IMPLEMENTASI *OPTICAL BACKPROPAGATION NEURAL NETWORK* UNTUK MENENTUKAN KUAT TEKAN DAN *SETTING TIME* BETON PADA KASUS *SELF COMPACTING CONCRETE*

SKRIPSI

oleh :

MOCH. LUTFI

0610960044-96

PROGRAM STUDI ILMU KOMPUTER

JURUSAN MATEMATIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS BRAWIJAYA

MALANG

2011


UNIVERSITAS BRAWIJAYA

IMPLEMENTASI *OPTICAL BACKPROPAGATION NEURAL NETWORK* UNTUK MENENTUKAN KUAT TEKAN DAN *SETTING TIME* BETON PADA KASUS *SELF COMPACTING CONCRETE*

SKRIPSI

Sebagai salah satu syarat untuk memperoleh gelar sarjana dalam bidang ilmu komputer

oleh :
MOCH. LUTFI
0610960044-96

PROGRAM STUDI ILMU KOMPUTER
JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS BRAWIJAYA
MALANG

2011


UNIVERSITAS BRAWIJAYA

LEMBAR PENGESAHAN

IMPLEMENTASI *OPTICAL BACKPROPAGATION NEURAL NETWORK* UNTUK MENENTUKAN KUAT TEKAN DAN *SETTING TIME* BETON PADA KASUS *SELF COMPACTING CONCRETE*

Oleh:
MOCH. LUTFI
0610960044-96

Setelah dipertahankan di depan Majelis Penguji
Pada tanggal 1 Februari 2011
Dan dinyatakan memenuhi syarat untuk memperoleh gelar Sarjana dalam bidang ilmu komputer

Pembimbing I	Pembimbing II
Candra Dewi, S.Kom.,MSc.	**Drs. Marji, MT**
NIP : 197711142003122001	**NIP: 196708011992031001**

Mengetahui,
Ketua jurusan matematika
Fakultas MIPA Universitas Brawijaya

Dr. Abdul Rouf Algofari, M.Sc.
NIP : 196709071992203001


UNIVERSITAS BRAWIJAYA

LEMBAR PERNYATAAN

Saya yang bertanda tangan di bawah ini :

Nama	**: Moch.Lutfi**
NIM	**: 0610960044**
Jurusan	**: Ilmu Komputer**
Penulis Skripsi berjudul	**: Implementasi *Optical Backpropagation Neural Network* Untuk Menentukan Kuat Tekan dan *Setting Time* Beton Pada Kasus *Self Compacting Concrete***

Dengan ini menyatakan bahwa :

1. **Isi dari Skripsi yang saya buat adalah benar-benar karya sendiri dan tidak menjiplak karya orang lain, selain nama-nama yang termaktub di isi dan tertulis di daftar pustaka dalam skripsi ini.**
2. **Apabila dikemudian hari ternyata Skripsi yang saya tulis terbukti hasil jiplakan, maka saya akan bersedia menanggung segala resiko yang akan saya terima.**

Demikian pernyataan ini dibuat dengan segala kesadaran.

Malang, 1 Februari 2011
Yang menyatakan,

(Moch. Lutfi)
NIM. 0610960044

IMPLEMENTASI *OPTICAL BACKPROPAGATION NEURAL NETWORK* UNTUK MENENTUKAN KUAT TEKAN DAN *SETTING TIME* BETON PADA KASUS *SELF COMPACTING CONCRETE*

ABSTRAK

Self Compacting Concrete (SCC) merupakan alternatif dari beton konvensional yang cenderung lebih mahal. Belum adanya formula khusus untuk menentukan campuran yang memiliki kuat tekan dan *setting time* tertentu sehingga untuk menentukan kuat tekan dan *setting time* diperlukan uji laboratorium yang mahal. Untuk mengatasi hal itu digunakanlah jaringan syaraf tiruan (JST) propagasi balik untuk menentukan kuat tekan dan *setting time* dari data yang sudah diuji di laboratorium. Namun JST propagasi balik masih lambat dan masih bisa terjebak pada solusi lokal. Salah satu solusinya yaitu *Optical Backpropagation Neural Network* (OBPNN). Penelitian dilakukan untuk mengimplementasikan OBPNN untuk menentukan kuat tekan dan *setting time* beton SCC serta untuk mengetahui pengaruh jumlah *hidden neuron* dan *learning rate* terhadap keakuratan dan kecepatan *training* OBPNN. Pada penelitian ini pengujian dilakukan dengan satu *hidden layer* dengan jumlah *hidden neuron* berkisar 6 sampai 12 dan *learning rate* 0.1 sampai 1. Hasil *training error* yang mampu dicapai yaitu 2,5% dengan lama *training* 1.8 detik. Selain itu hasil validasi dengan dataset yang berbeda diperoleh *error* sebesar 5,8%. Hal ini menunjukkan bahwa OBPNN mampu menghasilkan *error* yang cukup kecil dalam waktu yang singkat.

Kata kunci: *hidden neuron, learning rate, Optical Backpropagation Neural Network, kuat tekan, setting time*

IMPLEMENTATION OF OPTICAL BACKPROPAGATION NEURAL NETWORK FOR DETERMINING COMPRESSIVE CONCRETE STRENGTH AND SETTING TIME: CASE OF SELF COMPACTING CONCRETE

ABSTRACT

Self-compacting Concrete (SCC) is an alternative of conventional concrete which more expensive. There has been no specific mixture formula for determining compressive strength and setting time of concrete. Laboratory tests must be conducted to determine compressive strength and setting time of concrete for each mixture but that was expensive. Back propagation artificial neural network (ANN) can be used to determine the compressive strength and setting time of data that has been tested in the laboratory. Optical Backpropagation Neural Network (OBPNN) was introduced to overcome the weakness of back propagation neural network. The purposes of this research are applying OBPNN for detemining compressive concrete strength and setting time. In addition to know the effect of learning rate and number of hidden neurons for the accuracy and speed of training process. In this research, testing was done with one hidden layer and learning rate 0.1 to 1. The training error that can be achieved is 2.5% with 1.8 seconds long training. In addition, the validation results with other data obtained by 5.8% error. This shows that OBPNN able to generate small error in a short time.

Keyword: *hidden neuron, learning rate, Optical Backpropagation Neural Network, compressive strength, setting time*


UNIVERSITAS BRAWIJAYA

KATA PENGANTAR

Alhamdulillahi rabbil 'alamin. Puji syukur penulis panjatkan kehadirat Allah SWT yang telah melimpahkan rahmat dan hidayah-Nya, sehingga Skripsi yang berjudul "*Implementasi Optical Backpropagation Neural Network Untuk Menentukan Kuat Tekan Dan Setting Time Beton Pada Kasus Self Compacting Concrete*" dapat diselesaikan.

Skripsi ini disusun dan diajukan sebagai syarat untuk memperoleh gelar sarjana pada Program Studi Ilmu Komputer, Jurusan Matematika, Fakultas MIPA, Universitas Brawijaya.

Banyak pihak yang berperan atas terselesaikannya penelitian dan penulisan skripsi ini. Atas bantuan yang telah diberikan, penulis ingin menyampaikan penghargaan dan ucapan terima kasih yang sedalam-dalamnya kepada :

1. Candra Dewi,S.Kom.,MSc. selaku dosen pembimbing 1 dan Drs. Marji MT., selaku dosen pembimbing 2, yang selalu meluangkan waktunya untuk memberikan bimbingan dan saran demi tercapainya keutuhan pada penyusunan tugas akhir ini.
2. Dr. Abdul Rouf Algofari, M.Sc., selaku ketua jurusan Matematika FMIPA Universitas Brawijaya Malang
3. Drs. Marji, MT., selaku Ketua Program Studi Ilmu Komputer Universitas Brawijaya Malang.
4. Segenap bapak ibu dosen yang telah mendidik, dan mengamalkan ilmunya kepada penulis.
5. Segenap staf dan karyawan di Jurusan Matematika FMIPA Universitas Brawijaya, yang telah banyak membantu penulis dalam pelaksanaan penyusunan tugas akhir ini.
6. Pak Akhmad Suryadi, BS, MT., yang memberikan data uji pada tugas akhir ini.
7. Jeff Heaton, PhD., yang selalu memberikan inspirasi dan jawaban-jawaban atas pertanyaan saya mengenai JST.
8. Keluarga penulis, Bapak, Ibu dan Adik Malik yang selalu setia mendukung, mendo'akan dan memberikan semangat. Sekaligus sebagai contoh, bahwa tidak ada keberhasilan yang tercapai tanpa perjuangan dan Do'a.

9. Teman-teman seperjuangan di Program Studi Ilmu Komputer FMIPA Universitas Brawijaya yang slalu menemani, memberikan semangat dan bantuannya demi kelancaran pelaksanaan penyusunan tugas akhir ini.
10. Teman-teman MOST yang selalu mendukung dan menyemangati saya.
11. Semua pihak lain yang telah membantu terselesaikannya skripsi ini yang tidak bisa penulis sebutkan satu-persatu

Penulis menyadari bahwa skripsi ini masih jauh dari kesempurnaan, dan memiliki banyak kekurangan, sehingga dengan segala kerendahan hati, penulis mengharapkan kritik dan saran yang membangun dari pembaca. Semoga penulisan skripsi ini bermanfaat bagi pembaca.

Malang, 26 Januari 2011

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR LAMPIRAN

UNIVERSITAS BRAWIJAYA

BAB I
PENDAHULUAN

1.1 Latar Belakang Masalah

Dalam teknologi beton konvensional, campuran materialnya terdiri dari semen, pasir, kerikil, dan air. Campuran cenderung kaku, selama ini sering kali menimbulkan permasalahan. Diantaranya beton tidak dapat dipadatkan dengan baik, permukaan beton yang dihasilkan kasar, sehingga diperlukan pekerjaan tambahan untuk *finishing* permukaan beton supaya lebih halus. (Hajime & Ouchi, 2003)

Self-Compacting concrete (SCC) adalah teknologi baru dalam ilmu beton yang dapat mengatasi permasalahan yang ada pada beton konvensional (Hajime & Ouchi, 2003). SCC merupakan jenis beton yang dapat mengalir akibat beratnya sendiri, sehingga campuran beton SCC dapat masuk ke dalam setiap sudut cetakan, tanpa menggunakan alat penggetar/ *vibrator*, tidak memerlukan banyak tenaga manusia, dan memiliki permukaan yang halus (Esping, 2007).Jumlah tenaga kerja sedikit, sehingga teknologi SCC dapat mengefisienkan biaya sebesar 7.5% dibandingkan dengan teknologi beton konvensional (Ouchi, 2003).

Akan tetapi SCC juga mengalami permasalahan tentang belum adanya standard baku yang mengatur tentang komposisi campuran beton SCC (Efca, 2005). Selama ini dalam menentukan komposisi SCC yang tepat sesuai dengan rencana kekuatan beton yang diinginkan, biasanya dilakukan dengan cara *trial and error*. Tentunya dengan metode ini kecenderungan banyak material yang terbuang, mahal, dan memakan waktu lama untuk menyelesaikan suatu komposisi campuran beton yang tepat.

Salah satu metode yang dapat digunakan untuk mengatasi permasalahan tentang pengaturan campuran pada beton SCC dapat digunakan *Artificial Neural Networks*/ANN (Jaringan Syaraf Tiruan/JST). Penggunaan JST dikarenakan metode konvensional(pendekatan statistika atau program linear) tidak mampu untuk mengatasi permasalahan penentuan nilai dengan jumlah peubah yang cukup banyak apalagi jika datanya tidak lengkap. Selain itu JST ini sudah sering digunakan dan diaplikasikan pada bidang rekayasa industri termasuk dalam bidang Teknik Sipil. Ada keuntungan besar yang dapat diperoleh bila menggunakan

teknik JST dalam menentukan komposisi campuran, baik dipandang dari sudut teori maupun aplikasinya. JST dapat digunakan untuk identifikasi, prediksi, klasifikasi, penyesuaian pola, pengenalan pola, optimisasi dan kontrol permasalahan yang terkait. Dalam bidang Teknik Sipil penggunaan JST biasanya digunakan untuk memprediksi perilaku suatu data berdasarkan hasil ekspreimental yang digunakan untuk melatih, menguji, memverifikasi dan menvalidasi data (Yeh, 1998).

Pada penelitian (Altin, 2008) menjelaskan bahwa, dengan menggunakan JST *backpropagation* diperoleh hasil yang memuaskan dan tingkat ketelitian yang tinggi(Altin, dkk, 2008). *Backpropagation Neural Network* (BPNN) adalah jenis JST *supervised* yang memerlukan data awal sebagai acuan pembelajaran JST. BPNN merupakan pendekatan terbaik dalam memecahkan permasalahan dimana data *input* dan *output* yang besar serta tidak diketahuinya hubungan antara *input* dan *output*. Selain itu hubungan *input* dan *output* yang tidak dapat diketahui dengan metode algoritma tradisional seperti pendekatan statistik atau program linear biasa. Selain itu dengan BPNN mampu menghasilkan solusi dengan ketelitian yang tinggi. (Orr, 1999)

Namun hasil ketelitian yang tinggi tersebut diperoleh dengan konvergensi sangat lambat dalam pembelajaran sehingga proses pembelajaran menjadi lama selain itu juga terdapat permasalahan lain yaitu nilai yang dihasilkan kemungkinan terjebak di lokal minima sehingga rawan terjadi kesalahan solusi yang fatal jika hal itu terjadi.(Kasabov, 1998). Menurut (Otair & Salameh, 2005) solusinya adalah dengan menggunakan perbaikan BPNN yaitu *optical* BPNN (OBPNN). Karena itulah penulis mengambil judul "**Implementasi *Optical Backpropagation Neural Network* Untuk Menentukan Kuat Tekan dan *Setting Time* Beton Pada Kasus *Self Compacting Concrete***".

1.2 Rumusan Masalah

1. Bagaimana mengimplementasikan OBPNN untuk menentukan kuat tekan dan *setting time beton*?
2. Bagaimana pengaruh jumlah neuron dan *learning rate* pada keakuratan dan kecepatan proses OBPNN?

1.3 Tujuan

Tujuan penelitian yang ingin dicapai adalah:

1. Mengimplementasikan *optical backpropagation neural network* untuk menentukan kuat tekan dan *setting time* pada beton SCC.
2. Mengetahui pengaruh jumlah neuron dan *learning rate* pada keakuratan dan kecepatan proses dari *optical backpropagation neural network*.

1.4 Batasan Masalah

1. Arsitektur *Neural network* yang digunakan hanya menggunakan satu *hidden layer*.
2. Tidak dilakukan perbandingan dengan metode lain.

1.5 Manfaat Penelititan

1. Hasil penelitian pada skripsi ini dapat digunakan untuk menetukan kuat tekan dan *setting time* beton tanpa melakukan pengujian kuat tekan dan *setting time* di laboratorium.
2. Dapat diketahui kombinasi jumlah *neuron* dan *learning rate* yang terbaik untuk OBPNN pada kasus SCC.
3. Dari keakuratan dan kecepatan proses dapat diketahui kelayakan sistem ini.

1.6 Metode Penelitian

Metode penelitian yang dilakukan pada penelitian ini adalah:

1. Studi Literatur
   Mempelajari literatur baik berupa buku(*textbook*), jurnal dan artikel ilmiah, maupun website yang berhubungan dengan Jaringan Syaraf Tiruan dan juga referensi mengenai teknologi beton.
2. Pendefinisian dan analisis masalah
   Mendefinisikan dan menganalisis masalah untuk mencari solusi yang optimal.
3. Perancangan dan implementasi system
   Membuat perancangan perangkat luinak dengan analisis terstruktur dan mengimplementasikan hasil rancangan tersebut.
4. Uji coba dan analisa hasil implementasi.
   Menguji perangkat lunak, dan menganlisa hasil dari implementasi tersebut apakah sudah sesuai dengan tujuan yang dirumuskan sebelumnya, dan selanjutnya dievaluasi dan disempurnakan.

1.7 Sistematika Penulisan

Sistematika penulisan tugas akhir ini dibagi menjadi lima bab dengan masing-masing bab diuraikan sebagai berikut:

1. **BAB I PENDAHULUAN**

   Berisi latar belakang penelitian, perumusan masalah, batasan masalah, tujuan penelitian, manfaat, metode penelitian, dan sistematika penulisan.

2. **BAB II TINJAUAN PUSTAKA**

   Bab ini berisi teori-teori dari berbagai pustaka yang menunjang penelitian dalam penulisan skripsi. Adapun teori yang tercakup dalam bab ini yaitu mengenai beton SCC, definisi dan konsep jaringan syaraf tiruan, metode *backpropagation,* dan *optical backpropagation neural network*.

3. **BAB III METODOLOGI DAN PERANCANGAN SISTEM**

   Bab ini berisi mengenai perancangan perangkat lunak yang dibangun, meliputi analisa sistem, perancangan sistem, perancangan tabel dan perancangan uji coba.

4. **BAB IV IMPLEMENTASI DAN PEMBAHASAN**

   Bab ini berisi hasil dari implementasi perangkat lunak yang digunakan untuk menentukan kuat tekan dan *setting time* pada beton SCC, pembahasan analisa hasil uji coba dan evaluasi hasil uji coba.

5. **BAB V KESIMPULAN DAN SARAN**

   Bab ini memuat kesimpulan dari hasil penelitian dan saran-saran untuk pengembangan penelitian selanjutnya.

BAB II
TINJAUAN PUSTAKA

2.1 Perkembangan Teknologi Beton

Teknologi beton pada dunia industri konstruksi semakin hari semakin banyak digunakan untuk semua jenis bangunan, karena dari sifat beton yang mudah dibentuk sesuai dengan keinginan, bahan campuran beton mudah didapat, mempunyai kekuatan yang tinggi terutama kekuatan tekannya dan mempunyai durabilitas yang baik, sehingga teknologi beton terus dikembangkan oleh para ahli guna mendapatkan performa beton yang optimal dengan menekan biaya semurah-murahnya. Perkembangan teknologi beton yang ada saat ini terfokus pada penggunaan material beton dan metode pelaksanaan di lapangan.

Teknologi beton yang sudah biasa dan sering dikerjakan selama ini adalah teknologi beton konvensional. Akan tetapi dalam perkembangannya teknologi beton konvensional ini sering mengalami kendala-kendala saat pelaksanaan di lapangan, khususnya saat pengecoran. Kendala tersebut antara lain : penggunaan *vibrator* yang tidak optimal, karena posisi tulangan sangat rapat, posisi dan lokasi pengecoran yang tidak terjangkau oleh alat *vibrator*, karena konstruksi yang terlalu tinggi/dalam dan tipis, konstruksi yang mempunyai bentuk yang unik, sudut kemiringan yang tajam dan yang memerlukan permukaan yang halus (Ouchi, 2003).

Kendala-kendala yang ada pada teknologi dan metode beton konvensional ini dapat dieliminasi dengan mengembangkan teknologi beton yaitu teknologi beton yang dapat mengalir, masuk pada setiap sudut cetakan dan memiliki permukaan yang halus serta tanpa menggunakan alat getar/*vibrator*. Teknologi beton jenis ini disebut *Self-compacting concrete* (SCC) yaitu beton yang tidak memerlukan penggetaran (*vibrator*) untuk penempatan dan pemadatannya karena kemampuanya untuk mengalir akibat berat sendiri dari campuran beton tersebut (Efca, 2005).

SCC merupakan teknologi beton yang baru di gunakan di Negara-negara maju, seperti Jepang, Eropa dan Amerika Serikat (Ouchi, 2003). SCC, dikembangkan dan digunakan pertama kali di Jepang sejak tahun 1989 (Hajime & Ouchi, 2003), dalam rangka mendapatkan struktur beton yang memiliki durabilitas yang tinggi

dan memudahkan untuk menuangkan campuran beton kedalam cetakan. Penggunaan SCC di negara Jepang ini terus berkembang, hingga tahun 2000 saja penggunaan SCC baik untuk konstruksi *precast* maupun berupa *ready mix (cast-in-place)* sudah mencapai 400.000 m^3 (Ouchi, 2003).

SCC adalah campuran yang sensitif, yang sangat tergantung kepada material beton, yaitu semen, pasir, kerikil dan air. Campuran SCC harus memenuhi dua kriteria yang saling bertolak belakang yang harus tercapai pada saat yang bersamaan, yaitu SCC harus mempunyai sifat kemampuan mengalir tinggi dan harus bebas dari segregasi. Ini dapat tercapai dengan mengkombinasikan *admixture superplasticizer* dengan tambahan bahan halus dalam campuran SCC (Hajime & Ouchi, 2003). Mekanisme utama dalam mengontrol bahan halus ini sangat tergantung pada sifat fisik dan kimianya, akan tetapi campuran SCC sangat tergantung dari penggunaan *admixture* dan luasnya permukaan akibat bahan halus (Eric & David, 2007).

2.2 Kriteria Desain Campuran SCC

Seperti halnya beton pada umumnya, formula *SCC* sangat tergantung pada karakteristik material (semen, pasir, kerikil, air dan *admixture*) yang akan digunakan. *Mix* Desain SCC harus memenuhi tiga kriteria kunci (Ouchi, 2003) sebagai berikut:

1. Mempunyai kemampuan untuk mengalir dan mengisi semua ruang cetakan secara baik.
2. Mempunyai kemampuan untuk mengalir dan melalui tulangan yang rapat.
3. Campuran SCC mempunyai ketahanan tinggi terhadap segregasi agregat(pemisahan material kasar dan halus).

Dalam merancang campuran beton SCC ada beberapa hal yang harus diperhatikan. Pertama adalah penggunaan tipe bahan *cementitious* (*powder type*) yang akan digunakan dalam campuran SCC, seperti *silica fume*, *fly ash* atau *copper slag*. Kedua adalah pemilihan dan penggunaan bahan aditif (*Viscosity-modifying admixture*) yang tepat dalam campuran SCC baik tipe maupun jumlah yang akan digunakan. Ketiga adalah kombinasi dari kedua unsur diatas (Baldino, Frontera, Gabriele, & Candamano, 2007)

Banyak perusahaan *ready-mixed*, *precast*, institusi, akademisi dan kontraktor yang membuat *mix design* beton SCC sendiri (Efca, 2005). Ini menunjukkan, bahwa dalam merancang campuran beton SCC antara satu dengan lainnya dapat berbeda, sangat tergantung

dari beberapa hal, seperti : kondisi konstruksi, kualitas material yang akan digunakan, posisi elemen struktur, kondisi lapangan, dsb. Pada dasarnya material yang digunakan dalam campuran SCC meliputi : semen, agregat, air, bahan tambahan, dan *admixture* (lihat tabel 2.1) (ASTM C 33, 2003).

Tabel 2.1 Material Beton

MATERIAL	JENIS MATERIAL
Semen	• Tipe Semen • Kelas Kekuatan Semen
Agregat	• Agregat normal, ringan, dan berat • Natural/sintetis : pasir alam dan pasir *crusher* • Struktur Partikel, kurva
Air	Batasi jumlah material dalam air yang dapat merusak beton
Material Tambahan	• *Fly ash, Trass, silica fume* • Abu batu • *Fiber* (besi/baja, sintetis, gelas) • *Pigment*
Admixture	• *Plasticizer, super plasticizer* • *Accelerator*, *Retarder*, dll.

Berikut ini akan diberikan beberapa contoh *mix design* yang telah digunakan pada pelaksanaan proyek di lapangan. Bagaimana *mix design* satu dengan lainnya berbeda yang sangat tergantung dari kondisi masing-masing proyek dan nampak pula perbedaan *mix design* pada teknologi SCC dangan teknologi beton konvensional. Perbedaan yang menyolok antara teknologi SCC dan teknologi beton konvensional yaitu penggunaan bahan aditif *super plasticizer*.

2.3 Setting time

Setting time beton adalah transisi perubahan secara gradual dari beton cair menjadi beton keras. Waktu *setting* awal beton dapat memberi pengaruh pada performa beton keras (Schindler, 2003). *Setting* akhir beton erat kaitannya dengan terjadinya tegangan dan kekakuan beton mulai saat pertama beton segar dituangkan. Perubahan *gradient* panas pada waktu *setting* akhir dapat digunakan membuat pemodelan untuk memprediksi waktu *setting* beton dan

meprediksi perkembangan temperatur beton. Semakin panas udara lingkungan, akan semakin memperpendek waktu *setting* awalnya dibandingkan dengan pada temperatur normal (Schindler, 2003).

2.4 Kuat Tekan Beton

Kuat tekan beton adalah tekanan maksimum yang mampu ditahan oleh beton. Untuk memperoleh nilai kuat tekan tersebut dilakukan pengujian di laboratorium. Sebelum pengujian kuat tekan dilakukan, silinder beton di-*capping* terlebih dahulu untuk memperoleh permukaan yang rata dan baik. Benda uji silinder dibebani tekan *uniaxial* hingga benda uji hancur (*failure*) dengan kecepatan pembebanan (*loading rate*) antara 0.14 sampai 0.34 MPa/detik. Pengujian kuat tekan ini dilakukan pada umur 1, 7, 21 28, 56, dan 90 hari. Pada umumnya campuran beton yang menggunakan teknologi SCC mempunyai kuat tekan yang lebih baik dibandingkan dengan campuran beton yang konvensional.

Kuat tekan beton pada umur tertentu dapat dihitung dengan persamaan 2.1.

$$= \frac{}{} \qquad (2.1)$$

dimana: f_c' = kuat tekan beton pada umur tertentu (Mpa)

P = beban tekan maksimum (N)

A = luas penampang benda uji (mm^2)

2.5 Jaringan Syaraf Tiruan

Jaringan syaraf tiruan (JST) atau *neural* network adalah suatu metode komputasi yang meniru sistem jaringan syaraf biologis. Metode ini menggunakan elemen perhitungan non-linier dasar yang disebut *neuron* yang diorganisasikan sebagai jaringan yang saling berhubungan, sehingga mirip dengan jaringan syaraf manusia. Jaringan syaraf tiruan dibentuk untuk memecahkan suatu masalah tertentu seperti pengenalan pola atau klasifikasi karena proses pembelajaran.

Layaknya *neuron* biologi, JST juga merupakan sistem yang bersifat *fault tolerant* dalam 2 hal. Pertama, dapat mengenali sinyal *input* yang agak berbeda dari yang pernah diterima sebelumnya. Sebagai contoh, manusia sering dapat mengenali seseorang yang wajahnya pernah dilihat dari foto atau dapat mengenali seseorang

yang wajahnya agak berbeda karena sudah lama tidak menjumpainya. Kedua, tetap mampu bekerja meskipun beberapa *neuron*-nya tidak mampu bekerja dengan baik. Jika sebuah *neuron* rusak, *neuron* lain dapat dilatih untuk menggantikan fungsi *neuron* yang rusak tersebut.

Jaringan syaraf tiruan, seperti manusia, belajar dari suatu contoh karena mempunyai karakteristik yang adaptif, yaitu dapat belajar dari data-data sebelumnya dan mengenal pola data yang selalu berubah. Selain itu, JST merupakan sistem yang tak terprogram, artinya semua keluaran atau kesimpulan yang ditarik oleh jaringan didasarkan pada pengalamannya selama mengikuti proses pembelajaran/pelatihan.

Hal yang ingin dicapai dengan melatih JST adalah untuk mencapai keseimbangan antara kemampuan mengingat dan generalisasi. Yang dimaksud kemampuan mengingat adalah kemampuan JST untuk mengambil kembali secara sempurna sebuah pola yang telah dipelajari. Kemampuan generalisasi adalah kemampuan JST untuk menghasilkan respons yang bisa diterima terhadap pola-pola *input* yang serupa (namun tidak identik) dengan pola-pola yang sebelumnya telah dipelajari. Hal ini sangat bermanfaat bila pada suatu saat ke dalam JST itu dimasukkan informasi baru yang belum pernah dipelajari, maka JST itu masih akan tetap dapat memberikan tanggapan yang baik, memberikan keluaran yang paling mendekati (Puspitaningrum, 2006).

Jaringan syaraf tiruan berkembang secara pesat pada beberapa tahun terakhir. Jaringan syaraf tiruan telah dikembangkan sebelum adanya suatu komputer konvensional yang canggih dan terus berkembang walaupun pernah mengalami masa vakum selama beberapa tahun.

JST menyerupai otak manusia dalam dua hal, yaitu:

1. Pengetahuan diperoleh jaringan melalui proses belajar.
2. Kekuatan hubungan antar sel syaraf (*neuron*) yang dikenal sebagai bobot-bobot sinaptik digunakan untuk menyimpan pengetahuan.

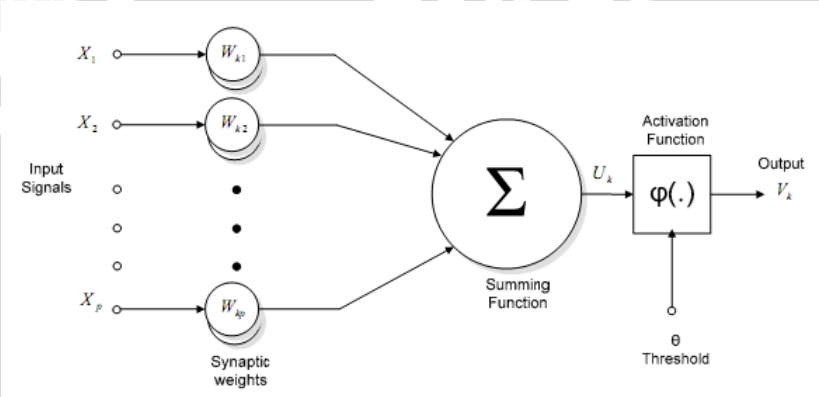
Menurut (Siang, 2004) JST ditentukan oleh 3 hal:

1. Pola hubungan antar *neuron* (disebut arsitektur jaringan). Bagian ini selanjutnya akan dijelaskan pada sub bab 2.7.
2. Metode untuk menentukan bobot penghubung (disebut metode pembelajaran/learning). Bagian ini selanjutnya akan dijelaskan pada sub bab 2.8.

3. Fungsi aktivasi, yaitu fungsi yang digunakan untuk menentukan keluaran suatu *neuron*. Bagian ini selanjutnya akan dijelaskan pada sub bab 2.9.

2.6 Model *Neuron*

Satu sel syaraf terdiri dari tiga bagian, yaitu: fungsi penjumlah (*summing function*), fungsi aktivasi (*activation function*), dan keluaran (*output*).



Gambar 2.1 Model *Neuron*

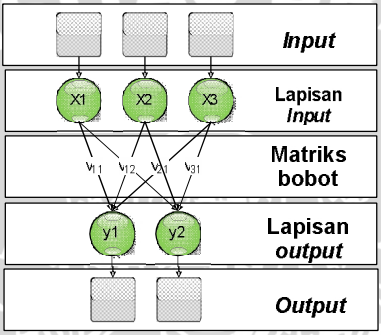
Pada gambar 2.1 bisa dilihat bahwa *neuron* buatan diatas mirip dengan sel *neuron* biologis. Informasi (*input*) akan dikirim ke *neuron* dengan bobot tertentu. *Input* ini akan diproses oleh suatu fungsi yang akan menjumlahkan nilai-nilai bobot yang ada. Hasil penjumlahan kemudian akan dibandingkan dengan suatu nilai ambang (*threshold*) tertentu melalui fungsi aktivasi setiap *neuron*. Apabila *input* tersebut melewati suatu nilai ambang tertentu, maka *neuron* tersebut akan diaktifkan, jika tidak, maka *neuron* tidak akan diaktifkan. Apabila *neuron* tersebut diaktifkan, maka *neuron* tersebut akan mengirimkan *output* melalui bobot-bobot *output*-nya ke semua *neuron* yang berhubungan dengan neuron tersebut.

2.7 Arsitektur Jaringan

JST memiliki beberapa arsitektur jaringan yang sering digunakan dalam berbagai aplikasi. Arsitektur JST tersebut, antara lain (Kusumadewi, 2003):

1. Jaringan layar tunggal (single *layer* network)

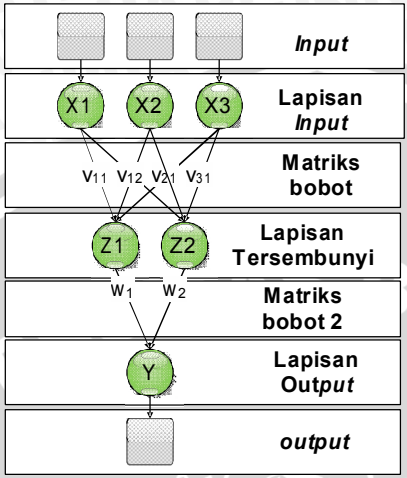
Jaringan dengan lapisan tunggal terdiri dari 1 *layer input* dan 1 *layer output*. Setiap *neuron*/unit yang terdapat di dalam lapisan/*layer input* selalu terhubung dengan setiap *neuron* yang terdapat pada *layer output*. Terlihat pada Gambar 2.2, jaringan ini hanya menerima *input* kemudian secara langsung akan mengolahnya menjadi *output* tanpa harus melalui lapisan tersembunyi.
Contoh algoritma JST yang menggunakan metode ini yaitu : *ADALINE*, *Hopfield*, *Perceptron*.



Gambar 2.2 Arsitektur *layer* tunggal

2. Jaringan layar jamak (multi *layer* network)

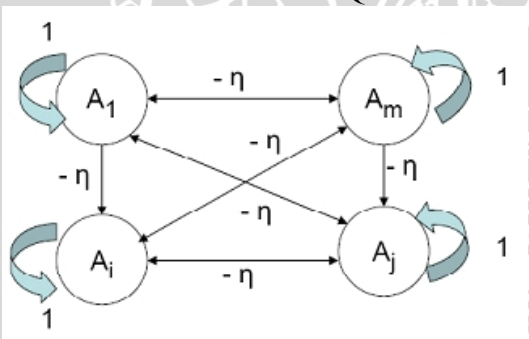
Jaringan dengan lapisan jamak memiliki ciri khas tertentu yaitu memiliki 3 jenis *layer* yakni *layer input*, *layer output*, dan juga *layer* tersembunyi seperti pada Gambar 2.3. Jaringan dengan banyak lapisan ini dapat menyelesaikan permasalahan yang lebih kompleks dibandingkan jaringan dengan lapisan tunggal. Namun, proses pelatihan sering membutuhkan waktu yang cenderung lama. Contoh algoritma JST yang menggunakan metode ini yaitu : *MADALINE*, *backpropagation*, *Neocognitron*.



Gambar 2.3 Arsitektur *layer* jamak

3. Jaringan dengan lapisan kompetitif (*competitive layer network*)

Seperti pada Gambar 2.4, pada jaringan ini sekumpulan *neuron* bersaing untuk mendapatkan hak menjadi aktif. Contoh algoritma yang menggunakan metode ini adalah LVQ.



Gambar 2.4 Arsitektur *layer* kompetitif

2.8 Metode Pelatihan/Pembelajaran

Cara berlangsungnya pembelajaran atau pelatihan JST dikelompokkan menjadi 3 yaitu (Puspitanigrum, 2006):

a) *Supervised learning* (pembelajaran terawasi)

Pada metode ini, setiap pola yang diberikan kedalam JST telah diketahui *output*-nya. Selisih antara pola *output* aktual (*output* yang dihasilkan) dengan pola *output* yang dikehendaki (*output* target)

yang disebut *error* digunakan untuk mengoreksi bobot JST sehingga JST mampu menghasilkan *output* sedekat mungkin dengan pola target yang telah diketahui oleh JST. Contoh algoritma JST yang menggunakan metode ini adalah : *Hebbian*, *Perceptron*, *ADALINE*, *Boltzman*, *Hopfield*, *Backpropagation*.

b) *Unsupervised learning* (pembelajaran tak terawasi)

Pada metode ini, tidak memerlukan target *output*. Pada metode ini tidak dapat ditentukan hasil seperti apakah yang diharapkan selama proses pembelajaran. Selama proses pembelajaran, nilai bobot disusun dalam suatu *range* tertentu tergantung pada nilai *input* yang diberikan. Tujuan pembelajaran ini adalah mengelompokkan unit-unit yang hampir sama dalam suatu area tertentu. Pembelajaran ini biasanya sangat cocok untuk klasifikasi pola. Contoh algoritma JST yang menggunakan metode ini adalah : *Competitive, Hebbian, Kohonen, LVQ (Learning Vector Quantization), Neocognitron.*

c) *Hybrid Learning* (pembelajaran hibrida)

Merupakan kombinasi dari metode pembelajaran *supervised learning* dan *unsupervised learning*. Sebagian dari bobot-bobotnya ditentukan melalui pembelajaran terawasi dan sebagian lainnya melalui pembelajaran tak terawasi. Contoh algoritma JST yang menggunakan metode ini yaitu : algoritma RBF.

2.9 Fungsi Aktivasi

Dalam JST, fungsi aktivasi digunakan untuk menentukan keluaran suatu *neuron*. Argumen fungsi aktivasi adalah net masukan (kombinasi linier masukan dan bobotnya yang ditunjukkan dengan persamaan 2.2) (Siang, 2004).

Jika terdapat suatu jaringan dengan persamaan yang ditunjukkan oleh persamaan 2.2 maka,

$$\text{net} = \sum x_i \, \text{w}_i \qquad (2.2)$$

Maka persamaan tersebut memiliki fungsi aktivasinya ditunjukkan oleh persamaan 2.3.

$$\text{f(net)} = \text{f}(\sum x_i w_i) \qquad (2.3)$$

Beberapa fungsi aktivasi yang digunakan adalah (Siang, 2004) :

a) Fungsi *threshold* (batas ambang)

$$y = \begin{cases} 0, jika & x < 0 \\ 1, jika & x >= 0 \end{cases} \qquad (2.4)$$

Fungsi *threshold* pada persamaan 2.4 merupakan fungsi *threshold* biner. Adakalanya dalam JST ditambahkan suatu unit masukan yang nilainya selalu 1. Unit tersebut dikenal dengan bias. Bias dapat dipandang sebagai sebuah *input* yang nilainya selalu 1. Bias berfungsi untuk mengubah *threshold* menjadi = 0.

b) Fungsi sigmoid

$$y = f(x) = \frac{1}{1 + e^{-\sigma x}} \tag{2.5}$$

Fungsi aktivasi pada persamaan 2.5 sering digunakan karena nilai fungsinya yang sangat mudah untuk didiferensiasikan menjadi persamaan 2.6.

$$f'(x) = f(x)[1 - f(x)] \tag{2.6}$$

c) Fungsi Sigmoid Bipolar (tansig)

Fungsi sigmoid bipolar hampir sama dengan fungsi sigmoid biner, hanya saja *output* dari fungsi ini memiliki *range* antara 1 sampai -1. Fungsi sigmoid bipolar ditunjukkan pada persamaan 2.7.

$$y = f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.7}$$

dengan persamaan 2.8 sebagai turunannya

$$f'(x) = [1 + f(x)][1 - f(x)] \tag{2.8}$$

c) Fungsi identitas

$$f(x) = x \tag{2.9}$$

2.10 Metode Backpropagation

Kelemahan ANN yang terdiri dari lapisan tunggal membuat perkembangan ANN menjadi terhenti pada sekitar tahun 1970 an. Penemuan *backpropagation* yang terdiri dari beberapa lapisan membuka kembali cakrawala baru. Terlebih setelah berhasil ditemukannya berbagai aplikasi yang dapat diselesaikan dengan metode *backpropagation*, membuat ANN semakin diminati orang, sekalipun dengan lapisan tunggal memiliki keterbatasan dalam pengenalan pola.

Kelemahan dengan lapisan tunggal, dapat ditanggulangi dengan menambahkan satu/beberapa lapisan tersembunyi diantara lapisan masukan dan keluaran. Meskipun penggunaan lebih dari satu lapisan tersembunyi memiliki kelebihan manfaat untuk beberapa kasus, tapi pelatihannya memerlukan waktu yang lama. Maka umumnya orang mulai mencoba dengan sebuah lapisan tersembunyi lebih dahulu.

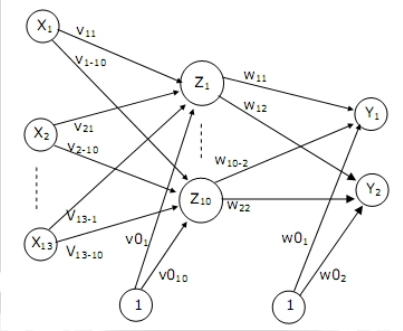
Seperti halnya model ANN lain, *backpropagation* melatih jaringan untuk mendapatkan keseimbangan antara kemampuan jaringan untuk mengenali pola yang digunakan selama pelatihan serta kemampuan jaringan untuk memberikan respon yang benar terhadap pola masukan yang serupa (tetapi tidak sama) dengan pola yang dipakai selama pelatihan.

2.10.1 Arsitektur Jaringan Metode *Backpropagation*

Ada beberapa pola arsitektur Jaringan syaraf tiruan, yaitu terdiri dari :

1. 2 (dua) lapisan, yaitu lapisan masukan/*input* dan lapisan *output*.
2. 3 (tiga) lapisan, yaitu lapisan masukan/*input*, satu lapisan hidden dan lapisan *output*.
3. Lebih dari 3 (tiga) lapisan, yaitu lapisan masukan/*input*, beberapa lapisan hidden dan lapisan *output*.

Pada Gambar 2.3 ditunjukkan salah satu jenis contoh arsitektur jaringan *backpropagation*, yaitu dengan model 3 (tiga) lapisan. Satu kelompok lapisan *input*, satu lapisan *hidden layer* dan satu lapisan *output*. Sedangkan variabel yaitu X_1 sampai dengan X_{13} adalah lapisan *input*. Artinya ada 13 variabel yang digunakan untuk data *input*. Variable Z_1 hingga Z_{10} dan lapisan keluaran/*output* terdiri atas 2 (dua) sel syaraf dengan variable Y_1 dan Y_2. Arsitektur jaringan *Backpropagation* ditunjukkan pada Gambar 2.5.



Gambar 2.5 Arsitektur Jaringan Backpropagation

2.10.2 Pengukuran Error

Perhitungan kesalahan merupakan pengukuran bagaimana jaringan dapat belajar dengan baik. Kesalahan pada keluaran dari jaringan merupakan selisih antara keluaran aktual (*current output*) dan keluaran target (*desired output*). Langkah berikutnya adalah menghitung nilai SSE *(Sum Square Error)* yang merupakan hasil penjumlahan nilai kuadrat *error neuron*1 dan *neuron*2 pada lapisan *output* tiap data, dimana hasil penjumlahan keseluruhan nilai SSE akan digunakan untuk menghitung nilai RMSE *(Root Mean Square Error)* tiap iterasi (Kusumadewi, 2003).

1. *Sum Square Error* (SSE).

SEE dihitung sebagai berikut :

1. Lapisan prediksi atau keluaran model dihitung untuk masukan pertama.
2. Selisih antara nilai luar prediksi (D) dan nilai target (T) atau sinyal latihan dihitung untuk setiap keluaran.
3. Setiap keluaran dikuadratkan kemudian seluruhnya dihitung.

$$= \quad (\quad - \quad) \qquad (2.10)$$

***2. Root Mean Square Error* (RMS *Error*).**

Dihitung sebagai berikut:

1. SSE dihitung terlebih dahulu seperti yang ditunjukkan pada persamaan 2.10.
2. Hasilnya dibagi dengan perkalian antara banyaknya data pada latihan dan banyaknya luaran, kemudian diakarkan dan diperoleh persamaan 2.11.

$$= \quad \frac{\sqrt{\quad}}{\quad_{*}} \qquad (2.11)$$

Dimana :
RMSE = *Root Mean Square Error*
SSE = *Sum Square Error*
N = Banyaknya data pada latihan
K = Banyaknya luaran/*output*.

2.10.3 Algoritma Backpropagation

Perambatan galat mundur (*Backpropagation*) adalah sebuah metode sistematik untuk pelatihan *multilayer* jaringan syaraf tiruan. Metode ini memiliki dasar matematis yang kuat, objektif dan algoritma ini mendapatkan bentuk persamaan dan nilai koefisien dalam formula dengan meminimalkan jumlah kuadrat galat *error* melalui model yang dikembangkan (*training set*) (Yeh, 2006).

1. Dimulai dengan lapisan *input*, keluaran dari setiap elemen pemroses melalui lapisan *output* dihitung.
2. Kesalahan pada lapisan *output* yang merupakan selisih antara data aktual dan target dihitung.
3. Kesalahan tersebut ditransformasikan pada kesalahan yang sesuai di sisi *input* elemen pemroses.
4. Dilakukan propagasi balik kesalahan-kesalahan ini pada keluaran setiap elemen pemroses ke kesalahan yang terdapat pada *input*. Proses ini diulangi sampai kesalahan output yang ditentukan tercapai.
5. Ubah seluruh bobot dengan menggunakan kesalahan pada sisi masukan elemen dan luaran elemen pemroses yang terhubung.

2.10.4 Langkah-langkah Algoritma *Backpropagation*.

Pelatihan suatu jaringan dengan algoritma *backpropagation* meliputi dua tahap : perambatan maju dan perambatan mundur. Selama perambatan maju, tiap unit masukan (x_i) menerima sebuah masukan sinyal ini ke tiap-tiap lapisan tersembunyi $z_1,\ldots..,z_p$. Tiap unit tersembunyi ini kemudian menghitung aktivasinya dan mengirimkan sinyalnya (z_j) ke tiap unit keluaran. Tiap unit keluaran (y_k) menghitung aktivasinya (y_k) untuk membentuk respon pada jaringan untuk memberikan pola masukan. Selama pelatihan, tiap unit keluaran membandingkan perhitungan aktivasinya y_k dengan nilai targetnya t_k untuk menentukan kesalahan pola tersebut dengan unit itu. Berdasarkan kesalahan ini, faktor δ_k $(k = 1,..,m)$ dihitung. δ_k digunakan untuk menyebarkan kesalahan pada unit keluaran *yk* kembali ke semua unit pada lapisan sebelumnya (unit-unit tersembunyi yang dihubungkan ke y_k). Juga digunakan (nantinya) untuk mengupdate bobot-bobot antara keluaran dan lapisan tersembunyi. Dengan cara yang sama, faktor ($j = 1,\ldots,p$) dihitung untuk tiap unit tersembunyi z_j. Tidak perlu untuk menyebarkan kesalahan kembali ke lapisan masukan, tetapi δ_j digunakan untuk

mengupdate bobot-bobot antara lapisan tersembunyi dan lapisan masukan.

Setelah seluruh faktor δ ditentukan, bobot untuk semua lapisan diatur secara serentak. Pengaturan bobot w_{jk} (dari unit tersembunyi z_j ke unit keluaran y_k) didasarkan pada faktor δ_k dan aktivasi z_j dari unit tersembunyi z_j. didasarkan pada faktor δ_j dan dan aktivasi x_i unit masukan. Menurut (Freeman & Skapura, 1991) langkah-langkahnya adalah sebagai berikut:

Pada prosedur pelatihan dilakukan beberapa langkah proses pelatihan yang diawali dengan inisialisasi bobot yang sebaiknya diatur pada nilai random yang cukup kecil atau menggunakan metode Nguyen Widrow(dijelaskan pada subbab 2.15),

1. Maksimum *epoch* (siklus tiap perubahan bobot), target *error*, dan learning rate (α), jika kondisi tidak tercapai, lakukan langkah 2-9,
2. Inisiasi Epoch = 0, RMSE = 1,
3. Langkah-langkah berikut selama (*epoch* < maksimum *epoch*) dan (*error* > target *error*). Untuk tiap-tiap pasangan elemen yang akan dilakukan pembelajaran lakukan :

Perambatan Maju (*feedfoward*):

4. Tiap unit masukan (x_i, $i = 1,\ldots, n$) menerima sinyal x_i dan menghantarkan sinyal ini ke semua unit lapisan di atasnya (unit tersembunyi),
5. Setiap unit tersembunyi (x_i, $i = 1,\ldots, p$) jumlahkan bobot sinyal masukannya dengan persamaan 2.12, kemudian dilakukan aktivasi dengan persamaan 2.13.

$$= + \sum \qquad (2.12)$$

$$= = \frac{}{-} \qquad (2.13)$$

v_{jo} = bias pada unit tersembunyi j aplikasikan fungsi aktivasinya untuk menghitung sinyal keluarannya, dan kirimkan sinyal ini keseluruh unit pada lapisan diatasnya (unit keluaran).

6. Tiap unit keluaran (y_k, $k = 1,2,3,\ldots, m$) jumlahkan bobot sinyal masukannya dengan persamaan 2.14, kemudian dilakukan aktivasi dengan menggunakan persamaan 2.15,

$$= \quad +$$

(2.14)

$$= \quad = \frac{}{}$$ (2.15)

w_{ko} = bias pada unit keluaran *k* dan aplikasikan fungsi aktivasinya untuk menghitung sinyal keluarannya.

Perambatan Mundur (*feedback*):

7. Tiap unit keluaran (y_k , *k = 1,2,3,…, m*) menerima pola target yang saling berhubungan pada masukan pola pelatihan, hitung kesalahan informasinya seperti yang ditunjukkan pada persamaan 2.16,

$$= - \quad = - \quad (1 - \quad)$$ (2.16)

Suku perubahan bobot w_{jk} dihitung dengan persamaan 2.17(digunakan untuk memperbaharui w_{jk} dengan laju pelatihan α).

$$\Delta \quad =$$ (2.17)

Hitung perubahan biasnya (digunakan untuk memperbaharui w_{ok}), dan kirimkan δ_k ke unit-unit pada lapisan dibawahnya,

8. Setiap unit lapisan tersembunyi (*dari unit ke-1 hingga ke-p;i=1…n;k=1…m*) dilakukan perhitungan kesalahan lapisan tersembunyi(δ_j). δ_j kemudian digunakan untuk menghitung besar koreksi bobot dan bias (Δv_{ij} dan Δv_{jo}) antara lapisan *input* dan lapisan tersembunyi.

$$_ \quad = \sum$$ (2.18)

Hasil dari perhitungan pada persamaan 2.18 dikalikan dengan turunan fungsi aktivasinya untuk menghitung informasi kesalahannya, sehingga untuk mencari δ_j dapat digunakan persamaan 2.19.

$$= \quad _ \quad (\quad)$$ (2.19)

Suku perubahan bobot Δv_{ij}(yang digunakan untuk perbaikan bobot v_{ij}) dihitung dengan persamaan 2.20.

Δ = (2.20)

Perubahan bias (untuk memperbaiki v_{jo}) dihitung dengan menggunakan persamaan 2.21.

Δ = (2.21)

9. Tiap unit keluaran (*yk, k = 1,2,3,.., m*) dilakukan pembaharuan bias dan bobotnya (*j = 0,1,2,3.., p*) sehingga menghasilkan bobot dan bias baru dari persamaan 2.22:

= + Δ (2.22)

Demikian juga untuk setiap unit tersembunyi mulai dari unit ke-1 sampai dengan unit ke-p dilakukan pembaharuan bias dan bobotnya dengan persamaan 2.23 :

= + Δ (2.23)

10. Uji kondisi berhenti(akhir iterasi).

2.11 Inisialisasi Nguyen-Widrow

Inisialisasi Nguyen-Widrow merupakan suatu metode penginisialisasian nilai bobot dan bias jaringan *backpropagation* yang mampu mempercepat proses *training* dibandingkan dengan penginisialisasian secara random/acak. Tahap ini digunakan untuk optimasi jaringan *backpropagation* sehingga lebih mudah mencapai konvergensi.

Pada inisialisasi Nguyen-Widrow, inisialisasi acak tetap digunakan untuk menginisialisasi bias dan bobot dari lapisan tersembunyi ke lapisan *output*. Untuk bias dan bobot dari lapisan masukan ke lapisan tersembunyi digunakan bias dan bobot yang diskalakan agar jatuh pada *range* tertentu. (Puspitanigrum, 2006)

Faktor skala Nguyen-Widrow (β) didefinisikan pada persamaan 2.24.

$$\beta = 0.7\ (p)^{1/n} \quad (2.24)$$

dimana:

n = banyak unit *input*

p= banyak unit tersembunyi

β=faktor skala

Untuk setiap unit tersembunyi dari unit ke-1 sampai unit ke-p :

1. Inisialisasi bobot-bobot antara unit *input* ke unit tersembunyi (j=1,2,…,p) dengan cara:
    a. Menentukan bobot-bobot antara unit *input* ke unit

tersembunyi (v_{ij}) dengan bilangan acak antara $-\beta$ dan β.

b. Menghitung dengan menggunakan persamaan 2.25.

$$= + + \cdots + \quad (2.25)$$

c. Menginisialisasikan kembali v_{ij} seperti yang ditunjukkan pada persamaan 2.26:

$$v = \frac{\cdot ( )}{} \quad (2.26)$$

2. Menentukan bias antara *input* ke unit tersembunyi (j=1,2,...p) dengan v_{oj} di set dengan bilangan acak yang terletak pada skala antara $-\beta$ dan β.

2.12 Metode Optical Backpropagation

Metode *optical backpropagation* (OBP), didesain untuk menyelesaikan masalah pada standar *backpropagation* (BP) yang menggunakan fungsi non-linear. Aspek penting dalam metode ini yaitu kemampuan untuk keluar dari lokal minima dengan kecepatan konvergensi yang tinggi pada saat pelatihan. Prinsip dasar dari OBP ini terletak pada pengaturan *error* yang ditransmisikan ke belakang dari lapisan *output* pada tiap unit pada lapisan tersembunyi. Dalam BP, kesalahan pada *output* tunggal didefinisikan sebagai berikut:

$$= (-) (1 -) \quad (2.27)$$

Pada persamaan 2.27 k adalah unit keluaran ke-k. Dan t_k adalah nilai *output* yang diinginkan serta y_k adalah *output* akutal dari unit ke-k, kemudian dilakukan propagasi balik untuk memperbaharui bobot dan bias. Namun pada OBP *error single output* diatur sebagai berikut:

Jika $- \geq 0$ maka

$$= (1 +) (1 -) \quad (2.28)$$

Jika $- < 0$ maka

$$= -(1 +) (1 -) \quad (2.29)$$

Untuk langkah-langkah OBP sama seperti BP pada subbab 2.10.4 hanya saja pada persamaan 2.16 diganti seperti pada persamaan 2.28 atau 2.29 sesuai dengan syarat mana yang dipenuhi. (Otair & Salameh, 2005)

2.13 Normalisasi dan Denormalisasi

Sebelum data diproses sebagai masukan dari suatu JST maka sebelumnya data juga perlu dilakukan normalisasi dengan menyesuaikan *range output* fungsi aktivasi. Misalkan menggunakan fungsi aktivasi sigmoid maka data masukan harus dirubah dengan *range* [0...1], namun karena merupakan fungsi kontinu maka nilai 0 dan 1 tidak pernah dicapai. Maka dari itu *range* dirubah menjadi [0.1...0.9], sehingga dapat dirumuskan pada persamaan 2.30.(Siang, 2004).

$$= \text{——} 0{,}8 + 0{,}1 \qquad (2.30)$$

Sedangkan untuk mengembalikan nilai awal data yang ternormalisasi dilakukan denormalisasi seperti pada persamaan 2.31

$$= \frac{(\quad ,)}{,} + \qquad (2.31)$$

Dimana
x : *input*
a : nilai minimum dari data
b : nilai maksimum dari data
: *input* yang telah ternormalisasi
0.8 merupakan jarak dari skala 0,9-0,1

2.14 Pemilihan jumlah layer dan neuron

Permasalahan yang sering terjadi dalam jaringan syaraf tiruan adalah menentukan berapa *hidden layer* yang dibutuhkan dan berapa banyak *neuron* dalam *layer* tersebut. Menurut(Heaton, 2010) menggunakan satu *hidden layer* sudah cukup untuk menyelesaikan berbagai macam kasus. Sedangkan untuk menetukan jumlah *neuron* dalam suatu *hidden layer* harus memenuhi aturan bahwa jumlah neuron disarankan 2/3 dari jumlah *input* ditambah dengan jumlah *output* dan jumlah neuron kurang dari dua kali jumlah input.

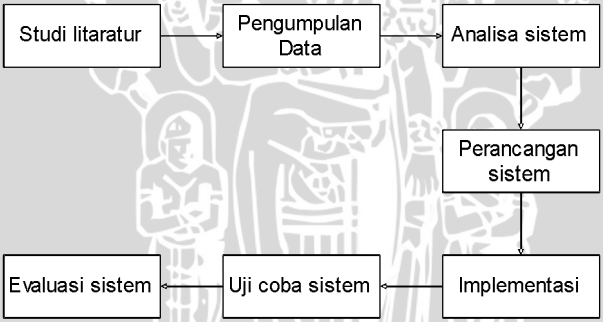
BAB III
METODOLOGI DAN PERANCANGAN SISTEM

Dalam bab ini akan dijelaskan mengenai analisis sistem dan perancangan sistem untuk menentukan kuat tekan dan *setting time* beton pada kasus SCC menggunakan jaringan syaraf tiruan dengan metode *optical backpropagation.*

Penelitian dilakukan dengan tahapan-tahapan sebagai berikut:

1. Mempelajari literatur yang terkait dengan masalah beton SCC dan jaringan syaraf tiruan *optical backpropagation.*
2. Mengumpulkan data-data hasil penelitian SCC di laboratorium.
3. Menganalisa system dan melakukan perancangan sistem menggunkan jaringan syaraf tiruan *optical bakcpropagation.*
4. Mengimplementasikan sistem
5. Melakukan uji coba sistem dengan memasukkan data yang berbeda dengan data *training* kedalam sistem.
6. Mengevaluasi hasil pengujian.

Alur penelitian yang dilakukan dapat digambarkan dalam bentuk diagram alir yang ditunjukkan pada Gambar 3.1.



Gambar 3.1 Langkah-langkah Penelitian

3.1 Studi Literatur

Untuk lebih memantapkan pengetahuan dalam merealisasikan tujuan dan pemecahan masalah penelitian ini dilakukan dengan studi literatur. Teori-teori mengenai teorti jaringan syaraf tiruan, dan penentuan kuat tekan dan *setting time* beton SCC digunakan sebagai dasar penelitian ini diperoleh dari buku, jurnal, dan sumber lain.

3.2 Data Yang Digunakan

Data yang digunakan untuk penelitian ini adalah data *mix design* SCC hasil penelitian tentang beton SCC. Data berisi 8 parameter, enam material campuran sebagai masukan yang terdiri dari jumlah semen (kg/m^3), jumlah kerikil(kg/m^3), jumlah pasir(kg/m^3), jumlah *fly ash*(kg/m^3), SP(lt/m^3), dan fas/faktor air semen(%), serta dua parameter sebagai target yaitu *setting time*(menit) dan kuat tekan(MPa).

3.3 Menentukan Pola Data

Dari data yang didapat dapat disusun suatu pola/ *pattern* data yang nantinya digunakan sebagai masukan dalam pelatihan jaringan syaraf tiruan. Tabel pola data ditunjukkan pada tabel 3.1.

Tabel 3.1 Pola Data

X1	X2	X3	X4	X5	X6	T1	T2

Keterangan dari pola input adalah sebagai berikut:

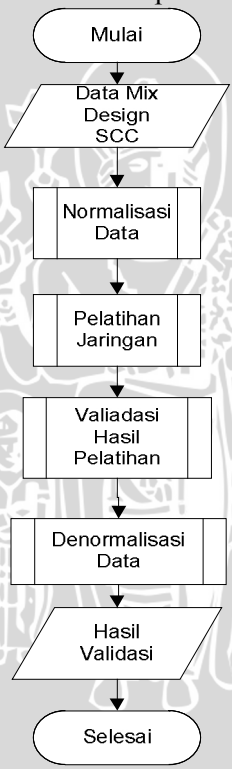
- $x_{1:}$ jumlah semen
- $x_{2:}$ jumlah kerikil
- $x_{3:}$ jumlah pasir
- x_{4} jumlah *fly ash*,
- $x_{5:}$ *superplasticiser*
- $x_{6:}$ fas/faktor air semen
- $t_{1:}$ s*etting time*
- $t_{2:}$ kuat tekan.

3.4 Deskripsi Umum Sistem

Secara umum system yang dibangun adalah perangkat lunak untuk menentukan kuat tekan dan *setting time* beton SCC yang mengimplementasikan jaringan syaraf tiruan *optical bakcpropagation.* Jaringan syaraf tiruan berfungsi sebagai pengidentifikasian pola *output.* Sistem bertujuan untuk menentukan kuat tekan dan *setting time* beton SCC dari data hasil penelitian di laboratorium agar tidak perlu lagi melakukan penelitian yang lama dan mahal untuk mengetahui nilai kuat tekan dan *setting time* dari campuran material yang telah disebutkan pada sub bab 3.2.

3.5 Perancangan Proses

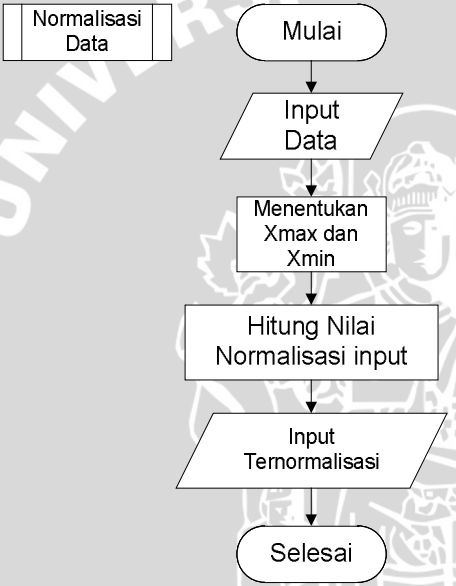
Tahapan proses dari sistem itu yaitu pengambilan data *mix design* SCC yang selanjutnya dinormalisasi dengan persamaan 2.30. Setelah data dinormalisasi maka data siap untuk dilatihkan dalam jaringan syaraf tiruan. Apabila error minimum telah didapatkan maka sistem siap melakukan *testing* atau validasi dengan data yang saling asing dengan data *training* sehingga menghasilkan *output* berupa kuat tekan dan *setting time* beton. Hasil dari validasi masih berupa data ternormalisasi sehingga perlu didenormalisasi agar didapatkan nilai output yang nyata(tidak dalam *range activation function*). Alur proses dalam sistem secara umum dapat dilihat pada gambar 3.2.



Gambar 3.2 Diagram Alir Sistem

3.5.1 Proses Normalisasi Data

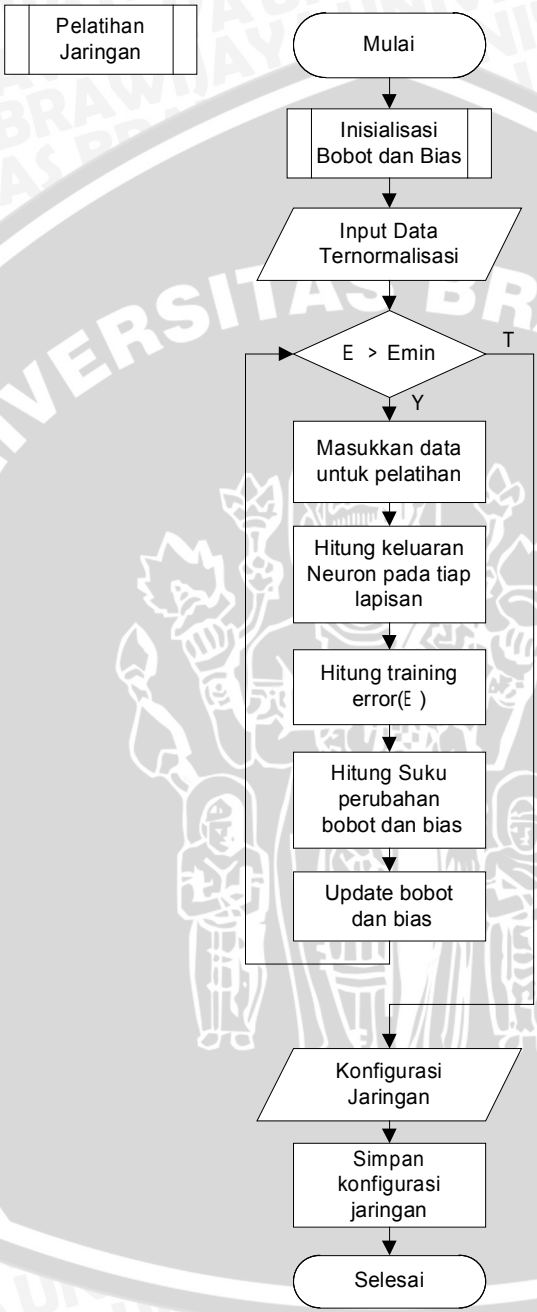
Proses ini bertujuan untuk merubah data kedalam domain fungsi aktivasi *sigmoid* (Persamaan 2.5) yaitu berada antara nilai 0 dan 1. Namun karena nilai 0 dan 1 tidak akan pernah dicapai oleh fungsi *sigmoid* maka digunakan 0,1 dan 0,9 untuk merepresentasikan nilai terkecil dan terbesar dari data.(Freeman & Skapura, 1991). Kemudian dilakukan normalisasi sesuai dengan persamaan 2.30. Langkah-langkah proses normalisasi data ditunjukkan pada gambar 3.3.



Gambar 3.3 Diagram Alir Proses Normalisasi

3.5.2 Proses Pelatihan Jaringan Syaraf Tiruan

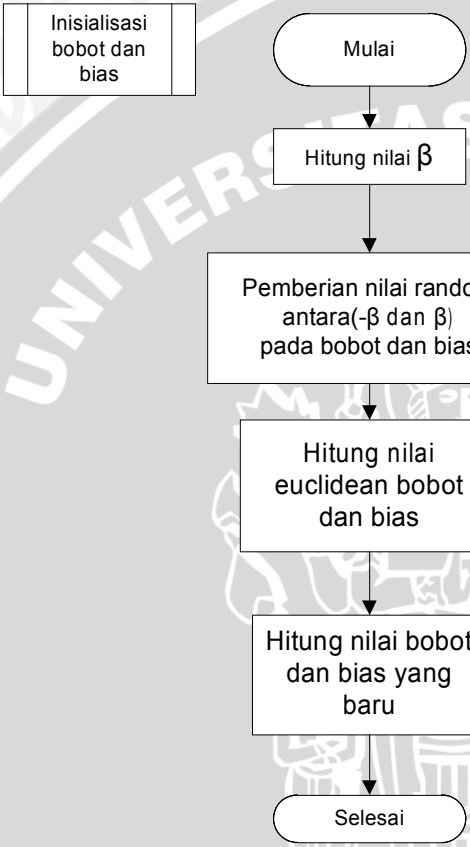
Proses pelatihan diawali dengan inisialisasi bobot awal untuk lapisan tersembunyi dan lapisan *output* dengan nilai random dengan metode Nguyen-widrow(lihat sub bab 2.11). Kemudian memasukkan data training yang sudah dinormalisasi sebelumnya. Selama *error* masih lebih besar dari *error* minimum maka dilakukan langkah 4 sampai 10 yang sudah dijabarkan pada sub bab 2.10.4.



Gambar 3.4 Diagram Alir Pelatihan Jaringan

3.5.3 Proses Inisialisasi Bobot dan Bias

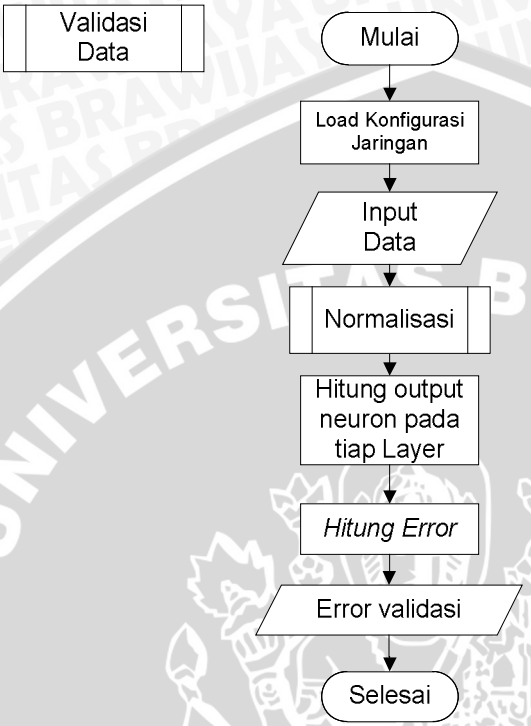
Pada gambar 3.5 ditunjukkan diagram alir proses inisialisasi bobot dan bias menggunakan metode Nguyen-Widrow. Untuk langkah-langkah secara rinci telah dijelaskan pada sub bab 2.13.



Gambar 3.5 Diagram Alir Inisialisasi Bobot dan Bias

3.5.4 Proses Validasi Hasil Penelitian

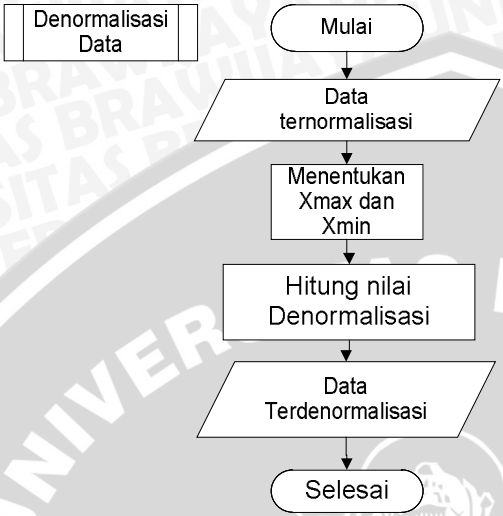
Proses validasi hasil penelitian merupakan pengecekan tingkat kebenaran sistem dalam menentukan kuat tekan dan *setting time* dengan cara memasukkan data selain dari yang digunakan untuk *training* kemudian dihitung tingkat kebenarannya. Langkah-langkah validasi ditunjukkan oleh diagram alir pada gambar 3.6.



Gambar 3.6 Diagram Alir Validasi Data

3.5.5 Proses Denormalisasi

Dari validasi dapat diketahui nilai kesalahan dari sistem pada saat pengujian dengan data selain data *training*. Selain itu juga menghasilkan nilai kuat tekan dan *setting time* dari data masukan yang diberikan. Namun besarnya kuat tekan dan *setting time* masih dalam *range* fungsi aktivasi sehingga perlu dilakukan denormalisasi agar diperoleh nilai yang sebenarnya. Proses denormalisasi ini ditunjukkan pada gambar 3.7.



Gambar 3.7 Diagram Alir Proses Denormalisasi Data

3.6 Perancangan Uji Coba

Pada bagian perancangan uji coba ini akan dijelaskan mengenai pengujian data. Dalam uji coba ini menggunakan 40 data *input* sebagai *training data* kemudian dilakukan validasi dengan data yang lain. Dengan data *training* yang sama dilakukan pengulangan uji coba dengan perubahan jumlah *hidden* unit dan learning rate.

Pengujian dilakukan dengan menggunakan satu *hidden layer*. *Training error* maksimal ditentukan 1%. Proses pengujian dilakukan dengan merubah-ubah kombinasi antara jumlah *hidden unit* dengan *learning rate* yang diujikan yaitu 0,1 sampai 1 dengan perubahan learning rate sebesar 0,1 dan dalam jumlah *epoch* maksimal yang tetap yaitu 1000. Jumlah *hidden Unit* atau *neuron* yang diujikan antara 6 sampai 12 neuron. Bentuk data hasil percobaan tersebut akan disajikan pada Tabel 3.2.

Setelah *training* selesai dilakukan maka tahap selanjutnya yaitu validasi hasil *training* dengan memasukkan data yang saling asing dengan data *training* kemudian dicek *error* yang didapatkan. Format validasi hasil *training* ditunjukkan pada Tabel 3.3.

Tabel 3.2 Pengujian Jumlah *Neuron* dan Learning Rate terhadap error(RMSE) dan waktu

No	*Hidden Unit*	*Learning Rate*	Uji ke-	RMSE(%)	Waktu(detik)
1	6	0.1	1		
			2		
			3		
			4		
			5		
2	6	0.2	1		
			2		
			3		
			4		
			5		

Tabel 3.3 Validasi hasil *training*

X1	**X2**	**X3**	**X4**	**X5**	**X6**	**T1**	**Y1**	**T2**	**Y2**

Keterangan:
$x_1 \ldots x_6$: *data input*
T1 : *setting time* target
Y1 : *setting time* hasil perhitungan sistem
T2 : kuat tekan target
Y2 : kuat tekan hasil perhitungan sistem

3.7 Perhitungan Manual

Diketahui 4 *sample* data penelitian tentang komposisi campuran material penyusun SCC yang ditunjukkan pada **Tabel 3.5**.

Tabel 3.4 Tabel Data Masukan

	Jml Semen (kg/m^3)	Jml Kerikil (kg/m^3)	Jml Pasir (kg/m^3)	Jml Fly Ash (kg/m^3)	SP (lt/m^3)	fas (%)	Setting Time (Jam)	Kuat Tekan (MPa)
1	460	600	915	0	2.5	0.38	3.21	39.5
2	425	900	880	75	2.5	0.35	165	53.79
3	400	800	880	100	1.65	0.35	180	54
4	368	900	915	92	1.5	0.38	172	43

Langkah yang pertama yaitu untuk menentukan pola data pelatihan yaitu x1 adalah jumlah semen, x2 adalah jumlah kerikil, x3 adalah jumlah pasir, x4 adalah jumlah *fly ash*, x5 adalah jumlah SP, sedangkan targetnya adalah *setting time*(t1) dan kuat tekan beton(t2). Kemudian dilakukan normalisasi data input dengan persamaan 2.30. Hasil dari normalisasi data ditunjukkan pada **Tabel 3.5**.

Tabel 3.5 Tabel Hasil Normalisasi Data Awal

	x1	**x2**	**x3**	**x4**	**x5**	**x6**	**t1**	**t2**
1	0.90	0.10	0.90	0.10	0.90	0.90	0.10	0.10
2	0.60	0.90	0.10	0.70	0.90	0.10	0.83	0.89
3	0.38	0.63	0.10	0.90	0.22	0.10	0.90	0.90
4	0.10	0.90	0.90	0.84	0.10	0.90	0.86	0.29

Parameter awal untuk pelatihan yaitu
$\alpha = 0.1$
$\varepsilon = 0.2$
jumlah unit *input* = 6
jumlah unit di *hidden layer* = 6
jumlah unit *output* = 2

Inisialisasi bobot awal dengan metode Nguyen Widrow. Pertama-tama nilai faktor β dihitung dengan menggunakan persamaan 2.24 sehingga dihasilkan β=0.94. Kemudian Nilai bobot awal diberi nilai random yang kecil(antara - β sampai β) kemudian dihitung $\|v_j\|$ dengan persamaan 2.25, dan hasilnya pada Tabel 3.6 dan Tabel 3.7.

Tabel 3.6 Tabel Data Nilai Bobot Awal V_{ij}

Vij	x1	x2	x3	x4	x5	x6	$\|\|V_j\|\|$
z1	0.06	0.54	0.46	-0.33	0.25	-0.16	0.837963
z2	-0.37	-0.36	0.34	0.07	-0.63	-0.10	0.88942
z3	0.30	0.10	-0.19	-0.24	-0.01	0.24	0.501946
z4	-0.69	0.20	0.05	-0.24	0.59	0.68	1.176175
z5	0.13	0.01	0.25	-0.28	0.38	-0.12	0.558817
z6	0.77	-0.53	0.08	0.05	0.36	-0.05	1.01399

Tabel 3.7 Tabel Data Nilai Bobot Awal W_{ij}

Wij	z1	z2	z3	z4	z5	z6	||Wj||
t1	0.17	-0.70	0.18	-0.80	-0.06	0.43	1.175116
t2	0.01	-0.16	-0.04	-0.08	0.05	0.48	0.518644

Setelah itu dilakukan perhitungan vij dan wij baru dengan persamaan 2.26, dan hasilnya pada Tabel 3.8 dan Tabel 3.9.

Tabel 3.8 Tabel Data Nilai Bobot V_{ij} Nguyen Widrow

Vij	x1	x2	x3	x4	x5	x6	1
z1	0.07	0.60	0.52	-0.37	0.28	-0.17	0.00
z2	-0.39	-0.38	0.36	0.08	-0.66	-0.11	0.30
z3	0.56	0.19	-0.36	-0.44	-0.01	0.45	-0.02
z4	-0.55	0.16	0.04	-0.19	0.47	0.54	-0.13
z5	0.22	0.02	0.42	-0.46	0.63	-0.20	-0.12
z6	0.72	-0.50	0.07	0.04	0.34	-0.05	0.04

Tabel 3.9 Tabel Data Nilai Bobot W_{ij} Nguyen Widrow

Wij	z1	z2	z3	z4	z5	z6	1
t1	0.14	-0.56	0.14	-0.64	-0.05	0.34	0.168053
t2	0.02	-0.30	-0.07	-0.14	0.09	0.87	-0.26123

1. Jika kondisi penghentian belum terpenuhi, maka langkah 2-7 diulangi terus menerus.
2. Untuk setiap pasang data pelatihan, langkah 3-7 diulangi sampai semua pola data dilatih.

Pola data 1

3. Setiap input mengirim sinyal ke unit tersembunyi. Hitung keluaran di unit tersembunyi(z_j) dengan persamaan 2.12 dan dilanjutkan dengan perhitungan aktivasinya dengan persamaan 2.13.Hasil perhitungannya ditunjukkan pada tabel Tabel 3.10 dan Tabel 3.11.

Tabel 3.10 Tabel Operasi pada *Hidden Neuron*

j	z_in
1	0.6442
2	-0.4472
3	0.5257
4	0.3251
5	0.8011
6	0.9723

Tabel 3.11 Tabel Aktivasi pada *Hidden Neuron*

j	Z
1	0.6557
2	0.39
3	0.6285
4	0.5806
5	0.6902
6	0.7256

4. Keluaran unit output dihitung dengan menggunakan persamaan 2.14 dan 2.15, sehingga menghasilkan *y_in* pada Tabel 3.12 dan menghasilkan hasil aktivasi *y_in* menghasilkan *y* yang ditunjukkan pada Tabel 3.13.

Tabel 3.12 Tabel Operasi pada *Output*

j	y_in
1	0.2653
2	-0.2489

Tabel 3.13 Tabel Hasil Aktivasi pada *output*

J	y
1	0.5659
2	0.4381

5. Kemudian mencari nilai faktor δ di unit *output* y_k dengan menggunakan persamaan 2.16. Karena menggunakan *optical backpropagation* maka nilai (t_k-y_k) dicek terlebih dahulu. Jika bernilai kurang dari 0 maka menggunakan persamaan 2,29 untuk mengganti nilai (t_k-y_k) dan jika lebih besar sama dengan 0 maka menggunakan persamaan 2.28.

$$(t_i-y_1)=0.1-0.5639= - 0.4659$$
$$(t_2-y_2)=0.1-0.4381= - 0.3381$$

Karena (t_k-y_k) semuanya bernilai negatif maka menggunakan persamaan 2.29 menggantikan nilai (t_k-y_k) pada persamaan 2.16.

$\delta_1 = -(1 +$ $)*0.5659*(1-0.5659) = -0.55086712077796$

$\delta_2 = -(1 +$ $)*0.4381*(1-0.4381) = -0.532353094974464$

Setelah itu dihitung suku perubahan bobot ΔW dengan α=0.1. Hasil perhitungan ΔW ditunjukkan pada Tabel 3.14.

Tabel 3.14 Tabel Suku Perubahan Bobot

ΔW	k1	k2
0	-0.05509	-0.05324
1	-0.03612	-0.03491
2	-0.02149	-0.02076
3	-0.03462	-0.03346
4	-0.03198	-0.03091
5	-0.03802	-0.03674
6	-0.03997	-0.03863

6. Kemudian dihitung penjumlahan kesalahan dari unit tersembunyi dengan menggunakan persamaan 2.18 dan dilanjutkan dengan

persamaan 2.19. Perhitungan pertama yang menggunakan persamaan 2.18 hasilnya ditunjukkan pada Tabel 3.15.

Tabel 3.15 Tabel Jumlah Kesalahan Pada *Hidden* Unit

	1	2	3	4	5	6
δ_in	-0.0859	0.47315	-0.03695	0.432001	-0.02417	-0.66761

Selanjutnya dihitung faktor kesalahan δ di unit tersembunyi yang hasilnya ditunjukkan oleh Tabel 3.16.

Tabel 3.16 Tabel Faktor kesalahan *hidden* unit

	1	2	3	4	5	6
δ	-0.01939	-0.02044	-0.02006	-0.02092	-0.01837	-0.0171

Dari nilai faktor kesalahan *hidden* unit dapat dihitung suku perubahan bobot pada unit tersembunyi(ΔV_{ij}) dengan menggunkan persamaan 2.20. Hasilnya ditunjukkan pada Tabel 3.17.

Tabel 3.17 Tabel Suku Perubahan Bobot ke *hidden* unit

ΔV	x1	x2	x3	x4	x5	x6	1
z1	-0.00175	-0.00019	-0.00175	-0.00019	-0.00175	-0.00175	-0.00194
z2	-0.00175	-0.00019	-0.00175	-0.00019	-0.00175	-0.00175	-0.00194
z3	-0.00175	-0.00019	-0.00175	-0.00019	-0.00175	-0.00175	-0.00194
z4	-0.00175	-0.00019	-0.00175	-0.00019	-0.00175	-0.00175	-0.00194
z5	-0.00175	-0.00019	-0.00175	-0.00019	-0.00175	-0.00175	-0.00194
z6	-0.00175	-0.00019	-0.00175	-0.00019	-0.00175	-0.00175	-0.00194

Langkah yang terakhir yaitu peng-*update*-an bobot dan bias dengan menggunakan persamaan 2.22 untuk W_{ij}(bobot *output* unit) dan persamaan 2.23 untuk V_{ij}(bobot *hidden* unit). Bobot unit *output* dan *hidden* unit yang baru ditunjukkan pada tabel Tabel 3.18 dan Tabel 3.19.

Tabel 3.18 Tabel Bobot *Output* Unit yang Baru

Wij	t1	t2
z1	0.082	-0.034
z2	-0.597	-0.333
z3	0.120	-0.095
z4	-0.676	-0.176
z5	-0.081	0.062
z6	0.302	0.835
1	0.128	-0.300

Tabel 3.19 Tabel Bobot *Hidden* Unit yang Baru

Vij	x1	x2	x3	x4	x5	x6	1
1	0.07	0.60	0.52	-0.37	0.27	-0.18	0.00
2	-0.39	-0.38	0.36	0.08	-0.67	-0.11	0.30
3	0.56	0.19	-0.37	-0.44	-0.01	0.44	-0.02
4	-0.55	0.16	0.04	-0.19	0.47	0.54	-0.13
5	0.22	0.02	0.42	-0.46	0.63	-0.20	-0.12
6	0.72	-0.50	0.07	0.04	0.33	-0.05	0.04

Proses ini diulangi terus-menerus sampai semua pola data dilatihkan dan akan berhenti jika memenuhi kondisi berhenti(misalnya error lebih kecil dari error minimal yang ditentukan atau sudah mencapai batas *epoch* maksimal.

3.8 Perancangan Antarmuka

Pada bagian ini akan dijelaskan antarmuka sistem yang akan dibuat. Gambar 3.8 menunjukkan tampilan rancangan antarmuka untuk memasukkan data training kedalam system. Berikut adalah penjelasan Gambar 3.8 berdasarkan nomor:

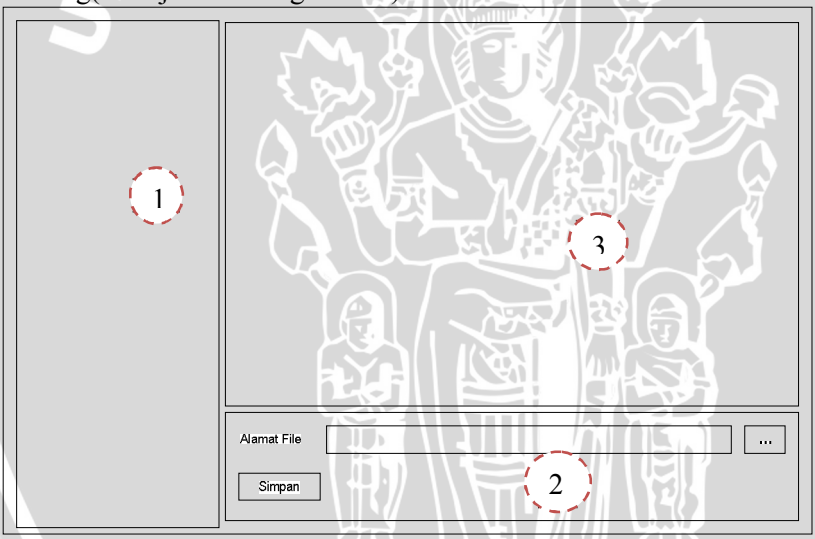
1. Menampilkan daftar nama data *training* yang sudah dimasukkan dalam sistem.
2. Bagian ini berisi tombol untuk load data training, *textbox* alamat file dan tombol untuk menyimpan data training kedalam kedalam sistem.

3. Pada bagian ini menampilkan data training dalam bentuk tabel.

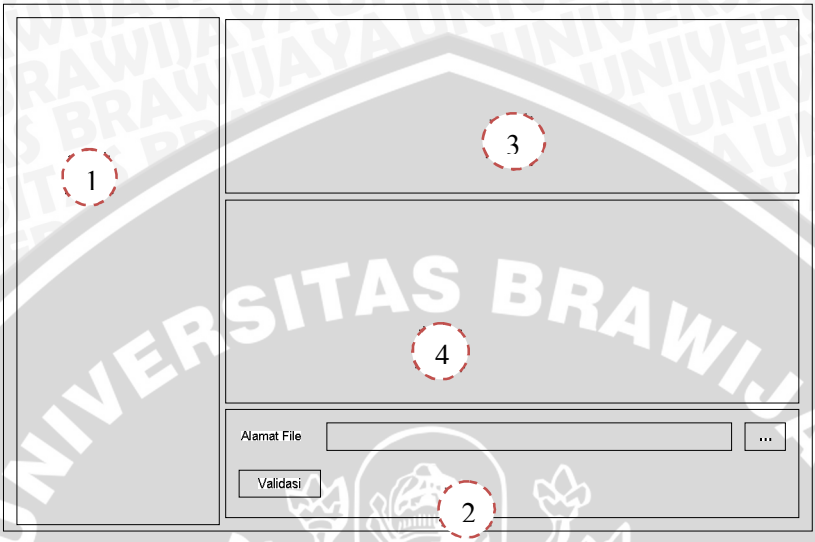
Antarmuka validasi data ditunjukkan pada Gambar 3.9, penjelasan berdasarkan nomor pada gambar adalah sebagai berikut:

1. Menampilkan daftar nama konfigurasi OBPNN yang telah dilakukan proses *training*.
2. Bagian ini berisi tombol untuk load data training, *textbox* alamat file dan tombol untuk validasi.
3. Menampilkan grafik.
4. Setelah tombol validasi pada bagian no 2 di klik maka pada bagian ini akan menampilkan data dalam bentuk tabel validasi dengan format seperti pada Tabel 3.3.

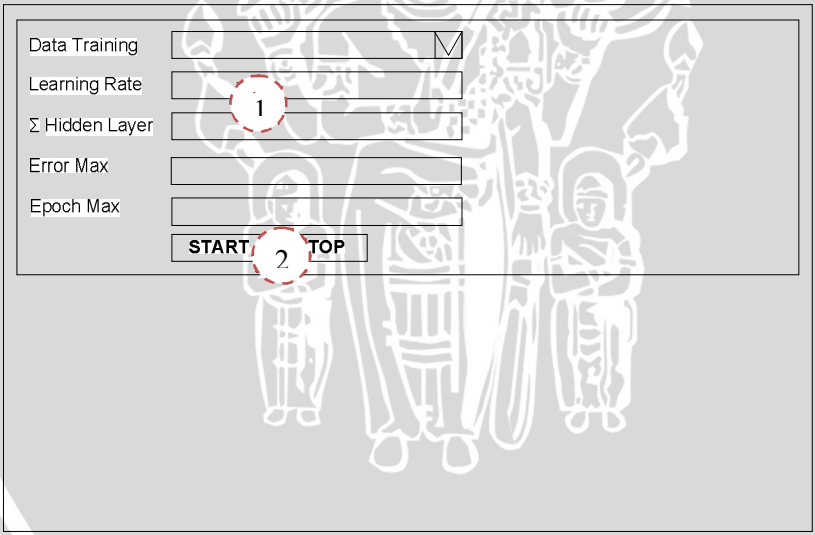
Gambar 3.10 merupakan antarmuka *Training Jaringan* . Pada antarmuka ini hanya terdapat parameter input (ditunjukkan dengan no 1) dan tombol untuk memulai dan menghentikan proses training(ditunjukkan dengan no 2).



Gambar 3.8 Tampilan Rancangan Antarmuka Data Training



Gambar 3.9 Tampilan Rancangan Antarmuka Validasi Data



Gambar 3.10 Tampilam Rancangan Antarmuka *Form Training*

BAB IV
IMPLEMENTASI DAN PEMBAHASAN

4.1 Lingkungan Implementasi

Lingkungan implementasi yang akan dijelaskan dalam sub bab ini adalah lingkungan implementasi perangkat keras dan perangkat lunak.

4.1.1. Lingkungan Perangkat Keras

Perangkat keras yang digunakan dalam pengembangan perangkat lunak ini adalah :

1. AMD Turion Neo X2 *Dual Core Processor* L625 @1,6 Ghz
2. Memori 4 GB
3. Harddisk 320 GB
4. Monitor 14'
5. Keyboard
6. Mouse

4.1.2. Lingkungan Perangkat Lunak

Perangkat lunak yang digunakan dalam pengembangan aplikasi *optical backpropagation neural network (OBPNN)* ini adalah :

1. Sistem operasi Windows 7 64 bit
2. Microsoft Visual Studio Express 2010

4.2 Implementasi Program

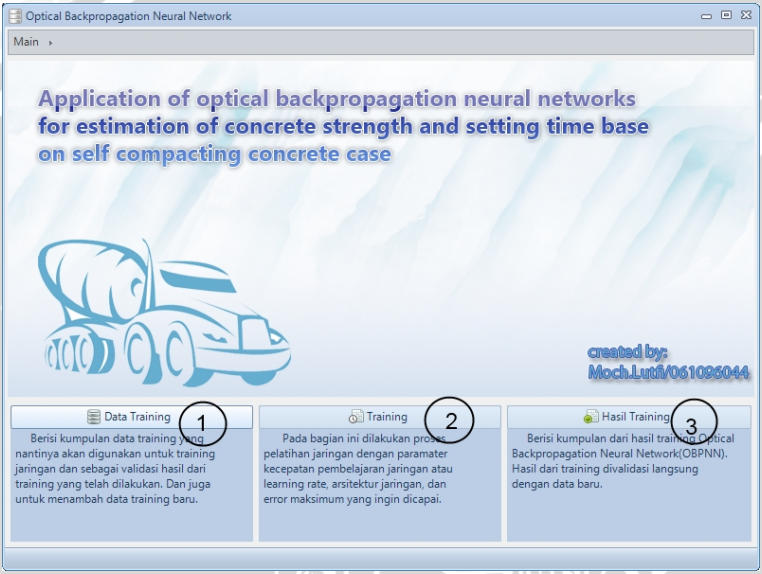
Pada subbab implementasi program ini akan dijelaskan mengenai implementasi dari rancangan perangkat lunak yang dijelaskan sebelumnya pada subbab 3.3.

4.2.1 Implementasi antarmuka

Implementasi antarmuka dari sistem terdiri dari 4 *form* yaitu *form* Utama yang berisi navigasi untuk menuju ke tiga *form* lainnya, *form* data *training* yaitu *form* yang mengatur *input output* data yang akan digunakan dalam pelatihan jaringan, *form training* yang mengatur data dan parameter-parameter yang digunakan pada proses *training* jaringan, dan yang terakhir yaitu *form* hasil *training* yang menangani validasi jaringan setelah proses *training*.

1. *Form* Utama

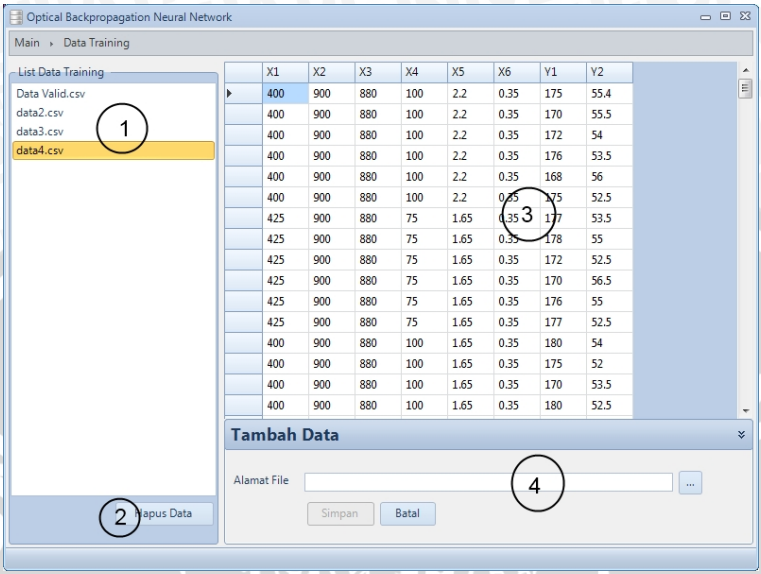
Form utama merupakan *form* yang pertama kali ditampilkan pada saat program dijalankan. Pada *form* ini terdapat 3 tombol untuk berpindah ke *form* yang lain yaitu tombol data *training*, tombol *training*, dan tombol hasil *training* secara berturut-turut ditunjukkan oleh nomor 1, 2, dan 3 pada Gambar 4.1.



Gambar 4.1 Tampilan utama program

2. *Form* Data *Training*

Form *Data* Training adalah *form* yang menangani proses tambah data *training*, hapus data *training,* dan juga menampilkan data training dalam bentuk tabel. *Form* utama diilustrasikan pada Gambar 4.2. Bagian yang ditunjukkan oleh nomor 1 pada Gambar 4.2 merupakan daftar nama file yang telah disimpan, bagian nomor 2 adalah tombol untuk menghapus dataset yang telah disimpan, bagian nomor 3 adalah tampilan data dari daftar nama file yang dipilih, dan pada bagian no 4 untuk menambahkan dataset baru kedalam media penyimpanan.



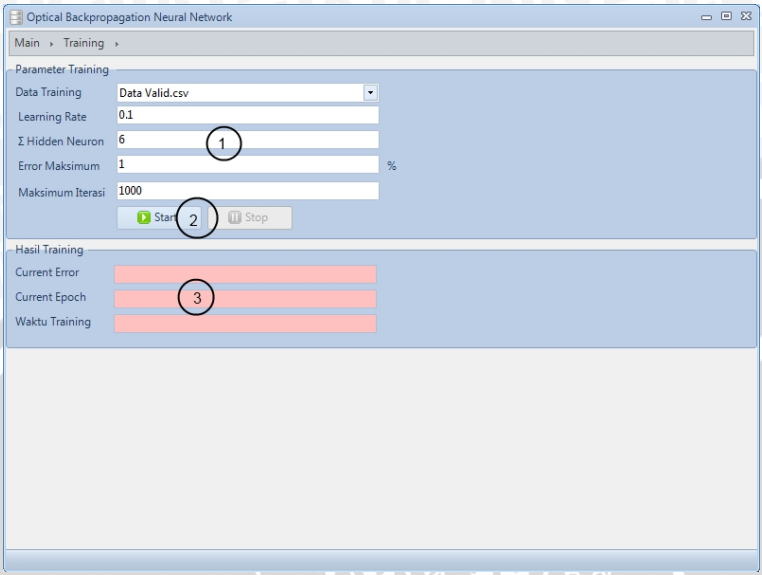
Gambar 4.2 Tampilan *Form* Data Training

3. *Form Training*

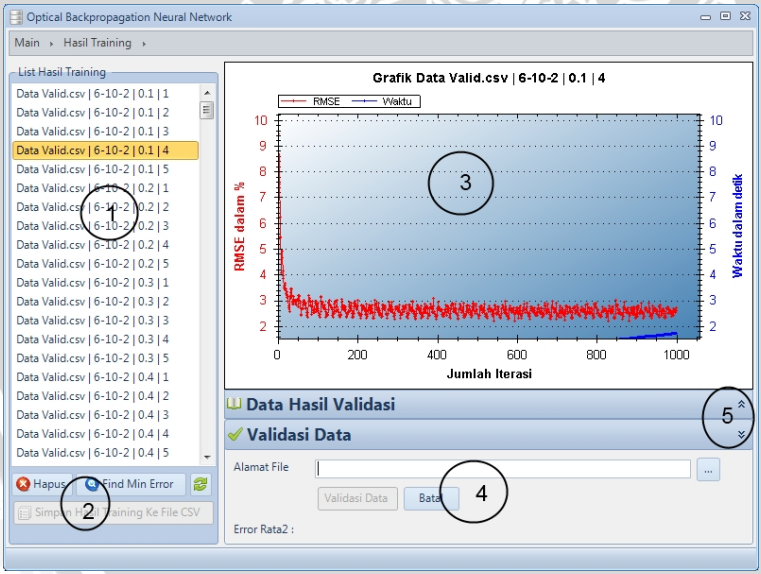
Form data *training* menangani proses pelatihan jaringan dan penyimpanan hasil dari pelatihan jaringan ke dalam media penyimpanan. Pada *form* ini terbagi menjadi 2 bagian tampilan, pada bagian atas menampilkan parameter training(ditunjukkan nomor 1 pada Gambar 4.3) , tombol *start* dan *stop*(ditunjukkan nomor 2 pada Gambar 4.3). Keterangan proses *training* ditunjukkan oleh nomor 3 pada Gambar 4.3.

4. *Form* Hasil *Training*

Form hasil *training* menangani proses validasi hasil training dan juga untuk menampilkan grafik hasil training. Pada bagian sebelah kiri(bagian nomor 3 pada Gambar 4.4) terdapat daftar hasil pengujian dan dibawahnya(ditunjukkan dengan no.2) merupakan tombol untuk hapus data, *refresh* daftar hasil pengujian, tombol pencarian training error minimum, dan tombol untuk menyimpan hasil proses validasi jaringan. Grafik ditunjukkan dengan no.3 dan bagian untuk validasi hasil training ditunjukkan oleh no 4 pada Gambar 4.4. Pada bagian no.5 merupakan tombol untuk melihat tabel hasil validasi.



Gambar 4.3 Tampilan *Form Training*



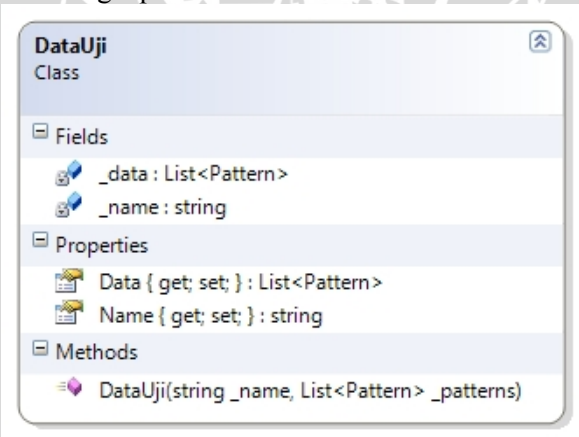
Gambar 4.4 Tampilan *Form* Hasil Training

4.2.2 Implementasi kelas

Pada implementasi program, daftar program dibuat secara modular yang diimplementasikan dalam bentuk kelas-kelas. Pada pembangunan sistem ini terdapat 2 kelompok kelas dengan fungsional yang berbeda, yang pertama yaitu untuk penanganan berkas yang terdiri dari kelas `DataUji`, kelas `HasilUji`, kelas `DataRepository`, dan kelas `NetworkRepository`. Kelompok kelas yang kedua adalah yang menangani proses proses jaringan syaraf tiruan *optical backpropagation,* dalam kelompok kelas ini terdiri dari kelas `Layer`, kelas `Network`, kelas `Neuron`, kelas `Pattern`, dan kelas `Weight`.

1. Kelas `DataUji`

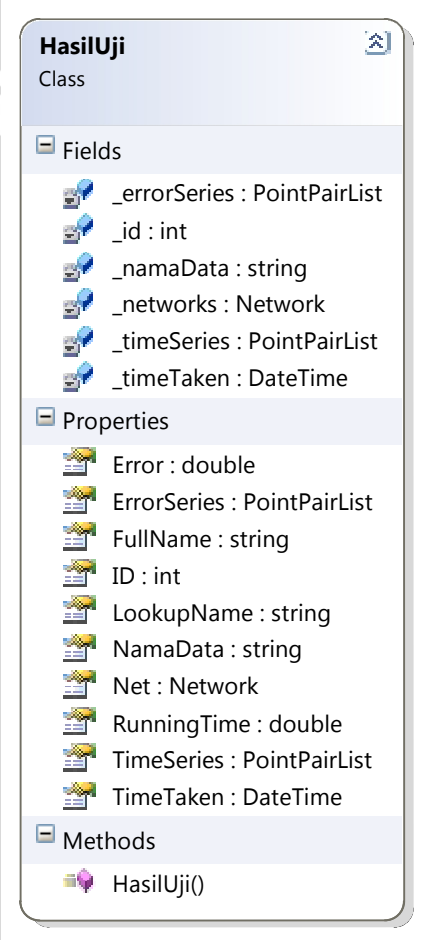
Kelas ini berfungsi sebagai wadah untuk penampung data uji yang direpresentasikan sebagai *list* dari object kelas `Pattern` dan menyimpan nama berkas data training. Pada kelas ini tidak mempunyai method selain konstruktor, karena pada dasarnya kelas `DataUji` ini sebagai satuan data yang akan diolah dalam kelas `DataRepository`. Pada Gambar 4.5 ditunjukkan diagram kelas `DataUji` secara lengkap.



Gambar 4.5 Kelas Diagram dari kelas `HasilUji`

2. Kelas `HasilUji`

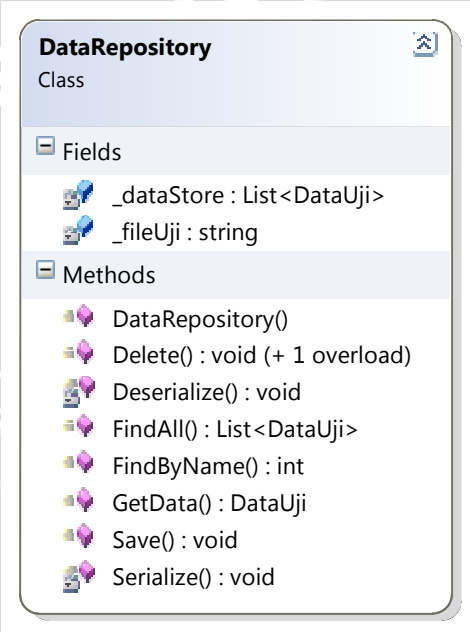
Sama halnya dengan kelas `DataUji`, pada kelas ini pun juga berfungsi sebagai penampung. Namun pada kelas ini menampung data hasil *training*, *list* error, dan *list* waktu proses. Kelas ini digunakan sebagai objek penyimpanan yang diproses dalam kelas `NetworkRepository`. Pada Gambar 4.6 ditunjukkan diagram kelas `DataUji`.



Gambar 4.6 Kelas Diagram kelas `DataUji`

3. Kelas `DataRepository`

Kelas ini berfungsi untuk menyimpan, menghapus, dan mengambil data uji. Data yang disimpan kedalam berkas merupakan data yang merupakan serialisasi objek kelas `DataUji`. Untuk penyimpanan data *training* jika terdapat nilai nama data *training* yang sama maka otomatis di *overwrite*. Diagram kelas `DataRepository` ditunjukkan pada Gambar 4.7.



Gambar 4.7 Kelas Diagram dari kelas `DataRepository`

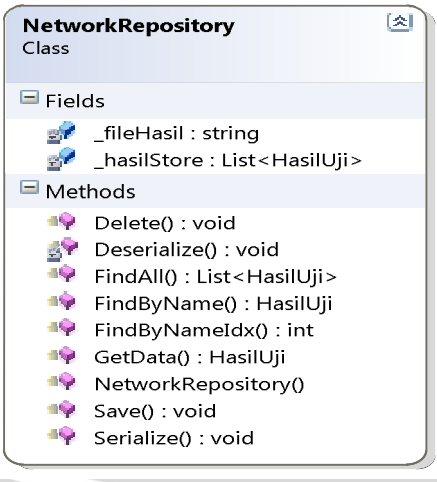
Semua fungsi/procedure dalam kelas ini dideskripsikan pada Tabel 4.1. Data yang akan disimpan perlu ditambahkan kedalam `_dataStore` setelah itu dilakukan proses serialisasi objek dan disimpan kedalam berkas. Objek yang disimpan kedalam berkas adalah `_dataStore`. Untuk pengambilan data dari berkas dilakukan proses deserialisasi. Hasil deserialisasi disimpan kedalam peubah `_dataStore`.

Tabel 4.1 Deskripsi Fungsi Kelas `DataRepository`

Nama Prosedur/Fungsi	Deskripsi
`public List<DataUji> FindAll()`	Fungsi untuk mendapatkan *list* `DataUji` yang telah disimpan
`public void Delete(DataUji data)`	Prosedur untuk menghapus data yang telah disimpan
`public DataUji GetData(int idx)`	Fungsi untuk mengambil data berdasarkan dari indeks data dalam *list* data
`public void Save(DataUji data)`	Prosedur untuk menyimpan `data`

4. Kelas `NetworkRepository`

Kelas ini digunakan sebagai pengatur data hasil *training* untuk menyimpan, menghapus, dan mengambil data hasil *training*. Sama halnya dengan DataRepository pada kelas ini pun data yang disimpan berupa serialisasi dari objek. Perbedaannya dengan `DataRepository` adalah pada saat penyimpanan datanya, pada kelas ini jika terdapat nama hasil training yang sama maka akan diberi penomoran. Pada Gambar 4.8 ditunjukkan kelas diagram `NetworkRepository`.



Gambar 4.8 Kelas Diagram dari kelas `NetworkRepository`

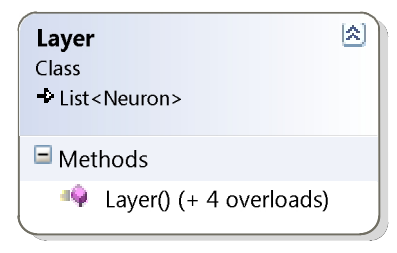
Deskripsi fungsi/prosedur dalam kelas ini ditunjukkan pada Tabel 4.2. Secara fungsional kelas ini dengan kelas `DataRepository` adalah sama. Perbedaanya terletak pada tipe data *list* dari data yang disimpan.

Tabel 4.2 Deskripsi Fungsi Kelas `NetworkRepository`

Nama Prosedur/Fungsi	Deskripsi
`public List<HasilUji> FindAll()`	Fungsi untuk mendapatkan *list* `HasilUji` yang telah disimpan
`public void Delete(DataUji data)`	Prosedur untuk menghapus data yang telah disimpan
`public DataUji GetData(int idx)`	Fungsi untuk mengambil data berdasarkan dari indeks data dalam *list* data
`public void Save(DataUji data)`	Prosedur untuk menyimpan `data`
`public int FindByNameIdx(string name)`	Mencari indeks data dalam list berdasarkan nama data
`public int FindByName(string name)`	Mencari data dalam list berdasarkan nama data

5. Kelas `Layer`

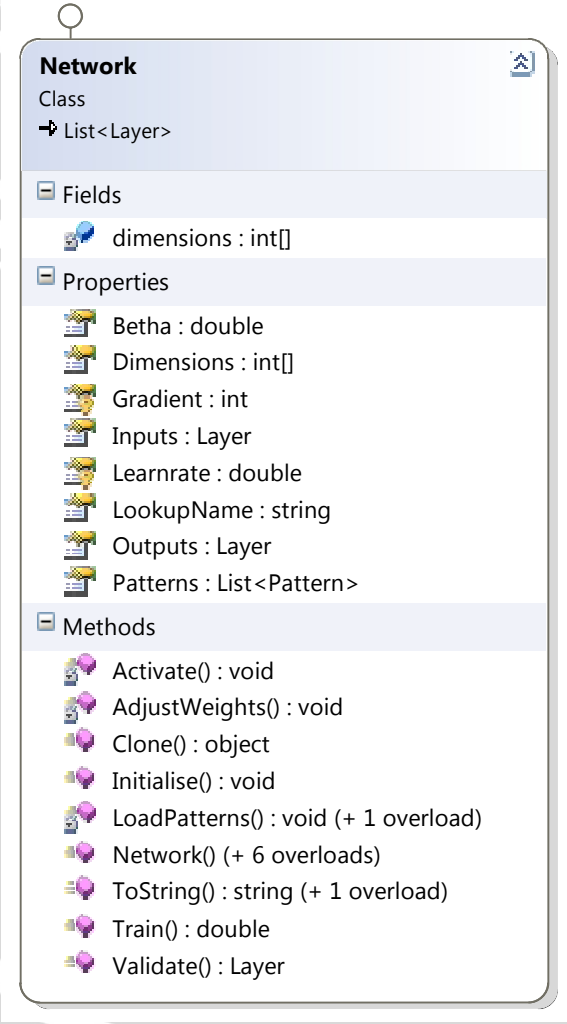
Kelas ini merupakan turunan dari *list* kelas `Neuron` dan hanya memiliki konstruktor. Konstruktor dalam kelas **`layer`** berfungsi untuk menambahkan *neuron-neuron* kedalam *layer*. Gambar 4.9 menunjukkan kelas diagram dari kelas `Layer`.



Gambar 4.9 Kelas Diagram dari kelas `Layer`

6. Kelas `Network`

Kelas ini merupakan kelas yang mengatur jalannya proses *training*. Kelas `Network` merupakan turunan dari *list* kelas `Layer`. Gambar 4.10 menunjukkan kelas diagram dari kelas `Network`. Dalam kelas ini tidak banyak terdapat proses komputasi, karena proses komputasi lebih banyak dilakukan didalam kelas `Neuron`.



Gambar 4.10 Kelas Diagram dari kelas `Network`

Deskripsi fungsi dan prosedur ditunjukkan pada Tabel 4.3. Pada Tabel 4.3 terdapat fungsi kloning kelas `Network` untuk menghindari *copy* objek berdasarkan alamat dalam *memory* pada saat penyimpanan objek kelas `Network` kedalam berkas.

Tabel 4.3 Deskripsi Fungsi Kelas `Network`

Nama Prosedur/Fungsi	Deskripsi
`public override string ToString()`	Untuk mengembalikan nilai string yang berisi nilai dari peubah `LookupName`
`private void LoadPatterns(List<Pattern> patterns)`	Untuk *load list* `Pattern` yang akan digunakan dalam *training*
`public object Clone()`	Untuk membuat kloning kelas `Network`
`public double Train()`	Fungsi untuk menjalankan *training* jaringan dengan *return value* berupa SSE(*Sum Square Error*)
`private void AdjustWeights()`	Prosedur untuk memperbarui bobot dalam jaringan.
`private void Activate(Pattern pattern)`	Prosedur untuk aktivasi *neuron-neuron* dalam jaringan.
`public Layer Validate(Pattern p)`	Fungsi untuk validasi jaringan setelah dilakukan proses training

Semua proses dalam *Optical Backpropagation Neural Network*(OBPNN) dirangkum dalam satu fungsi dalam kelas `Network`, yaitu fungsi `Train()`. *Source code* dari fungsi `Train()` ditunjukkan pada Gambar 4.11. Dalam fungsi `Train()` melibatkan prosedur `Activate(Pattern pattern)` yang melakukan proses *feedforward*(propagasi maju) dan `AdjustWeights()` merupakan proses propagasi balik untuk memperbarui bobot-bobot jaringan. Nilai kembalian dari fungsi `Train()` berupa SSE(*Sum Square Error*).

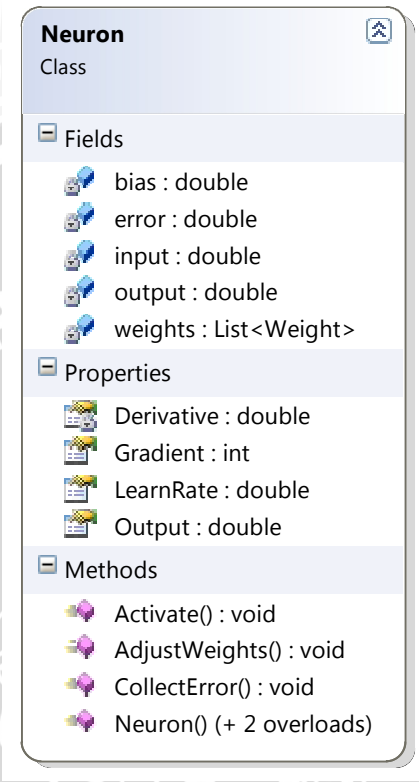
```
1   public double Train()
2   {
3       double error = 0;
4       double delta = 0;
5       double newDelta = 0;
6       foreach (Pattern pattern in Patterns)
7       {
8           Activate(pattern);
9           for (int i = 0; i < Outputs.Count; i++)
10          {
11              delta =
12              pattern.NormOutputs[i] - Outputs[i].Output;
13              newDelta = 1.0 + Math.Exp(delta * delta);
14
15              if (delta < 0)
16                  newDelta = -1 * newDelta;
17
18              Outputs[i].CollectError(newDelta);
19              error += Math.Pow(delta, 2);
20          }
21          AdjustWeights();
22      }
23      return error;
24  }
```

Gambar 4.11 *Source code* fungsi `Train` dalam kelas `Network`

7. Kelas `Neuron`

Proses komputasi dari proses *training* pada OBPNN ditangani oleh kelas `Neuron`. Proses tersebut meliputi aktivasi, pengumpulan *error,* dan *update* bobot. Konstruktor kelas `Neuron` merupakan proses inisialisasi dengan metode Nguyen-Widrow seperti yang telah dijelaskan pada subbab 2.3.6.

Pada Gambar 4.12 ditunjukkan kelas diagram dari kelas `Neuron`. Dalam kelas neuron memiliki peubah dengan tipe data `List<Weight>` yang berfungsi untuk menyimpan bobot dari *neuron* lain yang terhubung dengan *neuron*. Derivative dan Output merupakan peubah khusus yang pada saat pengambilan nilainya dilakukan proses komputasi terlebih dahulu. Untuk lebih jelasnya tentang prosedur dan fungsi yang digunakan pada kelas ini dideskripsikan pada Tabel 4.4.



Gambar 4.12 Kelas Diagram dari kelas `Neuron`

Tabel 4.4 Deskripsi Fungsi Kelas Neuron

Nama Prosedur/Fungsi	Deskripsi
`public void Activate()`	Prosedur untuk menjumlahkan perkalian antara
`public void AdjustWeights()`	Prosedur untuk *update* bobot.
`public void CollectError(double delta)`	Mengumpulkan error keseluruhan
`private double Derivative`	Merupakan turunan dari fungsi aktivasi
`public double Output`	Merupakan nilai hasil dari aktivasi neuron

a. Prosedur `Activate`

Prosedur ini pada dasarnya untuk mendapatkan besarnya nilai input pada neuron dengan cara mengalikan nilai bobot yang terhubung dengan nilai aktivasi/output neuron asal. *Source code* dari prosedur ini ditunjukkan pada Gambar 4.13.

```
1  public void Activate()
2  {
3      error = 0;
4      input = 0;
5      foreach (Weight w in weights)
6      {
7          input += w.Value * w.Input.Output;
8      }
   }
```

Gambar 4.13 *Source code* Prosedur `Activate`

b. Prosedur `CollectError`

Proses pengumpulan error berlangsung secara rekursif. Proses ini berfungsi untuk menghitung error yang nantinya akan digunakan dalam *update* bobot pada jaringan. Source code prosedur ini ditunjukkan pada

```
1  public void CollectError(double delta)
2  {
3      if (weights != null)
4      {
5          error += delta;
6          foreach (Weight w in weights)
7          {
8              w.Input.CollectError(error * w.Value);
9          }
10     }
   }
```

Gambar 4.14 *Source code* Prosedur `CollectError`

c. Prosedur `AdjustWeight`

Prosedur ini merupakan kelanjutan dari proses CollectError, setelah total error diketahui maka dilakukan pembaruan bobot. Proses pembaruan bobot secara matematis ditunjukkan pada persamaan 2.22 dan dituangkan dalam kode program seperti pada Gambar 4.15.

```
1  public void AdjustWeights()
2  {
3      for (int i = 0; i < weights.Count; i++)
4      {
5          weights[i].Value += error * Derivative *
   LearnRate * weights[i].Input.Output;
6      }
7      bias += error * Derivative * LearnRate;
8  }
```

Gambar 4.15 *Source code* Prosedur `AdjustWeight`

d. *Property* `Derivative`

Pada dasarnya property merupakan versi singkat dari penulisan *set get* dalam program. Namun *property* ini hanya memiliki *get method* yang merupakan turunan dari fungsi aktivasi neuron(lihat persamaan 2.16). Source code *property* Derivative ditunjukkan pada Gambar 4.16.

```
1  private double Derivative
2  {
3      get
4      {
5          double activation = Output;
6          return activation * (1 - activation);
7      }
8  }
```

Gambar 4.16 *Source code Property* `Derivative`

e. *Property* `Output`

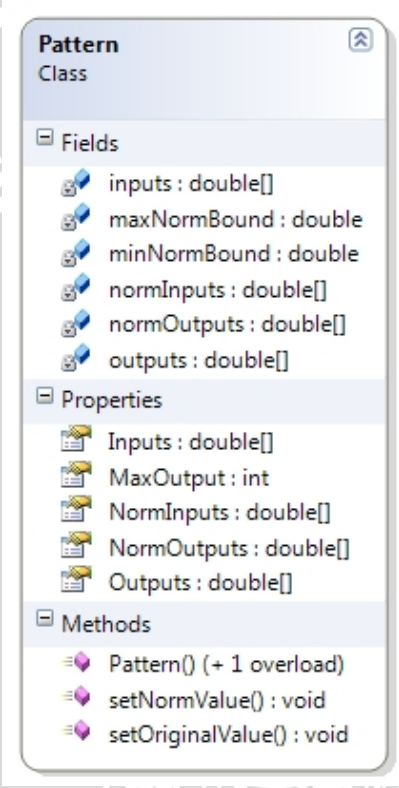
Sama halnya *property* `Derivative`, pada *get method* *property* Output terjadi proses aktivasi sigmoid tetapi sebelum itu dilakukan pengecekan terlebih dahulu apakah nilai peubah `output` sudah diisi nilai dari proses aktivasi atau belum. Jika belum maka *return value* dari *get method* adalah hasil perhitungan dari fungsi aktivasi. Source code property ini ditunjukkan pada Gambar 4.17.

```
1  private double Derivative
2  {
3      get
4      {
5          double activation = Output;
6          return activation * (1 - activation);
7      }
8  }
```

Gambar 4.17 Source code *property* `Output`

8. Kelas `Pattern`

Kelas `Pattern` menangani proses pengolahan awal data, karena tipe data masukan untuk proses *training* berupa `Pattern` maka semua data dari harus dikonversi menjadi `Pattern` agar bisa dilakukan *training*. Kelas diagram ditunjukkan pada Gambar 4.18.



Gambar 4.18 Kelas Diagram dari kelas `Pattern`

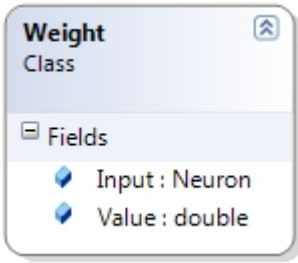
Deskripsi fungsi kelas `Pattern` ditunjukkan pada Tabel 4.5. Prosedur `setNormValue` digunakan untuk melakukan proses normalisasi input dan output dari kelas `Pattern`. Prosedur `setOriginalValue` digunakan untuk mengembalikan data ternormalisasi kedalam bentuk awal. Proses normalisasi dan denormalisasi ini hanya untuk menangani tiap pattern, bukan normalisasi/denormalisasi keseluruhan data training.

Tabel 4.5 Deskripsi Fungsi Kelas Pattern

Nama Prosedur/Fungsi	Deskripsi
`public void setOriginalValue(MaxMin[] boundaryInputs, MaxMin[] boundaryOutputs)`	Untuk merubah data input ternormalisasi menjadi bentuk asli/tidak ternormalisasi
`public void setNormValue(MaxMin[] boundaryInputs, MaxMin[] boundaryOutputs)`	Untuk merubah data input menjadi data ternormalisasi

9. Kelas `Weight`

Kelas ini berisi struktur data untuk menyimpan bobot dari jaringan syaraf tiruan. Pada kelas ini terdapat variable `Value` yang bertipe *double* dan peubah `Input` bertipe data `Neuron`. Kelas diagram kelas Weight ini ditunjukkan pada Gambar 4.19.



Gambar 4.19 Kelas Diagram dari kelas `Weight`

4.3 Implementasi Uji Coba

Pada subbab ini akan dilakukan pembahasan mengenai pengujian yang telah dilakukan pada sistem dan hasil evaluasi dari ringkasan hasil sistem.

4.3.1 Hasil Uji

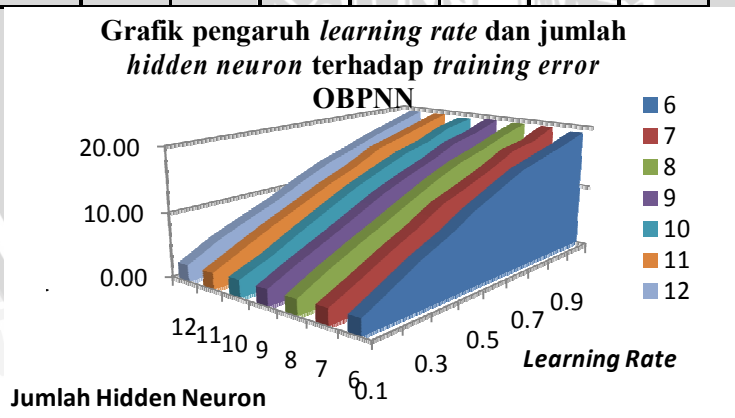
Hasil pengujian yang dilakukan terhadap *data training* dengan variasi jumlah *neuron*/*hidden unit* antara 6 sampai 12 *neuron* dan *learning rate (α)* antara 0.1 sampai dengan 1. Data yang digunakan dalam proses *training* menggunakan data pada Lampiran 1 dan untuk validasi hasil *training* menggunakan data pada Lampiran 2(data2.csv) dan 3(data3.csv). Untuk data hasil pengujian terdapat pada Lampiran 4.

Nilai *error* pada *training Optical Backpropagation Neural Network*(OBPNN) ditunjukkan pada Tabel 4.6. Nilai *error* pada hasil pengujian tidak mengalami perubahan yang signifikan apabila dilakukan pada nilai *learning rate* yang sama meskipun jumlah *hidden neuron* berubah.

Dengan bertambahnya nilai *learning rate* pada jumlah *hidden neuron* yang sama, *error* yang dihasilkan pada saat training semakin meningkat. Untuk lebih jelasnya, hasil pengujian pada Tabel 4.6 direpresentasikan dalam bentuk grafik pada Gambar 4.20.

Tabel 4.6 Hasil pengujian *training error* OBPNN

Learning Rate	**Jumlah *Hidden* Neuron**						
	6	**7**	**8**	**9**	**10**	**11**	**12**
0.1	2.52	2.53	2.46	2.56	2.70	2.50	2.56
0.2	4.85	4.70	4.96	4.93	4.67	4.93	5.17
0.3	7.27	7.08	7.18	7.06	7.12	7.22	7.18
0.4	9.19	9.20	9.24	9.27	9.41	9.30	9.04
0.5	11.58	11.65	11.42	11.53	11.67	11.44	11.36
0.6	13.53	13.17	13.49	13.32	13.50	13.24	13.37
0.7	15.40	14.86	15.26	14.93	15.27	15.52	14.92
0.8	16.53	16.83	16.45	16.85	16.44	16.73	16.54
0.9	17.56	17.60	17.98	17.86	18.23	18.06	17.92
1	18.82	19.20	19.19	19.28	19.02	19.31	19.29



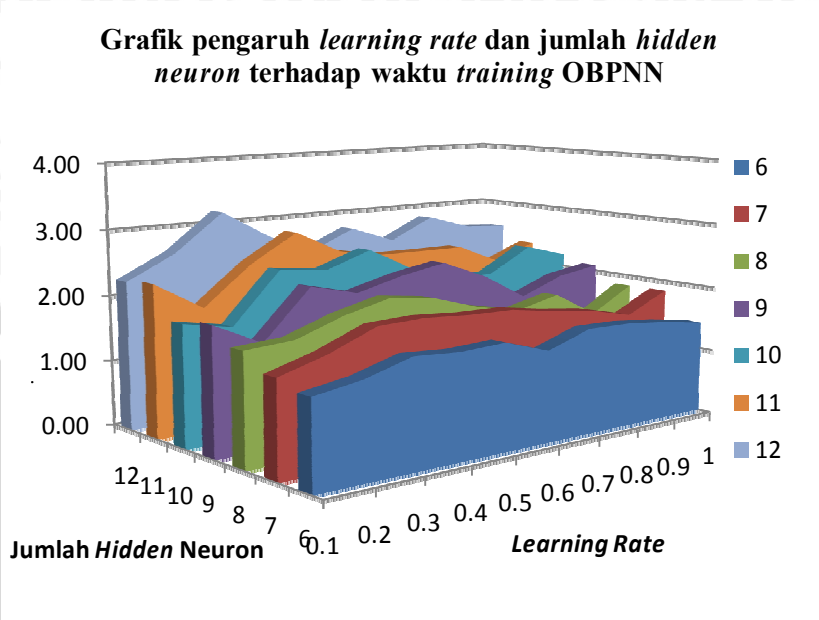
Gambar 4.20 Grafik error hasil *training* OBPNN

Gambar 4.20 menunjukkan bahwa pada semua jumlah neuron yang diujikan menghasilkan nilai *error* pada *training* OBPNN yang hampir sama. Selain itu pada nilai *learning rate* 0.1 mampu menghasilkan nilai *error* yang paling kecil pada pengujian ini yaitu rata-rata *error* 2,5% dan semakin besar nilai *learning rate* diikuti semakin besar *error* yang dihasilkan.

Untuk pengujian lamanya waktu yang dibutuhkan OBPNN untuk proses *training* ditunjukkan pada Tabel 4.7. Lama proses training pada hasil pengujian cenderung mempunyai nilai yang tidak terpaut jauh apabila dilakukan dengan jumlah *hidden neuron* yang sama. Gambar 4.21 menunjukkan bahwa untuk setiap kenaikan jumlah *hidden neuron* juga diikuti dengan bertambahnya lama proses training.

Tabel 4.7 Hasil pengujian waktu *training* OBPNN

Learning Rate	**Jumlah *Hidden* Neuron**						
	6	**7**	**8**	**9**	**10**	**11**	**12**
0.1	1.30	1.43	1.68	1.93	1.83	2.32	2.27
0.2	1.41	1.63	1.72	1.57	1.70	1.88	2.62
0.3	1.62	1.92	1.97	2.30	2.46	2.46	3.19
0.4	1.57	1.94	2.14	2.12	2.37	2.88	2.76
0.5	1.60	1.89	2.04	2.30	2.61	2.52	2.44
0.6	1.38	1.88	1.83	2.42	2.25	2.40	2.75
0.7	1.60	1.77	1.68	2.13	2.02	2.39	2.50
0.8	1.59	1.69	1.84	1.73	2.02	2.38	2.82
0.9	1.52	1.53	1.45	1.98	2.39	2.08	2.61
1	1.41	1.83	1.83	2.09	2.20	2.32	2.56



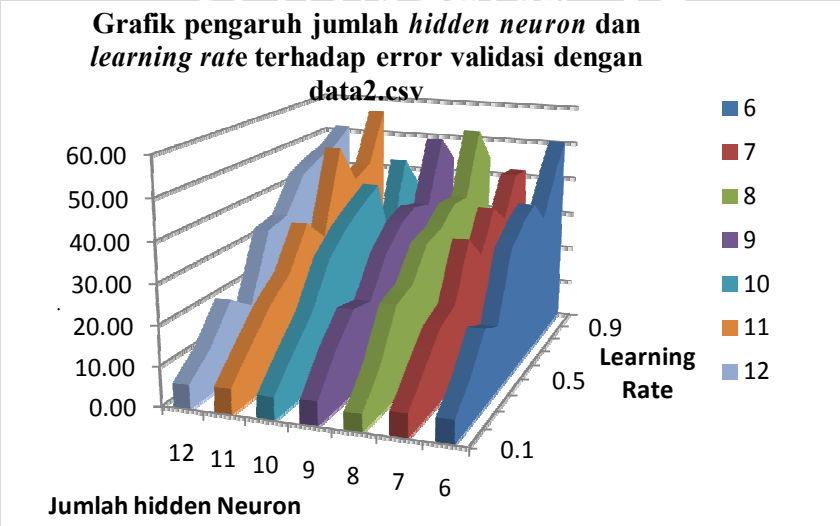
Gambar 4.21 Grafik kecepatan training OBPNN

Pengujian selanjutnya yaitu validasi hasil training dengan dengan data set baru. Proses validasi yang pertama menggunakan data2.csv sebagai data set untuk validasi dan validasi yang kedua menggunakan data3.csv. Hasil pengujian *error* pada validasi hasil training dengan data2.csv ditunjukkan pada Tabel 4.8 dan validasi hasil training dengan dengan data3.csv ditunjukkan pada Tabel 4.9.

Gambar 4.22 menunjukkan representasi grafik dari hasil error pada validasi hasil training dengan data2.csv. Grafik tersebut memiliki kemiripan bentuk dengan grafik pada Gambar 4.23. Pada setiap jumlah layer yang sama kedua grafik tersebut memliki kemiripan pola yang dihasilkan. Tetapi secara umum hampir serupa dengan grafik error yang dihasilkan pada saat proses training yang ditunjukkan pada Gambar 4.20. Berbeda dengan *error* yang dihasilkan pada saat *training, error* yang dihasilkan pada proses validasi ini peningkatannya sangat besar bahkan pada *learning rate* 0.1 mencapai 50% lebih. Karena pada saat training banyaknya *hidden layer* tidak terlalu berpengaruh terhadap *error* yang dihasilkan pada saat training maka pada proses validasi pun *error* yang dihasilkan tidak terlalu dipengaruhi oleh jumlah *hidden neuron.*

Tabel 4.8 Error validasi hasil *training* dengan data2.csv

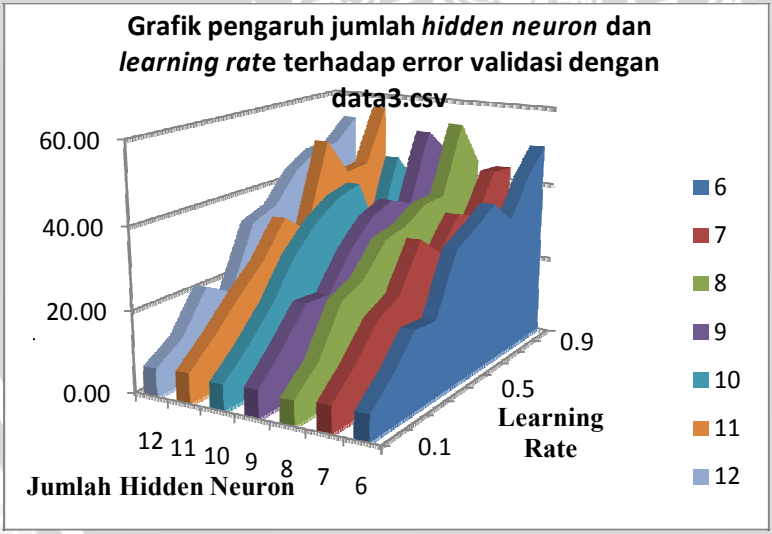
Learning Rate	Jumlah *Hidden* Neuron						
	6	7	8	9	10	11	12
0.1	5.99	6.07	4.55	6.14	5.81	6.46	6.08
0.2	11.86	12.72	11.87	14.72	12.69	13.46	11.46
0.3	19.73	18.79	23.35	21.11	19.11	20.24	20.31
0.4	16.83	20.86	25.07	19.91	29.72	24.96	15.55
0.5	32.97	34.55	32.67	29.18	34.88	35.25	32.63
0.6	37.02	27.88	33.96	34.41	38.36	29.03	33.95
0.7	39.29	37.49	37.79	36.37	41.52	50.89	42.82
0.8	33.08	32.24	37.28	34.49	22.82	40.15	45.55
0.9	42.35	41.91	53.59	50.76	44.16	42.53	47.07
1	50.14	41.21	44.30	43.58	35.54	55.89	50.98



Gambar 4.22 Grafik error validasi hasil training OBPNN dengan data2.csv

Tabel 4.9 Error validasi hasil *training* dengan data3.csv

Learning Rate	Jumlah *Hidden* Neuron						
	6	7	8	9	10	11	12
0.1	6.50	6.72	6.09	6.87	6.43	7.43	6.93
0.2	12.24	12.96	12.06	13.83	12.61	13.48	11.40
0.3	19.53	20.41	23.46	21.60	19.88	20.41	21.57
0.4	18.90	23.56	26.07	21.05	29.55	26.56	17.60
0.5	33.38	34.61	33.35	29.85	35.73	35.50	33.26
0.6	36.37	29.10	34.55	35.74	39.84	30.31	35.64
0.7	40.15	36.63	38.07	38.07	41.30	51.64	43.10
0.8	34.20	33.91	38.42	35.43	25.58	41.88	46.91
0.9	42.94	43.72	55.31	52.62	45.04	42.55	47.49
1	49.63	43.19	44.20	46.11	36.92	56.41	53.01

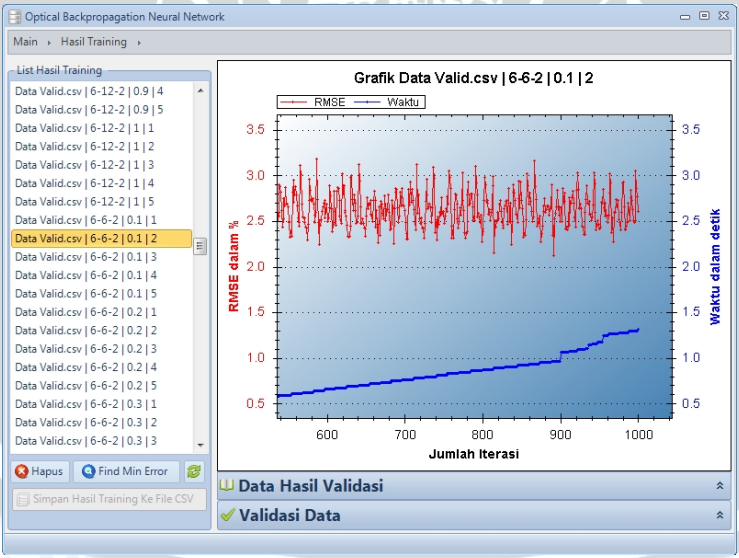


Gambar 4.23 Grafik error validasi hasil training OBPNN dengan data3.csv

4.3.2 Analisa Hasil

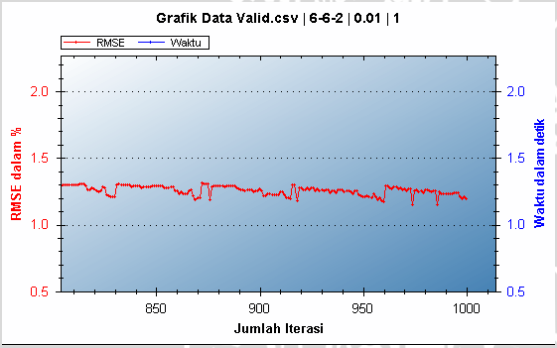
1. Analisa Pengaruh *Learning Rate* Terhadap *Training Error*

Pada dasarnya *learning rate* digunakan untuk merubah nilai bobot jaringan syaraf tiruan pada saat proses *training*. Semakin besar nilai *learning rate* maka semakin besar pula perubahan bobot tiap siklus proses training (*epoch*). Dan pada jaringan syaraf tiruan propagasi balik pada umumnya akan menghasilkan error training yang diperoleh dari hasil selisih nilai target dengan nilai hasil perhitungan seperti yang ditunjukkan pada Persamaan 2.16. Error yang dihasilkan tersebut akan selalu bernilai positif karena merupakan selisih target dengan hasil perhitungan jaringan. Sedangkan pada *Optical Backpropagation Neural Network*(OBPNN), nilai error yang dihasilkan bukan selisih melainkan pengurangan antara output target dengan output hasil perhitungan jaringan. Kemudian dilakukan perhitungan lagi seperti yang ditunjukkan pada Persamaan 2.28 dan 2.29. Dengan demikian pada Δ bisa bernilai positif ataupun negative sehingga pada *epoch* selanjutnya nilai *error* yang dihasilkan bisa bertambah atau berkurang. Sehingga dimungkinkan dihasilkan grafik error yang naik turun pada setiap *epoch*. Contoh nilai *error training* yang tidak stabil ditunjukkan pada Gambar 4.24.

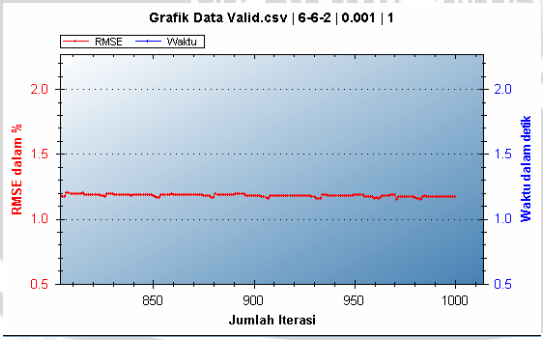


Gambar 4.24 Grafik *training error* dengan learning rate 0.1 dan 6 *hidden neuron*

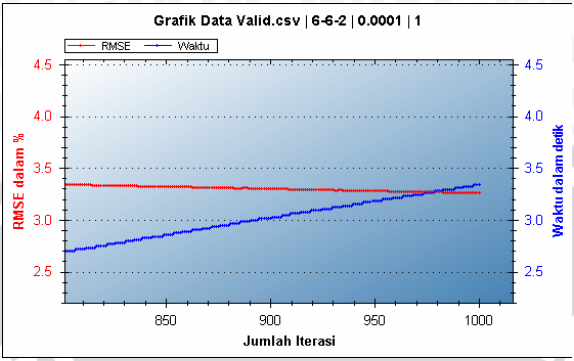
Pada pengujian ini tidak ada satupun nilai error yang konvergen, semua hasil pengujian seperti pada Gambar 4.24. Ini disebabkan prinsip dasar OBPNN adalah dengan memperbesar nilai error output untuk memperbarui bobot agar lebih cepat konvergen dan juga pada pengujian ini nilai *learning rate* 0.1 sampai dengan 1 ternyata masih terlalu besar sehingga konvergensi tidak dapat dicapai. Selain itu faktor peubah bobot Δ yang bisa bernilai negatif juga sangat berpengaruh pada hasil *error* yang membentuk pola *zig-zag*. Jadi diperlukan nilai *learning rate* yang lebih kecil untuk memperoleh nilai *error training* konvergen, untuk mendukung pernyataan tersebut dilakukan pengujian dengan nilai *learning rate* 0.01, 0.001 dan 0.0001 untuk melihat seberapa konvergen hasil *error* yang didapat. Grafik yang menunjukkan konvergensi ketiga pengujian tersebut secara berturut-turut ditunjukkan pada Gambar 4.25, Gambar 4.26, dan Gambar 4.27.



Gambar 4.25 Grafik *error training* dengan *learning rate* 0.01 dan *hidden neuron* 6



Gambar 4.26 Grafik error training dengan learning rate 0.001 dan *hidden neuron* 6

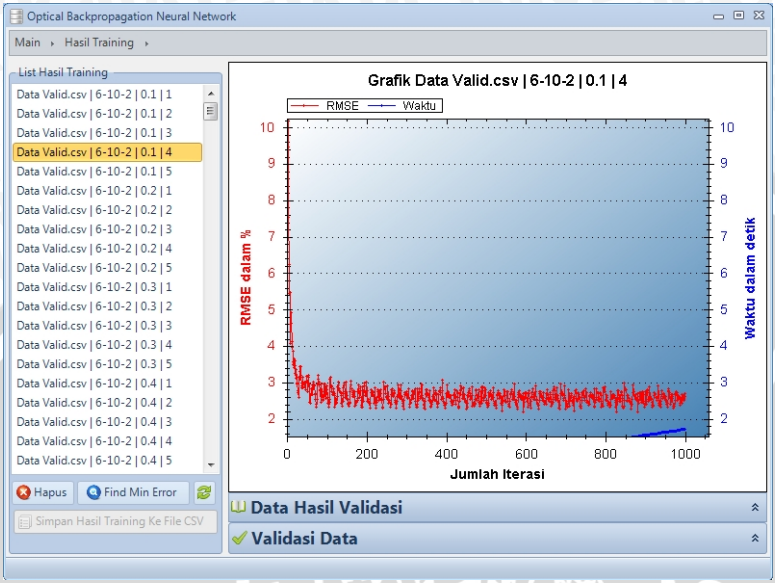


Gambar 4.27 Grafik *error training* dengan *learning rate* 0.0001 dan *hidden neuron* 6

Dari Gambar 4.25, Gambar 4.26, dan Gambar 4.27 bahwa semakin kecil nilai *learning rate* maka akan berpengaruh juga terhadap konvergensinya. Apabila *learning rate* semakin kecil maka jumlah *epoch* yang dibutuhkan untuk mencapai error yang diinginkan juga akan bertambah. Semakin kecil nilai *learning rate* akan membuat perubahan bobot jaringan semakin kecil sehingga tidak terjadi perubahan nilai error yang ekstrim seperti pada pengujian dengan *learning rate* 0.1 sampai 1.

2. Analisa Pengaruh Jumlah *Hidden Neuron* Terhadap *Training Error*

Pada arsitektur jaringan syaraf tiruan dengan menggunakan 1 atau lebih *hidden layer,* jumlah *hidden neuron* sangat berpengaruh terhadap kecepatan konvergensi *error* pada saat training. Semakin sedikit jumlah *hidden neuron* maka semakin cepat konvergen, begitu juga sebaliknya semakin banyak *hidden neuron* maka akan semakin lambat konvergensinya. Namun pada pengujian OBPNN ini konvergensi error tidak dapat dicapai sehingga memiliki nilai *error* yang sama untuk jumlah *hidden neuron* yang berbeda. Konvergensi yang terjadi pada pengujian ini berupa perulangan pola perubahan error yang terus-menerus sehingga menyebabkan nilai error training yang didapat jika *hidden neuron* berjumlah 6-12 hampir tidak berpengaruh. Karena pada awal *epoch* sudah terjadi pola perubahan *error* yang tetap. Sehingga sampai *epoch* yang ke-1000 pun akan terulang pola error yang sama seperti yang terjadi sebelumnya. Pada Gambar 4.28 menunjukkan terjadinya perubahan nilai error yang hampir konstan.



Gambar 4.28 Pola *error* pada *training*

3. Analisa Pengaruh *Learning Rate* Terhadap Kecepatan *Training*

Besarnya *learning rate* sangat berpengaruh dalam kecepatan untuk mencapai *training error* yang ditentukan. Namun pada hasil pengujian penelitian ini, target *error* tidak ada yang dipenuhi sehingga kecepatan *training* bergantung pada jumlah maksimum *epoch* yang ditentukan sebelumnya. Karena itulah pada Gambar 4.21 untuk *hidden neuron* yang sama maka kecepatan training antara *learning rate* 0.1 dengan 1 hampir sama. Hal ini disebabkan pada *hidden neuron* yang sama maka banyaknya proses perhitungan yang dilakukan adalah sama dan *error* target tidak tercapai sehingga kecepatan *training* OBPNN yang dihasilkan hampir sama untuk *hidden neuron* yang sama.

4. Analisa Pengaruh Jumlah *Hidden Neuron* Terhadap Kecepatan *Training*

Banyaknya *hidden neuron* pada jaringan syaraf tiruan berpengaruh pada jumlah koneksi jaringan antara *neuron input layer* dengan *hidden layer* dan antara *hidden layer* dengan *output layer*.
Pada setiap koneksi jaringan menyimpan suatu bobot dan bobot tersebut digunakan untuk memperoleh pada neuron. Semakin banyak

jumlah *hidden neuron* maka semakin banyak pula koneksi jaringan dan apabila koneksi jaringan semakin banyak maka proses komputasi pada setiap kali siklus training(*epoch)* juga semakin banyak. Karena itulah jika jumlah *hidden neuron* bertambah, maka kecepatan training jaringan akan berkurang. Seperti yang ditunjukkan pada Gambar 4.21, pada gambar tersebut terjadi sedikit peningkatan waktu training apabila jumlah *hidden neuron* bertambah.

5. Analisa kesalahan Validasi Hasil *Training*

Error validasi hasil *training* yang dihasilkan pada pengujian ini dipengaruhi oleh dua hal, yaitu *training error* dan data yang digunakan sebagai validasi. Semakin kecil *training error* maka semakin baik pula untuk memprediksi nilai *setting time* dan kuat tekan dari data input baru meskipun data tersebut tidak ada dalam data *training*. Namun apabila *training error* semakin besar maka *error* hasil prediksi juga akan semakin besar. Selain itu juga dipengaruhi oleh konsistensi nilai data set baru, jika pada data set untuk validasi terdapat banyak nilai input yang sama dengan output yang berbeda tentunya akan memperbesar *error* validasi karena OBPNN hanya mampu memprediksi salah satu dari data tersebut.

Pada pengujian penelitian ini data set untuk setiap 5 data set untuk validasi yang digunakan terdapat input yang sama dan output yang berbeda. Karena itulah semakin besar *training error* yang digunakan untuk validasi data maka semakin besar pula nilai error hasil validasinya.


UNIVERSITAS BRAWIJAYA

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Kesimpulan yang diperoleh dari penelitian ini adalah :

1. Pada implementasi *Optical Backpropagation Neural Network* pada kasus SCC diperoleh kombinasi *learning rate* dan jumlah *hidden neuron* yang terbaik yaitu 0.1 dengan 6 *hidden neuron* yang menghasilkan nilai *training error* sebesar 2.5% dan error pada validasi 5.8%.
2. Pada *Optical Backpropagation Neural Network, training error* pada kasus SCC dengan *learning rate* 0.1 sampai 1 tidak dipengaruhi oleh jumlah *hidden neuron.* Besarnya *training error* hanya dipengaruhi oleh besarnya *learning rate.* Semakin besar *learning rate* semakin besar *training error* yang dihasilkan, begitu juga sebaliknya.
3. Untuk kecepatan dari *training Optical Backpropagation Neural Network* pada kasus SCC ini hampir tidak terpengaruh oleh besarnya *learning rate*, karena *training error* tidak tercapai, sehingga lama training mencapai batas maksimal *epoch*. Namun kecepatan *training* dipengaruhi oleh jumlah *hidden neuron* karena semakin banyak *hidden neuron* akan semakin banyak proses komputasi yang dilakukan pada setiap *epoch* sehingga waktu *training* menjadi lebih lama.
4. *Error* validasi hasil dari *training Optical Backpropagation Neuran Network,* dipengaruhi oleh besarnya *error* pada saat *training*. Apabila *training error* kecil maka nilai *error* validasi juga semakin kecil. Selain itu juga dipengaruhi oleh konsistensi nilai input dan output dari data yang digunakan dalam validasi.

5.2 Saran

Beberapa saran yang mungkin menjadi pertimbangan adalah sebagai berikut :

1. Dalam penelitian ini perlu dilakukan perbandingan dengan metode JST propagasi balik lainnya.
2. Dilakukan pengujian dengan nilai *learning rate* yang lebih kecil dari 0.1.

DAFTAR PUSTAKA

Altin, M., Tasdemir, S., Saritas, I., Kamanli, M., Cogurcu, M. T., & Kaltakci, M. Y. (2008). Determination of the Resistance Characteristics of Self-Compacting Concrete Samples by Artificial Neural Network. *International Conference on Computer Systems and Technologies- CompSysTech'08.*

ASTM C 33. (2003). Standard Spesification for Concrete Aggregate. *American Society for Testing and Materials*. Philadelphia.

Baldino, C. N., Frontera, F., Gabriele, D., & Candamano, S. (2007). Properties of SCC Using a Fine Synthetic Zeolite. *5th International RILEM Symposium on SCC*, (hal. 863-868). Ghent.

Efca. (2005). The European Guidelines for Self-Compacting Concrete Spesification, Product and Use.

Eric, K. P., & David, F. W. (2007). Summary of Concrete WorkabilityTest Methods. *International Center for Aggregate Reseach (Icar)*. Austin.

Esping, O. (2007). *Early Age Properties of Self-Compacting Concrete*. Göteborg: Chalmers University of Technology.

Freeman, J. A., & Skapura, D. M. (1991). *Neural networks : algorithms, applications, and programming techniques*. Addison-Wesley Publishing Company, Inc.

Hajime, O., & Ouchi, M. (2003). Self-Compacting Concrete. *Journal of Advance Technology*, (hal. 5-15).

Heaton, J. (2010). *Introduction to Neural Networks for C#, Second Edition.* St. Louis: Heaton Research, Inc.

Holt, E. E. (2001). Early age autogenous shrinkage of concrete. *Technical Research Centre of Finland* (hal. 446). VTT Publications.

Kasabov, N. K. (1998). *Foundations of Neural Networks, Fuzzy Systems, and Knowledge Engineering.* Massachusetts: MIT PRESS.

Kusumadewi, S. (2003). *Artificial Intelligence (Teknik dan Aplikasinya)*. Yogyakarta: Graha Ilmu.

Orr, G. (1999). *CS-449: Neural Networks.* Dipetik April 7, 2010, dari www.willamette.edu/_gorr/classes/cs449/

Otair, M. A., & Salameh, W. A. (2005). Speeding Up Back-Propagation Neural Networks. *Proceedings of the 2005*

Informing Science and IT Education Joint Conference. Arizona.

Ouchi, M. (2003). Aplication of Self-Compacting Concrete in Japan, Europe, and The United State. *International Seminar High Performance Concrete.*

Puspitanigrum, D. (2006). *Pengantar Jaringan Syaraf Tiruan.* Yogyakarta: Penerbit Andi.

Reinhardt, H. W., Grobe, C. U., & Herb, A. T. (2000). Ultrasonic monitoring of setting and hardening of cement mortar – A new device. *Materials and Structures, Vol. 33*, (hal. 580-583).

Schindler, K. A. (2003). *Prediction of Concrete Setting.* Alabama: Auburn University.

Siang, J. J. (2004). *Jaringan Syaraf Tiruan & Pemrogramannya Menggunakan Matlab.* Yogyakarta: Penerbit Andi.

Yeh, I. C. (2006). Analysis of Strength of Concrete Using Design of Experiments and Neural Networks. *Journal of Materials in Civil Engineering* .

Yeh, I. C. (1998). Modelling of Strength of High Performance Concrete Using Artificial Neural Networks. *Cement And Concrete Research*, (hal. 1797–1808).

LAMPIRAN

Lampiran 1. **Data.csv**

x1	x2	x3	x4	x5	x6	Y1	Y2
400	900	880	100	2.2	0.35	172	54
425	900	880	75	1.65	0.35	175	54
400	900	880	100	1.65	0.35	175	53.5
425	900	880	75	2.5	0.35	186.6	53.2
400	900	880	100	2.5	0.35	183	54
425	900	880	75	2.2	0.35	180	54
400	900	880	100	2.2	0.35	189.5	53
425	900	880	75	1.65	0.35	179	53.7
400	900	880	100	1.65	0.35	178.2	52.8
425	900	880	75	2.5	0.35	182.4	53.25
400	900	880	100	2.5	0.35	184.2	54
425	900	880	75	2.2	0.35	180.6667	54.41667
400	900	880	100	2.2	0.35	182	53.41667
425	900	880	75	1.65	0.35	182.1667	53.68333
475	900	880	25	2.5	0.37	159.3333	53.31333
450	900	880	50	2.5	0.37	171.6667	53.555
425	900	880	75	2.2	0.35	186.3333	54.40167
400	900	880	100	2.2	0.35	191.3333	53.07833
375	900	880	125	2	0.33	200.8333	51.56833
350	900	880	150	2	0.33	207.1667	49.445
325	900	880	175	1.75	0.31	202	46.285
300	900	880	200	1.75	0.3	216.6667	44.53833
275	900	880	225	1.5	0.3	220	40.52833
250	900	880	250	1.5	0.3	234.6667	37.69833
425	900	880	75	2.5	0.35	161.25	54.40167
425	900	880	75	2.2	0.35	168.1667	54.31667
400	900	880	100	2.5	0.35	169.3167	53.07833
425	900	880	75	1.65	0.35	175	54.16667
400	900	880	100	2.2	0.35	172.6667	54.48333

425	900	880	75	2.5	0.35	184.6667	53.33333
400	900	880	100	1.65	0.35	175	53.58333
425	900	880	75	2.2	0.35	180.6667	54.15
400	900	880	100	2.5	0.35	183.1667	53.99
400	900	880	100	2.2	0.35	189.5	53.21667
400	900	880	100	1.65	0.35	178.5	52.86667
425	900	880	75	1.65	0.35	179.1667	53.75
425	900	880	75	2.2	0.35	180.6667	54.41667
425	900	880	75	2.5	0.35	180.6667	53.28333
400	900	880	100	2.5	0.35	183.5	54.62
400	900	880	100	2.2	0.35	182	53.41667

Lampiran 2. **Data2.csv**

x1	x2	x3	x4	x5	x6	y1	y2
400	900	880	100	2.2	0.35	175	55.4
400	900	880	100	2.2	0.35	170	55.5
400	900	880	100	2.2	0.35	172	54
400	900	880	100	2.2	0.35	176	53.5
400	900	880	100	2.2	0.35	168	56
400	900	880	100	2.2	0.35	175	52.5
425	900	880	75	1.65	0.35	177	53.5
425	900	880	75	1.65	0.35	178	55
425	900	880	75	1.65	0.35	172	52.5
425	900	880	75	1.65	0.35	170	56.5
425	900	880	75	1.65	0.35	176	55
425	900	880	75	1.65	0.35	177	52.5
400	900	880	100	1.65	0.35	180	54
400	900	880	100	1.65	0.35	175	52
400	900	880	100	1.65	0.35	170	53.5
400	900	880	100	1.65	0.35	180	52.5
400	900	880	100	1.65	0.35	175	55
400	900	880	100	1.65	0.35	170	54.5
425	900	880	75	2.5	0.35	175	54
425	900	880	75	2.5	0.35	180	53.2
425	900	880	75	2.5	0.35	190	53
425	900	880	75	2.5	0.35	185	51.8
425	900	880	75	2.5	0.35	186	56
425	900	880	75	2.5	0.35	192	52
400	900	880	100	2.5	0.35	190	56
400	900	880	100	2.5	0.35	185	55.5
400	900	880	100	2.5	0.35	187	52.94
400	900	880	100	2.5	0.35	190	53
400	900	880	100	2.5	0.35	187	54
400	900	880	100	2.5	0.35	160	52.5
425	900	880	75	2.2	0.35	175	55.3

425	900	880	75	2.2	0.35	180	55.3
425	900	880	75	2.2	0.35	175	54
425	900	880	75	2.2	0.35	190	52.5
425	900	880	75	2.2	0.35	180	53.5
425	900	880	75	2.2	0.35	184	54.3
400	900	880	100	2.2	0.35	187	52.5
400	900	880	100	2.2	0.35	190	56.8
400	900	880	100	2.2	0.35	195	52.5
400	900	880	100	2.2	0.35	185	53.5
400	900	880	100	2.2	0.35	190	51.5
400	900	880	100	2.2	0.35	190	52.5
425	900	880	75	1.65	0.35	180	54
425	900	880	75	1.65	0.35	186	52.5
425	900	880	75	1.65	0.35	175	54
425	900	880	75	1.65	0.35	179	55.5
425	900	880	75	1.65	0.35	180	52.5
425	900	880	75	1.65	0.35	175	54
400	900	880	100	1.65	0.35	180	53
400	900	880	100	1.65	0.35	184	54
400	900	880	100	1.65	0.35	176	52
400	900	880	100	1.65	0.35	180	51.5
400	900	880	100	1.65	0.35	175	54.4
400	900	880	100	1.65	0.35	176	52.3
425	900	880	75	2.5	0.35	170	53.5
425	900	880	75	2.5	0.35	175	52
425	900	880	75	2.5	0.35	180	54.4
425	900	880	75	2.5	0.35	185	55.3
425	900	880	75	2.5	0.35	184	52.5
425	900	880	75	2.5	0.35	190	52
400	900	880	100	2.5	0.35	180	55.2
400	900	880	100	2.5	0.35	175	52.8
400	900	880	100	2.5	0.35	185	56

400	900	880	100	2.5	0.35	184	53.2
400	900	880	100	2.5	0.35	191	54
400	900	880	100	2.5	0.35	186	54.7
425	900	880	75	2.2	0.35	175	55.4
425	900	880	75	2.2	0.35	187	53.5
425	900	880	75	2.2	0.35	180	53.5
425	900	880	75	2.2	0.35	182	54.2
425	900	880	75	2.2	0.35	180	53.5
425	900	880	75	2.2	0.35	180	56.4
400	900	880	100	2.2	0.35	179	55.2
400	900	880	100	2.2	0.35	180	54.3
400	900	880	100	2.2	0.35	185	52.5
400	900	880	100	2.2	0.35	180	53.5
400	900	880	100	2.2	0.35	190	52.5
400	900	880	100	2.2	0.35	178	52.5
425	900	880	75	1.65	0.35	182	52.4
425	900	880	75	1.65	0.35	184	54.3
425	900	880	75	1.65	0.35	185	53.5
425	900	880	75	1.65	0.35	180	55.3
425	900	880	75	1.65	0.35	182	52.8
425	900	880	75	1.65	0.35	180	53.8
475	900	880	25	2.5	0.37	155	52.65
475	900	880	25	2.5	0.37	160	52.94
475	900	880	25	2.5	0.37	165	52.37
475	900	880	25	2.5	0.37	156	53.5
475	900	880	25	2.5	0.37	162	54.07
475	900	880	25	2.5	0.37	158	54.35
450	900	880	50	2.5	0.37	170	53.22
450	900	880	50	2.5	0.37	175	54.1
450	900	880	50	2.5	0.37	165	53.5
450	900	880	50	2.5	0.37	170	53.22
450	900	880	50	2.5	0.37	175	53.79

450	900	880	50	2.5	0.37	175	53.5
425	900	880	75	2.2	0.35	170	55.2
425	900	880	75	2.2	0.35	180	54.64
425	900	880	75	2.2	0.35	190	52.94
425	900	880	75	2.2	0.35	185	54.64
425	900	880	75	2.2	0.35	195	55.2
425	900	880	75	2.2	0.35	198	53.79
400	900	880	100	2.2	0.35	185	53.5
400	900	880	100	2.2	0.35	195	53.79
400	900	880	100	2.2	0.35	186	52.94
400	900	880	100	2.2	0.35	198	52.37
400	900	880	100	2.2	0.35	190	53.5
400	900	880	100	2.2	0.35	194	52.37
375	900	880	125	2	0.33	200	51.8
375	900	880	125	2	0.33	205	50.96
375	900	880	125	2	0.33	195	51.52
375	900	880	125	2	0.33	205	51.8
375	900	880	125	2	0.33	198	50.96
375	900	880	125	2	0.33	202	52.37
350	900	880	150	2	0.33	210	49.54
350	900	880	150	2	0.33	204	50.39
350	900	880	150	2	0.33	205	50.11
350	900	880	150	2	0.33	215	48.12
350	900	880	150	2	0.33	205	49.54
350	900	880	150	2	0.33	204	48.97
325	900	880	175	1.75	0.31	202	47.28
325	900	880	175	1.75	0.31	195	46.71
325	900	880	175	1.75	0.31	198	45.86
325	900	880	175	1.75	0.31	202	45.29
325	900	880	175	1.75	0.31	205	46.14
325	900	880	175	1.75	0.31	210	46.43
300	900	880	200	1.75	0.3	215	43.31

300	900	880	200	1.75	0.3	210	44.73
300	900	880	200	1.75	0.3	215	44.44
300	900	880	200	1.75	0.3	225	44.16
300	900	880	200	1.75	0.3	220	45.58
300	900	880	200	1.75	0.3	215	45.01
275	900	880	225	1.5	0.3	210	41.05
275	900	880	225	1.5	0.3	215	40.2
275	900	880	225	1.5	0.3	225	40.76
275	900	880	225	1.5	0.3	215	39.63
275	900	880	225	1.5	0.3	225	41.05
275	900	880	225	1.5	0.3	230	40.48
250	900	880	250	1.5	0.3	215	38.78
250	900	880	250	1.5	0.3	235	38.22
250	900	880	250	1.5	0.3	240	36.8
250	900	880	250	1.5	0.3	242	37.65
250	900	880	250	1.5	0.3	236	39.35
250	900	880	250	1.5	0.3	240	35.39
425	900	880	75	2.5	0.35	165	55.2
425	900	880	75	2.5	0.35	157.5	54.64
425	900	880	75	2.5	0.35	157.5	52.94
425	900	880	75	2.5	0.35	157.5	54.64
425	900	880	75	2.5	0.35	165	55.2
425	900	880	75	2.5	0.35	165	53.79
400	900	880	100	2.5	0.35	178	53.5
400	900	880	100	2.5	0.35	171	53.79
400	900	880	100	2.5	0.35	170	52.94
400	900	880	100	2.5	0.35	172.5	52.37
400	900	880	100	2.5	0.35	159.4	53.5
400	900	880	100	2.5	0.35	165	52.37
425	900	880	75	2.2	0.35	170	55.3
425	900	880	75	2.2	0.35	166	55.1
425	900	880	75	2.2	0.35	170	53

425	900	880	75	2.2	0.35	165	54
425	900	880	75	2.2	0.35	168	55
425	900	880	75	2.2	0.35	170	53.5
400	900	880	100	2.2	0.35	175	55.4
400	900	880	100	2.2	0.35	170	55.5
400	900	880	100	2.2	0.35	172	54
400	900	880	100	2.2	0.35	176	53.5
400	900	880	100	2.2	0.35	168	56
400	900	880	100	2.2	0.35	175	52.5
425	900	880	75	1.65	0.35	177	53.5
425	900	880	75	1.65	0.35	178	55
425	900	880	75	1.65	0.35	172	52.5
425	900	880	75	1.65	0.35	170	56.5
425	900	880	75	1.65	0.35	176	55
425	900	880	75	1.65	0.35	177	52.5
400	900	880	100	1.65	0.35	180	54
400	900	880	100	1.65	0.35	175	52
400	900	880	100	1.65	0.35	170	53.5
400	900	880	100	1.65	0.35	180	52.5
400	900	880	100	1.65	0.35	175	55
400	900	880	100	1.65	0.35	170	54.5
425	900	880	75	2.5	0.35	175	54
425	900	880	75	2.5	0.35	180	53.2
425	900	880	75	2.5	0.35	190	53
425	900	880	75	2.5	0.35	185	51.8
425	900	880	75	2.5	0.35	186	56
425	900	880	75	2.5	0.35	192	52
400	900	880	100	2.5	0.35	190	56
400	900	880	100	2.5	0.35	185	55.5
400	900	880	100	2.5	0.35	187	52.94
400	900	880	100	2.5	0.35	190	53
400	900	880	100	2.5	0.35	187	54

400	900	880	100	2.5	0.35	160	52.5
425	900	880	75	2.2	0.35	175	55.3
425	900	880	75	2.2	0.35	180	55.3
425	900	880	75	2.2	0.35	175	54
425	900	880	75	2.2	0.35	190	52.5
425	900	880	75	2.2	0.35	180	53.5
425	900	880	75	2.2	0.35	184	54.3
400	900	880	100	2.2	0.35	187	52.5
400	900	880	100	2.2	0.35	190	56.8
400	900	880	100	2.2	0.35	195	52.5
400	900	880	100	2.2	0.35	185	53.5
400	900	880	100	2.2	0.35	190	51.5
400	900	880	100	2.2	0.35	190	52.5
425	900	880	75	1.65	0.35	180	54
425	900	880	75	1.65	0.35	186	52.5
425	900	880	75	1.65	0.35	175	54
425	900	880	75	1.65	0.35	179	55.5
425	900	880	75	1.65	0.35	180	52.5
425	900	880	75	1.65	0.35	175	54
400	900	880	100	1.65	0.35	180	53
400	900	880	100	1.65	0.35	184	54
400	900	880	100	1.65	0.35	176	52
400	900	880	100	1.65	0.35	180	51.5
400	900	880	100	1.65	0.35	175	54.4
400	900	880	100	1.65	0.35	176	52.3
425	900	880	75	2.5	0.35	170	53.5
425	900	880	75	2.5	0.35	175	52
425	900	880	75	2.5	0.35	180	54.4
425	900	880	75	2.5	0.35	185	55.3
425	900	880	75	2.5	0.35	184	52.5
425	900	880	75	2.5	0.35	190	52
400	900	880	100	2.5	0.35	180	55.2

400	900	880	100	2.5	0.35	175	52.8
400	900	880	100	2.5	0.35	185	56
400	900	880	100	2.5	0.35	184	53.2
400	900	880	100	2.5	0.35	191	54
400	900	880	100	2.5	0.35	186	54.7
425	900	880	75	2.2	0.35	175	55.4
425	900	880	75	2.2	0.35	187	53.5
425	900	880	75	2.2	0.35	180	53.5
425	900	880	75	2.2	0.35	182	54.2
425	900	880	75	2.2	0.35	180	53.5
425	900	880	75	2.2	0.35	180	56.4
400	900	880	100	2.2	0.35	179	55.2
400	900	880	100	2.2	0.35	180	54.3
400	900	880	100	2.2	0.35	185	52.5
400	900	880	100	2.2	0.35	180	53.5
400	900	880	100	2.2	0.35	190	52.5
400	900	880	100	2.2	0.35	178	52.5

Lampiran 3 Data3.csv

x1	x2	x3	x4	x5	x6	y1	y2
475	900	880	25	2.5	0.37	155	52.65
475	900	880	25	2.5	0.37	160	52.94
475	900	880	25	2.5	0.37	165	52.37
475	900	880	25	2.5	0.37	156	53.5
475	900	880	25	2.5	0.37	162	54.07
475	900	880	25	2.5	0.37	158	54.35
450	900	880	50	2.5	0.37	170	53.22
450	900	880	50	2.5	0.37	175	54.1
450	900	880	50	2.5	0.37	165	53.5
450	900	880	50	2.5	0.37	170	53.22
450	900	880	50	2.5	0.37	175	53.79
450	900	880	50	2.5	0.37	175	53.5
425	900	880	75	2.2	0.35	170	55.2
425	900	880	75	2.2	0.35	180	54.64
425	900	880	75	2.2	0.35	190	52.94
425	900	880	75	2.2	0.35	185	54.64
425	900	880	75	2.2	0.35	195	55.2
425	900	880	75	2.2	0.35	198	53.79
400	900	880	100	2.2	0.35	185	53.5
400	900	880	100	2.2	0.35	195	53.79
400	900	880	100	2.2	0.35	186	52.94
400	900	880	100	2.2	0.35	198	52.37
400	900	880	100	2.2	0.35	190	53.5
400	900	880	100	2.2	0.35	194	52.37
375	900	880	125	2	0.33	200	51.8
375	900	880	125	2	0.33	205	50.96
375	900	880	125	2	0.33	195	51.52
375	900	880	125	2	0.33	205	51.8
375	900	880	125	2	0.33	198	50.96
375	900	880	125	2	0.33	202	52.37
350	900	880	150	2	0.33	210	49.54

350	900	880	150	2	0.33	204	50.39
350	900	880	150	2	0.33	205	50.11
350	900	880	150	2	0.33	215	48.12
350	900	880	150	2	0.33	205	49.54
350	900	880	150	2	0.33	204	48.97
325	900	880	175	1.75	0.31	202	47.28
325	900	880	175	1.75	0.31	195	46.71
325	900	880	175	1.75	0.31	198	45.86
325	900	880	175	1.75	0.31	202	45.29
325	900	880	175	1.75	0.31	205	46.14
325	900	880	175	1.75	0.31	210	46.43
300	900	880	200	1.75	0.3	215	43.31
300	900	880	200	1.75	0.3	210	44.73
300	900	880	200	1.75	0.3	215	44.44
300	900	880	200	1.75	0.3	225	44.16
300	900	880	200	1.75	0.3	220	45.58
300	900	880	200	1.75	0.3	215	45.01
275	900	880	225	1.5	0.3	210	41.05
275	900	880	225	1.5	0.3	215	40.2
275	900	880	225	1.5	0.3	225	40.76
275	900	880	225	1.5	0.3	215	39.63
275	900	880	225	1.5	0.3	225	41.05
275	900	880	225	1.5	0.3	230	40.48
250	900	880	250	1.5	0.3	215	38.78
250	900	880	250	1.5	0.3	235	38.22
250	900	880	250	1.5	0.3	240	36.8
250	900	880	250	1.5	0.3	242	37.65
250	900	880	250	1.5	0.3	236	39.35
250	900	880	250	1.5	0.3	240	35.39
425	900	880	75	2.5	0.35	165	55.2
425	900	880	75	2.5	0.35	157.5	54.64
425	900	880	75	2.5	0.35	157.5	52.94

425	900	880	75	2.5	0.35	157.5	54.64
425	900	880	75	2.5	0.35	165	55.2
425	900	880	75	2.5	0.35	165	53.79
400	900	880	100	2.5	0.35	178	53.5
400	900	880	100	2.5	0.35	171	53.79
400	900	880	100	2.5	0.35	170	52.94
400	900	880	100	2.5	0.35	172.5	52.37
400	900	880	100	2.5	0.35	159.4	53.5
400	900	880	100	2.5	0.35	165	52.37
425	900	880	75	2.2	0.35	170	55.3
425	900	880	75	2.2	0.35	166	55.1
425	900	880	75	2.2	0.35	170	53
425	900	880	75	2.2	0.35	165	54
425	900	880	75	2.2	0.35	168	55
425	900	880	75	2.2	0.35	170	53.5
400	900	880	100	2.2	0.35	175	55.4
400	900	880	100	2.2	0.35	170	55.5
400	900	880	100	2.2	0.35	172	54
400	900	880	100	2.2	0.35	176	53.5
400	900	880	100	2.2	0.35	168	56
400	900	880	100	2.2	0.35	175	52.5
425	900	880	75	1.65	0.35	177	53.5
425	900	880	75	1.65	0.35	178	55
425	900	880	75	1.65	0.35	172	52.5
425	900	880	75	1.65	0.35	170	56.5
425	900	880	75	1.65	0.35	176	55
425	900	880	75	1.65	0.35	177	52.5
400	900	880	100	1.65	0.35	180	54
400	900	880	100	1.65	0.35	175	52
400	900	880	100	1.65	0.35	170	53.5
400	900	880	100	1.65	0.35	180	52.5
400	900	880	100	1.65	0.35	175	55

400	900	880	100	1.65	0.35	170	54.5
425	900	880	75	2.5	0.35	175	54
425	900	880	75	2.5	0.35	180	53.2
425	900	880	75	2.5	0.35	190	53
425	900	880	75	2.5	0.35	185	51.8
425	900	880	75	2.5	0.35	186	56
425	900	880	75	2.5	0.35	192	52

Lampiran 4 Data Hasil Pengujian

No	Σ hidden neuron	Learning Rate	Training Error	Waktu	Validasi	
					data2	data3
1	6	0.1	2.524	1.304	6.5	5.988
2	7	0.1	2.528	1.429	6.718	6.07
3	8	0.1	2.458	1.6756	6.094	4.552
4	9	0.1	2.56	1.925	6.872	6.138
5	10	0.1	2.704	1.8346	6.434	5.81
6	11	0.1	2.504	2.3214	7.432	6.456
7	12	0.1	2.564	2.265	6.934	6.084
8	6	0.2	4.848	1.4132	12.24	11.86
9	7	0.2	4.704	1.6348	12.958	12.72
10	8	0.2	4.96	1.7194	12.056	11.87
11	9	0.2	4.93	1.5694	13.828	14.718
12	10	0.2	4.674	1.6974	12.614	12.686
13	11	0.2	4.932	1.8784	13.478	13.464
14	12	0.2	5.174	2.6208	11.404	11.464
15	6	0.3	7.272	1.616	19.532	19.728
16	7	0.3	7.076	1.9218	20.406	18.792
17	8	0.3	7.176	1.9656	23.464	23.348
18	9	0.3	7.056	2.2962	21.596	21.112
19	10	0.3	7.12	2.4586	19.882	19.114
20	11	0.3	7.224	2.4586	20.406	20.238
21	12	0.3	7.176	3.189	21.57	20.306
22	6	0.4	9.19	1.5696	18.902	16.826
23	7	0.4	9.196	1.9438	23.562	20.864
24	8	0.4	9.242	2.137	26.072	25.068
25	9	0.4	9.272	2.1186	21.052	19.906
26	10	0.4	9.41	2.3746	29.55	29.716
27	11	0.4	9.304	2.8798	26.562	24.96

28	12	0.4	9.04	2.7582	17.596	15.548
29	6	0.5	11.584	1.5972	33.382	32.966
30	7	0.5	11.65	1.8938	34.61	34.548
31	8	0.5	11.418	2.0436	33.346	32.672
32	9	0.5	11.532	2.2962	29.852	29.184
33	10	0.5	11.668	2.6146	35.728	34.876
34	11	0.5	11.444	2.521	35.502	35.25
35	12	0.5	11.364	2.4396	33.256	32.632
36	6	0.6	13.526	1.3762	36.368	37.016
37	7	0.6	13.168	1.8782	29.098	27.88
38	8	0.6	13.492	1.8282	34.548	33.958
39	9	0.6	13.322	2.4182	35.74	34.41
40	10	0.6	13.502	2.2466	39.836	38.356
41	11	0.6	13.238	2.3962	30.306	29.032
42	12	0.6	13.374	2.752	35.636	33.948
43	6	0.7	15.404	1.6008	40.152	39.286
44	7	0.7	14.86	1.7692	36.63	37.494
45	8	0.7	15.262	1.6816	38.068	37.792
46	9	0.7	14.926	2.1342	38.072	36.366
47	10	0.7	15.272	2.0156	41.304	41.522
48	11	0.7	15.518	2.3866	51.638	50.894
49	12	0.7	14.92	2.496	43.102	42.822
50	6	0.8	16.526	1.585	34.2	33.076
51	7	0.8	16.826	1.685	33.91	32.242
52	8	0.8	16.454	1.8406	38.42	37.284
53	9	0.8	16.846	1.7286	35.426	34.49
54	10	0.8	16.438	2.0154	25.584	22.816
55	11	0.8	16.732	2.3834	41.876	40.152
56	12	0.8	16.544	2.8234	46.908	45.554
57	6	0.9	17.562	1.5162	42.942	42.348
58	7	0.9	17.6	1.526	43.724	41.914
59	8	0.9	17.984	1.4542	55.306	53.592

60	9	0.9	17.856	1.9844	52.624	50.764
61	10	0.9	18.226	2.3896	45.04	44.156
62	11	0.9	18.058	2.0778	42.548	42.53
63	12	0.9	17.916	2.6114	47.488	47.066
64	6	1	18.816	1.4098	49.63	50.138
65	7	1	19.204	1.8316	43.194	41.206
66	8	1	19.188	1.8286	44.2	44.296
67	9	1	19.28	2.0876	46.11	43.578
68	10	1	19.024	2.1964	36.922	35.538
69	11	1	19.312	2.3244	56.41	55.886
70	12	1	19.29	2.5614	53.014	50.978