BAB 2. LANDASAN TEORI

Pada bab ini dijelaskan mengenai semua teori yang digunakan di dalam pembuatan Pengkonversian Suara menjadi Teks dengan Machine Learning untuk Aplikasi Membantu Tunarungu. Berikut adalah hal-hal yang dijelaskan pada bab ini antara lain pengertian mengenai Teori Umum, MFCC *(Mel-Frequency Cepstral Coefficient)*, *Machine Learning*, dan *Language Model*. Berikut adalah penjelasan mengenai teori-teori tersebut.

2.1. Teori Umum

Pada subbab ini dijelaskan secara umum teori yang mendukung secara mendasar untuk mendasari penelitian ini.

2.1.1. Tuna Rungu

Dalam Kamus Besar Bahasa Indonesia (KBBI) arti dari tuna rungu adalah tidak dapat mendengar; tuli. Tuli, tunarungu, atau gangguan dengar dalam kedokteran adalah kondisi fisik yang ditandai dengan penurunan atau ketidakmampuan seseorang untuk mendengarkan suara.

Tunarungu memiliki tingkatan berdasarkan ketidakmampuan seseorang untuk mendengar, mulai dari ringan yaitu kurang dengar *(hard of hearing)* hingga tuli *(deaf)*. Hallahan & Kauffman (1991:266) dan Hardman, et *al* (1990:276) mengemukakan bahwa orang yang tuli (*a deaf person*) adalah orang yang mengalami ketidakmampuan mendengar, sehingga mengalami hambatan dalam memproses informasi bahasa melalui pendengarannya dengan atau tanpa menggunakan alat bantu dengar (*hearing aid*). Sedangkan orang yang kurang dengar (*a hard of hearing person*) adalah seseorang yang biasanya menggunakan alat bantu dengar, sisa pendengarannya cukup memungkinkan untuk keberhasilan memproses informasi bahasa, artinya apabila orang yang kurang dengar tersebut menggunakan *hearing aid*, ia masih dapat menangkap pembicaraan malalui pendengarannya. (Hernawati, 2007)

2.1.2. Suara

Suara atau gelombang akustik sesungguhnya merupakan kasus khusus dari suatu gelombang elastik pada medium udara atau fluida. Manusia mulai memperhatikan suara sejak lama, bahkan alat musik yang menghasilkan suara sudah ada di negara Mesir, yang kemudian dikembangkan secara terstruktur oleh Al-Farabi, Al-Kindi dan masyarakat China.Sebuah kenyataan yang cukup unik bahwa pada awalnya, musik adalah sebuah disiplin ilmu yang mempelajari suara dan bunyi-bunyian, kemudian oleh Al-Farabi digolongkan ke dalam ilmu hitung dan bukan ilmu seni. (Ishaq, 2007)

Suara membawa serangkaian informasi berupa gelombang longitudinal yang unik. Setiap gelombang yang dihasilkan memiliki panjang berbeda – beda. Perubahan sedikit saja dapat menyebabkan perbedaan frekuensi yang jika didengar akan berbeda dari satu dengan yang lain. Setiap hari kita mendengar suara sekecil gelombang apapun yang kita tangkap melalui sistem pendengaran kita.

2.1.3. Frekuensi

Dalam Kamus Besar Bahasa Indonesia (KBBI) arti dari frekuensi adalah 1 kekerapan: 2 jumlah pemakaian suatu unsur bahasa dalam suatu teks atau rekaman; 3 jumlah getaran gelombang suara per detik; 4 jumlah getaran gelombang elektrik per detik pada gelombang elektromagnetik.

Frekuensi yang dapat didengar oleh manusia adalah 20Hz hingga 20kHz. Frekuensi memiliki satuan *Hertz,* artinya terjadi n getaran dalam 1 detik, sehingga dapat dinotasikan rumus (2.1).

$$f = \frac{n}{t} \tag{2.1}$$

f = frekuensi dalam satuan *Hertz* (Hz)

n = jumlah getaran

t = waktu untuk bergetar

2.1.4. Amplitudo dan Desibel

Dalam Kamus Besar Bahasa Indonesia (KBBI) arti dari amplitudo 1 simpangan yang paling jauh dari titik keseimbangan pada getaran; 2 selisih suhu tahunan atau suhu harian; 3 jarak antara puncak gelombang bunyi dan titik rata-rata.

Dalam Kamus Besar Bahasa Indonesia (KBBI) arti dari desibel 1 satuan ukuran untuk mengukur kerasnya suara; satuan ukuran untuk mengukur ketajaman pendengaran; 2 satuan untuk mengukur rasio dua daya atau intensitas, atau rasio antara suatu daya dan daya acuan; satuan yang sama dengan 10 kali logaritme rasio tersebut; 3 satuan untuk menyatakan intensitas bunyi relatif pada skala dari 0 untuk rata-rata bunyi yang dapat didengar sampai 130 untuk rata-rata bunyi pada ambang pendengaran tertinggi.

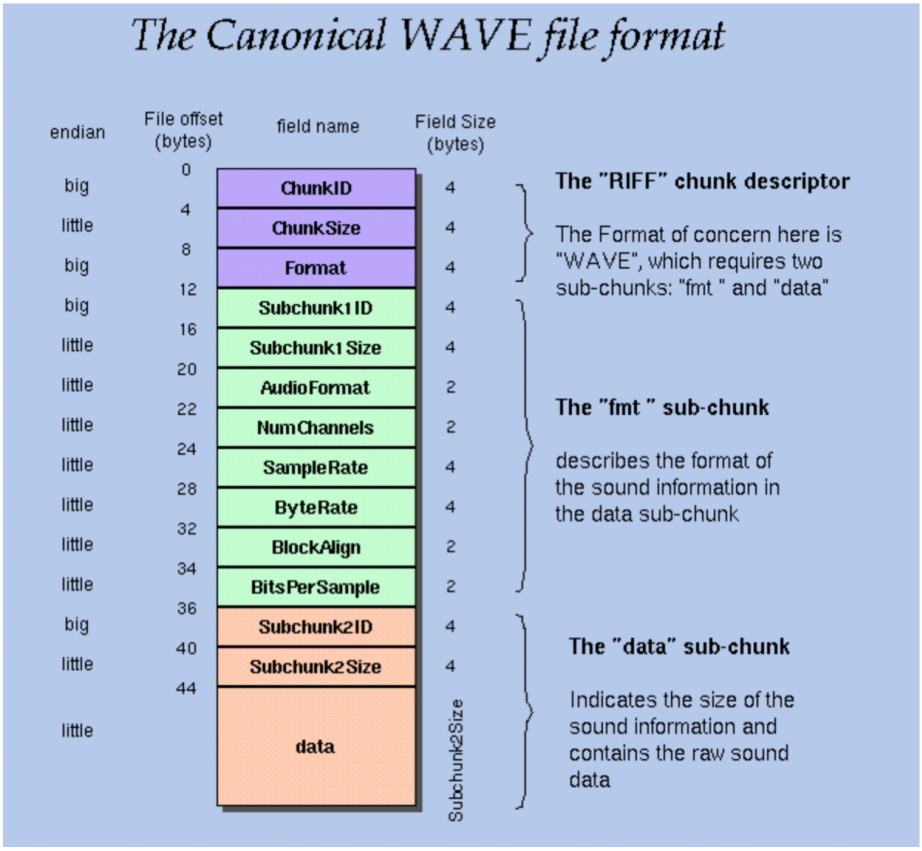
2.2. Pengambilang Fitur MFCC *(Mel-Frequency Cepstral Coefficient)*

Pada subbab ini dijelaskan fitur utama yang digunakan dalam penelitian ini. MFCC *(Mel-Frequency Cepstral Coefficient)* memiliki tahapan proses mulai dari WAV, *Pre-Emphasis*, *Hamming Window*, *Discrete Fourier Transform*, *Discrete Cosine Transform*.

2.2.1. Pengambilan Suara Dengan Format WAV *(Waveform Audio Format)*

WAV adalah singkatan dari istilah dalam bahasa Inggris *waveform audio format* merupakan standar format berkas audio yang dikembangkan oleh Microsoft dan IBM. WAV merupakan varian dari format *bitstream* RIFF dan mirip dengan format IFF dan AIFF yang digunakan komputer Amiga dan Macintosh. Baik WAV maupun AIFF kompatibel dengan sistem operasi Windows dan Macintosh. Walaupun WAV dapat menampung audio dalam bentuk terkompresi, umumnya format WAV merupakan audio yang tidak terkompres.

WAV memiliki header file untuk mendefiniskan konfigurasi data suara dalam file tersebut. Secara umum header file juga memuat konfigurasi audio seperti *sample rate*, jumlah *channel (stereo / mono)*, *bit ber sample (bit-depth)*. Total byte yang digunakan oleh header file ini adalah 44 *bytes*. Header file pada WAV dapat dilihat pada Gambar 2.1.



Gambar 2.1. *Header* wav *file* beserta kegunaannya

2.2.2. *Pre Emphasis*

Pre-Emphasis yang merupakan proses sistem yang dirancang untuk meningkatkan sebuah band frekuensi, langkah ini memproses lewat sinyal melalui filter yang menekankan frekuensi yang lebih tinggi. Tujuan dari *pre-emphasis* ini adalah mengurangi *noise ratio* pada sinyal dan menyeimbangkan *spectrum* dari suara mikrofon. Proses ini akan meningkatkan energi sinyal pada frekuensi yang lebih tinggi.

Bentuk yang paling umum digunakan dalam Pre-Emphasis dirumuskan pada rumus (2.2).

$$H(z) = 1 - \alpha z^{-1}, 0.9 \leq \alpha \leq 1.0, \alpha \in R \qquad (2.2)$$

$H(z)$ = hasil *pre-emphasis* signal z

z = signal yang akan dilakukan *pre-emphasis*

Sehingga fungsi tersebut diatas akan dijadikan sebagai *first order differentiator,* turunan rumus (2.2) akan jadi seperti rumus (2.3).

$$y[i] = x[i]\text{-}\alpha x[i\text{-}1] \tag{2.3}$$

$y[i]$ = hasil *pre-emphasis* signal y pada indeks ke-i

$x[i]$ = sample signal y pada indeks ke-i

2.2.3. *Hamming Window*

Hamming Window adalah salah satu fungsi matematika dimana memiliki nilai 0 jika memasuki di luar rentang interval tertentu. Fungsi *Hamming Window* ini diperkenalkan oleh Richard W. Hamming. *Window* ini mengoptimasi sehingga meminimalisir nilai maksimum pada sisi lobus. Rumus *Hamming Window* dinotasikan pada rumus (2.4). Hasil visualisasi Hamming Window dapat dilihat pada sebelah kiri Gambar 2.2 sedangkan hasil Fourier Transform yang difungsikan terhadap Hamming Window dapat dilihat pada sebelah kanan Gambar 2.2

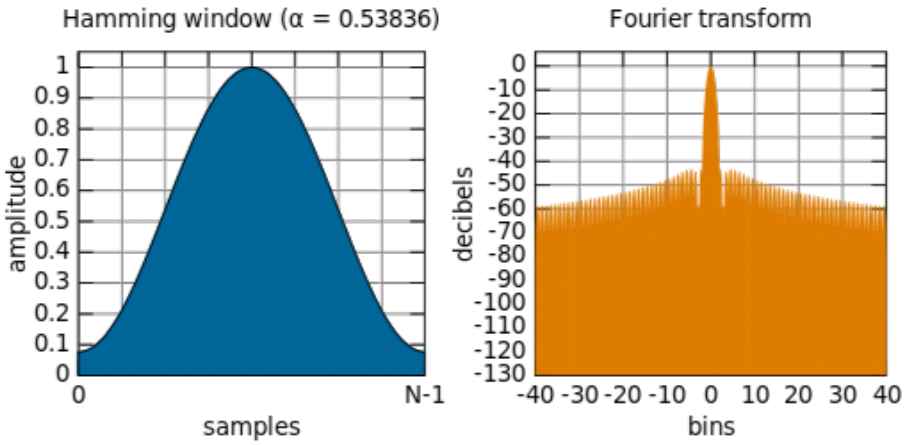
$$w(n) = \alpha\text{-}\beta \cos\left(\frac{2\pi n}{N\text{-}1}\right) \tag{2.4}$$

$w(n)$ = hasil *hamming window*

α = konstanta sebesar 0.54

β = konstanta sebesar 1.0

N = panjang window



Gambar 2.2. Grafik Hamming Window

2.2.4. *Fourier Transform*

Fourier transform merupakan fungsi transformasi yang mengubah dari domain waktu menjadi domain frekuensi. Hasil dari *Fourier transform* adalah

bilangan kompleks, dimana nilai real merepresentasikan jumlah frekuensi, dan nilai imajiner merepresentasikan fase pada grafik sinus pada frekuensi. Secara umum, *Fourier transform* dinotasikan rumus (2.5).

$$X(f) = \int_{-\infty}^{\infty} x(t)e^{-i2\pi ft}dt \tag{2.5}$$

$X(f)$ = hasil tranformasi fourier

$x(t)$ = signal kontinu pada waktu t

e = bilangan *Euler*

Fourier transform memiliki masalah jika harus berhadapan dengan data yang diskrit sehingga perlu adanya perhitungan pendekatan diskrit ke kontinu. Sehingga didapatkan rumus *Discrete Fourier transform (DFT)* pada rumus (2.6).

$$X_k = \sum_{i=0}^{N-1} x_i e^{-\frac{2\pi kn}{N}}, 0 \leq k \leq N\text{-}1 \tag{2.6}$$

X_k = hasil transformasi diskrit fourier

x_i = signal kontinu pada indeks ke-i

k = variable frekuensi diskrit $k = \frac{N}{2}, k \in N$

2.2.5. *Periodogram*

Dalam *signal processing*, periodogram digunakan untuk meng-estimasi *spectral density (power spectrum).* Istilah ini diciptakan oleh Arthur Schuster pada tahun 1898. Periodogram untuk mengestimasi power spectrum dapat dinotasikan rumus (2.7).

$$P_i(k) = \frac{1}{N}|S_i(k)|^2 \tag{2.7}$$

$P_i(k)$ = hasil periodogram pada frame ke-i

$S_i(k)$ = hasil DFT pada frame ke-i

2.2.6. Skala *Mel*

Skala *mel* adalah perseptual skala atas tangga nada suara dengan jarak yang antar tangga nada. Referensi posisi titik antara skala dan penghitungan frekuensi normal dinotasikan rumus (2.8).

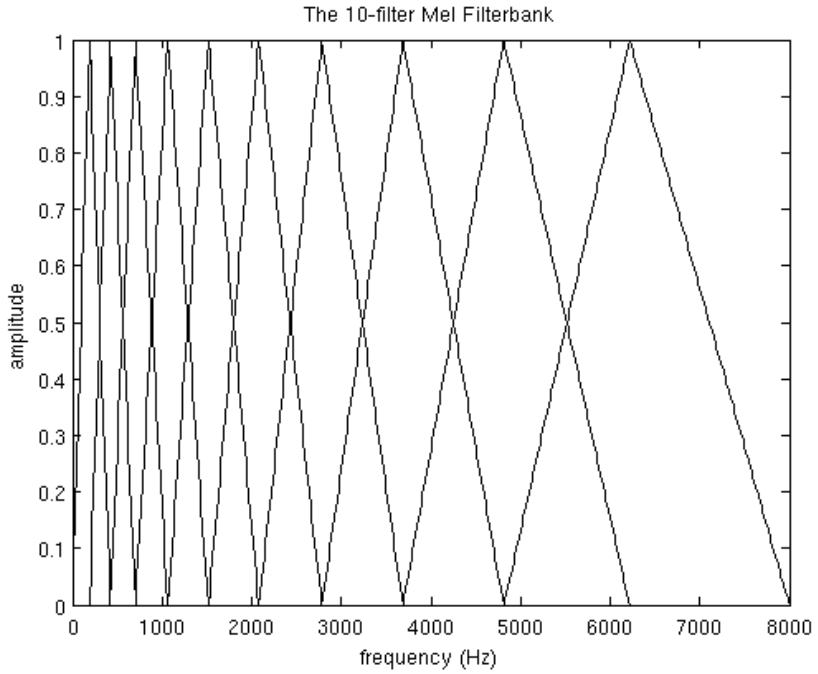
$$M(f) = 1125 \ln\left(1 + \frac{f}{700}\right) \quad (2.8)$$

$M(f)$ = hasil skala mel pada frekuensi f

Nama mel berasal dari kata *melody* untuk mengindikasikan skala berdasarkan perbedaan tangga nada.

2.2.7. *Mel Filter Bank*

Filterbank adalah kumpulan daripada *band-pass filter* dimana akan memisahkan *signal* menjadi beberapa komponen, setiapnya mewakili satu frekuensi sub-band daripada signal aslinya. *Mel filter ban*k menggunakan skala mel dalam menetapkan kumpulan band-pass filter, jumlah kumpulan biasanya 20 - 40 filterbank. Visualisasi data pengambilan 10-*filter mel filterbank* untuk suara dengan frekuensi 8kHz dapat dilihat pada Gambar 2.3.



Gambar 2.3. Contoh 10-filter mel filterbank dengan 8000Hz frekuensi.

2.2.8. *Discrete Cosine Transform*

Pada dasarnya konsep dari *Discrete Cosine Transform* (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. Oleh karena itu, masing-masing

masukan ucapan berubah menjadi urutan vektor akustik *(acoustic model)*. DCT dinotasikan rumus (2.9).

$$C_k = \sum_{i=0}^{N-1} (\log(S_i)) \cos\left[\frac{\pi}{N}\left(i + \frac{1}{2}\right)k\right] \quad (2.9)$$

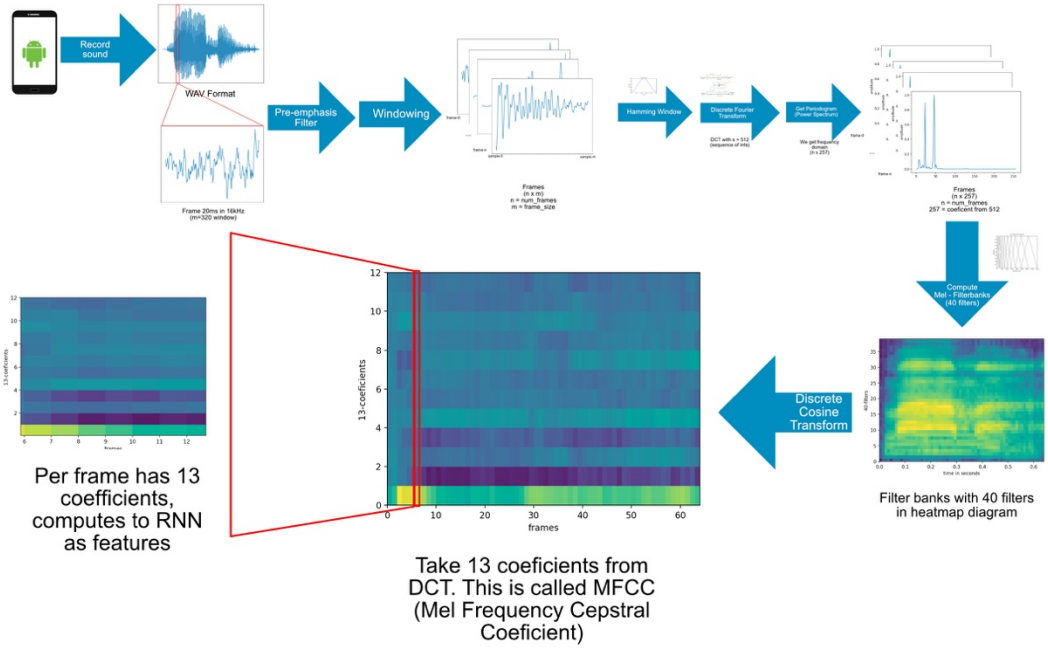
C_k = hasil *Discrete Cosine Transform* pada indeks ke-k

2.2.9. *MFCCs (Mel-Frequency Cepstral Coefficients)*

MFC *(mel-frequency cepstrum)* adalah representasi *power spectrum* suara berdasarkan transformasi cosinus linear dari *log power spectrum* pada skala mel frekuensi.

MFCC adalah koefisien dari koleksi yang dihasilkan oleh MFC. MFCC menggunakan skala mel pada *frequency band*-nya daripada *normal cepstrum* karena *frequency band* pada mel scale meng-aproksimasi sistem respon suara dan pendengaran manusia.

Pada subab 2.2.1 hingga 2.2.8 telah dijelaskan bagian – bagian yang diperlukan dalam meng-ekstrak fitur MFCC, pada Gambar 2.4 adalah alur pengambilan fitur MFCC mulai dari *raw audio* WAV hingga mendapat MFCC dengan 13 koefisien.



Gambar 2.4. *Flowchart* / diagram alur pengambilan MFCC.

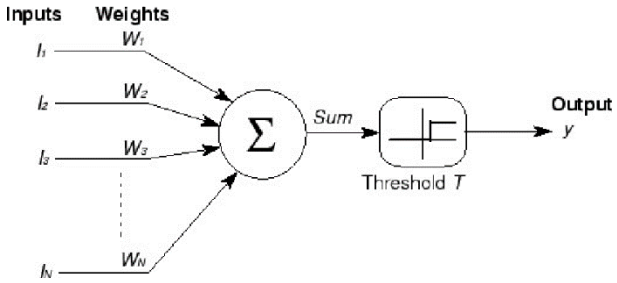
Pada Gambar 2.4 dapat dijabarkan langkah – langkah sebagai berikut :

1. Dilakukan *Pre-Emphasis filter* dengan *alpha* = 0.9.
2. Ambil *frame* sebanyak 0.02s * 16000 Hz = 320 sample, dengan *striding* 0.01s * 16000 Hz = 160 sample.
3. Dilakukan *window filter* menggunakan *Hamming window*.
4. Dilakukan *Discrete Fourier transform* dengan N = 512 *sample*.
5. Dilakukan *periodogram* untuk menghasilkan *power spectrum* setiap *frame*.
6. Hitung *mel-filterbank* dengan 40-*filter,* dan hitung *log* setiap *filter*-nya sejumlah 26 koefisien, maka dihasilkan 26 *log filterbank energies*.
7. Ubah dari domain frekuensi menjadi kembali ke domain waktu dengan *Discrete Cosine Transform* sejumlah 13 koefisien.

2.3. Artificial Neural Network

Artificial Neural Network atau disebut juga Jaringan Syaraf Tiruan adalah sistem komputasi aygn terinspirasi dari cara kerja otak makhluk hidup. Sistem yang mempelajari dalam melakukan tujuan tertentu tanpa perlu adanya teknik programming atau metode programming yang spesifik. Secara umum sistem akan belajar dengan sendirinya dengan menggunakan jaringan-jaringan yang terhubung atau disebut dengan *weighted-matrix*. Contohnya pada pengenalan citra, untuk mencapai tujuan tertentu misalnya membedakan kucing atau bukan. Dengan dataset yang diberi label mana saja citra yang berupa kucing dan bukan, sistem akan memperbaiki jaringan-jaringan yang terhubung hingga mendapatkan hasil dan tujuan yang diinginkan.

Sistem komputasi ini pertama kali dibuat oleh Warren McCulloch dan Walper Pitts pada tahun 1943. Pada saat itu komputasi dibuat berdasarkan perhitungan matematika dan algoritma yang disebut *threshold logic* (lihat Gambar 2.5).



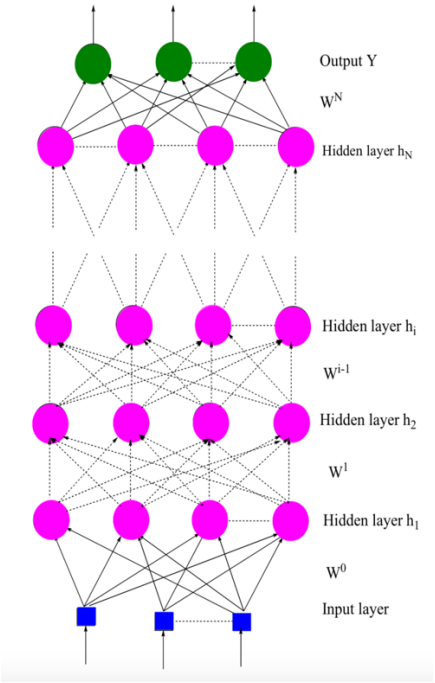
Gambar 2.5. Diagram cara kerja *threshold logic.*

Pada subab ini dijelaskan perkembangan Neural Network mulai dari *(Multi-Layer Perceptron),* RNN *(Recurrent Neural Network)*, *Bi-directional RNN,* LSTM *(Long-Short Term Memory)* dan *Connectionist Temporal Classification.* Pada subab ini juga akan dijelaskan Backpropagation yang digunakan yaitu Adam.

2.3.1. MLP *(Multi-Layer Perceptron)*

MLP *(Multi-Layer Perceptron)* adalah model perceptron yang dikemukakan oleh Rosenblatt pada tahun 1950. Pada model ini berbeda dengan model pertama kali perkembangan *Neural Network* pertama kali (lihat subbab 2.3). Perbedaan yang paling menonjol adalah pada *hidden layer* tidak hanya berjumlah satu melainkan banyak *neuron*. Jumlah banyak neuron yang digunakan disesuaikan dengan pola masalah yang akan diselesaikan. Semua layer yang terkoneksi dari *input layer*, *hidden layer*, dan *output layer* disebut juga arsitektur utama dimana akan di-*train.* sehingga dicapai parameter yang optimal sesuai dengan sebuah hasil sistem klasifikasi atau regresi.

Contoh sederhana jaringan pada MLP dapat dilihat pada Gambar 2.6. MLP juga sering disebut juga dengan *Feed-Forward Neural Network* karena cara kerjanya yang beralur maju dan terhubung selalu menuju output tidak ada layer yang saling terkonek dalam keadaan sejajar (bandingkan subab 2.3.2)



Gambar 2.6. Jaringan sederhana MLP

Bentuk umum penghitungan MLP dapat dinotasikan dengan rumus (2.10)

$$f(x) = W * x + b \tag{2.10}$$

$f(x)$ = *activation function*

W = *weight*

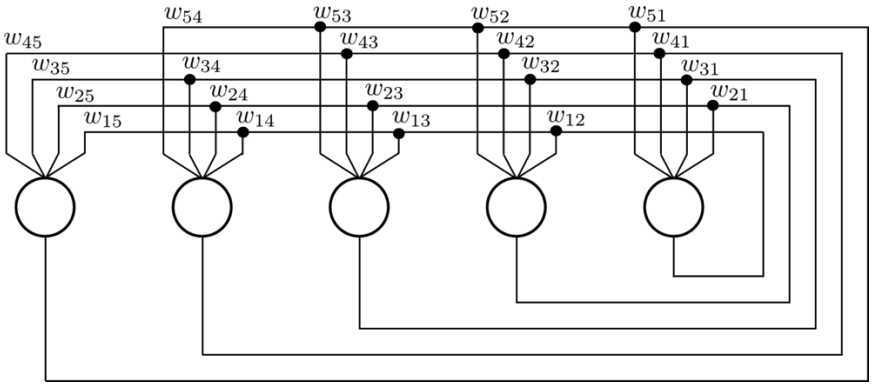
x = *input matrix*

b = *bias*

Activation function adalah fungsi non-linear seperti ReLU (Rectified Linear Unit), TanH, Sigmoid, dan Sign / Step. Fungsi non-linear diperlukan karena kasus pada *neural network* tidak selalu linear.

2.3.2. RNN *(Recurrent Neural Network)*

RNN *(Recurrent Neural Network)* adalah perkembangan model dari Hopfield Network yang dikemukakan J. J. Hopfield pada tahun 1982. Pada model ini *layer* yang satu dapat terkoneksi dengan *layer* yang lain dan dinamakan sebagai *state*. Berbeda dengan McCulloch maupun Rosenblatt dimana antar layer diharuskan meneruskan sinyal ke layer berikutnya tanpa memperdulikan state atau urutannya sekarang. Model Hopfield network dapat dilihat pada Gambar 2.7.



Gambar 2.7. Model Hopfield Network

Perkembangan model setelah Hopfield Network lebih dikenal dengan nama RNN. Permasalahan *sequence* atau berurutan dalam kasus *neural network* sering digunakan RNN sebagai solusi utama.

2.3.3. *Bi-RNN (Bi-directional Recurrent Neural Network)*

Bi-RNN (Bi-directional Recurrent Neural Network) adalah hasil modifikasi pada layer RNN dimana state tidak hanya terkoneksi dengan state berikutnya namun juga pada state sebelumnya. Rumus pada RNN dapat dinotasikan sebagai rumus (2.11)

$$h_t^{(j)} = f\left(W^{(j)T} h_t^{(j-1)} + W^{(f)T} h_{t-1}^{(j)} + b^{(j)}\right) \tag{2.11}$$

$h_t^{(j)}$ = hasil layer ke-j pada waktu t

T = panjang frame-t

Sedangkan rumus pada Bi-RNN dapat dinotasikan sebagai rumus (2.12), (2.13), dan (2.14)

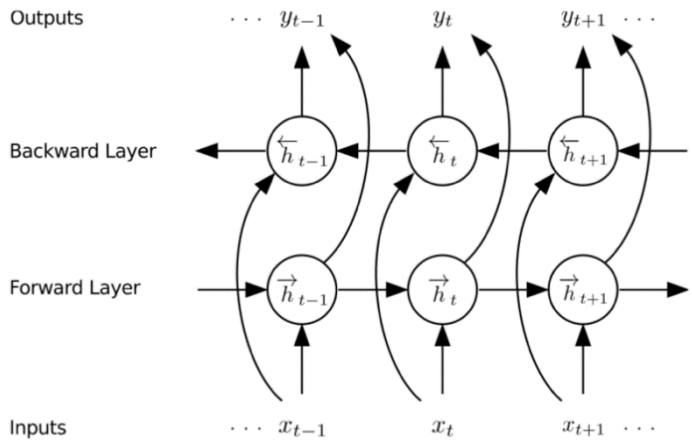
$$h_t^{(f)} = g\left(W^{(j)T} h_t^{(j)} + W^{(j)} h_{t-1}^{(j)} + b^{(j)}\right) \tag{2.12}$$

$$h_t^{(b)} = g\left(W^{(j)T} h_t^{(j)} + W^{(j)} h_{t-1}^{(j)} + b^{(j)}\right) \tag{2.13}$$

Dimana pada layer ke-j tidak hanya dihitung pada *forward state* saja, namun dengan *backward state*. Sehingga hasil dari $h_t^{(f)}$ dan $h_t^{(b)}$ dijumlahkan dengan rumus (2.14).

$$h_t^{(j)} = h_t^{(f)} + h_t^{(b)} \tag{2.14}$$

Pada rumus RNN (2.11) dengan Bi-RNN (2.14), perbedaan yang menonjol adalah adanya perhitungan *backward state,* keduanya sama-sama menghitung *forward state.* (lihat Gambar 2.8



Gambar 2.8. Alur kerja Bi-RNN

2.3.4. LSTM (Long-Short Term Memory)

LSTM (Long-Short Term Memory) dikemukakan oleh Sepp Hochreiter and Jurgen Schmidhuber pada tahun 1997. Permasalahan utama pada RNN adalah *backpropagation through time*, yaitu pengoptimalan parameter pada jaringan dengan menelusuri setiap state dari waktu t hingga T. Masalah pada *backpropagation through time* adalah vanishing gradient problem. State yang paling jauh dari waktu awal mengalami gradient error yang mendekati nol akibatnya state – state akhiran tidak mengalami perbaikan parameter dengan tepat.

LSTM terdiri dari sel memori *(memory cells)* dan unit gerbang *(gate units)*. Pada Gambar 2.9 sel memori dilambangkan dengan kotak keseluruhan dengan *input* dari x_t pada *input gate* dan *forget gate*, akan dilakukan komputasi pada rumus (2.15) hingga (2.19) sehingga menghasilkan output h_t.

$$f(t) = \sigma_g\left(W_f x_t + U_f h_{t\text{-}1} + b_f\right) \quad (2.15)$$

$$i(t) = \sigma_g(W_i x_t + U_i h_{t\text{-}1} + b_i) \quad (2.16)$$

$$o(t) = \sigma_g(W_o x_t + U_o h_{t\text{-}1} + b_o) \quad (2.17)$$

$$c(t) = f_t \circ c_{t\text{-}1} + i_t \circ c_{t\text{-}1}\sigma_c(W_c x_t + U_c h_{t\text{-}1} + b_c) \quad (2.18)$$

$$h_t = h_t \circ \sigma_h(c_t) \quad (2.19)$$

x_t = *input vector* LSTM

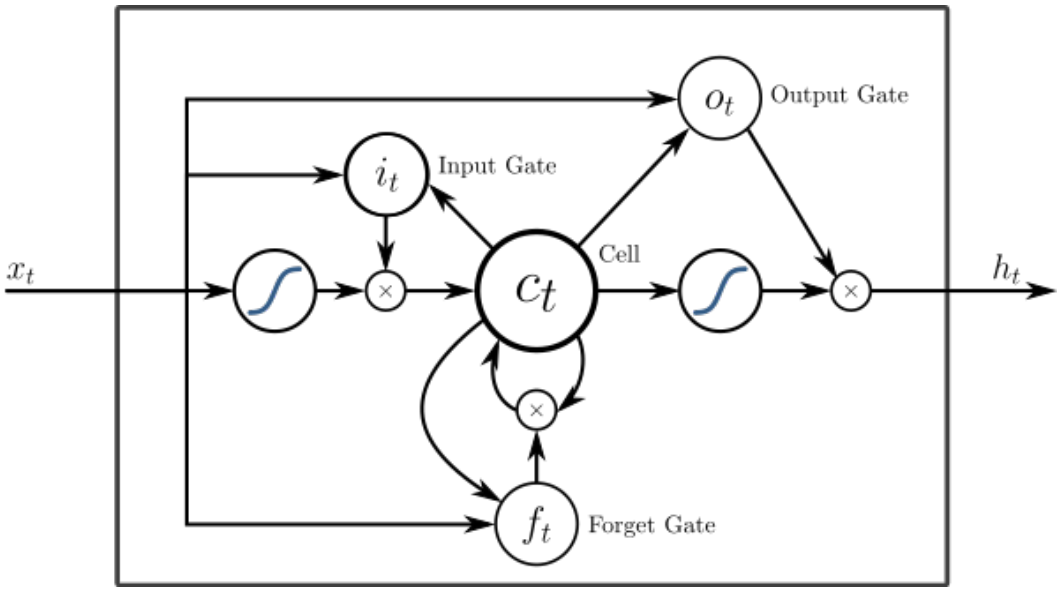
f_t = *forget gate's activation vector* LSTM

i_t = *input gate's activation vector* LSTM

o_t = *input gate's activation vector* LSTM

h_t = hasil *output vector* LSTM

c_t = *cell state vector* LSTM



Gambar 2.9. Alur kerja di dalam sel memori pada LSTM

2.3.5. CTC *(Classification Temporal Classification)*

CTC *(Classification Temporal Classification)* adalah metode pengklasifikasian dengan label yang berurutan / sekuensial tanpa perlu mensegmentasi secara eksplisit. CTC dikemukakan oleh Alex Graves pada tahun 2006. CTC sangat populer digunakan pada *Speech Recognition,* tidak hanya performa bagus yang dicapai, namun keunggulan utama pada metode ini, tidak perlunya segmentasi pada label target teks. CTC didasari ide dari HMM *(Hidden Markov Model)* dan CFR *(Conditional Field Random)*. CTC terlebih dahulu dimulai dengan perhitungan probabilitas dari tiap – tiap prediksi label yang ditetapkan. Misal saja label = { a, b, c }, maka tiap waktu t akan memiliki 3 indeks

probabilitas tersebut. Rumus untuk ditribusi daripada probabilitas terdapat pada rumus (2.20).

$$p(\pi|x) = \prod_{t=0}^{T} y_{\pi_t}^{t}, \forall \pi \in L^{'T} \tag{2.20}$$

$p(\pi|x)$ = distribusi probabilitas label x melalui *path* dinotasikan π

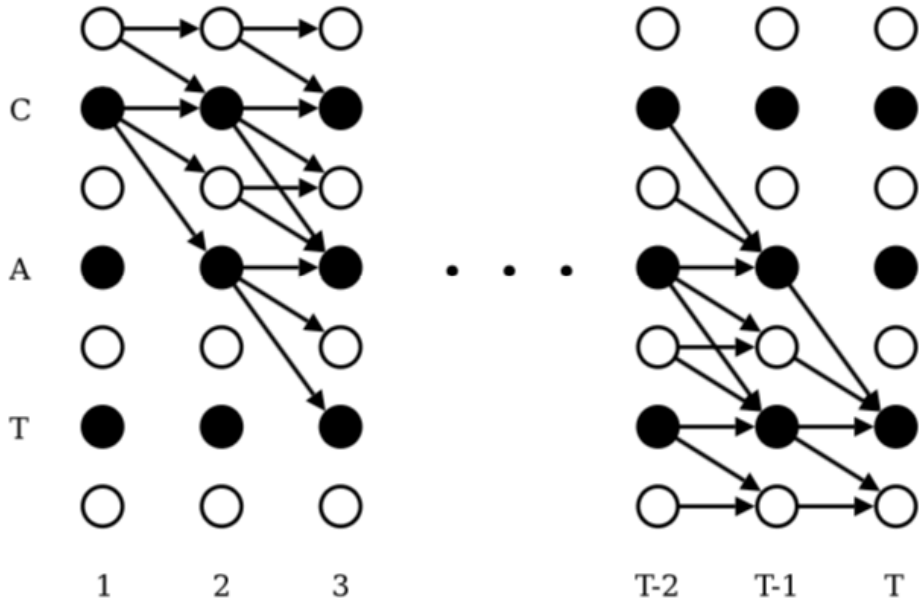
$y_{\pi_t}^{t}$ = hasil output dari RNN menggunakan operasi softmax

$L^{'T}$ = elemen dari pada semua kemungkinan *path*

Penghitungan $y_{\pi_t}^{t}$ dihitung dengan operasi softmax pada rumus (2.21).

$$h_{t,k}^{(6)} = \hat{y}_{t,k} \equiv P(c_t = k|x) = \frac{\exp\left(W_k^{(6)} h_t^{(5)} + b_k^6\right)}{\sum_j \exp\left(W_j^{(6)} h_t^{(5)} + b_j^6\right)} \tag{2.21}$$

Pada Gambar 2.10 dapat dilihat hasil prediksi dengan kata “cat” dengan melihat semua path yang mungkin. Algoritma menghitungnya menggunakan *Forward* dan *Backward Algorithm.*



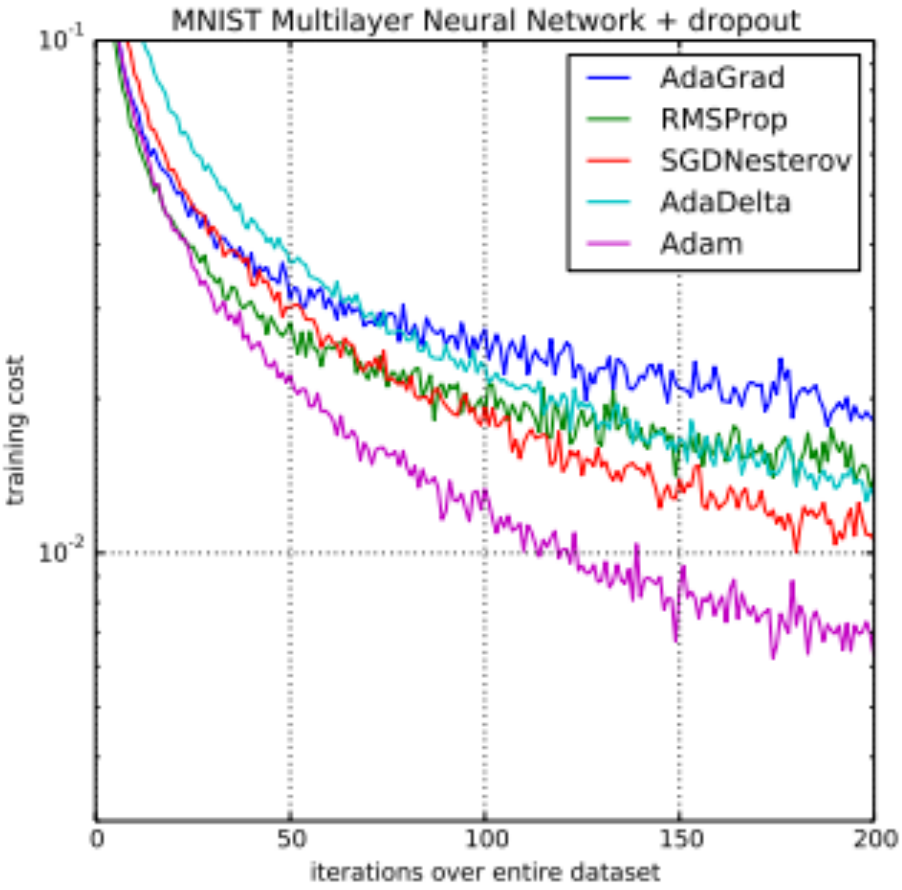
Gambar 2.10. Semua kemungkinan path untuk kata “cat”

2.3.6. Optimisasi Menggunakan Adam

Adam adalah metode yang menggabungkan dengan mengambil kelebihan daripada Adagrad dimana metode ini bekerja dengan baik pada *sparse gradient* dan RMSProp bekerja baik pada *on-line* dan konfigurasi yang tidak seimbang / *noisy*. Konfigurasi parameter yang digunakan oleh Adam adalah :

1. Alpha : *learning rate* / laju belajar
2. Beta1 : besaran momentum pertama dimana menurun secara eksponensial
3. Beta2 : besaran momentum kedua dimana menurun secara eksponensial
4. Epsilon : besaran yang hampir mendekati nol untuk menghindari hasil yang absolut nol.

Adam merupakan metode *state-of-the-art* untuk metode optimisasi *(backpropagation)* pada *neural network*, ini dapat dibuktikan dari hasil grafik pada Gambar 2.11.



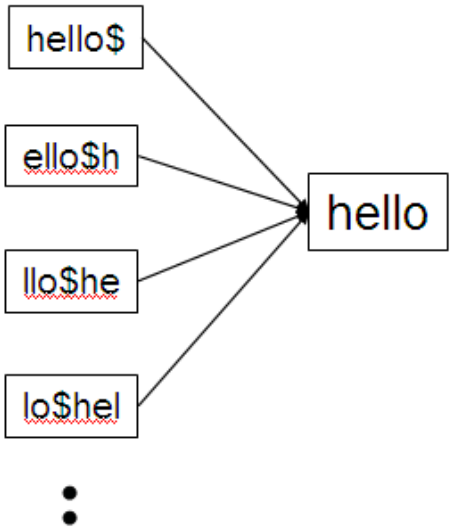
Gambar 2.11. Grafik perbandingan training cost

2.4. Language Model

Pada *language model* ini yang ingin dicapai adalah dapat mengkoreksi hasik teks yang dihasilkan oleh neural network. Hasil koreksi ini dapat diuji nantinya dengan menggunakan WER. Metode yang digunakan adalah *permuterm indexes*, *N-gram queries,* dan *Edit Distance (Levenshtein Distance).*

2.4.1. *Permuterm Indexes*

Permuterm Indexes adalah metode untuk mengkonstruksi ulang kata dari memutar indeks akhiran dari kata tersebut guna untuk mendapatkan *query* kata (*list* kata). Cara kerja metode ini dapat dilihat pada Gambar 2.12. Kata "hello$" diakhiri dengan $ yang menandakan akhiran suatu kata. Dengan men-shift indeks $ akan mendapatkan semua permutasi indeks yang memungkinkan (seakan – akan memutar).

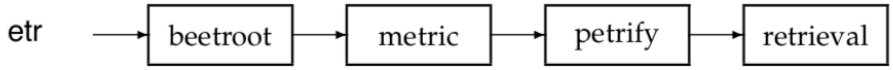


Gambar 2.12. Permuterm *indexes* pada kata "hello"

2.4.2. N-gram

Permasalahan pada *permuterm indexes* adalah meledaknya hasil permutasi yang menyebabkan lambatnya proses *language model.* Sehingga hanya diperlukan n-gram untuk mendapatkan indeks *query*, misalnya 3-gram pada $makanan$ adalah { ma, kan, an }. Maka hasil dari 3-gram digunakan sebagai *query* kata, artinya semua kata yang diawali ma, mengandung kan, dan diakhiri an. Permasalahan selanjutnya adalah hampir tidak semua kata – kata yang ingin dikoreksi adalah

match dengan kata tersebut. Contoh *query* kata kamus bahasa ingris dengan mengadung "etr" dapat dilihat pada



2.4.3. Edit Distance *(Levenshtein Distance)*

Edit distance adalah metode pengukuran tingkat kemiripan pada level bahasa mulai dari karakter, kata, dan kalimat. Edit distance juga dikenal dengan *Levenshtein Distance.* Edit distance merupakan metode pengukuran yang sangat umum pada *language model.* Edit distance menggunakan asumsi kata untuk membandingkan prediksi kata dan target kata sebagai berikut dalam penghitungannya :

1. Kata diganti dengan tidak tepat, disebut *subtitute error.*
2. Kata dihapus ketika panjang kata pada prediksi kata sudah habis dicek dengan target kata, disebut *deletion error.*
3. Kata ditambahkan ketika panjang target kata telah habis dicek semua, disebut *insertion error.*
4. Kata yang tepat diprediksi, disebut *correct.*

Sehingga dapat dirumuskan pada rumus (2.22).

$$ED = \frac{S + D + I}{S + D + C} \tag{2.22}$$

ED = hasil *edit distance*

S = *subtitute error*

D = *deletion error*

I = *insertion error*

C = *correct*