame* dari keseluruhan sinyal ucapan. Adapun

cara untuk menentukan *frame* terakhir dari CF hampir sama dengan cara mencari sinyal pertama yaitu dengan mengalikan EF dengan jumlah *frame* dari keseluruhan sinyal ucapan. Jika dalam perhitungan SF dan EF menghasilkan bilangan desimal, maka harus dibulatkan ke bilangan bulat terdekat. Dalam melakukan *time alignment* nilai SF harus lebih kecil dari EF [15].

2.5.5 Windowing

Proses *framing* dapat menyebabkan terjadinya kebocoran spektral (*spectral leakage*) atau *aliasing*. *Aliasing* adalah sinyal baru dimana memiliki frekuensi yang berbeda dengan sinyal aslinya. Efek ini dapat terjadi karena rendahnya jumlah sampling rate, ataupun karena proses *frame blocking* dimana menyebabkan sinyal menjadi *discontinue*. Untuk mengurangi kemungkinan terjadinya kebocoran spektral, maka sampel hasil dari proses *framing* harus melewati proses *window* [10].

Sebuah fungsi *window* yang baik harus menyempit pada bagian *main lobe* dan melebar pada bagian *side lobe*-nya. Berikut ini adalah representasi dari fungsi window terhadap sinyal suara yang diinputkan.

$$w_i = p_i \,.\, \mathrm{fw_i} \qquad \textbf{(2. 7)}$$

w_i = nilai titik sinyal hasil *windowing*

p_i = hasil pre emphasis ke-l dari *frame* ke-i

$\mathrm{fw_i}$ = fungsi *window* ke-l untuk *frame* ke-i

NPE = banyaknya *sample point* pada *frame*

l = *sample point* ke-l pada *frame* (dimana nilai n = 0,1,…,NPE-1)

Untuk dapat mendapatkan hasil windowing diperlukan fungsi *window*. Ada banyak fungsi *window*, namun yang paling sering digunakan dalam untuk *speech recognition* adalah *hamming window*. Fungsi *window* ini menghasilkan *sidelobe level* yang tidak terlalu tinggi (kurang lebih -43 dB), selain itu *noise* yang dihasilkan pun tidak terlalu besar [16].

Fungsi *Hamming window* adalah sebagai berikut :

$$\mathrm{fw_i} = 0.54 - 0.46\, cos\, \frac{2\pi l}{SP-1} \qquad \textbf{(2. 8)}$$

fw_i = fungsi *hamming window* untuk titik sinyal ke-l pada frame ke-i

SP = *sample point* (dapat disebut sebagai panjang *frame)*

l = 0,1,...,SP-1

2.5.6 Fast Fourier Transform

Fast Fourier Transform (FFT) merupakan perluasan dari transformasi fourier yang berlaku untuk sinyal-sinyal diskrit dengan panjang yang terhingga. Semua sinyal periodik terbentuk dari gabungan sinyal-sinyal sinusoidal yang menjadi satu yang dapat dirumuskan sebagai berikut [14]:

$$f_k = \sum_{i=0}^{SP-1}\left(w_i \cos\frac{2\pi ik}{SP}\right) - j\sum_{l=0}^{SP-1}\left(w_i \sin\frac{2\pi ik}{SP}\right), 0 \le k \le SP-1 \quad \textbf{(2. 9)}$$

$f_k =$ spectrum FFT ke-k

N = jumlah sampel yang akan diproses ($N \in N$)

l = 0,1,2,…,SP-1 (jumlah sample dalam *frame*)

k = 0,1,2,…,SP-1 (variable frekuensi diskrit, merupakan hasil FFT)

w_i = nilai titik sinyal ke-n (hasil windowing ke-n pada frame)

j = bilangan imajiner

Untuk menghitung magnitude hasil FFT digunakan rumus.

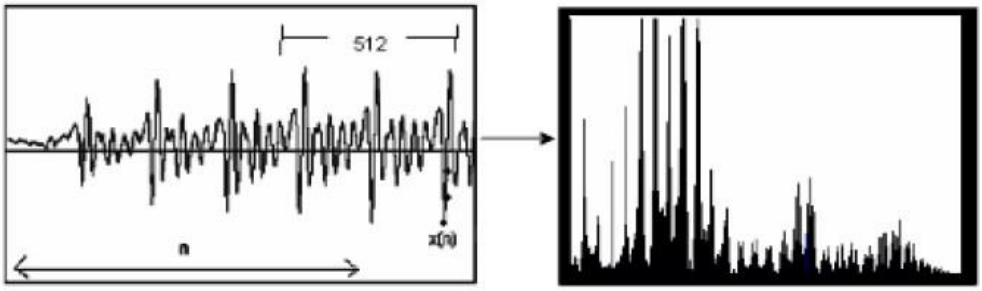
$$\text{fft}_k = [R^2 + I^2]^{\frac{1}{2}} \quad \textbf{(2. 10)}$$

Dimana :

R = bilangan real (hasil perhitungan $\left(w_i cos\frac{2\pi lk}{N}\right)$)

I = bilangan imajiner (hasil perhitungan $j\left(w_i sin\frac{2\pi lk}{N}\right)$)

Dengan persamaan **(2. 10**) suatu sinyal suara dalam domain waktu dapat kita cari frekuensi pembentuknya. Hal inilah tujuan penggunaan analisa fourier pada data suara, yaitu untuk merubah data dari domain waktu menjadi data spektrum di domain frekuensi. Untuk pemrosesan sinyal suara, hal ini sangatlah menguntungkan karena data pada domain frekuensi dapat diproses dengan lebih mudah dibandingkan data pada domain waktu, karena pada domain frekuensi, keras lemahnya suara tidak seberapa berpengaruh [14].



Gambar 2.4 Domain waktu menjadi domain Frekuensi

Pada Gambar 2.4 dapat dilihat bahwa gambar di sebelah kiri menunjukan gambar domain waktu dan gambar sebelah kanan menunjukan gambar domain frekuensi.

2.5.7 Mel Frequency Warping

Mel Frequency Wrapping umumnya dilakukan dengan menggunakan *filterbank. Filterbank* adalah salah satu bentuk dari filter yang dilakukan dengan tujuan untuk mengetahui ukuran energi dari *frequency band* tertentu dalam sinyal suara. *Filterbank* dapat diterapkan baik pada domain waktu maupun pada domain frekuensi, tetapi untuk keperluan MFCC, *filterbank* harus diterapkan dalam domain frekuensi [14].

Filterbank menggunakan representasi konvolusi dalam melakukan filter terhadap sinyal. Konvolusi dapat dilakukan dengan melakukan perkalian antara spektrum sinyal dengan koefisien *filterbank*. Berikut ini adalah rumus yang digunakan dalam perhitungan *filterbank* [17].

$$mel\ (f) = 2595 \times log10(1 + \frac{f}{700}) \quad \textbf{(2. 11)}$$

Dimana :

mel (*f*) = Fungsi Mel Scale

f = frekuensi yang digunakan

$$mel^{-1}(f) = 700 \times (10^{\frac{f}{2595}} - 1) \quad \textbf{(2. 12)}$$

Dimana :

$mel^{-1}(f)$ = Fungsi Mel Scale

f = frekuensi yang digunakan

$$fb_m = \left(\frac{SP}{FS}\right) \times mel^{-1}\left(mel(Nlow) + m\frac{mel(Nhigh)-mel(Nlow)}{Nfbank}\right) \quad \textbf{(2. 13)}$$

Dimana :

fb_m = hasil *boundary point filter*

SP = banyaknya sample point

FS = Frekuensi Sampling

$$H_{mk} = \begin{cases} 0,\ untuk\ k < fb_m \\ \frac{k-fb_{m-1}}{fb_m - fb_{m-1}},\ untuk\ fb_{m-1} \le k < fb_m \\ \frac{fb_{m+1}-k}{fb_{m+1}-fb_{m-1}},\ untuk\ fb_m \le k < fb_{m+1} \\ k > fb_{m+1} \end{cases} \quad (2.\ 14)$$

Dimana :

fb_m = hasil *boundary point filter*

H_{mk} = koefisien dari mel filterbank

$$MF_m = \sum_{k=0}^{SP-1} \text{fft}_k \times H_{mk} \quad \textbf{(2. 15)}$$

Dimana :

MF_m = Hasil Mel Frequency Warping ke-*m*

fft_k = Hasil FFT ke-*k*

SP = Banyaknya data hasil FFT

m = 0,1,2,..,*Nfbank*

Nfbank = Jumlah *Mel Filterbank*

2.5.8 Discrete Cosine Transform

DCT merupakan langkah terakhir dari proses utama MFCC *feature extraction.* Konsep dasar dari DCT adalah mendekorelasikan *mel spectrum* sehingga menghasilkan representasi yang baik dari property spektral local. Pada dasarnya konsep dari DCT sama dengan *inverse fourier transform.* Namun hasil dari DCT mendekati PCA (*principle component analysis*). PCA adalah metode static klasik yang digunakan secara luas dalam analisa data dan kompresi. Hal inilah yang menyebabkan seringkali DCT menggantikan *inverse fourier transform* dalam proses MFCC *feature extraction* [16].

Berikut adalah formula yang digunakan untuk menghitung DCT [17].

$$C_{koef} = \sqrt{\frac{2}{Nfbank}} \sum_{m=0}^{Nfbank-1} (\log MF_{\mathrm{m}}) \cos\left[koef\left(\frac{2m-1}{2}\right)\frac{\pi}{Nkoef}\right] \quad \textbf{(2.16)}$$

Dimana:

MF_{m} = Hasil *Mel Frequency Warping* ke-m

m = 0, 1, 2, … , SP-1(indeks hasil *filterbank* ke-m)

$Nfbank$ = Jumlah *Mel Filterbank* (40)

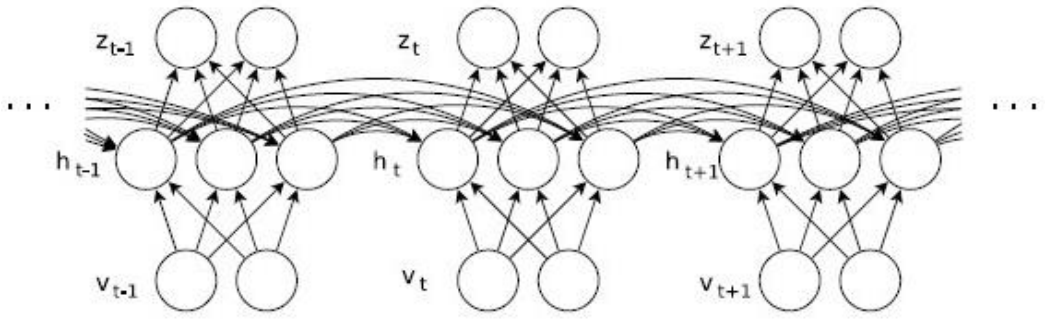
Nkoef = Jumlah koefisien yang diharapkan (pada penelitian ini 13)

koef = 0, 1, 2, … , $Nkoef$ (indeks untuk koefisien)

Koefisien ke nol dari DCT pada umumya akan dihilangkan, walaupun sebenarnya mengindikasikan energi dari *frame* sinyal tersebut. Hal ini dilakukan karena, berdasarkan penelitian-penelitian yang pernah dilakukan, koefisien ke nol ini tidak reliable terhadap *speaker recognition*. Tetapi koefisien ke nol dari DCT sangatlah berguna dalam menentukan *end point* dari suatu suku kata maupun kata. Hal ini disebabkan karena pada *end point* dari suatu suku kata maupun kata mempunyai energi yang lebih rendah daripada *point-point*[14].

2.6 *Recurrent Neural Network*

Recurrent Neural Network (RNN) merupakan salah satu bagian dari *artificial neural network* (ANN) dimana koneksi dari setiap unit neuron membentuk sebuah siklus searah. Tidak seperti *Feedforward Neural Network*, RNN bisa menggunakan memori internal untuk memproses beberapa *input sequence*s, hal inilah yang menjadikan RNN dapat digunakan untuk melakukan *speech recognition*.



Gambar 2.5 Arsitektur RNN

2.6.1 Normalisasi dan Thresholding

Normalisasi adalah penskalaan terhadap nilai-nilai masuk ke dalam suatu *range* tertentu. Hal ini dilakukan agar nilai *input* dan target *output* sesuai dengan *range* dari fungsi aktivasi yang digunakan dalam jaringan [18]. Pada penelitian ini persamaan normalisasi yang akan digunakan yaitu persamaan **(2. 17)**

$$normal_n = \frac{0.8(Cn-a)}{b-a} + 0.1 \quad \textbf{(2. 17)}$$

Dengan keterangan:

$normal_n$ = nilai hasil normalisasi ke-n

Cn = nilai *sample* ke-n (hasil DCT)

b = nilai hasil ekstraksi ciri maksimal pada *frame*

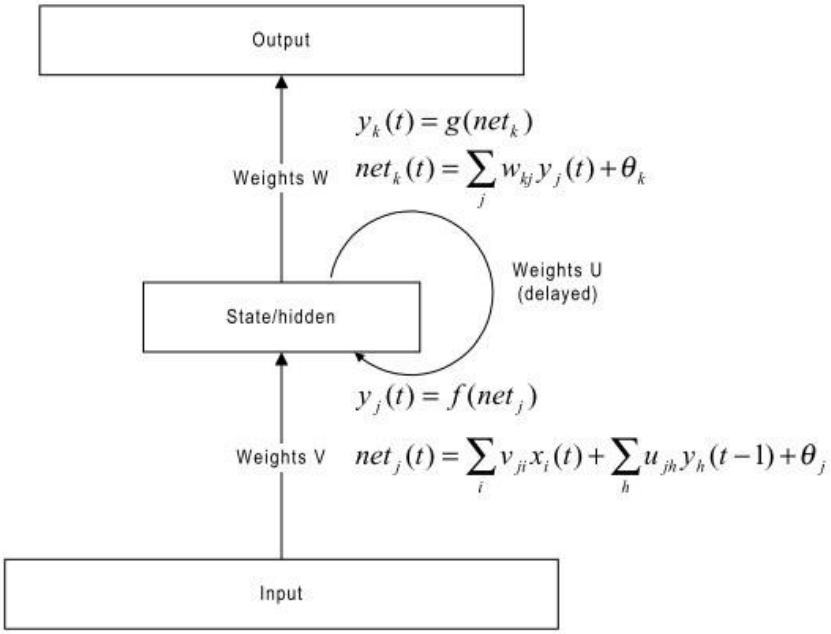
a = nilai hasil ekstraksi ciri minimal pada *frame*

setelah dilakukan normalisasi terhadap data Input maka perlu dilakukan thresholding agar data hasil normalisasi yang bersifat kontinu, dapat berubah menjadi bersifat diskrit. Perubahan data input untuk sebuah *neural network* diubah kedalam bentuk diskrit bertujuan untuk mempermudah proses pelatihan *neural network*, karena *neural network* lebih mudah dilatih dengan data diskrit dibanding kontinu.

2.6.2 Elman Recurrent Neural Network

Elman Recurrent Neural Network (ERNN) merupakan salah satu arsitektur dari beberapa arsitektur yang dimiliki oleh *Recurrent Neural Network*. *Elman Recurrent Neural Network* (ERNN) termasuk kedalam *Simple Recurrent Neural Network* (SRNN) [19]. Selain ERNN arsitektur *Simple Recurrent Neural Network* adalah *Jordan Reccurrent Neural Network*, perbedaannya adalah pada *Jordan Recurrent Neural Network* nilai dari *context* merupakan *value* dari *output layer* pada *sequence* sebelumnya sedangkan pada *Elman Recurrent Neural Network value* dari *context* merupakan *value* dari *hidden layer* pada *sequence* sebelumnya. *Elman Recurrent Neural Network* termasuk kedalam RNN sederhana karena hanya mengandalkan *context* yang didapatkan dari hasil perhitungan pada hidden layer di *t* (*sequence*) sebelumnya (*t-1*), berbeda dengan beberapa arsitektur lain seperti *Long-Short Term Memory* (LSTM) yang menerapkan sistem *gating unit* (adanya

gerbang untuk menggunakan *context* sebelumnya atau melupakan *context* yang sebelumnya).



Gambar 2.6 Arsitektur *Elman Recurrent Neural Network* (ERNN)

Pada gambar di atas *context value* dilambangkan dengan u yang mana adalah hasil dari aktivasi yang telah dilakukan dari *hidden neuron* pada arus sebelumnya. Dalam melakukan perhitungannya secara *forward pass Elman Recurrent Neural Network* menggunakan persamaan sebagai berikut.

$$netHid_t(t) = \sum_i v_{ji} x_i(t) + \sum_h u_{jh} outHid_h(t-1) + \theta_j \quad \textbf{(2. 18)}$$

Dengan ketentuan:

$net_j(t)$ = *hidden layer vector* pada timestep ke-t

v_{ji} = bobot dari neuron *input* ke-i menuju *neuron* output ke-j

x_i = nilai neuron input ke-i

u_{jh} = bobot dari neuron *context* ke-h menuju *neuron* output ke-j

y_h = nilai neuron *context* ke-h (nilai neuron *hidden* ke-h pada timestep sebelumnya)

t = timestep

Sedangkan nilai pada output layer dihasilkan dengan menggunakan persamaan berikut

$$net_k(t) = \sum_j w_{kj}\, outHid_j(t) + \theta_k \quad \textbf{(2. 19)}$$

Dengan ketentuan:

$net_k(t)$ = *output layer vector*

w_{kj} = bobot dari neuron *hidden* ke-j menuju *neuron* output ke-k

y_j = nilai aktif neuron *hidden* ke-j

θ_k = nilai bias menuju neuron *output* ke-k

Secara sederhana cara kerja algoritma *Elman Recurrent Neural Network* dalam mengenali sebuah pola dapat ditulis sebagai berikut.

1. Masukkan nilai neuron *input* pada setiap layer input di setiap timestep.
2. Masukkan nilai ***t***.yang diinginkan.
3. Masukkan nilai n_timestep = 0.
4. Jika nilai n_timestep tidak sama dengan ***t***, maka lakukan langkah no. 5
5. Lakukan perhitungan menggunakan persamaan **(2. 18)** pada timestep ke-n_timestep untuk menghasilkan nilai setiap *hidden* neuron pada layer *hidden* di timestep ke-n_timestep.
6. Cek nilai n_timestep, jika n_timestep = ***t*** ,maka lakukan langkah ke-7, jika tidak lakukan langkah no. 5.
7. Hitung nilai *output* neuron menggunakan persamaan **(2. 19)**.
8. Hasil pengenalan pola didapatkan.

2.6.3 *Activation Function* (Fungsi Aktivasi)

Fungsi aktivasi biasa digunakan dalam sebuah neural network. Fungsi aktivasi memiliki beberapa peranan penting dalam sebuah neural network. Alasan utama mengapa fungsi aktivasi digunakan adalah agar neural network dapat mengnali data yang non-linear, karena output yang diharapkan dari neural network jarang sekali bersifat linear [20]. Fungsi aktivasi merupakan sebuah fungsi yang dapat digunakan untuk menentukan keluaran pada neuron. Terdapat banyak jenis

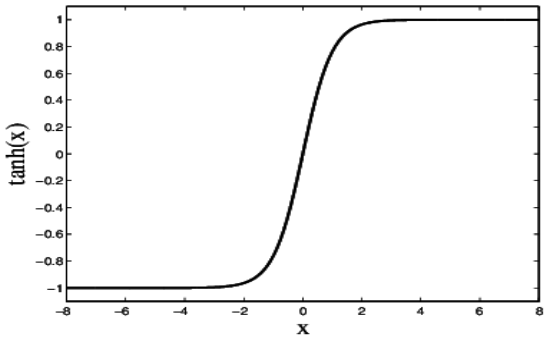
fungsi aktivasi, namun yang akan digunakan pada penelitian kali ini adalah fungsi aktivasi *hyperbolic tangent* dan *softmax*.

2.6.3.1 *Hyperbolic Tangent* (tanh)

Salah satu fungsi aktivasi yang umumnya digunakan adalah *hyperbolic tangent*. Fungsi aktivasi *hyperbolic tangent* akan menghasilkan nilai -1 sampai 1. Fungsi aktivasi *hyperbolic* tangent sebenarnya adalah sebuah fungsi trigonometri. Namun begitu, dalam neural network fungsi ini sama sekali tidak ada hubungannya dengan trigonometri. Fungsi aktivasi ini digunakan karena bentuk dari grafik nilai yang dihasilkan[20]. Persamaan dari fungsi *hyperbolic tangent* dapat dilihat pada persamaan **(2. 20)**.

$$outHid_j(t) = \frac{e^{2\times netHid}-1}{e^{2\times netHid}+1} \quad \textbf{(2. 20)}$$

Adapun grafik nilai yang dihasilkan oleh fungsi aktivasi *hyperbolic* tangent dapat dilihat pada gambar.



Gambar 2.7 Grafik Fungsi Aktivasi *Hyperbolic Tangent* (tanh)

Sedangkan untuk turunan dari fungsi aktivasi *hyperbolic tangent* (tanh) dapat dilihat pada persamaan. Pada penelitian ini fungsi aktivasi *hyperbolic tangent* (bisa juga ditulis tanh [21]) akan digunakan pada *hidden neuron*.

$$\text{derivative}_j = (1 - outHid) \times (1 + outHid) \quad \textbf{(2. 21)}$$

2.6.3.2 *Softmax*

Fungsi aktivasi softmax digunakan untuk memetakan probabilitas pada output layer, dimana nilai dari seluruh output neuron jika dijumlahkan akan bernilai

1, yang mana hal ini merupakan syarat untuk setiap distribusi *multivariate probability* yang *valid* [22]. Pada penelitian ini fungsi aktivasi *softmax* akan digunakan pada layer output. Adapun persamaan dari fungsi aktivasi *softmax* dapat dilihat pada persamaan **(2. 22)**.

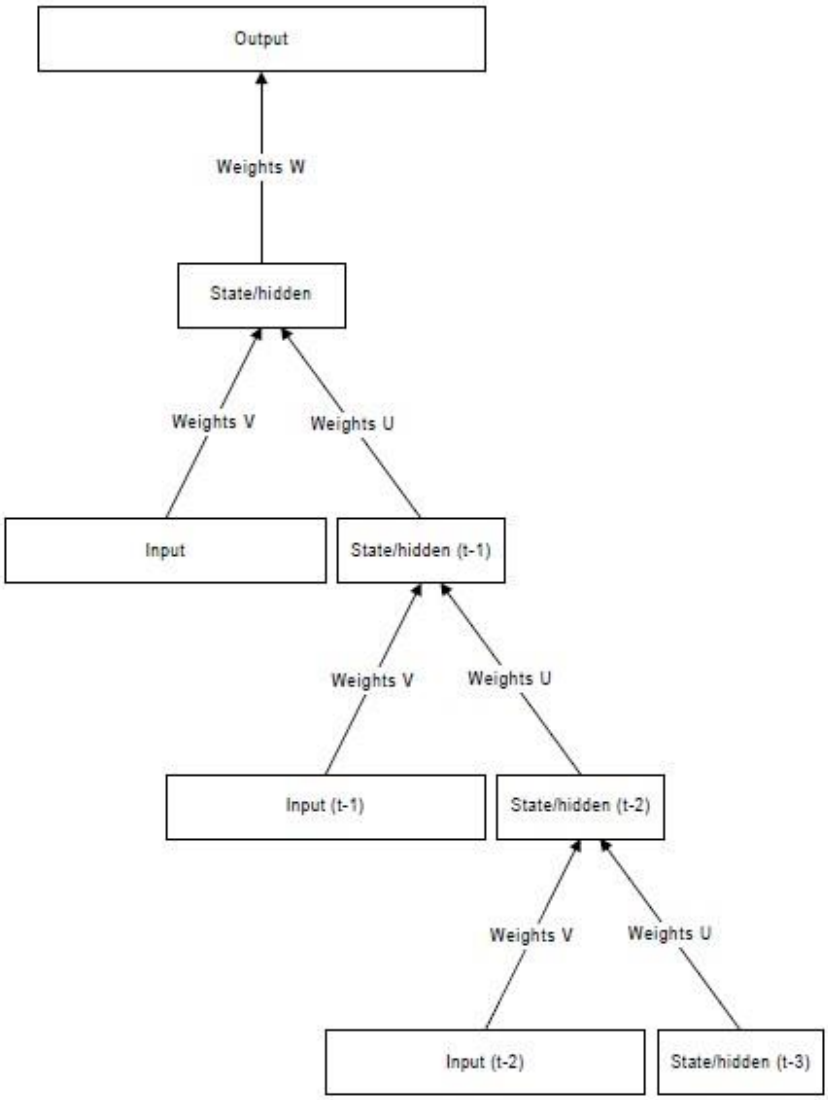
$$Output_k = \frac{e^{netOut_k}}{\sum_{k=1}^{K} e^{netOut_k}} \quad \textbf{(2. 22)}$$

Sedangkan untuk turunan dari fungsi aktivasi *softmax* dapat dilihat pada persamaan

$$\text{oError}_k = (\text{Output}_k * (1 - \text{Output}_k)) \times (\text{Output Target}_k - \text{Output}_k) \quad \textbf{(2. 23)}$$

2.6.4 Backpropagation Through Time (BPTT)

Dalam sebuah *recurrent neural network* sebuah nilai error dapat di-*backpropagate* lebih jauh daripada *feedforward neural network* biasa. Penyebaran error yang lebih jauh ini disebut dengan *backpropagation through time* (BPTT). Prinsip dasar dari BPTT adalah adanya *unfolding*[19]. Semua bobot yang *recurrent* dapat diduplikasi secara spasial untuk sejumlah *timestep* tertentu, pada penelitian ini dinyatakan sebagai ***t***. Sebagai konsekuensinya, setiap neuron yang berpengaruh terhadap nilai aktivasi neuron *recurrent* akan memiliki duplikat sebanyak ***t*** juga. Sebagai contoh, jika timestep dari *recurrent neural network* yang dimiliki adalah 3 (t=3) dengan melakukan *unfolding* maka arsitektur dari *recurrent neural network* tersebut dapat digambarkan sebagai berikut.



Gambar 2.8 Hasil *Unfolding* Sebuah *Network* (t=3)

Dengan memperhatikan gambar tersebut dapat terlihat bahwa semakin besar nilai *timestep* maka bobot dan nilai errornya yang perlu dimasukkan kedalam memori akan semakin besar. Dalam prakteknya penggunaan *timestep* yang terlalu besar tidak terlalu disarankan karena semakin besar nilai *timestep* maka neural network akan semakin rentan terhadap fenomena *vanishing gradient* [19]. Adapun langkah-langkah dari pelatihan menggunakan *backpropagation through time* dapat dituliskan sebagai berikut.

1. Inisialisasi bobot dengan nilai random untuk setiap bobot antara neuron input-hidden, context-hidden, hidden-output, dan bias-biasnya.

2. Tentukan nilai *hyperparameter* seperti learning_rate, max_epoch.
3. Lakukan langkah no. 4 sampai no. 8 sebanyak max_epoch kali.
4. Hitung secara *feedforward*. Dalam proses perhitungan *feedforward* gunakan persamaan **(2. 18)** untuk menghitung nilai pada neuron *hidden*, kemudian gunakan persamaan **(2. 20)** untuk menghitung hasil aktivasi dari tiap-tiap *hidden* neuron, jika proses perhitungan untuk setiap nilai *hidden* neuron telah dilakukan maka hitung nilai pada setiap neuron *output* dengan menggunakan persamaan **(2. 19)** kemudian lakukan aktivasi terhadap setiap neuron output tersebut dengan menggunakan persamaan **(2. 22)**.
5. Hitung niai *error* pada setiap neuron output, (nilai *errror* pada neuron *output* didapatkan dari selisih nilai *output* yang diharapkan dengan *output* yang dihasilkan pada setiap neuron). Gunakan persamaan **(2. 23)**
6. Hitung nilai *error*/gradient pada setiap *hidden* neuron, di setiap timestep. Adapun nilai *error* pada setiap *hidden* neuron dapat dihitung dengan menggunakan persamaan **(2. 24)**

   $hError_j = derivative_j \times sum_j$ **(2. 24)**

   - Untuk mendapatkan nilai dari $derivative_j$ digunakan persamaan **(2. 21)**,
   - Sedangkan untuk mendapatkan nilai sum digunakan persamaan **(2. 25)**

   $sum_j = \sum_{k=0}^{k} oError_k \times w_{jk}$ **(2. 25)**
7. Setelah nilai error pada setiap hidden neuron di setiap timestep didapatkan maka hitung delta untuk setiap bobot menggunakan persamaan **(2. 26)**

   $delta_{ij} = \alpha \times downstream\ gradient \times input$ **(2. 26)**

   dimana :
   α = Learning rate
   $downstream\ gradient$ = nilai gradient (error) pada layer ke-j
   $input$ = Nilai input dari neuron ke-i pada layer input
8. Lakukan *update* terhadap nilai bobot dengan cara nilai delta + nilai bobot asal.

   $bobot_{ij} = delta_{ij} + bobot_{ij}$ **(2. 27)**
9. Fase pelatihan selesai.

2.7 *Object Oriented Programming* (OOP)

Object Oriented Programming (OOP) atau Pemrograman Berorientasi Objek (PBO) adalah sebuah paradigma pemrograman yang berorientasikan kepada objek. Dalam paradigma ini semua data dan fungsi dimasukkan kedalam *class* atau *objek*. Perbedaan dari pemrograman berorientasi objek dengan pemrograman terstruktur adalah jika pada pemrograman terstruktur setiap objek atau prosedur dapat menerima pesan memproses data dan mengirim pesan ke objek lainnya karena tidak memiliki hak akses (semua objek memiliki hak akses yang sama), sedangkan pada pemrograman berorientasi objek terdapat hak akses yang berbeda. Masing-masing objek harus berisikan informasi mengenai dirinya sendiri (*encapsulation*) dan objek dapat dikaitkan (*inheritance*) [23].

Dalam OOP, *class* adalah sebuah modul yang mereprentasikan dan mendefinisikan sekumpulan objek yang sama dan anggotanya (*method* dan *atribut*). *Class* pada OOP harus memiliki anggota yaitu *atribut* dan *method*. Pada OOP *procedure* disebut sebagai *method*. Objek dalam OOP merupakan representasi baik dari *properties* maupun *behavior* [24]. Objek adalah sebuah abstraksi dari sesuatu pada domain masalah, menggambarkan kemampuan untuk menyimpan informasi mengenai hal tersebut (*properties*), berinteraksi dengan hal tersebut (*behavior*) atau keduanya. Selain *class* Pemrograman Berorientasi Objek juga memiliki *superclass* yaitu sebuah *class* induk yang nantinya mempunyai *class* lain yang terdiri dari *class* dan *subclass*. Terdapat beberapa istilah pada konsep Pemrograman Berorientasi Objek diantaranya: *encapsulation, polymorphism,* dan *inheritance*.

Encapsulation (pengkapsulan) adalah teknik pembatasan ruang lingkup pada program terhadap data yang diproses supaya data terlindungi oleh prosedur atau objek lain, kecuali prosedur yang berada di objek itu sendiri [24].

Polymorphism adalah konsep yang menyatakan bahwa sesuatu yang sama dapat mempunyai bentuk dan perilaku yang berbeda, bahwa operasi yang sama mungkin memiliki perbedaan dalam *class* yang berbeda [24].

Inheritance (pewarisan) merupakan konsep dimana kepemilikan yang bersifat *implicit* dari fitur *subclass* yang didefinisikan dalam *superclass*. Dalam

Pemrograman Berorientasi Objek *inheritance* (pewarisan) yang terjadi mencakup *atribut* dan *method* [24].

2.8 *Unified Modeling Language* (UML)

Unified Modeling Language (UML) adalah sebuah bahasa standar pemodelan untuk pengembangan perangkat lunak dan sistem [25]. Dalam merancang sebuah desain untuk sistem yang besar tentu bukan hal yang mudah. Mulai dari aplikasi desktop yang sederhana sampai sistem *multi-tier* dapat dibangun dari ratusan, bahkan ribuan komponen perangkat lunak dan perangkat keras. Untuk dapat mengetahui komponen apa saja yang dibutuhkan, apa saja yang akan dikerjakan, bagaimana memenuhi kebutuhan pengguna, dan bagaimana menjelaskan sistem ke anggota tim lainnya maka dibutuhkan sebuah pemodelan, UML dapat mempermudah dalam memenuhi kebutuhan tersebut.

Memanajemeni kompleksitas merupakan salah satu alasan mengapa rancangan dari sebuah sistem harus dimodelkan. Pemodelan membantu para pengembang perangkat lunak dan sistem untuk dapat melihat gambaran sistem baik secara umum maupun detail, fokus pada saat melakukan dokumentasi, dan dalam mengkomunikasikan aspek penting dalam rancangan sistem yang telah dibuat.

UML dapat digunakan untuk memodelkan sistem dari mulai memodelkan sistem informasi untuk perusahaan hingga aplikasi web, bahkan untuk sistem yang rumit. UML menggunakan *class* dan *operation* dalam konsep dasarnya, oleh karena itu UML lebih cocok bila program ditulis dalam bahasa berorientasi objek seperti C++, Java, C# atau VB.NET. walaupun demikian, UML tetap dapat digunakan untuk perancangan aplikasi prosedural dalam VB atau C [26].

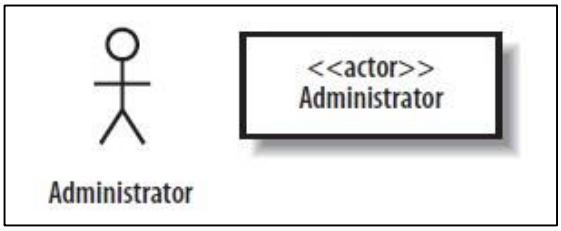
UML mendefinisikan diagram-diagram, antara lain:

1. Use case diagram

Use case diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah “apa” yang diperbuat sistem, dan bukan “bagaimana”. Sebuah *use case* merepresentasikan sebuah interaksi antara aktor dengan sistem. *Use case* merupakan sebuah pekerjaan tertentu, misalnya *login* ke sistem, membuat sebuah daftar belanja, dan sebagainya. Seorang/sebuah aktor

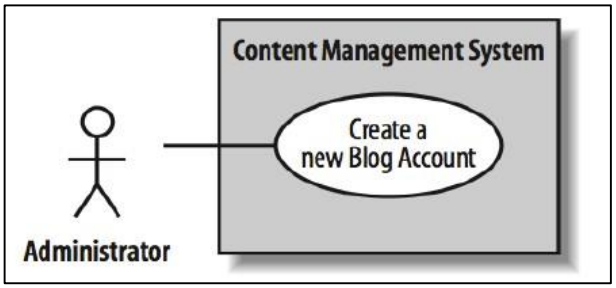
adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu [26]. Dengan kata lain yang ditekankan pada *use case diagram* adalah apa yang dapat dilakukan oleh sistem setelah ada interaksi dengan user. *Use case* juga membantu dalam memetakan kebutuhan sistem [24].

Diluar sistem terdapat aktor yang digambarkan dengan simbol *stickman* atau sebuah box dengan tulisan. Adapun contoh dari notasi aktor pada *use case diagram* dapat dilihat pada gambar 2.11 dimana terdapat dua cara untuk menyatakan notasi aktor.



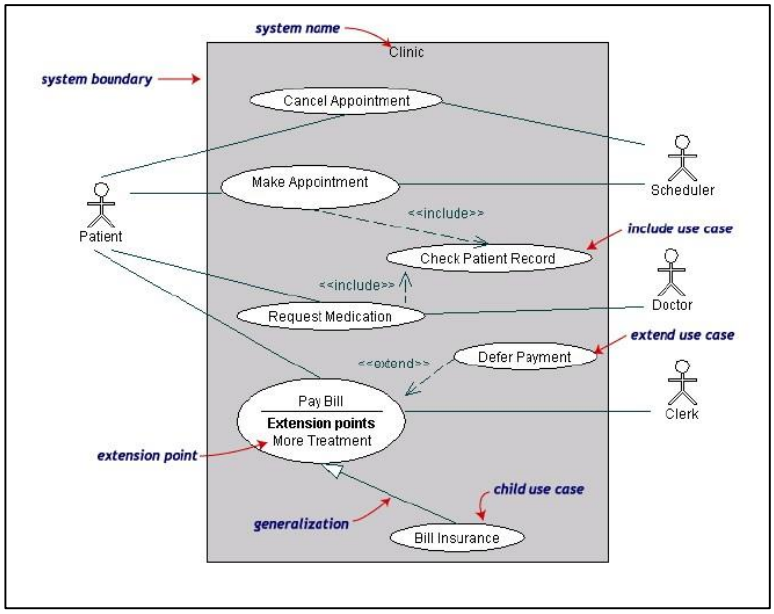
Gambar 2.9 Simbol Aktor pada *Use case*

Selain aktor, terrdapat sebuah notasi *use case*, *communication line*, dan *system boundaries* pada sebuah *use case diagram* yang utuh. Sebuah notasi *use case* menunjukkan bagaimana sistem digunakan untuk menyelesaikan pekerjaan dari aktor. *Communication line* menghubungkan aktor dan *use case* untuk menunjukkan bahwa aktor tersebut berpartisipasi di dalam sebuah *use case*. Sedangkan *system boundaries* digunakan untuk menandakan pemisahan antara eksternal sistem (aktor) dan internal sistem (*use cases*). Berikut merupakan contoh dari penggunaan notasi aktor, *use case*, *communication line* dan *system boundaries* [25].



Gambar 2.10 Contoh dari sebuah *use case* yang utuh

Sebuah *use case* dapat melakukan *include* (pemanggilan) terhadap fungsionalitas *use case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa *use case* yang di-*include* akan dipanggil setiap kali *use case* yang memanggilnya dieksekusi secara normal. Sebuah *use case* dapat di-*include* (dipanggil) oleh lebih dari satu *use case* lain, sehingga duplikasi fungsionalias dapat dihindari dengan cara menarik keluar fungsionalitas yang sama (*common*). Selain itu sebuah *use case* juga dapat melakukan *extend* kepada *use case* lain dengan *behaviour* (*method*) yang dimilikinya.



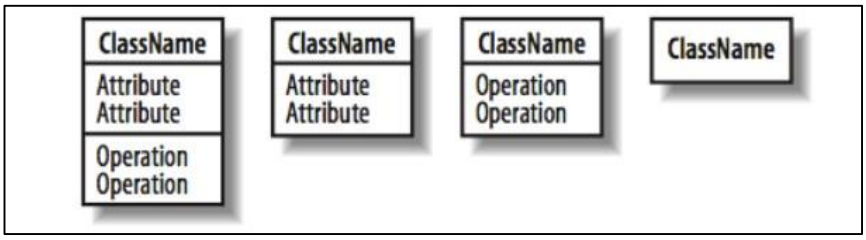
Gambar 2.11 Sebuah *use case diagram* dengan *include* dan *extend*

2. Class diagram

Class bisa disebut sebagai bagan yang digunakan untuk membangun dan mendefinisikan objek-objek. Sehingga setiap hal yang dibangun dari sebuah *class* disebut sebagai *objek* atau *instances*. Sederhananya, deskripsi dari sebuah *class* mencakup dua bagian: informasi yang akan dimiliki oleh objek dan *behaviour* yang akan didukung objek tersebut. Ini yang membedakan *object-oriented* dengan yang lainnya, yaitu *state information* dan *behaviour* digabungkan dalam satu definisi *class*, dimana objek-objek dapat terbentuk dari *class* tersebut [25].

Class dalam UML digambarkan sebagai persegi panjang dibagi menjadi tiga bagian. Bagian paling atas berisi nama *class*, bagian tengah berisi atribut atau

informasi yang dimiliki oleh *class* tersebut, dan bagian akhir berisi operasi yang menunjukkan *behaviour* dari *class* [25].



Gambar 2.12 Gambar *class diagram*

Class memiliki tiga area pokok yaitu *name, atribut,* dan *behaviour* (*method*). Baik *class*, *atribut*, ataupun *method* memiliki sebuah sifat *visibility*, yaitu sebuah sifat yang dapat digunakan untuk mengatur akses ke atribut atau *behaviour* tertentu. Adapun visibility yang dapat digunakan ada beberapa jenis yaitu:

1. Private

   Setiap *class*, *atribut,* dan *method* yang memiliki *visibility* (hak akses) private tidak dapat dipanggil dari luar class yang bersangkutan. Hak akses *private* biasanya memiliki simbol tanda "-".

2. Protected

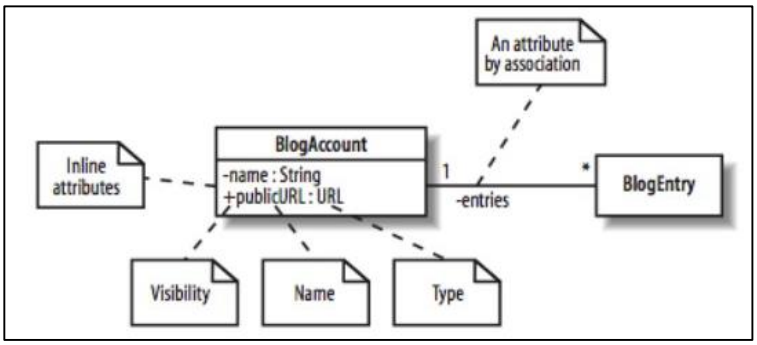
   Setiap *class*, *atribut*, dan *method* yang memiliki *visibility* (hak akses) *protected* hanya dapat dipanggil dari luar class yang bersangkutan dan anak-anak yang mewarisinya.

3. Public

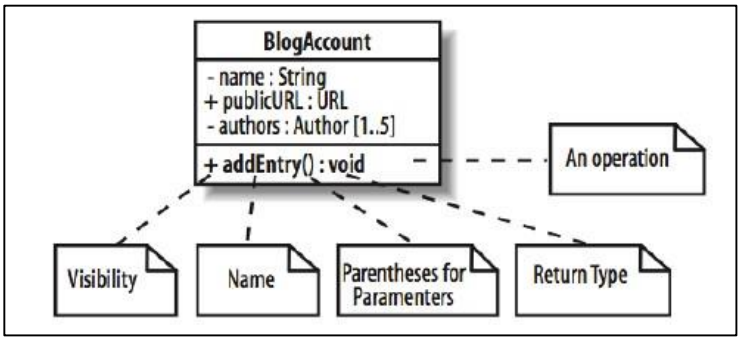
   Setiap *class*, *atribut,* dan *method* yang memiliki *visibility* (hak akses) public dapat dipanggil dari class manapun. Hak akses *public* biasanya memiliki simbol tanda "+".

4. Package

   Setiap *class*, *atribut,* dan *method* yang memiliki *visibility* (hak akses) package hanya dapat dipanggil dari luar class yang berada dalam package yang sama.



Gambar 2.13 State pada *atribut* dari sebuah *class*



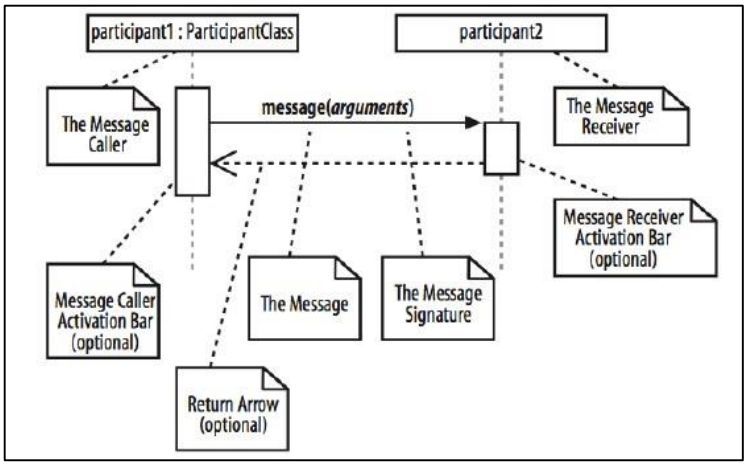
Gambar 2.14 State pada *behaviour* dari sebuah c*lass*

3. Sequence diagram

Sequence diagram memodelkan bagaimana sistem yang sedang dibangun akan menangani tugasnya. Diagram ini menggambarkan semua urutan interaksi pada bagian-bagian dari sistem. Dengan menggunakan *sequence diagram*, pengembang bisa menjelaskan interaksi apa yang akan dipanggil ketika sebuah *use case* dieksekusi dan dalam urutan seperti apa interaksi-interaksi tersebut akan terjadi [25].

Dalam *sequence diagram* terdapat bagian-bagian yang disebut dengan participant, time, message, dan *activation bars*. *Participant* adalah bagian-bagian dari sistem yang berinteraksi dengan satu sama lain selama sequence berjalan. *Time* menunjukkan urutan dimana semua interaksi berlangsung sesuai dengan waktu. *Time* pada *sequence diagram* ditunjukkan dengan garis titik-titik vertikal. *Message* dalam *sequence diagram* ditunjukkan menggunakan panah dari *participant* yang ingin mengirimkan pesan ke *participant* lain. *Message caller* adalah sebutan untuk *participant* yang mengirim *message* dan *Message Receiver* untuk *participant* yang

menerima *message*. *Activation bars* menunjukkan bahwa sebuah *participant* sedang melakukan sesuatu untuk jangka waktu tertentu. *Activation bars* dapat ditampilkan pada awal pengiriman *message* dan ujung dari penerimaan *message.* Hal ini menunjukkan bahwa *Message Caller* sibuk pada saat mengirim *message* dan *Message* [25].



Gambar 2.15 Contoh interaksi pada *sequence diagram*

2.9 Perangkat Pendukung

2.9.1 Netbeans

NetBeans adalah sebuah proyek *open-source* yang didedikasikan untuk menghasilkan produk *software development* yang baik (contohnya NetBeans IDE dan NetBeans Platform) yang dibutuhkan oleh developer, user dan pengguna bisnis yang mengandalkan NetBeans sebagai basis untuk produknya; khususnya untuk memungkinkan para pengguna untuk mengembangkan produknya dengan cepat, efisien dan mudah dengan mengandalkan kekuatan platform Java dan standar industri lainnya yang relevan.

Pada Juni 2000 , NetBeans dijadikan *open source* oleh Sun Microsystems yang mana merupakan sponsor Netbeans sampai Januari 2010 dimana saat itu Sun Microsystems menjadi anak perusahaan dari *Oracle*.

Kedua *base products* dari Netbeans yaitu NetBeans IDE dan NetBeans Platform dapat digunakan secara gratis baik untuk kepentingan non-komersial maupun

komersial. *Source code* pada kedua produk tersebut dapat digunakan ulang dan dimodifikasi sesuai kebutuhan selama sesuai dengan aturan [27].

2.9.2 Mikrofon

Mikrofon merupakan sebuah jenis transduser yang mengubah energi akustik dari gelombang suara menjadi sinyal listrik (digital). Mikrofon merupakan salah satu alat yang membantu manusia dalam berkomunikasi. Penggunaan dari mikrofon sendiri ditemukan pada banyak alat seperti telepon, alat perekam, alat bantu dengar, dan masih banyak lagi.

Istilah *mikrofon* berasal dari bahasa Yunani yaitu *mikros* yang berarti kecil dan *fon* yang berarti suara atau bunyi. Istilah ini mengacu kepada alat bantu dengar untuk suara berintensitas rendah. Pada awal penemuannya mikrofon digunakan pada telepon.

Pada tahun 1827, Sir Charles Wheatstone telah mengembangkan mikrofon. Ia merupakan orang pertama yang membuat “mikrofon frase". Selanjutnya, pada tahun 1876, Emile Berliner menciptakan mikrofon pertama yang digunakan sebagai pemancar suara telepon. Mikrofon praktis komersial pertama adalah mikrofon karbon yang ditemukan pada bulan Oktober 1876 oleh Thomas Alfa Edison. Pada tahun 1878, David Edward Hughes juga mengambil andil dalam perkembangan mikrofon karbon. Mikrofon karbon tersebut mengalami perkembangan hingga tahun 1920-an [14].