SKRIPSI

MEMPREDIKSI KEMACETAN DI KOTA BANDUNG MENGUNAKAN JARINGAN SYARAF TIRUAN

STEVEN DANIEL

NPM: 2012730021

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS
UNIVERSITAS KATOLIK PARAHYANGAN
2015

UNDERGRADUATE THESIS

TRAFFIC ANALYSIS AT BANDUNG CITY USING ARTIFICIAL NEURAL NETWORK

STEVEN DANIEL

NPM: 2012730021

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY AND SCIENCES
PARAHYANGAN CATHOLIC UNIVERSITY
2015

LEMBAR PENGESAHAN

MEMPREDIKSI KEMACETAN DI KOTA BANDUNG MENGUNAKAN JARINGAN SYARAF TIRUAN

STEVEN DANIEL

NPM: 2012730021

Bandung, «tanggal» «bulan» 2015

Menyetujui,

Pembimbing Tunggal

Pascal Alfadian, M.Com.

Ketua Tim Penguji	Anggota Tim Penguji
«penguji 1»	«penguji 2»

Mengetahui,

Ketua Program Studi

Thomas Anung Basuki, Ph.D.

PERNYATAAN

Dengan ini saya yang bertandatangan di bawah ini menyatakan bahwa skripsi dengan judul:

MEMPREDIKSI KEMACETAN DI KOTA BANDUNG MENGUNAKAN JARINGAN SYARAF TIRUAN

adalah benar-benar karya saya sendiri, dan saya tidak melakukan penjiplakan atau pengutipan dengan cara-cara yang tidak sesuai dengan etika keilmuan yang berlaku dalam masyarakat keilmuan.

Atas pernyataan ini, saya siap menanggung segala risiko dan sanksi yang dijatuhkan kepada saya, apabila di kemudian hari ditemukan adanya pelanggaran terhadap etika keilmuan dalam karya saya, atau jika ada tuntutan formal atau non-formal dari pihak lain berkaitan dengan keaslian karya saya ini.

Dinyatakan di Bandung,
Tanggal «tanggal» «bulan» 2015

Meterai

Steven Daniel
NPM: 2012730021

ABSTRAK

«Tuliskan abstrak anda di sini, dalam bahasa Indonesia» Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Kata-kata kunci: «Tuliskan di sini kata-kata kunci yang anda gunakan, dalam bahasa Indonesia»

ABSTRACT

«Tuliskan abstrak anda di sini, dalam bahasa Inggris» Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Keywords: «Tuliskan di sini kata-kata kunci yang anda gunakan, dalam bahasa Inggris»

«kepada siapa anda mempersembahkan skripsi ini. . . ?»

KATA PENGANTAR

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Bandung, «bulan» 2015

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB 1

PENDAHULUAN

1.1 Latar Belakang

Setiap tahun pertumbuhan jumlah kendaraan di Bandung selalu mengalami peningkatan [6]. Namun, pertumbuhan tersebut tidak diimbangi dengan pertumbuhan pembangunan jalan yang seimbang [7]. Akibatnya, hal tersebut mengakibatkan kepadatan lalu lintas.

Twitter adalah sebuah jaringan informasi [8]. Melalui Twitter, *netizens* dapat saling bertukar informasi secara cepat. Pengguna Twitter di Bandung ternyata cukup aktif dalam menggunakan Twitter. Sebuah lembaga pemantau media sosial bernama Semiocast mencatat, pengguna Twitter di kota Bandung menyumbang lebih dari 100 juta *tweets* sepanjang bulan juni 2012.

API (Application Programming Interface) merupakan sebuah cara yang didefinisikan sebuah program untuk menyelesaikan sebuah tugas, biasanya dengan menerima atau memodifikasi data [9]. Twitter menyediakan sebuah API yang memberikan hak akses kepada pengembang perangkat lunak untuk membaca dan menulis data dari server Twitter. Dalam pemograman bahasa Java ada sebuah *library* tidak resmi bernama Twitter4J yang membungkus Twitter API. *Library* ini memudahkan *programmer* Java dalam mengembangkan sebuah perangkat lunak yang memanfaatkan Twitter API [10].

Jaringan Saraf Tiruan (JST) adalah model komputasi yang terinspirasi dari cara kerja sistem saraf biologi [11]. Sama seperti sistem saraf biologi, JST memiliki neuron-neuron yang dapat meneruskan sinyal apabila sinyal yang dihantarkan melewati nilai tertentu. JST sendiri digunakan untuk menyelesaikan masalah yang rumit. JST juga digunakan untuk membuat sebuah kesimpulan berdasarkan informasi yang ada.

Salah satu solusi mengatasi kemacetan pada penelitian ini adalah dengan membuat sebuah perangkat lunak yang dapat memprediksi tingkat kemacetan berdasarkan *tweets* dari Twitter. Perangkat lunak akan mengambil *tweets* menggunakan *library* Java Twitter4J. Perangkat lunak akan memproses *tweets* menggunakan Jaringan Syaraf Tiruan. Keberadaan perangkat lunak diharapkan dapat membantu *netizens* dalam menghindari kemacetan.

1.2 Identifikasi Masalah

Dari observasi awal yang telah dilakukan, ada masalah yang dapat diidentifikasikan, diantaranya sebagai berikut :

1. Kemacetan di kota Bandung.

1.3 Rumusan Masalah

Berdasarkan deskripsi diatas, rumusan masalah adalah sebagai berikut :

1. Bagaimana melakukan pengambilan *tweets* dari Twitter?
2. Bagaimana memilih, memodelkan, dan menyimpan *tweets* dari Twitter menjadi sinyal-sinyal pada Jaringan Syaraf Tiruan?
3. Bagaimana menentukan jenis dan konfigurasi pada JST untuk memprediksi kemacetan?
4. Bagaimana cara melatih Jaringan Syaraf Tiruan?
5. Bagaimana memprediksi tingkat kemacetan di kota Bandung?

1.4 Tujuan

Karya tulis ilmiah ini bertujuan untuk :

1. Mengambil *tweets* dari Twitter secara *programmatic*.
2. Memilih, memodelkan, dan menyimpan *tweets* menjadi sinyal-sinyal pada JST.
3. Menentukan jenis dan konfigurasi pada JST untuk memprediksi kemacetan.
4. Melatih Jaringan Syaraf Tiruan.
5. Memprediksi tingkat kemacetan di kota Bandung.

1.5 Batasan Masalah

Karena keterbatasan waktu yang dimiliki penulis, maka ruang lingkup penelitian yang dilakukan dibatasi untuk beberapa hal berikut :

1. Penelitian ini hanya memprediksi tingkat kemacetan hanya di 10 jalan di kota Bandung.
2. Penelitian ini hanya menggunakan data yang berasal dari Twitter dengan kriteria *tweets* berumur kurang dari 5 tahun, bahasa Indonesia, dan berlokasi Bandung.

1.6 Metode Penelitian

Berikut adalah Metode Penelitian yang digunakan :

1. Melakukan studi mengenai Twitter API, *library* Java Twitter4J, Jaringan Syaraf Tiruan, SQL.
2. Merancang penyimpanan data.
3. Mengambil dan menyaring *tweets* dari Twitter sebagai input JST menggunakan Twitter4j
4. Mengimplementasikan JST kedalam bahasa Java.
5. Melatih JST
6. Melakukan eksperimen dan pengujian

1.7 Sistematika Penulisan

Sistematika penulisan setiap bab pada skripsi ini adalah sebagai berikut :

1. Bab Pendahuluan
   Bab 1 berisikan latar belakang, identifikasi masalah, rumusan masalah, tujuan, metode penelitian, sistematika penulisan dari penelitian yang dilakukan
2. Bab 2 Dasar Teori
   Bab 2 berisikan teori-teori yang menunjang penelitian yang dilakukan. Teori yang digunakan dalam penelitian ini, sebagai berikut : Twitter API, SQL, *library* Twitter4j, Jaringan Syaraf Tiruan.
3. Bab 3 Analisis
   Bab 3 berisikan analisis yang dilakukan pada penelitian ini, Analisis yang dilakukan adalah sebagai berikut : Analisis Twitter API, analisis sifat / karakter *tweets* yang akan digunakan, analisis JST yang akan digunakan, analisis perangkat lunak yang akan dibangun.
4. Bab 4 Perancangan perangkat lunak
   Bab 4 berisikan perancangan dari aplikasi JST yang dapat memprediksi kemacetan di Kota Bandung
5. Bab 5 Implementasi perangkat lunak
   Bab 5 berisikan implementasi dan pengujian dari perangkat lunak JST yang dapat memprediksi kemacetan di Kota Bandung
6. Bab 6 Kesimpulan
   Bab 6 berisikan kesimpulan dan saran dari penelitian JST memprediksi kemacetan di Kota Bandung

BAB 2

STUDI PUSTAKA

2.1 Twitter

Twitter adalah sebuah jaringan informasi yang terdiri dari pesan-pesan sepanjang 140 karakter yang disebut *tweet* [8]. Pengguna Twitter di Indonesia terutama di kota Bandung ternyata cukup aktif dalam menggunakan Twitter. Dalam data yang dirilis lembaga pemantau media sosial Semiocast ditunjukan pada gambar 2.1 "Top 20 Cities by Number of posted tweets" pengguna Twitter di Kota Bandung berada di posisi 5 dengan perkiraan 1.2 persen dari 10 miliar *tweets* selama Juni 2012. Tingginya popularitas Twitter dapat dimanfaatkan untuk berbagai keperluan dalam berbagai aspek. Contohnya sebagai sarana komunikasi, memberikan opini, kampanye poilitik, bisnis, mendapatkan informasi, dan banyak lainya.

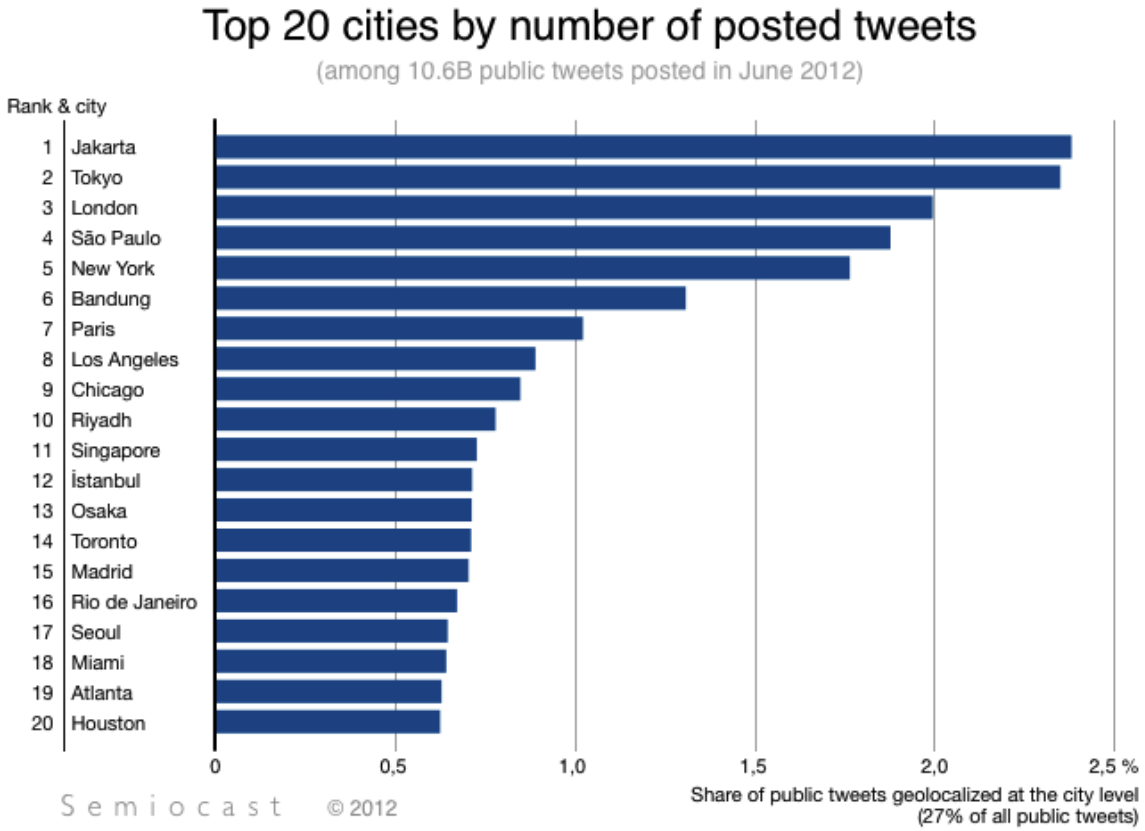
2.2 Twitter API

API (Application Programming *interface*) merupakan sebuah cara yang didefinisikan sebuah program untuk menyelesaikan sebuah tugas, biasanya dengan menerima atau memodifikasi data [9]. Twitter menyediakan sebuah API yang memberikan hak akses kepada pengembang perangkat lunak untuk membaca dan menulis data dari *server* Twitter. *Programmer* menggunakan Twitter API untuk membuat aplikasi, *website*, *widgets*, dan proyek lainya yang berinteraksi dengan Twitter. Program akan berkomunikasi dengan Twitter API melalui HTTP. Twitter menyediakan beberapa jenis dan fungsi API yang berbeda, diantaranya REST API, Streaming API, dan ads API.

2.2.1 REST API

REST API menyediakan akses secara program untuk membaca dan menulis data Twitter. Beberapa akses yang disediakan oleh Twitter seperti membuat *tweet* baru, membaca profil *author*, data *follower* dan lain-lain [3]. REST API mengidentifikasi aplikasi dan pengguna Twitter menggunakan OAuth. Twitter menyarankan jika pengembang berniat untuk memonitor atau memproses *tweets* secara *real-time* lebih baik menggunakan Streaming API daripada REST API, dikarenakan REST API memiliki *rate limits*.

Rate limiting pada API versi 1.1 ditujukan per-*user* basis atau per *access token*. *Rate limits* pada API versi 1.1 dibagi kedalam selang 15 menit. Ada 2 buah kelompok GET *request* : 15 panggilan setiap 15 menit, dan 180 panggilan setiap 15 menit. Setiap *endpoints* membutuhkan autentikasi.



Gambar 2.1: Top 20 cities by number of posted tweets, dari [1]

Ini bertujuan agar mencegah kebiasaan yang buruk, dan juga dapat membantu Twitter mengerti lebih jauh bagaimana mengkategorikan aplikasi yang menggunakan API.

2.2.2 Streaming API

Streaming API memberikan latensi yang rendah kepada pengembang untuk mengakses Twitter *global stream* dari data *tweets*. Pengimplementasian secara tepat dan benar dapat menghindari *overhead*. Twitter menawarkan beberapa *streaming endpoints*, setiap *endpoints* telah disesuaikan untuk kasus tertentu [2].

1. *Public Streams*
   Public Stream menyediakan data publik yang mengalir melewati Twitter. Jenis ini cocok untuk memantau spesifik *user* atau topik, dan penambangan data.

2. *User Streams*
   User Streams menyediakan aliran data dan spesifik kejadian untuk *user* yang terautentikasi. *User Streams* tunggal, mengandung semua data yang sesuai dengan *view* yang dimiliki *user* tunggal dari Twitter.

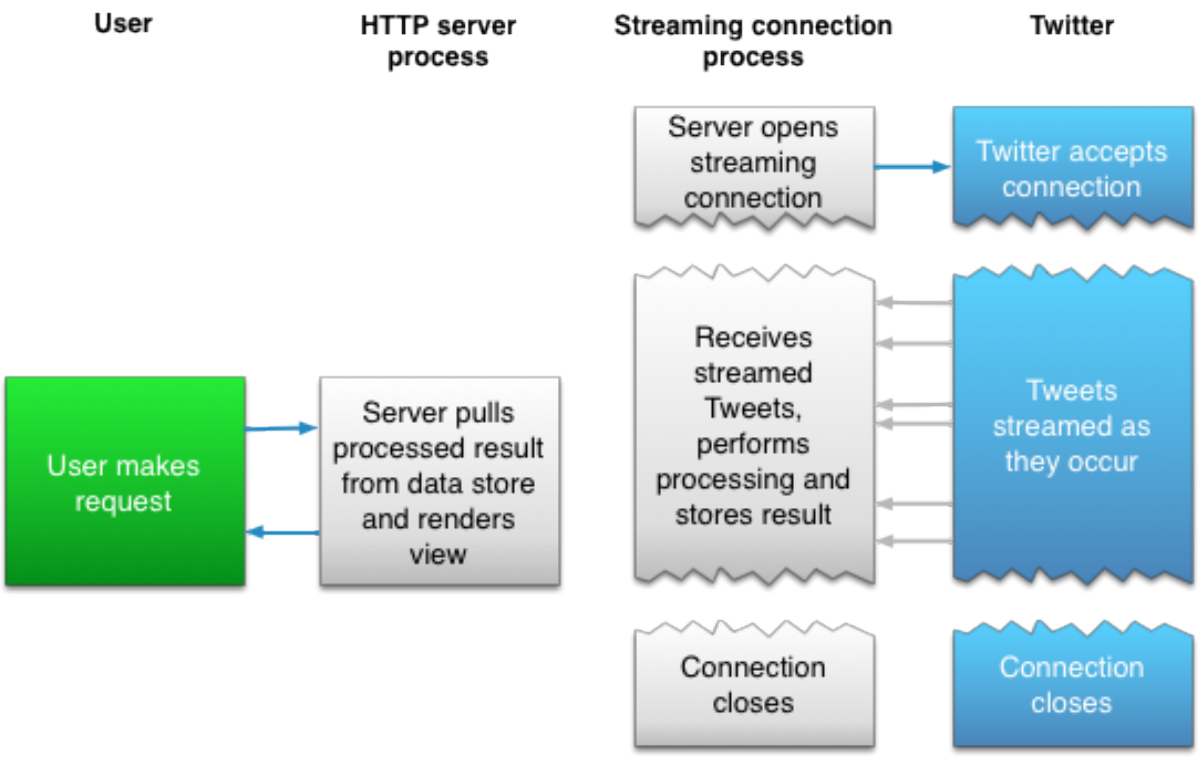
3. *Site Streams*
   Site Streams adalah versi beberapa *user* dari *User Streams*. *Site stream* ditujukan untuk *server* yang terhubung ke Twitter atas nama banyak *user*. *Site streams* berguna untuk menerima *udates* secara *real-times* dari jumlah *user* yang banyak.

2.2.3 Perbedaaan antara Streaming dan REST

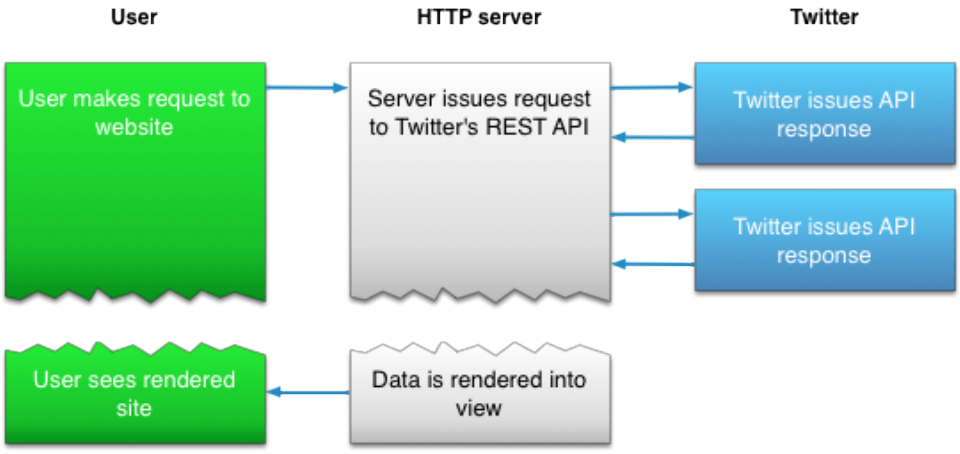
Streaming API dan REST API memiliki beberapa perbedaan. Karena adanya perbedaan ini, pengembang harus memikirkan jenis API mana yang cocok digunakan dalam aplikasinya. Setiap API memiliki karakteristik dan kelebihan yang berbeda-beda [2].

Pada Streaming API, aplikasi memerlukan koneksi HTTP yang tetap terbuka antara *streaming process* dengan *server* Twitter. Streaming API tidak dapat merespon permintaan *user* secara langsung, namun *user request* harus ditangani oleh HTTP *server* yang dimiliki oleh aplikasi. Pada gambar 2.2, aplikasi memiliki 2 buah penanganan yang terdiri dari *server* yang menangani *user request*, dan *server* yang menangani *streaming process*.

Dalam menggunakan REST API aplikasi idealnya memiliki 2 buah koneksi HTTP. Koneksi *user* dengan HTTP *server* dan Twitter *server* dengan HTTP *server*. Permintaan *user* akan diteruskan oleh HTTP *server* melalui REST API ke *server* Twitter. Respon dari *server* Twitter akan diproses oleh HTTP *server* dan diteruskan kepada *user*. Gambar 2.3 menggambarkan cara kerja dari REST API.



Gambar 2.2: Streaming API *Architecture*, dari [2]



Gambar 2.3: REST API *Architecture*, dari [3]

2.3 OAuth

OAuth 2.0 *authorization* adalah sebuah *framework* yang memungkinkan sebuah aplikasi pihak ketiga untuk mendapatkan akses terbatas pada layanan HTTP, tanpa memberikan data autentikasi [4]. Dalam model tradisional autentikasi *client*-server, ketika *client* meminta *requests* sebuah akses terhadap sumber daya yang dilindungi, pemilik dari sumber daya harus memberikan surat kepercayaan kepada *client*. Dalam hal mendukung aplikasi pihak ke-3, pemilik sumber daya harus membagi surat kepercayaanya kepada pihak ke-3. Hal ini menyebabkan beberapa masalah dan batasan :

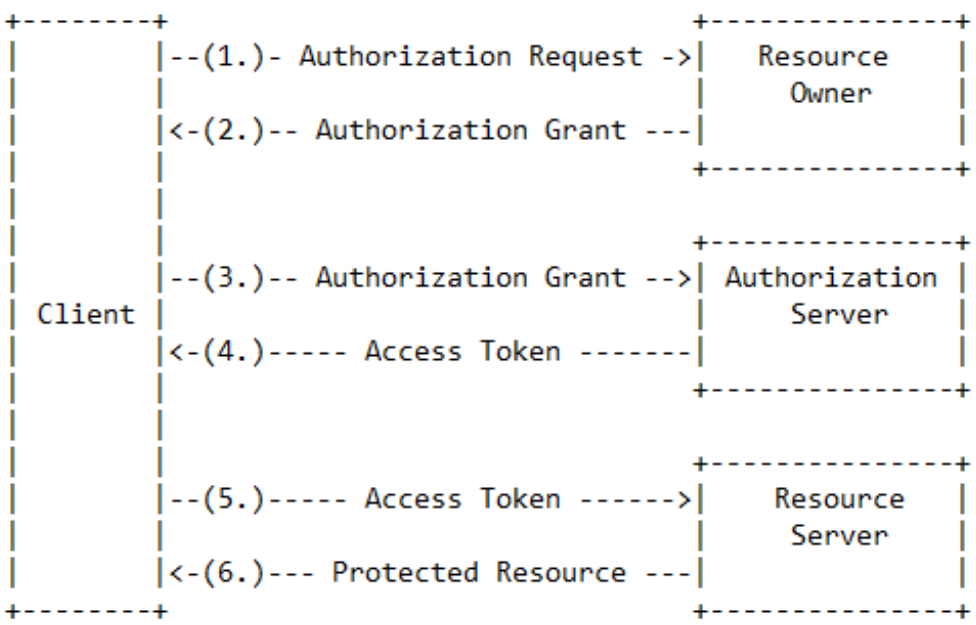
1. Aplikasi pihak ke-3 perlu menyimpan surat kepercayaan pemilik sumber daya untuk penggunaan dimasa depan, biasanya adalah sebuah textitpassword dalam teks.
2. *Server* perlu mendukung autentikasi *password*, walaupun kelemahan keamanan melekat pada *passwords*.
3. Aplikasi pihak ke-3 mendapatkan seluruh akses pada sumber daya yang dilindungi. Pemilik sumber daya tidak bisa memberikan batasan akses kepada pihak ke-3.
4. Pemilik sumber daya tidak bisa mencabut akses aplikasi pihak ke-3 tanpa mencabut seluruh akses pihak ke-3, dan harus mengganti password.

OAuth mengatasi masalah ini dengan memperkenalkan sebuah *author*ization *layer* dan membagi peran *client* untuk mengakses sumber daya. Sebagai ganti menggunakan surat kepercayaan pemilik untuk mengakses sumber daya yang dilindungi. *Client* mendapatkan *access token* (sebuah string yang berisi cakupan yang spesifik, waktu akses, dan attribut akses lainya.) *access token* diberikan kepada *client* atau pihak ketiga oleh *author*ization *server* dengan persetujuan pemiliki sumber daya. *Client* menggunakan *access token* untuk mengakses sumber daya yang dilindungi oleh *resource server*.

Dalam OAuth didefinisikan ada 4 peran:

1. Resource Owner
   Sebuah entitas yang dapat memberika akses kepada sumber daya yang dilindungi. Ketika resource owner seorang manusia, ini merujuk pada end-*user*.
2. Resource Server
   server yang menyediakan sumber daya yang dilindungi, dapat menerima dan merespon permintaan sumber daya yang dilindungi menggunakan *access token*s.
3. *client*
   Sebuah aplikasi yang meminta hak akses kepada sumber daya yang dilindungi.
4. *authorization server*
   server yang memberikan *access token*s kepada *client* ketika proses autentikasi berhasil.

Protocol Flow dari Oauth akan dijelaskan pada gambar 2.4



Gambar 2.4: *Protocol Flow* Oauth, dari [4]

1. *client* melakukan meminta *authorization grant* kepada pemiliki sumber daya. Permintaan bisa dilakukan secara langsung kepada pemilik sumber daya, atau secara tidak langsung melewati *authorization server* sebagai penengah.
2. *client* mendapatkan *author*ization grant.
3. *authorization grant* digunakan untuk meminta *access token* kepada *author*ization Server.
4. *authorization server* mengautentikasi apakah *authorization grant* yang dimiliki *client* valid, dan jika valid, *client* diberikan sebuah *access token*.
5. *client* meminta sumber daya yang dilindungi dari sumber daya *server* dan melakukan autentikasi dengan memperlihatkan *acces token*.
6. *resource server* memvalidasi *access token*, dan jika valid, *server* akan melayani permintaan.

2.4 Twitter4J

Twitter4j adalah sebuah Java library untuk Twitter API bersifat *open source* dan gratis. Dengan Twitter4j, *user* dapat dengan mudah mengintegrasikan aplikasi Java dengan Twitter service. Twitter4j dapat di unduh pada situs `http://twitter4j.org` [10]. Berikut beberapa kelas yang dimiliki oleh Twitter4j:

Twitter adalah sebuah *interface* yang digunakan untuk membungkus Twitter REST API. Method yang dimiliki *interface* ini sebagai berikut:

1. **TimelinesResources timelines()**
   Berfungsi mendapatkan objek TimelinesResources

Kembalian Mengembalikan sumber daya berupa *timelines* yang dimiliki oleh pemilik sumber daya berdasarkan *access token* Oauth.

2. **TweetsResources tweets()**
   Berfungsi mendapatkan objek TweetResources
   Kembalian Mengembalikan sumber daya berupa *tweets* yang dimiliki oleh pemilik sumber daya berdasarkan *access token* Oauth.

3. **SearchResource search()**
   Berfungsi mendapatkan objek SearchResource
   Kembalian Mengembalikan kelas SearchResource yang dimiliki oleh pemilik sumber daya berdasarkan *access token* Oauth.

TwitterFactory adalah sebuah kelas denggan pattern singleton yang digunakan untuk menginstansiasi kelas Twitter berdasarkan *config tree*. Method yang dimiliki kelas ini sebagai berikut:

1. **static Twitter getSingleton()**
   Berfungsi menginstansiasi kelas Twitter hanya satu kali.
   Kembalian instansiasi default singleton Twitter

TwitterStream adalah sebuah *interface* yang digunakan untuk membungkus Twitter Streaming API. Method yang dimiliki *interface* ini sebagai berikut:

1. **void addListener(StreamListener listener)**
   Berfungsi menambahkan *listener*.
   Kembalian void

2. **void removeListener(StreamListener listener)**
   Berfungsi menghapus *listener*.
   Kembalian void

3. **void filter(String query)**
   Berfungsi mengkonsumsi status publik yang sesuai dengan satu atau lebih predikat *filter*.
   parameter query - String
   Kembalian void

4. **void shutdown()**
   Berfungsi mematikan thread yang dibagikan oleh instansiasi TwitterStream
   Kembalian void

TwitterStreamFactory adalah sebuah kelas yang digunakan untuk menginstansiasi kelas TwitterStream. Method yang dimiliki kelas ini sebagai berikut:

1. **static TwitterStream getSingleton()**
   Berfungsi mendapatkan instansiasi dari kelas TwitterStream
   Kembalian instansiasi default singleton TwitterStream

Paging adalah sebuah kelas yang digunakan untuk mengontrol *pagination*. Method yang dimiliki kelas ini sebagai berikut:

1. **void setCount(int count)**
   Berfungsi untuk mengubah banyaknya status dalam 1 page
   Parameter count - int
   Kembalian void

2. **void setPage(int page)**
   Berfungsi untuk mengubah nomor page yang dipilih
   Parameter page - int
   Kembalian void

TimelinesResources adalah sebuah kelas yang digunakan untuk mengolah timelines berdasarkan hak akses dari *access token*. Method yang dimiliki kelas ini sebagai berikut:

1. **ResponseList<Status> getUserTimeline(String screenName,Paging paging)**
   Berfungsi untuk mendapatkan status dari timeline seseorang sejumlah paging yang diinginkan.
   Parameter screenName - String, paging - Paging
   Kembalian status dari timeline spesifik *user*.

SearchResources adalah sebuah kelas yang digunakan untuk mengolah pencarian berdasarkan hak akses dari *access token*. Method yang dimiliki kelas ini sebagai berikut:

1. **QueryResult search(Query query)**
   Berfungsi untuk mendapatkan tweets yang sesuai dengan spesifik query.
   Parameter query - Query
   Kembalian result dari query diminta

Query adalah sebuah kelas yang digunakan untuk mengolah query untuk menentukan pencarian. Method yang dimiliki kelas ini sebagai berikut:

1. **void setGeoCode(GeoLocation location, double radius, Unit unit**
   Berfungsi untuk mencari *tweets* berdasarkan radius tertentu.
   Parameter location - GeoLocation, radius - doouble, unit - Query.Unit)
   Kembalian void

2. **void setLocale(String locale)**
   Berfungsi untuk mencari *tweets* hanya pada bahasa tertentu.
   Parameter locale - String
   Kembalian void

3. **void setSince(String since)**
   Berfungsi untuk mencari *tweets* sejak tanggal tertentu.
   Parameter since - String
   Kembalian void

Status adalah sebuah *interface* yang merepresentasikan sebuah status dari seorang *user*. Method yang dimiliki *interface* ini sebagai berikut:

1. **java.util.Date getCreatedAt()**
   Berfungsi untuk mengetahui tanggal status tersebut dibuat.
   Kembalian tanggal status dibuat.

2. **long getId()**
   Berfungsi untuk mengetahui id status tersebut.
   Kembalian id status.

3. **java.lang.String getText()**
   Berfungsi untuk mengambil isi dari status.
   Kembalian isi dari status

4. **GeoLocation getGeoLocation()**
   Berfungsi untuk mengetahui lokasi tweet bila diketahui.
   Kembalian lokasi tweets berasa jika lokasi diketahui.

5. **User getUser()**
   Berfungsi untuk mengetahui user yang terasosiasi dengan status.
   Kembalianuser yang terasosiasi dengan status.

StatusListener adalah sebuah *interface* yang merepresentasikan bagaimana status yang dilisten akan ditangani. Method yang dimiliki *interface* ini sebagai berikut:

1. **void onDeletion**
   Berfungsi menangani ketika *deletionNotices* disampaikan.
   Kembalian void.

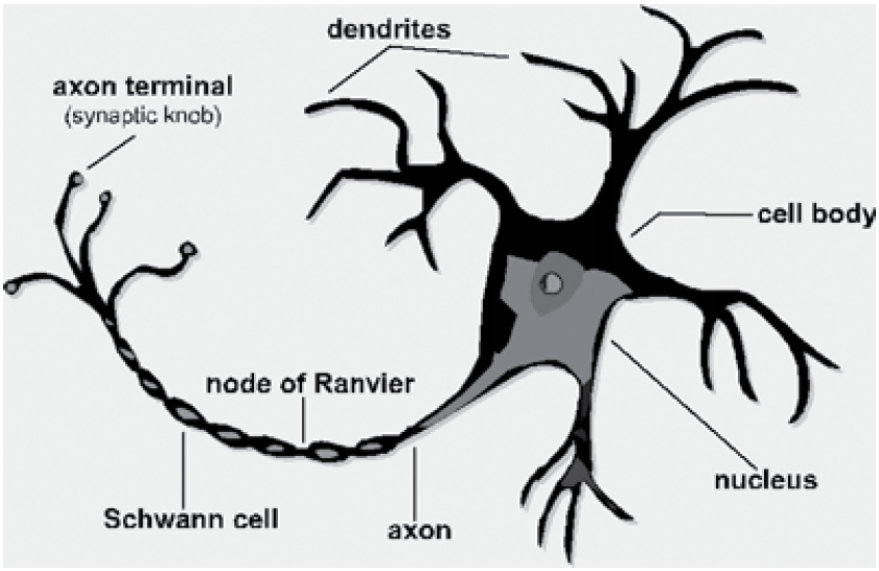
2. **void onStatus**
   Berfungsi menangani ketika mendapatkan status ketika melakukan proses *listen*.
   Kembalian void.

2.5 Jaringan Saraf Tiruan

Jaringan Saraf Tiruan (JST) adalah paradigma memproses sebuah informasi yang terinspirasi dari cara kerja sistem saraf biologi, seperti otak, memproses informasi [11]. Kunci utama dari paradigma ini adalah struktur dari sistem proses informasi. Sistem ini terdiri dari dari banyak unsur proses yang saling terhubung (neuron) bekerja secara serempak untuk menyelesaikan suatu masalah. JST dikonfigurasi untuk sebuah penerapan yang spesifik, seperti pengenalan pola atau pengklasifikasian data, melalui proses belajar. Di dalam sistem biologi, belajar melibatkan penyesuaian hubungan sinaptik yang berada diantara neuron.

Dalam sub bab dibawah peneliti akan membahas beberapa hal mengenai Jaringan Saraf Tiruan meliputi :

1. **Cara Kerja Jaringan Saraf Biologi**, untuk mengerti JST bekerja kita harus mengetahui cara kerja dari jaringan saraf biologi itu sendiri, dalam sub bab ini akan dibahas mengenai jaringan saraf biologi.



Gambar 2.5: Neuron Biologi, dari [5]

2. **Masalah yang tidak cocok diselesaikan dengan JST**, dalam sub bab ini akan dibahas bagaimana JST bisa menyelesaikan masalah
3. **Masalah yang dapat diselesaikan dengan JST**, dalam sub bab ini akan dibahas jenis permasalahan yang efektif dikerjakan oleh JST
4. **Metode Pembelajaran**, agar JST dapat menyelesaikan suatu masalah diperlukan proses belajar seperti halnya jaringan saraf biologi. Bab ini akan membahas metode pembelajaran.
5. **Perhitungan Galat**, dalam bab ini akan dibahas cara menghitung galat dari masukan dan keluaran JST.
6. **Feedforward Neural Network**, pada bab ini akan dibahas jenis JST yang bernama Feedforward Neural Network.

2.5.1 Cara Kerja Jaringan Saraf Biologi

Untuk membangun sebuah komputer yang dapat berfikir seperti manusia, para peneiliti harus memodelkanya seperti otak manusia. Komposisi utama otak manusia adalah sel neuron. JST harus mencoba mensimulasikan sifat-sifat dari sel neuron itu sendiri.

Sebuah sel neuron, seperti pada gambar 2.5, menerima sinyal dari dendrit. Ketika neuron menerima sebuah sinyal, ada kemungkinan neuron tersebut meneruskan sinyal tersebut. Ketika neuron meneruskanya, sinyal tersebut ditransimisikan melewati axon neuron. Sinyal tersebut akan melewati terminal axon, dan ditransmisikan ke neuron lain.

2.5.2 Masalah yang tidak cocok diselesaikan dengan JST

JST dapat memproses kumpulan informasi yang akan menghasilkan keluaran. Keluaran JST nantinya akan digunakan untuk membantu pengambilan sebuah keputusan. Faktanya tidak semua

masalah cocok diselesaikan dengan JST. Ada beberapa jenis masalah yang kurang cocok diselesaikan oleh JST :

1. Masalah yang dapat diselesaikan dengan program yang mudah untuk di tuliskan kedalam *flowcharts*.
2. Masalah yang dapat diselesaikan dengan program yang langkahnya dapat didefnisikan dengan terperinci.
3. Masalah yang harus diketahui cara solusinya diturunkan.
4. Algoritma yang digunakan untuk menyelesaikan sebuah masalah tidak berubah-rubah (Statik).

2.5.3 Masalah yang dapat diselesaikan dengan JST

Walau tidak semua masalah dapat diselesaikan oleh JST, namun ada banyak hal yang dapat diselesaikan oleh JST. Jenis masalah yang sering diselesaikan oleh JST sebagai berikut :

1. **Classification** adalah proses mengklasifikasian informasi menjadi beberapa jenis kelompok. Contohnya, perusahaan asuransi ingin mengklasifikasikan permohononan asuransi menjadi beberapa kategori risiko yang berbeda, atau sebuah organisasi online ingin membuat sistem email mereka dapat mengklasifikasikan pesan masuk menjadi kelompok *spam* dan bukan *spam*. Untuk mencapai hal tersebut JST harus dilatih menggunakan beberapa contoh kelompok data dan instruksi. Setiap kelompok data diklasifikasikan menjadi anggota himpunan tertentu. JST dapat belajar dari contoh kelompok data tersebut. Setelah proses pembelajaran diharapkan JST dapat mengindikasi anggota kelompok data yang baru.
2. **Prediction** adalah penerapan lain yang sering digunakan untuk JST. Dengan memberikan serangkaian masukan-keluaran data berdasarkan basis waktu, sebuah JST digunakan untuk memprediksi masa depan. Akurasi prediksi akan bergantung pada banyak faktor, seperti quantiti dan relevansi dari masukan data. JST biasanya diterapkan pada masalah yang melibatkan prediksi pergerakan dalam pasar finansial.
3. **Pattern Recognition** adalah sebuah bentuk pengklasifikasian. Pattern recognition adalah kemampuan untuk mengenali pola. Pola harus bisa dikenali bahkan ketika datanya berubah. Sebagai contoh dalam kehidupan kita pengemudi harus dapat dengan tepat mengidentifikasi lampu lalu lintas. Walaupun tidak semua lampu stopan bentuknya sama, pengemudi tetap dapat mengenalinya. Hal ini juga harus dicapai oleh JST agar komputer dapat melakukan pengenalan pola.
4. **Pattern Recognition** adalah sebuah bentuk pengklasifikasian. Pattern recognition adalah kemampuan untuk mengenali pola. Pola harus bisa dikenali bahkan ketika datanya berubah. Sebagai contoh dalam kehidupan nyata, pengemudi harus dapat dengan tepat mengidentifikasi lampu lalu lintas. Walaupun tidak semua lampu stopan bentuknya sama, pengemudi tetap dapat mengenalinya. Hal ini juga harus dicapai oleh JST agar komputer dapat melakukan pengenalan pola.

5. **Optimization** suatu kemampuan JST untuk mencari solusi yang optimal. Biasanya digunakan ketika suatu masalah memiliki *state space* yang sangat besar. JST mungkin tidak selalu menemukan solusi optimal, namun JST dapat mencari solusi yang dapat diterima. Salah satu masalah optimasi yang paling terkenal ialah Traveling Sales Problem.

2.5.4 Metode Pembelajaran

Ada banyak cara untuk membuat JST dapat belajar. Setiap algoritma pembelajaran nantinya akan melibatkan perubahan bobot setiap penghubung neuron. Proses pelatihan sangatlah penting bagi JST. Ada dua bentuk dari pelatihan yang dapat digunakan, *supervised* dan *unsupervised*. *supervised training*melibatkan JST dengan serangkaian masukan-keluaran yang diinginkan. Pada *unsupervised training* dibutuhkan juga kumpulan pelatihan, namun pelatihan tersebut tidak perlu disertai keluaran.

1. *Unsupervised training* adalah salah satu metode pembelajaran yang disediakan data masukan namun tidak perlu disediakan antisipasi keluaran. *Unsupervised training* biasanya digunakan untuk melatih JST klasifikasi. Penerapan lainya digunakan untuk data mining. *Unsupervised training* juga biasa digunakan untuk *self-organizing maps* (SOM). *Unsupervised training* dapat diterapkan kedalam banyak situasi. Pada gambar 2.6 dijelaskan *flowcharts* proses *unsupervised training*.

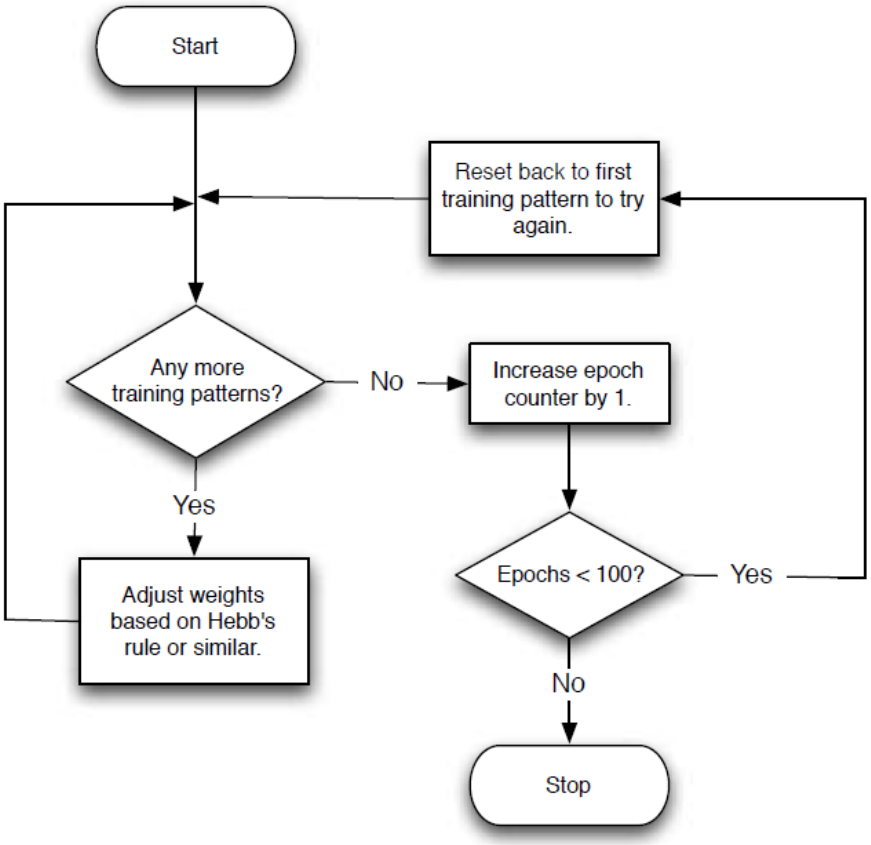
2. *Supervised Training* adalah metode pembelajaran, yang memiliki sekumpulan pelatihan. Perbedaan utama antara *supervised training* dan *unsupervised training* adalah pada *supervised training* disediakan harapan keluaran. Hal ini memungkinkan JST untuk menyesuaikan nilai dari bobot matriks berdasarkan perbedaan antara keluaran yang diharapkan dengan keluaran yang sesunguhnya. Pada gambar 2.7 dijelaskan *flowcharts* proses *supervised training*.

2.5.5 Perhitungan Galat (*Error*)

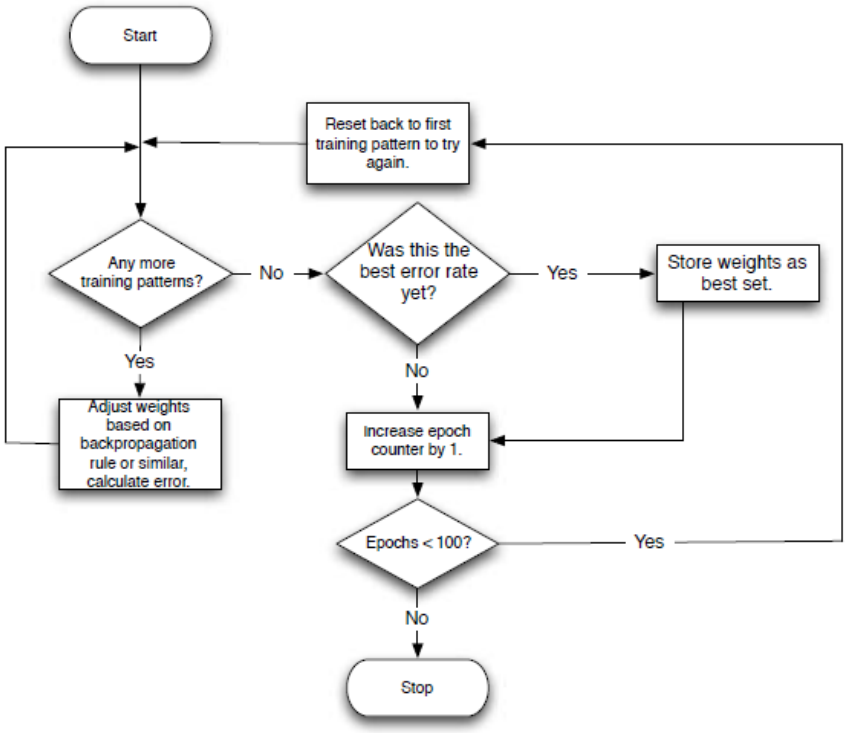
Perhitungan galat adalah salah satu aspek penting dari setiap pelatihan JST. Apakah pelatihan itu adalah *supervised* atau *unsupervised*, sebuah rata-rata galat harus dihitung. Tujuan dari setiap algoritma pelatihan ialah untuk meminimalisasi rata-rata galat.

Perhitungan Galat dan *supervised* Training

Ada dua nilai yang harus dipertimbangkan dalam menentukan rata-rata galat untuk *supervised training*. Pertama kita harus menghitung galat untuk setiap element pelatihan. Kedua, kita harus menghitung rata-rata galat untuk semua element pelatihan untuk setiap sampel. (Root Mean Square) RMS adalah salah satu metode untuk menghitung rata-rata galat untuk sebuah pelatihan. Metode RMS efektif dalam perhitungan rata-rata galat, tanpa memperhatikan apakah hasil yang sebenarnya lebih tinggi atau lebih kecil daripada hasil yang diharapkan. Untuk menghitung RMS



Gambar 2.6: *Unsupervised training*, dari [5]



Gambar 2.7: *Supervised training*, dari [5]

digunakan formula.

$$x_{rms} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(aktual_i - ideal_i)^2}$$

Dimana n adalah jumlah pasangan masukan dan keluaran, aktual adalah keluaran yang dihasilkan oleh JST, dan ideal adalah keluaran yang diinginkan. Dengan mengetahui rata-rata galat dari keluaran yang diharapkan dan yang sebenarnya, kita dapat menentukan apakah JST tersebut sudah cukup baik atau belum.

2.5.6 Feedforward Neural Network

Feedforward adalah sebuah arsitektur JST yang populer dan banyak digunakan sebagai model dalam banyak penerapan. Feedforward dikenal juga sebagai "multi-layer perceptron". Dalam JST feedforward, setiap layer dari JST mengandung hubungan ke layer berikutnya (contohnya dari layer masukan dihubungkan ke layer tersembunyi) ilustrasi feedforward pada gambar 2.8 menunjukan sebuah JST yang terdiri dari 3 lapisan (*input, hidden,* dan *output*)setiap layer memiliki jumlah neuron yang berbeda beda pada *input layer* terdiri dari 2 neuron, pada *hidden layer* terdiri 3 neuron, dan pada *output layer* hanya ada 1 neuron. Hubungan antar neuron pada feedforward hanya satu arah. Feedfoward selalu dimulai dari layer masukan. Jika masukan terhubung dengan sebuah layer tersembunyi, layer tersembunyi dapat terhubung dengan layer tersembunyi lainya atau dapat langsung terhubung dengan layer keluaran. Jumlah layer tersembunyi bisa banyak. Kebanyakan JST biasanya akan memiliki satu buah layer tersembunyi, dan akan sangat jarang JST memiliki lebih dari dua buah layer tersembunyi.

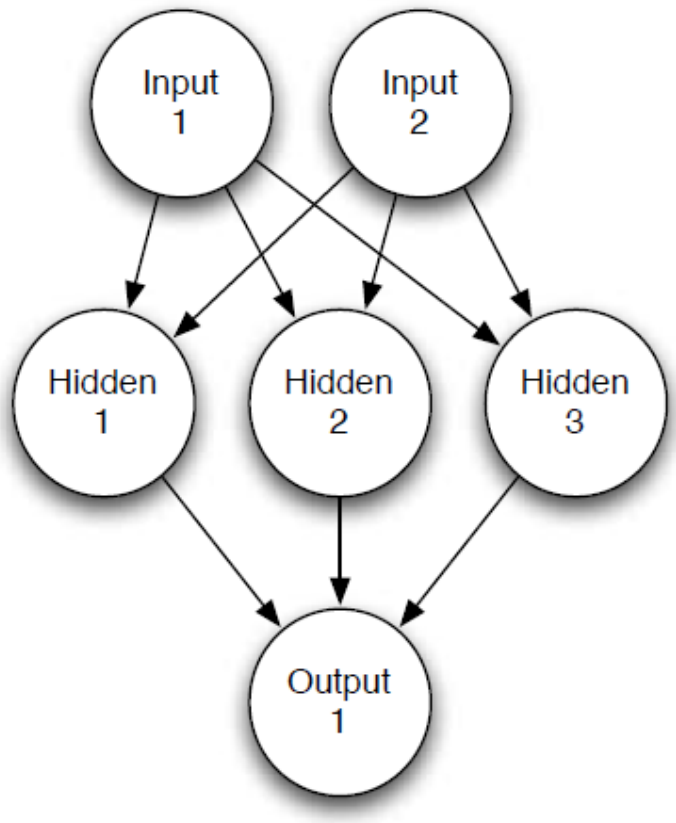
Memilih Struktur Jaringan

Ada banyak cara untuk membangun JST feedforward. Kita harus menentukan berapa jumlah neuron pada layer masukan dan layer keluaran. Selain itu layer tersembunyi harus ditentukan. Ada banyak teknik untuk memilih parameter tersebut. Untuk menentukan struktur yang optimal pada feedforward dibutuhkan pengalaman dan percobaan.

1. **Layer masukan**
   Jumlah neuron pada layer masukan dapat ditentukan bergantung pada data yang kita miliki. Parameter ini biasanya ditentukan secara unik ketika kita mengetahui data pelatihan kita. Secara spesifik, jumlah dari neuron setara dengan banyaknya kolum pada data kita. Biasanya kita dapat menambahkan satu buah titik bias.

2. **Layer Keluaran**
   Seperti layer masukan, setiap JST memiliki satu buah layer keluaran. Untuk jumlah neuron di dalamanya ditentukan pada model apa yang kita gunakan. Apakah JST kita *Machine Mode* atau *Regression Mode*. Jika pada *Machine Mode* JST akan mengembalikan kelas, sedangkan pada *Regression Mode* mengembalikan sebuah nilai. bila JST menggunakan sebuah regressor, maka keluarannya akan memiliki 1 neuron. Bila keluarannya sebuah pengklasifikasian, maka akan memiliki satu neuron atau lebih.

3. **Layer Tersembunyi**
   Dalam menentukan layer tersembunyi ada dua buah keputusan yang harus diperhatikan. Per-



Gambar 2.8: Contoh Feed Forward Neural Network, dari [5]

Tabel 2.1: Menentukan jumlah hidden layer

Jumlah Hidden Layer	Kegunaan
tidak ada	Hanya dapat merepresentasikan fungsi linear atau pemilihan linear.
1	Dapat memperikaran semua fungsi yang mengandung pemetaan kontinu dari satu *space* terbatas ke yang lainya.
2	Dapat merepresentasikan sebuah keputusan yang batasanya dan akurasi yang berubah-ubah dengan fungsi aktivasi rasional dan dapat memperkirakan setiap pemetaan halus untuk akurasi apapun.

tama menentukan berapa jumlah layer tersembunyi yang dibutuhkan. Kedua menentukan berapa jumlah neuron pada setiap layer.

Secara teori tidak ada alasan untuk menggunakan layer tersembunyi lebih dari dua. Faktanya pada masalah yang ada pada kehidupan sehari-hari, dengan 1 buah layer tersembunyi banyak masalah dapat diselesaikan. Tabel 2.1 merupakan kegunaan JST berdasarkan banyaknya layer tersembunyi

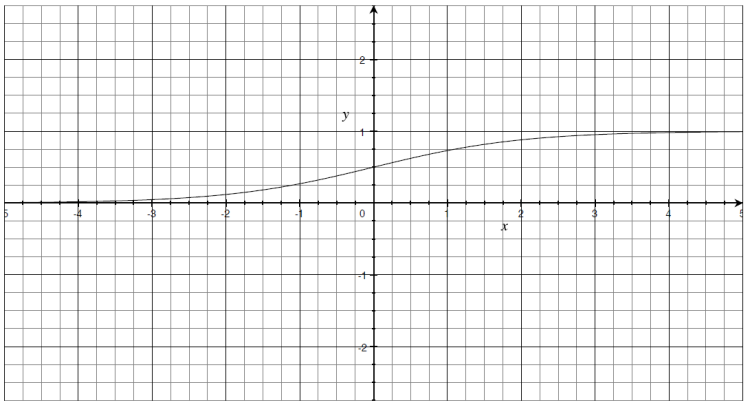
Untuk menentukan jumlah neuron dalam layer tersembunyi adalah bagian yang sangat penting dalam menentukan keseluruhan arsitektur JST. Walaupun layer ini tidak secara langsung berinteraksi dengan lingkungan luar, layer ini memiliki pengaruh yang sangat besar pada akhir keluaran. Jumlah layer tersembunyi dan jumlah neuronya sangat penting diperhatikan. Jika kita terlalu sedikit menggunakan neuron dalam layer tersembunyi ini akan mengakibatkan sesuatu yang disebut *underfitting*. Namun bila terlalu banyak menggunakan neuron pada layer tersembunyi ini akan mengakibatkan *overfitting*. *Overfitting* terjadi ketika JST terlalu banyak memiliki informasi yang diproses daripada jumlah batas dari informasi yang terkandung dalam sejumlah pelatihan. Ada beberapa tips untuk menentukan jumlah neuron: Pertama jumlah neuron harus berada diantara jumlah masukan dan keluaran. Kedua jumlah neuron seharusnya 2/3 ukuran dari layer masukan, ditambah dengan layer keluaran. Ketiga jumlah neuron seharusnya kurang dari dua kali dari layer masukan.

4. **Fungsi Aktivasi**
   Kebanyakakn JST mengeluarkan keluaran dari layernya menggunakan fungsi aktivasi. Fungsi aktivasi ini menskalakan keluaran dari JST dalam jangkauan tertentu. Fungsi aktivasi dapat kita buat sendiri namun umumnya menggunakan fungsi yang sudah sering digunakan. Ada beberapa jenis fungsi aktivasi yang sering digunakan diantara sebagai berikut:

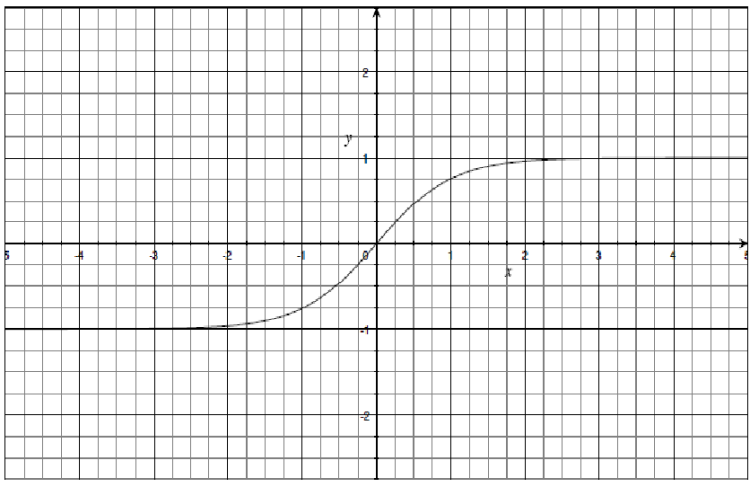
   Sigmoid adalah fungsi aktivasi yang menggunakan fungsi sigmoid untuk menentukan aktivasi. fungsi sigmoid di definisikan sebagai berikut:

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

   Kurva 2.9 menggambarkan fungsi sigmoid. Dalam perhitungan sigmoid berapapun parameter x maka y tidak akan lebih dari satu dan kurang dari nol Hal penting yang perlu diperhatikan dalam menggunakan fungsi sigmoid. Fungsi ini hanya akan menghasilkan nilai positif.



Gambar 2.9: Kurva Sigmoid, dari [5]



Gambar 2.10: Kurva Tangent, dari [5]

Hyperbolic Tangent Jika pada fungsi sigmoid y akan selalu selalu bernilai positif. Dengan menggunakan fungsi Hyperbolic Tangent maka nilai y bisa bernilai negatif. Jika keluaran yang kita ingin dapat bernilai negatif dan positif, kita dapat menggunakan fungsi Hyperbolic Tangent yang di definisikan sebagai berikut:

$$f(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

Kurva 2.10 menggambarkan fungsi Hyperbolic Tangent dimana nilai y dapat bernilai positif atau negatif bergantung pada x yang diberikan.

2.5.7 Backpropagation

Backpropagation adalah salah satu metode untuk melakukan pelatihan pada jaringan feedforward yang memiliki beberapa layer. *Backpropagation* dapat digunakan untuk setiap jaringan feedforward yang menggunakan sebuah fungsi aktivasi yang dapat diturunkan. Pada backpropagation bobot setiap penghubung antar neuron akan dirubah agar JST dapat menghasilkan keluaran yang diharapkan sesedikit mungkin mengalami kesalahan.

2.6 Database Management System

2.6.1 JDBC

JDBC adalah sebuah API yang di rancang untuk menghubungkan basis data SQL dengan pemograman berbahasa Java. JDBC menyediakan metode untuk melakukan *query* dan *updating* data kedalam database. JDBC berorientasi dengan basis data relasional. Berikut beberapa kelas yang dimiliki oleh JDBC:
Connection adalah sebuah *interface* yang berfungsi melakukan koneksi dengan sebuah spesifik basis data. Method yang dimiliki *interface* ini sebagai berikut:

1. **Statement createStatement()**
   Berfungsi membuat sebuah objek Statement untuk mengirimkan SQL statments kedalam basis data.
   Kembalian Statement

2. prepareStatement(String sql) Berfungsi membuat objek PrepareStatement untuk mengirimkan parameter SQL kedalam database.
   Parameter sql - String **Kembalian** sebuah objek PreparedStatement

DriverManager Layanan dasar untuk mengelola satu set JDBC driver. Method yang dimiliki kelas ini sebagai berikut:

1. **Connection getConnection(String url)**
   Berfungsi mendapatkan objek Connection **Parameter** url - String
   Kembalian objek Connection

PreparedStatement Adalah sebuah kelas yang merepresentasikan sebuah *precomplied SQL statment*. Method yang dimiliki kelas ini sebagai berikut:

1. **void Execute()** Berfungsi melakukan eksekusi terhadap query **Kembalian** void

Statement Adalah sebuah *interface* yang berfungsi mengeksekusi perintah SQL statik dan mengembalikan hasil perintah tersebut. Method yang dimiliki *interface* ini sebagai berikut:

1. **ResultSet executeQuery(String sql)**
   Berfungsi mengeksekusi perintah SQL yang diberikan.
   Parameter sql - String **Kembalian** sebuah objek ResultSet

BAB 3

ANALISIS

3.1 Arsitektur Perangkat Lunak

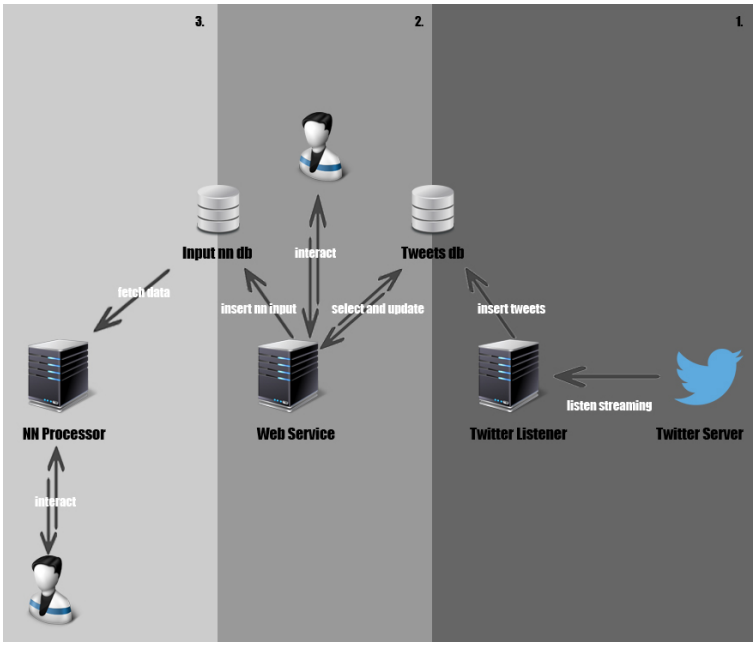
Pada penelitian ilmiah ini akan dibangun sebuah perangkat lunak yang bertujuan membantu penggunanya untuk memprediksi kemacetan di kota Bandung. Perangkat lunak ini terdiri dari 3 lapisan utama: Lapisan pertama berfungsi mendengarkan *tweets* yang telah disaring dan berasal dari Twitter *streaming*. Lapisan kedua adalah pra-proses data agar *tweets* yang telah disaring dirubah menjadi bahan pembelajaran JST yang terstruktur dengan baik, pada lapisan ini dibutuhkan interaksi manusia. Pada lapisan ketiga dalam perangkat lunak ini berfungsi memprediksi kemacetan berdasarkan nama jalan dan waktu. Arsitekturnya dapat dilihat pada gambar 3.1

3.2 REST API atau Streaming API

Pada lapisan pertama arsitektur digunakan untuk menarik *tweets* yang berasal dari Twitter. Untuk menarik *tweets* dari Twitter kedalam perangkat lunak akan digunakan *library* Java bernama Twitter4j, library ini telah membungkus Twitter API agar dapat diimplementasikan kedalam pemograman bahasa Java. Pada bab dasar teori membahas bahwa Twitter API memiliki 2 cara dalam penarikan data yaitu REST API dan Streaming API. Bila kita teliti pada kebutuhan perangkat lunak, diperlukan jumlah data yang tidak sedikit untuk melakukan pemrosesan data pada arsitektur lapisan ketiga, maka harus dipilih jenis Twitter API yang dapat mengambil data secara berkala. Jika kita melihat salah satu API Twitter, REST API memiliki *rate limits* yang membatasi perangkat lunak hanya dapat memanggil 180 perintah setiap 15 menit, karena batasan ini ada kemungkinan *tweets* yang berharga dapat terlewat bahkan terjadi duplikasi *tweets*. Berbeda seperti REST API, Streaming API tidak memiliki batasan untuk melakukan *listen* pada server Twitter, dengan hal ini dapat memudahkan perangkat lunak dalam mengambil *tweets* yang berlalu lalang di Server Twitter. Oleh karena itu perangkat lunak akan menggunakan Streaming API yang telah dibungkus oleh Twitter4j.

3.3 Jenis Jaringan Syaraf Tiruan

Tujuan penelitian ini adalah membangun perangkat lunak yang dapat memprediksi kemacetan. Jaringan Syaraf Tiruan adalah salah satu teknik yang tepat untuk memprediksi suatu kejadian berdasarkan kejadian-kejadian yang terjadi sebelumnya. Salah satu jenis JST yang popular dan sering digunakan untuk memprediksi ialah *Feed Forward*. Untuk menggunakan JST *Feed Foward*



Gambar 3.1: Arsitektur Perangkat Lunak

ada beberapa hal yang perlu ditentukan agar JST *Feed Foward* dapat bekerja secara optimal, hal yang perlu ditentukan antar lain :

1. Jumlah Hidden Layer dan Neuron setiap *layer*
2. Fungsi Aktivasi
3. Metode Pembelajaran JST

3.3.1 Jumlah Hidden Layer dan Neuron setiap *layer*

Pada JST *Feed Forward* terdiri 3 jenis *layer*, *input layer*, *hidden layer*, *output layer*. Setiap *layer* terdiri dari neuron-neuron. Jumlah neuron ini perlu ditentukan,hal ini bergantung pada masalah dan keperluan JST. Dalam sub-bab ini peneliti akan menganalisis jumlah neuron setiap *layer*-nya berdasarkan teknik yang telah dipaparkan pada bab-2.
Sebelum menentukan jumlah neuron setiap *layer* pertama yang akan kita akan menentukan jumlah *hidden layer*. Jumlah *hidden layer* akan menentukan fungsi dari JST itu sendiri. Kita dapat melihat kembali pada tabel 2.1 untuk menentukan jumlah hidden layer berdasarkan kegunaanya. Dapat hal memprediksi kemacetan dapat dimodelkan sebagai fungsi yang mengandung pemetaan kontinu. Oleh karena itu dengan hanya menggunakan satu buah hidden layer sudah cukup agar JST dapat memodelkan fungsi prediksi kemacetan.

Setelah menentukan jumlah *hidden layer* kita akan menentukan jumlah neuron dalam *input neuron*. Jumlah neuron pada *input layer* bergantung pada data yang kita miliki, dan parameter apa saja yang diperlukan untuk melatih JST. Jika kita meninjau struktur data pelatihan terdiri dari hari, jam, nama lokasi, dan status kemacetan. Oleh karena itu jumlah neuron dalam *hidden layer* akan berjumlah 4 buah.

Sama seperti *input layer*, *output layer* ditentukan berdasarkan kebutuhan. Karena *output layer* berfungsi menentukan output tersebut berstatus macet atau tidak, makan jumlah neuron dalam output layer akan memiliki 2 buah neuron. Dimana setiap neuron akan merepresentasikan status dari kemacetan.

Berbeda dengan menentukan *input* dan textitoutput layer tidak ada cara baku untuk menentukan jumlah *hidden layer* namun ada teknik-teknik yang membantu kita menentukan jumlah *hidden layer*. Pada bab-2 telah dipaparkan jumlah neuron harus berada diantara jumlah masukan dan keluaran. Jumlah neuron seharusnya 2/3 ukuran dari layer masukan, ditambah dengan layer keluaran. Ketiga jumlah neuron seharusnya kurang dari dua kali dari layer masukan. Bila menggunakan teknik tersebut jumlah neuron pada *hidden layer* berjumlah 4 neuron.

3.3.2 Fungsi Aktivasi

Dalam bab-2 dipaparkan bahwa fungsi aktivasi dapat kita buat dan tentukan berdasarkan kebutuhan. Namun keluaran yang dihasilkan JST hanya akan menentukan apakah prediksi tersebut macet atau tidak, *range* angka nol sampai satu sudah dapat merepresentasikan apakah keluaran bernilai macet atau tidak. Karena *range*nya bernilai nol sampai satu, kita dapat menggunakan fungsi aktivasi yang sering digunakan yaitu sigmoid. Fungsi aktivasi sigmoid hanya akan menghasilkan nilai nol sampai dengan satu dapat kita lihat pada gambar 2.9 pemetaan fungsi sigmoid.

3.3.3 Metode Pembelajaran

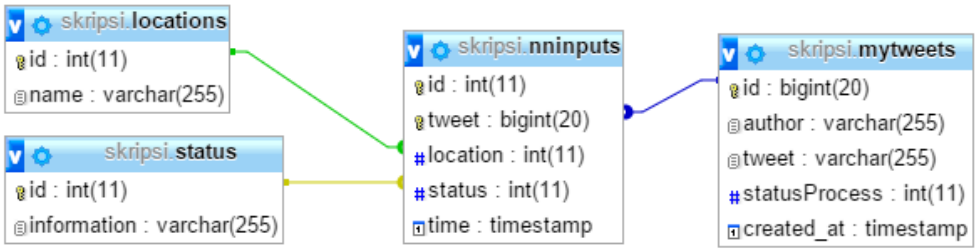
3.4 Pengumpulan data pelatihan JST

Agar JST dapat memprediksi kemacetan dengan tepat, maka JST perlu dilatih dilakukan pembelajaran. Karena JST perlu belajar maka dibutuhkanya kumpulan data pelatihan. Data yang digunakan untuk memprediksi ialah data kejadian yang pernah terjadi di masa lalu. Data pelatihan ialah *tweets* yang bersumber dari media sosial Twitter yang akan diambil oleh sistem secara kontinu. *Tweets* yang akan dijadikan metode pelatihan akan disaring berdasarkan *query* tertentu. Kriteria dibawah akan menjadi penyaring jenis *tweets* apa yang akan diambil sebagai pra-data pelatihan.

1. Pemilihan Nama Jalan
2. Pemilihan Sumber *tweets*
3. Bahasa

3.4.1 Merubah tweets menjadi input JST

Setelah kita melakukan penyaringan terhadap *tweets*, agar menjadi data pelatihan bagi JST kita harus melakukan pra-proses agar *tweets* menjadi masukan yang dapat diterima bagi JST. Cara yang kita gunakan memprosesnya dengan cara manual yang menggunakan manusia untuk memparsing *tweets* menjadi input JST. Pertama manusia akan memilih apakah *tweets* memenuhi kriteria sebagai



Gambar 3.2: Tabel Relational

input JST. Kedua *tweets* yang memenuhi kriteria akan di pra-proses informasi apa yang terdapat dari *tweets*, jalan apa subjectnya dan keterangan dari kondisi jalan.

3.5 Rancangan Basis Data

Perangkat lunak ini memerlukan sebuah tempat penyimpanan. Salah satu tempat penyimpanan yang populer dan mudah diimplementasikan kedalam berbagai bahasa ialah mysql. Mysql ialah salah satu basis data yang menggunakan sql sebagai bahasa untuk melakukan perintah. Perangkat lunak akan menyimpan *tweets* yang disaring dan juga menyimpan data pelatihan yang sudah dilakukan pra-proses. Rancangan basis data yang dibuat dapat dilihat pada gambar 3.2

DAFTAR REFERENSI

[1] Semiocast, "Top 20 cities by number of posted tweets," June 2012.

[2] Twitter, "The Streaming APIs," September 2015.

[3] Twitter, "The REST APIs," September 2015.

[4] E. D. Hardt, "The OAuth 2.0 Authorization Framework," October 2012.

[5] J. Heaton, *Introduction to Neural Networks for Java 2nd Edition.* 2008.

[6] Dinas Perhubungan Kota Bandung, "Jumlah Kendaraan Bermotor (Umum Dan Tidak Umum)," September 2014.

[7] Dinas Perhubungan Kota Bandung, "Panjang Jalan Menurut Status di Wilayah Kota Bandung," September 2014.

[8] Twitter, "Getting started with Twitter," September 2015.

[9] Twitter, "FAQ Twitter," September 2015.

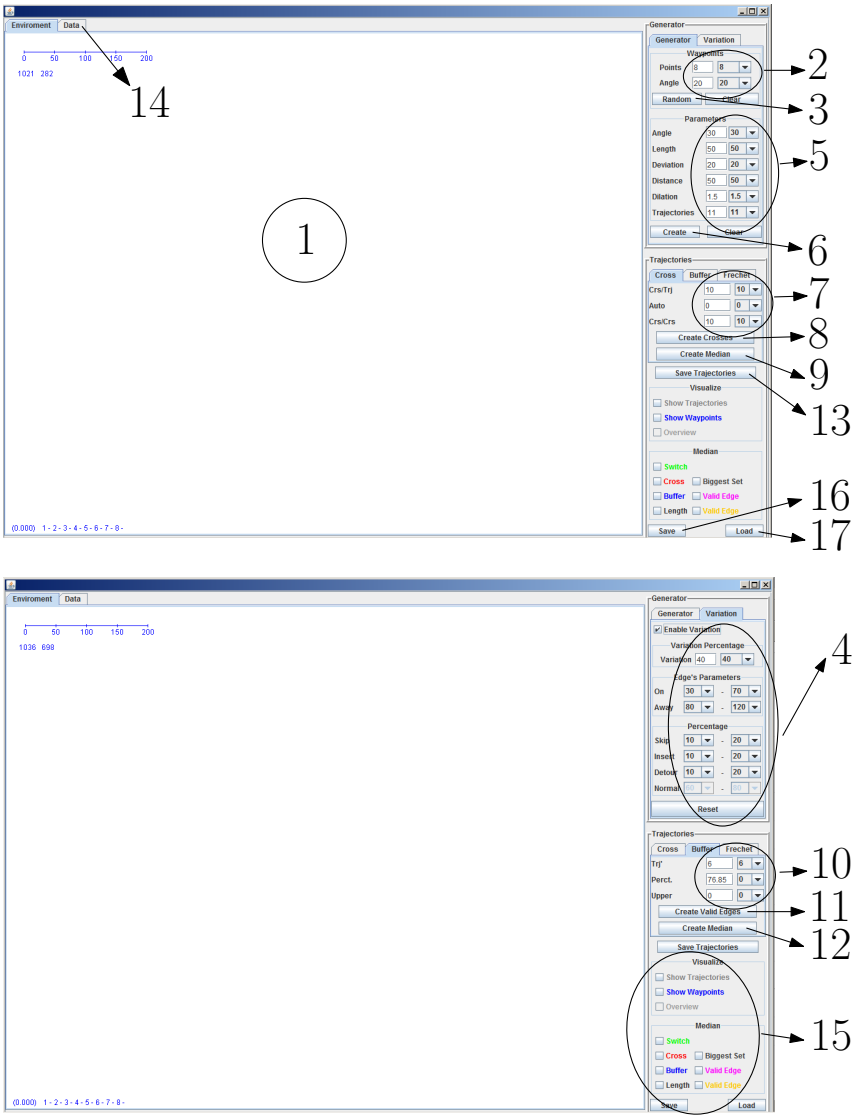
[10] Twitter, "Twitter4j," September 2015.

[11] C. Stergiou, "What is a Neural Network?," September 2015.

LAMPIRAN A

THE PROGRAM

The interface of the program is shown in Figure A.1:



Gambar A.1: Interface of the program

Step by step to compute the median trajectory using the program:

1. Create several waypoints. Click anywhere in the "Environment" area(1) or create them automatically by setting the parameters for waypoint(2) or clicking the button "Random"(3).

2. The "Variation" tab could be used to create variations by providing values needed to make them(4).

3. Create a set of trajectories by setting all parameters(5) and clicking the button "Create"(6).

4. Compute the median using the homotopic algorithm:

   - Define all parameters needed for the homotopic algorithm(7).
   - Create crosses by clicking the "Create Crosses" button(8).
   - Compute the median by clicking the "Compute Median" button(9).

5. Compute the median using the switching method and the buffer algorithm:

   - Define all parameters needed for the buffer algorithm(10).
   - Create valid edges by clicking the "Create Valid Edges"button(11).
   - Compute the median by clicking the "Compute Median"button(12).

6. Save the resulting median by clicking the "Save Trajectories" button(13). The result is saved in the computer memory and can be seen in "Data" tab(14)

7. The set of trajectories and its median trajectories will appear in the "Environment" area(1) and the user can change what to display by selecting various choices in "Visualize" and "Median" area(15).

8. To save all data to the disk, click the "Save"(16) button. A file dialog menu will appear.

9. To load data from the disk, click the "Load"(17) button.

LAMPIRAN B

THE SOURCE CODE

Listing B.1: MyFurSet.java

```
1
2  import java.util.ArrayList;
3  import java.util.Collections;
4  import java.util.HashSet;
5
6  /**
7   *
8   * @author Lionov
9   */
10
11 //class for set of vertices close to furthest edge
12 public class MyFurSet {
13     protected int id;                                    //id of the set
14     protected MyEdge FurthestEdge;                       //the furthest edge
15     protected HashSet<MyVertex> set;                     //set of vertices close to furthest edge
16     protected ArrayList<ArrayList<Integer>> ordered;     //list of all vertices in the set for each
           trajectory
17     protected ArrayList<Integer> closeID;                //store the ID of all vertices
18     protected ArrayList<Double> closeDist;               //store the distance of all vertices
19     protected int totaltrj;                              //total trajectories in the set
20
21     /**
22      * Constructor
23      * @param id : id of the set
24      * @param totaltrj : total number of trajectories in the set
25      * @param FurthestEdge : the furthest edge
26      */
27     public MyFurSet(int id,int totaltrj,MyEdge FurthestEdge) {
28         this.id = id;
29         this.totaltrj = totaltrj;
30         this.FurthestEdge = FurthestEdge;
31         set = new HashSet<MyVertex>();
32         ordered = new ArrayList<ArrayList<Integer>>();
33         for (int i=0;i<totaltrj;i++) ordered.add(new ArrayList<Integer>());
34         closeID = new ArrayList<Integer>(totaltrj);
35         closeDist = new ArrayList<Double>(totaltrj);
36         for (int i = 0;i <totaltrj;i++) {
37             closeID.add(-1);
38             closeDist.add(Double.MAX_VALUE);
39         }
40     }
41
42     /**
43      * set a vertex into the set
44      * @param v : vertex to be added to the set
45      */
46     public void add(MyVertex v) {
47         set.add(v);
48     }
49
50     /**
51      * check whether vertex v is a member of the set
52      * @param v : vertex to be checked
53      * @return true if v is a member of the set, false otherwise
54      */
55     public boolean contains(MyVertex v) {
56         return this.set.contains(v);
57     }
58 }
```