# PREDIKSI SKOR PERTANDINGAN SEPAK BOLA MENGGUNAKAN NEUROEVOLUTION OF AUGMENTING TOPOLOGIES DAN BACKPROPAGATION

Oleh:

Lily Puspa Dewi

Alvin Nathaniel Tjondrowiguno

Welly Winata

PROGRAM STUDI TEKNIK INFORMATIKA


**FAKULTAS TEKNOLOGI INDUSTRI**

**UNIVERSITAS KRISTEN PETRA**

**SURABAYA**

**2019**

**LAPORAN PENELITIAN**

**NO: 01021952/INF/2020**

# PREDIKSI SKOR PERTANDINGAN SEPAK BOLA MENGGUNAKAN NEUROEVOLUTION OF AUGMENTING TOPOLOGIES DAN BACKPROPAGATION


Oleh:

Lily Puspa Dewi

Alvin Nathaniel Tjondrowiguno

Welly Winata

PROGRAM STUDI TEKNIK INFORMATIKA

**FAKULTAS TEKNOLOGI INDUSTRI**

**UNIVERSITAS KRISTEN PETRA**

**SURABAYA**


**2019**

**LEMBAR IDENTITAS DAN PENGESAHAN**

**LAPORAN HASIL PENELITIAN**

| | | | |
|---|---|---|---|
| 1. | a. Judul Penelitian | : | Prediksi Pertandingan Sepak Bola Menggunakan *Neuroevolution of Augmenting Topologies* dan *Backpropagation* |
| | b. Nomor Penelitian | : | 01021952/INF/2020 |
| | c. Jalur Penelitian | : | I ~~/ II / III / IV~~ |
| 2. | Ketua Peneliti | | |
| | a. Nama lengkap dan Gelar | : | Lily Puspa Dewi S.T., M.Kom. |
| | b. Jenis Kelamin | : | Perempuan |
| | c. Pangkat / Golongan / NIP | : | - / - / 98-011 |
| | d. Bidang Ilmu yang diteliti | : | - |
| | e. Jabatan Akademik | : | - |
| | f. Fakultas / Program Studi | : | Fakultas Teknologi Industri / Teknik Informatika |
| | g. Universitas | : | Universitas Kristen Petra |
| 3. | Anggota Tim Peneliti (I) | | |
| | a. Nama lengkap dan Gelar | : | Alvin Nathaniel Tjondrowiguno, S.Kom. |
| | b. Jenis Kelamin | : | Laki-laki |
| | c. Pangkat / Golongan / NIP | : | - / - / 45-201 |
| | d. Bidang Ilmu yang diteliti | : | - |
| | e. Jabatan Akademik | : | - |
| | f. Fakultas / Program Studi | : | Fakultas Teknologi Industri / Teknik Informatika |
| | g. Universitas | : | Universitas Kristen Petra |
| 4. | Anggota Tim Peneliti (II) | | |
| | a. Nama lengkap dan Gelar | : | Welly Winata |
| | b. Jenis Kelamin | : | Laki-laki |
| | c. Pangkat / Golongan / NIP | : | - / IID / - |
| | d. Bidang Ilmu yang diteliti | : | - |
| | e. Jabatan Akademik | : | - |
| | f. Fakultas / Program Studi | : | Fakultas Teknologi Industri / Teknik Informatika |
| | g. Universitas | : | Universitas Kristen Petra |
| 5. | Lokasi penelitian | : | Surabaya |
| 6. | Tanggal Penelitian | : | - |
| 7. | Biaya | : | - |

Surabaya, 29 November 2019

Mengetahui,

| Ketua Program Studi | Ketua Peneliti |
| --- | --- |
| Henry Novianus Palit, Ph.D. | Ir. Kartika Gunadi, M.T. |
| NIP: 14-001 | NIP: 88-004 |

Menyetujui,

Dekan Fakultas Teknologi Industri

Dr. Juliana Anggono, S.T., M.Eng.

NIP: 94-016

## ABSTRAK

Welly Winata:
Skripsi
Prediksi Skor Pertandingan Sepak Bola Menggunakan *Neuroevolution of Augmenting Topologies* dan *Backpropagation*

Sepak bola merupakan olahraga yang memiliki penggemar paling banyak di dunia. Hal yang membuat sebak pola menjadi sangat populer adalah hasil yang tidak pasti dan sulit ditebak. Ada banyak faktor yang mempengaruhi hasil dari sebuah pertandingan sepak bola, diantarnya *strategy*, *skill*, bahkan sampai keberuntungan. Karena itu, menebak hasil pertandingan sepak bola merupakan masalah yang menarik.

Penelitian dimulai dengan *neuroevolution of augmenting topologies*, yang berfungsi untuk melakukan pencarian struktur dari sebuah *neural network*. Lalu, *network* yang dihasilkan oleh NEAT akan dioptimasi menggunakan *backpropagation*. *Rating* pemain, *rating team*, dan posisi pemain akan digunakan sebagai *features*.

Tingkat akurasi terbaik yang didapat sebesar 81.5% pada akurasi hasil pertandingan, dan 48% pada akurasi skor pertandingan diperoleh melalui proses NEAT yang telah dioptimasi oleh *backpropagation* menggunakan *rating* pemain, *rating team*, dan jumlah masing-masing posisi pada setiap sektor sebagai *features*.

Tetapi, pada pengujian *real life*, *rating* pemain dan *team* tidak diketahui, sehingga digunakan metode rata-rata untuk menghitung rating dari pemain dan *team*. Namun, akurasi yang didapat pada pengujian ini sangat rendah, inkonsistensi dari pemain menyebabkan metode rata-rata yang digunakan tidak mampu bekerja dengan baik.



Kata kunci: *Machine Learning, Artificial Neural Network, Neuroevolution, Neuroevolution of Augmenting Topologies, Backpropagation*

## ABSTRACT

Welly Winata:

Undergraduate Thesis

Predicting Football Matches Score Using Neuroevoluiton of Augmenting Topologies and Backpropagation.

Football, or soccer is the most popular sport in the world. What makes football special is the uncertainty and unpredictable result. There are a lot of factors that can affect the result of a football match, such as strategy, skill, or even luck. Therefore, predicting the outcome of football match can be challenging yet interesting task.

This research started with neuroevolution of augmenting topologies, which useful to find the structur of a neural network. Then, the network produced by NEAT is optimized using backpropagation. Player ratings, team ratings, and player position are used as features of neural network.

The hightest accuracies achieved are 81.5% on the final result predicting, and 48% on score predicting, were obtained through NEAT network that optimized by backpropagation, with player ratings, team ratings, and total position from each sectors are used as features.

However, on real life test, the player and team ratings are unknown. To calculate the player and team ratings, averages methods are used. Unfortunately, the network performed poorly causing the accuracies to dropped significantly. Lack of consistency from player ratings are believed to be the main problem on calculating the player and team ratings.



Keywords: *Machine Learning, Artificial Neural Network, Neuroevolution, Neuroevolution of Augmenting Topologies, Backpropagation*

# DAFTAR ISI

# DAFTAR TABEL

## DAFTAR GAMBAR

## DAFTAR SEGMEN PROGRAM


Segmen Program 4.1 Pre-processing pada data dengan feature berupa player ratings dan team ratings .....37
Segmen Program 4.2 Fungsi `load_data()` .....40
Segmen Program 4.3 Fungsi `start()` untuk menjalankan proses NEAT.....41
Segmen Program 4.4 *Fitness function* pertama.....41
Segmen Program 4.5 *Fitness function* kedua.....42
Segmen Program 4.6 Fungsi `calculate_winner()` .....42

# 1. PENDAHULUAN

## 1.1 Latar Belakang

Dalam artikel yang diterbitkan oleh *Bloomberg* pada tahun 2018, 4 dari 10 orang menyatakan bahwa mereka adalah penggemar sepak bola (Boudway, 2018). Ini menjadikan sepakbola sebagai olahraga paling populer di dunia. Ketidakpastian merupakan sifat alami dari sepak bola (Pappalardo & Cintia, 2018) yang menjadikan ini sebagai salah satu faktor mengapa sepak bola sangat disukai.

Seperti yang dikatakan *The New York Times* dalam artikelnya yang berjudul *Soccer, a Beautiful Game by Chance* banyak sekali komponen yang mempengaruhi hasil akhir dari sebuah pertandingan, seperti *strategy*, *skill*, dan *luck* (Tierney, 2014). Faktor-faktor tersebutlah yang membuat hasil dari setiap pertandingan unik dan sulit diprediksi. Namun, dari setiap pertandingan sepak bola dapat diperoleh data yang dapat digunakan untuk menganalisa bagaimana jalannya pertandingan.

Dengan semakin berkembangnya teknologi, data-data penting yang berkaitan dengan pertandingan sepak bola semakin mudah didapat. Data-data tersebut dapat diolah dan digunakan untuk melakukan prediksi pada pertandingan yang akan datang. Salah satu bidang dalam *Computer Science* yang banyak digunakan untuk melakukan prediksi berdasarkan data adalah *Machine Learning.*

*Machine Learning*, dalam definisinya, adalah suatu bidang dalam *Computer Science* yang dapat mempelajari pola tertentu dari kumpulan data dan membuat prediksi atau klasifikasi berdasarkan kumpulan data tersebut. Penggunaan *Machine Learning* dalam masalah seperti ini sangat cocok, karena selain banyaknya data yang tersedia, sepak bola juga sulit diprediksi berdasarkan logika, maupun alasan-alasan ekplisit lainnya (Simeone, 2018). Beberapa contoh algoritma Machine Learning yang sedang populer saat ini adalah *Artificial Neural Network* (ANN) dan *Support Vector Machine* (SVM).

Pada penelitian sebelumnya (Igiri, 2015), SVM dan ANN pernah digunakan untuk melakukan prediksi sepakbola, tetapi hasil yang didapat oleh SVM sangat mengecewakan, akurasi yang didapat hanya sebesar 53.3%, sedangkan ANN secara

impresif mampu menghasilkan akurasi diatas 80%. Berkaca dari hasil penelitian tersebut, metode yang akan digunakan dalam penelitian kali ini adalah *Neuroevolution of Augmenting Topologies* (NEAT).

NEAT adalah algoritma penyempurnaan dari *Neuroevolution* (NE) yang berasal dari penggabungan antara ANN dan *Evolutionary Algorithm* (EA). Salah satu kelebihan Perbedaan NE jika dibandingkan dengan ANN tradisional adalah topologi jaringan yang dapat melakukan evolusi seiring berjalannya proses *training* (Morse & Stanley, 2016). Tetapi, NE juga memiliki kekurangan, yaitu saat terjadi *crossover* antara 2 jaringan, adanya kemungkinan *offspring* yang dihasilkan memiliki informasi yang tidak lengkap.

Kekurangan yang ada pada NE dapat diselesaikan oleh NEAT. NEAT menyelesaikan masalah ini dengan cara melacak *innovation number* ketika terjadi *crossover*. Setelah proses NEAT selesai, akan dilakukan optimasi menggunakan metode yang biasa digunakan pada ANN pada umumnya, yaitu *backpropagation*. Berbeda dengan penelitian terdahulu, penelitian kali ini akan mencoba melakukan prediksi hasil sebuah pertandingan beserta skornya. Bukan hanya tim mana yang akan memenangkan pertandingan.

## 1.1 Rumusan Masalah

Berdasarkan latar belakang yang telah diuraikan di atas, maka rumusan masalahnya adalah:

1. Bagaimana pengaruh dari *features* yang digunakan terhadap akurasi dari *Neuroevolution of Augmenting Topologies* (NEAT) dalam melakukan prediksi skor pertandingan sepak bola ?
2. Bagaimana akurasi dari network yang dihasilkan oleh *Neuroevolution of Augmenting Topologies* (NEAT) yang telah dioptimasi dengan *Backpropagation* dalam melakukan prediksi skor pertandingan sepak bola ?

## 1.2 Tujuan Penelitian

Tujuan dari Skripsi ini adalah menerapkan *Neurovolution of Augmenting Topologies* (NEAT) yang dioptimasi dengan *Backpropagation* untuk melakukan prediksi hasil akhir dari sebuah pertandingan sepak bola.

## 1.3 Ruang Lingkup

Ruang lingkup skripsi ini dibatasi pada:

- *Dataset*
    - *Dataset* yang digunakan berupa pertandingan pada *English Premier*. *League* (EPL) pada musim 2014 – 2015 sampai dengan 2017-2018.
    - *Dataset* diperoleh dari situs *whoscored.com*.
    - Pembagian data untuk *training* dan *testing* ialah 75:25.
- Metode yang digunakan adalah *Neuroevolution of Augmenting Topologies* (NEAT) dan *Backpropagation*.
- *Training* dan *Testing*
    - *Training* dan *testing* akan dibagi menjadi 3 tahapan, yaitu :
        1. Menggunakan *rating* pemain sebagai *feature*.
        2. Menggunakan *rating* pemain dan *rating* tim sebagai *feature*.
        3. Menggunakan *rating* pemain, *rating* tim, dan posisi setiap pemain sebagai *feature*.
    - Tujuan dari pembagian ini adalah untuk melihat pengaruh *feature* yang digunakan terhadap akurasi prediksi dari model.
- *Output* model berupa prediksi skor hasil akhir dari pertandingan.
- Model tidak dapat melakukan prediksi pemain mana yang mencetak goal.
- Bahasa pemrograman yang digunakan adalah *Python*, *Javascript*, dan *PHP*.
- Model dirancang menggunakan *library neat-python* dan *neataptic*.

## 1.4 Metodologi Penelitian

Langkah-langkah dalam pengerjaan Skripsi:

- Studi literatur tentang:
    - *Artificial Neural Network*
    - *Neuroevolution*
- Pengumpulan dan pengolahan data berupa:
    - *Rating* pemain
    - Pertandingan-pertandingan *English Premier League* yang terdahulu

- Perencanaan dan Pembuatan Perangkat Lunak:
    - Perancangan model ANN untuk melakukan prediksi skor pertandingan sepak bola
- Pengujian dan Analisis Perangkat Lunak:
    - Pengujian model yang telah dibuat
    - Analisis hasil model
- Pengambilan Kesimpulan:
    - Pengambilan kesimpulan

## 1.5 Sistematika Penulisan

Penulisan laporan Skripsi ini dibagi menjadi beberapa bab, yaitu:

BAB I : PENDAHULUAN

Bab I berisikan judul, latar belakang, perumusan masalah, ruang lingkup, tujuan skripsi, metodologi penelitian, dan sistematika penulisan yang akan digunakan

BAB II : LANDASAN TEORI

Bab II berisikan teori-teori serta metode-metode yang digunakan dalam pembuatan skripsi

BAB III : ANALISIS DAN DESAIN SISTEM

Bab III berisikan analisis dan desain sistem yang dibuat

BAB IV : IMPLEMENTASI SISTEM

Bab IV berisikan tentang implementasi sistem berdasarkan desain sistem seperti pada Bab III

BAB V : PENGUJIAN SISTEM

Bab V berisikan pengujian sistem yang telah dibuat pada Bab IV

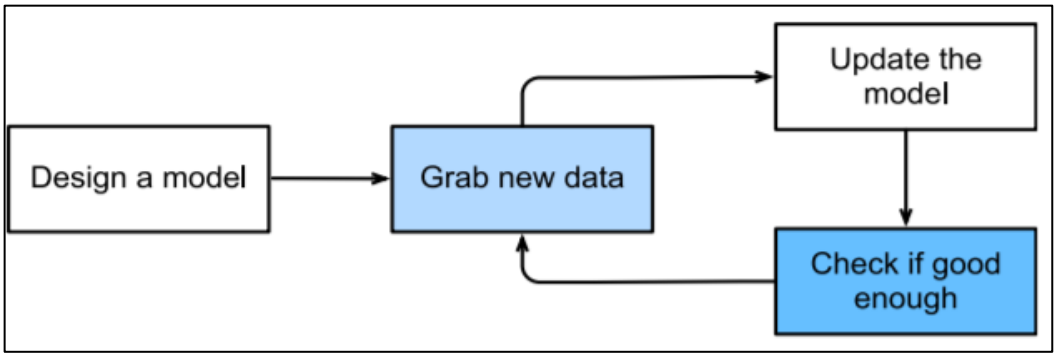
BAB VI : KESIMPULAN DAN SARAN

Bab VI berisikan kesimpulan yang dapat diambil terhadap hasil yang dicapai, dan saran – saran yang berguna bagi pengembangan selanjutnya.

# 2. LANDASAN TEORI

Pada bab ini akan dijelaskan mengenai teori-teori yang digunakan untuk melakukan prediksi skor pertandingan speak bola pada penelitian ini.

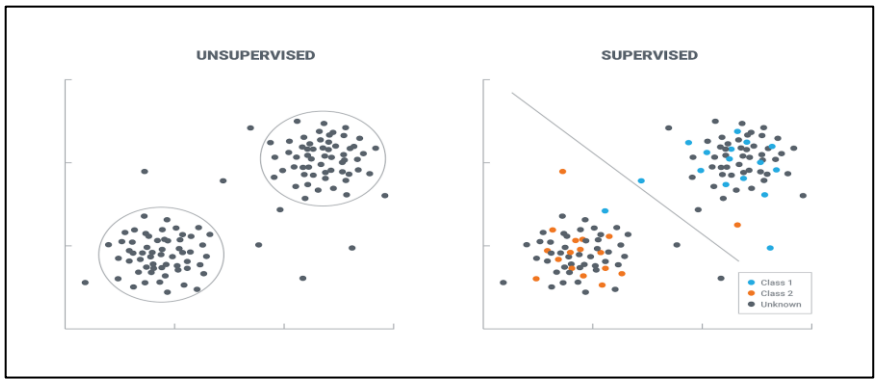
## 2.1 *Machine Learning*




Gambar 2.1 Ilustrasi cara kerja *Machine Learning*

Sumber: https://www.d2l.ai/chapter_introduction/intro.html

*Machine Learning* (ML) merubakan sebuah cabang dari *Computer Science* yang berhubungan pada pembangunan sebuah model berdasarkan data-data dari sebuah fenomena [7]. ML bekerja secara iteratif untuk mempelajari, menggambarkan, dan melakukan prediksi dari sebuah kumpulan data (Hurwitz & Kirsch, 2018). Dengan mempelajari sebuah kumpulan data, ML kemudian mampu membuat suatu model berdasarkan data tersebut.



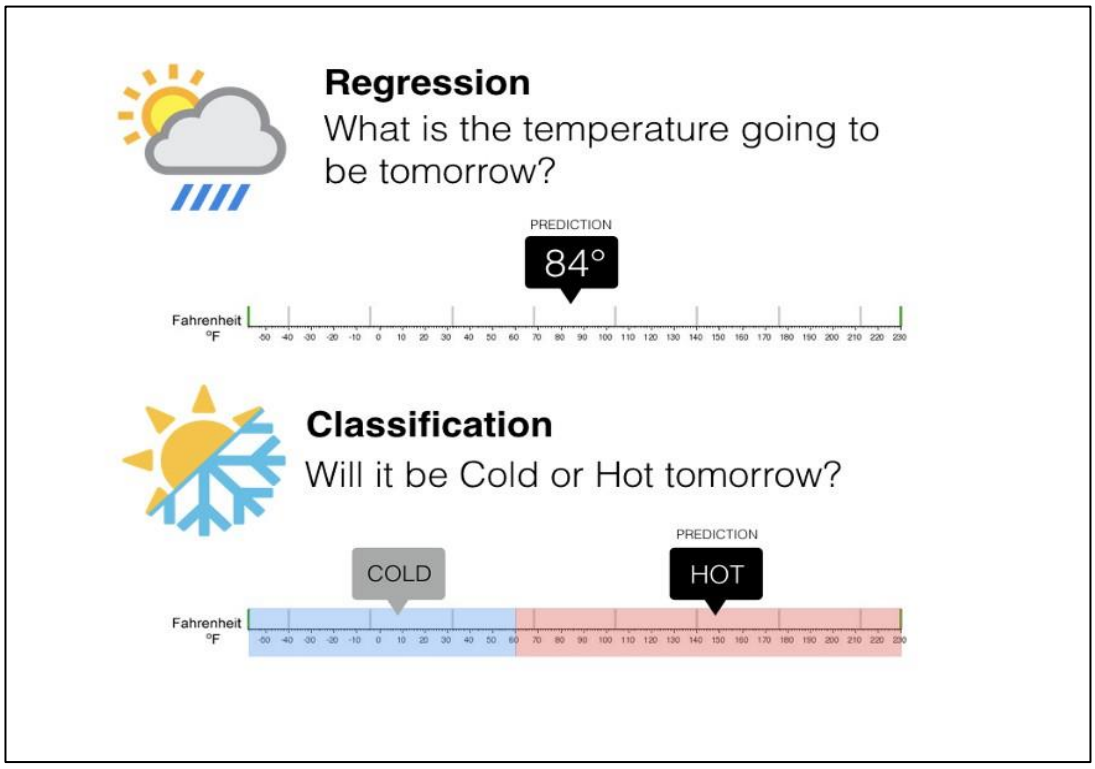

Gambar 2.2 Ilustrasi supervised learning dan unsupervised learning

Sumber: https://lawtomated.com/supervised-vs-unsupervised-learning-which-is-better/

Proses learning dalam machine learning dibagi menjadi beberapa kategori, di antaranya *supervised learning* dan *unsupervised learning*.

Untuk melakukan prediksi skor pada pertandingan sepak bola, proses learning yang akan digunakan adalah *supervised learning*

### *2.1.1 Supervised Learning*




Gambar 2.3 Ilustrasi masalah regressi (*regression*) dan klasifikasi (*classification*)

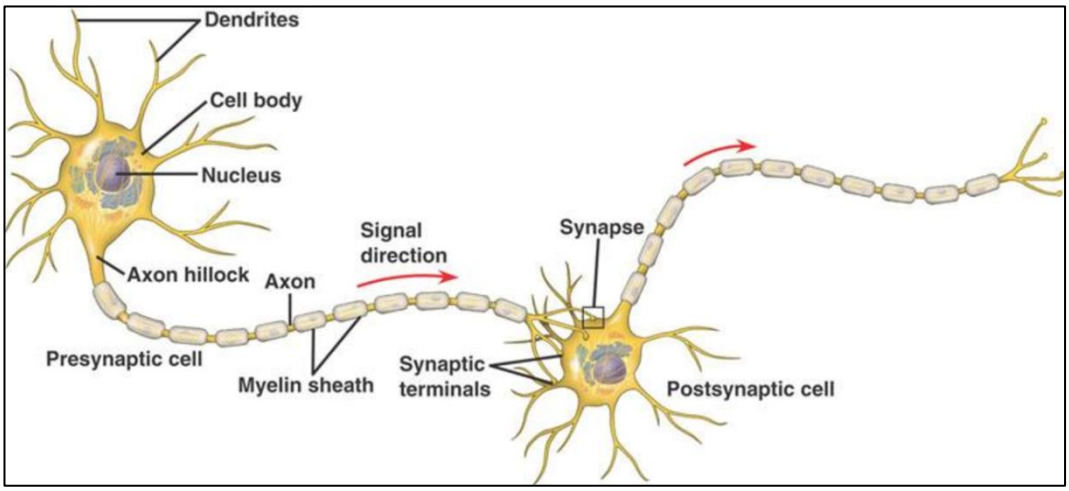
Sumber: https://lawtomated.com/supervised-vs-unsupervised-learning-which-is-better/

*Supervised Learning* memiliki tujuan untuk melakukan sebuah prediksi dari input data yang diberikan. Hasil prediksi dari *supervised learning*, yang biasa disebut label, dapat dinotasikan sebagai $y$, Sedangkan input data, yang biasa disebut *examples* juga dinotasikan sebagai $x$. Tujuan dari *supervised learning* adalah untuk membuat sebuah model $f\theta$ yang mampu memetakan input $x$ kepada prediksi $f\theta(x)$ (Zhang, Lipton, & Mu Li, 2019).

Pada supervised learning, data yang digunakan adalah data yang sudah memiliki label. Contohnya pada penelitian ini, data yang digunakan memiliki label berupa skor akhir pada sebuah pertandigan, seperti [1,0], [2,0], dan [1,1]. Pada umumnya, *supervised learning* digunakan melakukan klasifikasi (hasil prediksi berupa kategori, seperti untuk melakukan prediksi apakah cuaca akan panas atau dingin) dan regresi (hasil prediksi berupa angka, seperti untuk melakukan prediksi temperatur).

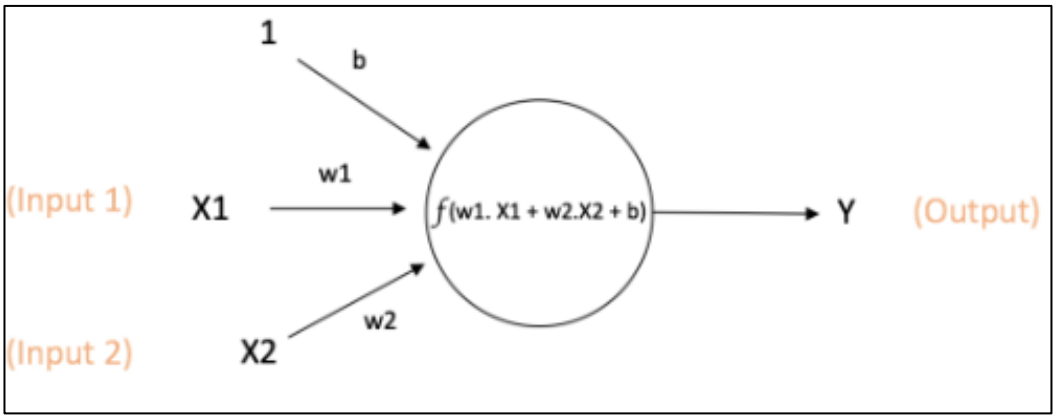
## 2.2 Jaringan Saraf Tiruan

Jaringan saraf tiruan (JST) atau *Artificial Neural Network* (ANN) merupakan cabang dari *machine learning* yang menggambarkan representasi buatan dari otak manusia yang mencoba untuk mensimulasikan proses pembelajaran dari otak manusia (Negnevitsky, 2005). Bentuk representasi ANN adalah berupa jaringan yang terdiri dari kumpulan unit pemroses kecil yang biasa disebut neuron, yang bersifat adaptif karena mampu mengubah struktur unit-unit tersebut untuk menyelesaikan masalah berdasarkan informasi (input) baik informasi eksternal, maupun informasi internal. ANN mampu belajar layaknya otak manusia dengan cara memberi bobot pada tiap neuron. Saat proses pembelajaran sedang berlangsung, neuron akan di *update* berdasarkan *error* yang didapat.




Gambar 2.4 Ilustrasi neuron pada otak manusia

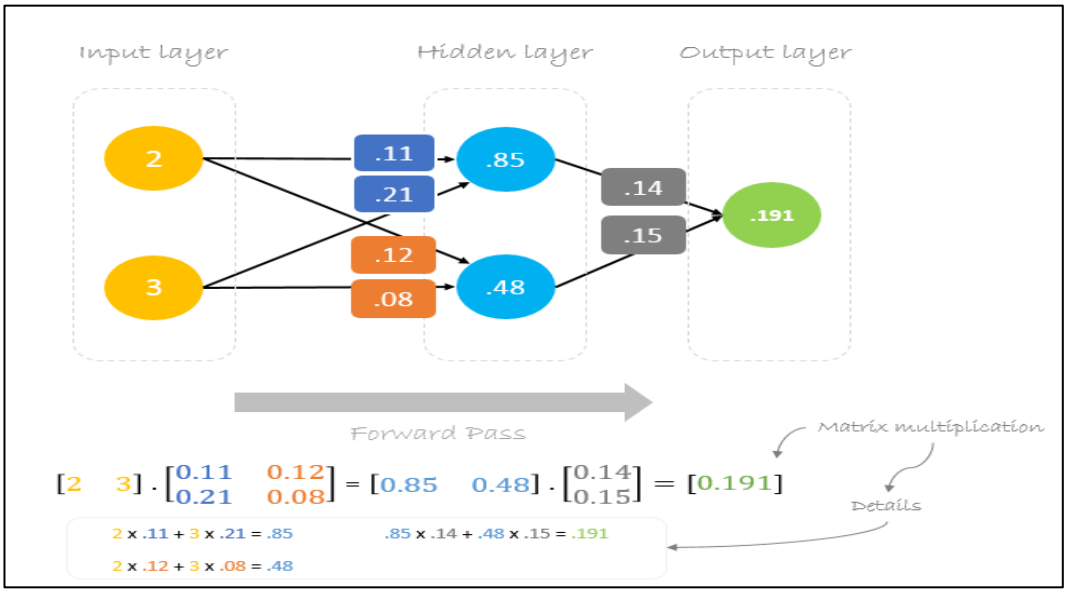
Sumber: https://www.semanticscholar.org/paper/An-Introduction-to-Artificial-Neural-Networks-(-ANN-Heger/7524e25abbdea2f982eb673ca1e60773c118cd66/figure/1




Gambar 2.5 Ilustrasi neuron pada *artificial neural network*

Sumber: http://hmkcode.github.io/images/ai/bp_forward.png

Setiap neuron pada JST memiliki bobot atau *weight* yang menghubungkan suatu neuron kepada neuron lainnya. Selain menghubungkan tiap neuron, bobot juga berguna untuk melakukan *scaling* terhadap input yang diterima oleh setiap neuron. *Output* dari tiap neuron kemudian akan diteruskan ke neuron selanjutnya hingga mencapai neuron pada lapisan terakhir. Proses ini disebut dengan *feed-forward* atau *forward-pass.*

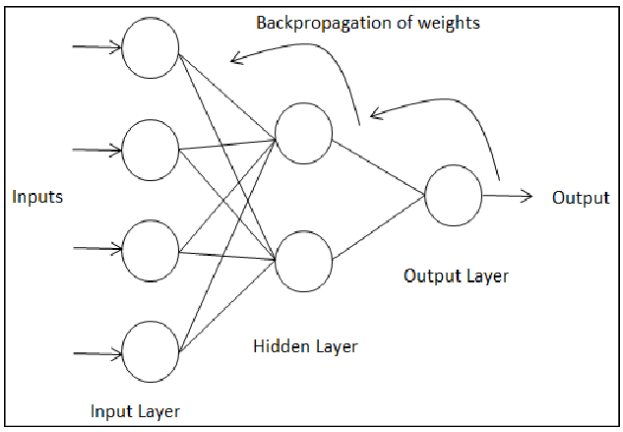



Gambar 2.6 Ilustrasi *forward-pass* pada *artificial neural network*

Sumber: http://hmkcode.github.io/images/ai/bp_forward.png

## 2.3 *Backpropagation*

*Backpropagation* adalah suatu proses pembelajaran bertipe *supervised learning* pada JST (Negnevitsky, 2005). Pada umumnya, terdapat 3 lapisan pada JST, yaitu lapisan input, lapisan tersembunyi, dan lapisan *output*. Berbeda dengan *forward-pass* yang dimulai dari lapisan terdepan atau lapisan input, *backpopagation* dimulai dari lapisan paling akhir atau lapisan *output*.




Gambar 2.7 Ilustrasi *Backpropagation*

Sumber: https://i.imgur.com/qNGcRby.png

*Backpropagation* bekerja dengan cara menghitung *error* dari sebuah data, *error* didapat dengan cara *error = (target – trediction)*$^2$ yang kemudian *error* tersebut akan digunakan untuk menyesuaikan atau meng-*update* bobot-bobot yang ada pada JST, sehingga kesalahan atau *error rate* dapat diperkecil yang kemudian akan menghasilkan prediksi yang lebih akurat [11].

Berikut adalah langkah-langkah untuk melakukan *backpropagation* (Negnevitsky, 2005) :

1. Tentukan semua bobot atau *weight* yang ada pada JST secara acak dalam *range* [-0.5,0.5].
2. Hitung *output* semua neuron, dimulai dari lapisan tersembunyi sampai dengan lapisan *output* menggunakan persamaan berikut:

$$Y(p) = step\left[\sum_{i=1}^{n} x_i(p)w_i(p) - \theta\right] \quad (2.1)$$

dimana n adalah jumlah *input* pada setiap neuron, x adalah *input signal*, w adalah bobot, $\theta$ adalah *threshold*, p adalah iterasi, dan *step* merupakan *activation function* yang digunakan.

3. Hitung *error signal* untuk neuron pada lapisan *output* menggunakan persamaan berikut:

$$e(p) = y_d(p) - y(p) \tag{2.2}$$

dimana $y_d$ adalah label, atau *output* yang diinginkan.

Hitung *error gradient* untuk neuron pada lapisan *output* menggunakan persamaan berikut:

$$\delta(p) = y(p) \times [1 - y(p)] \times e(p) \tag{2.3}$$

Hitung *weight correction* untuk neuron pada lapisan *output* menggunakan persamaan berikut:

$$\Delta w(p) = \alpha \times y(p) \times \delta(p) \tag{2.4}$$

dimana $\alpha$ adalah *learning* rate yang digunakan.

*Update* bobot untuk neuron pada lapisan *output* menggunakan persamaan berikut:

$$w(p+1) = w(p) + \Delta w(p) \tag{2.5}$$

Hitung *error gradient* untuk neuron pada lapisan tersembunyi menggunakan persamaan berikut:

$$\delta_j(p) = y_j(p) \times \left[1 - y_j(p)\right] \times \sum_{k=1}^{l} \delta_k(p) \times w_{jk}(p) \tag{2.6}$$

dimana $l$ adalah jumlah neuron pada lapisan sebelumnya.

Hitung *weight correction* untuk neuron pada lapisan tersembunyi menggunakan persamaan berikut:

$$\Delta w(p) = \alpha \times x(p) \times \delta(p) \tag{2.7}$$

dimana $a$ adalah *learning rate* yang digunakan.

Update bobot untuk neuron pada lapisan tersembunyi menggunakan persamaan berikut:

$$w(p+1) = w(p) + \Delta w(p) \tag{2.8}$$

4. Ulangi proses diatas mulai dari langkah ke-2 hingga didapatkan *Sum of Squared Errors* yang ideal

## 2.4 Algoritma Genetika (*Genetic Algorithm*)

Algoritma Genetika atau GA merupakan sebuah *search heuristic* yang terinspirasi dari teori evolusi yang dikemukakan oleh Charles Darwin. Algoritma ini mencerminkan proses dari seleksi alam dimana individu terbaiklah yang dipilih untuk melakukan reproduksi yang bertujuan untuk menghasilkan keturunan atau terciptanya sebuah generasi baru (Mallawaarachchi, 2017).

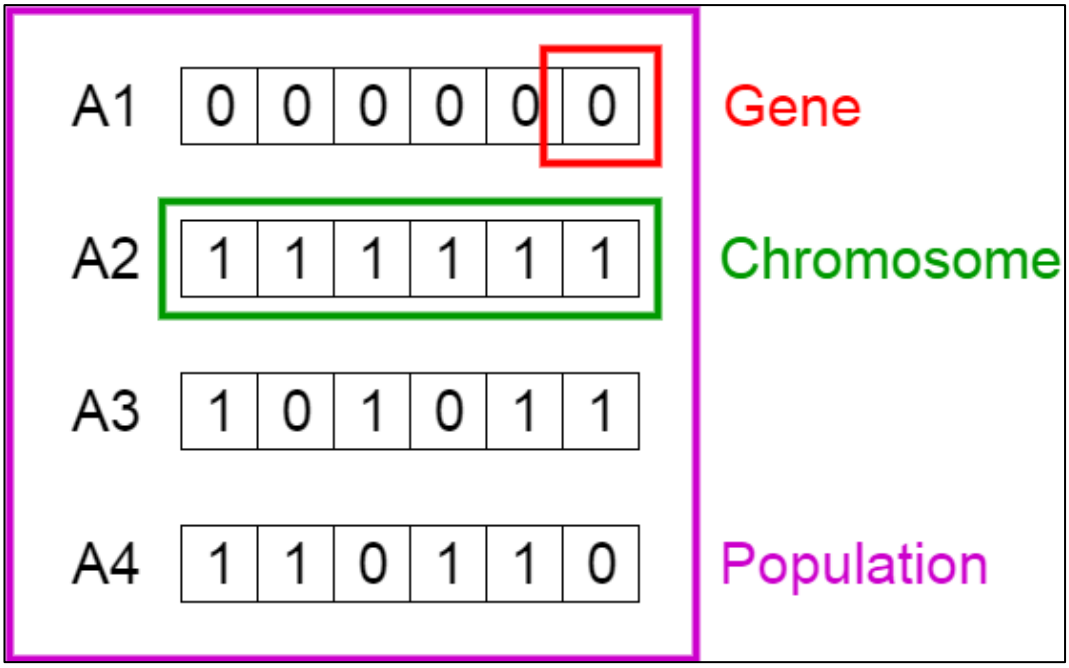
Proses seleksi alam dalam GA dimulai dengan melakukan seleksi dalam sebuah populasi yang bertujuan untuk menemukan individu-individu yang terbaik. Individu-individu terbaik tersebut akan dipilih menjadi *parents* untuk melakukan reproduksi. *Crossover* dilakukan agar terciptanya keterunan yang mewarisi karakteristik dari *parents*nya. Jika *parents* memiliki tingkat *fitness* yang baik, maka keturunannya akan memiliki kesempatan yang lebih besar untuk bertahan dalam sebuah populasi atau bahkan memiliki *fitness* yang lebih baik dari *parents*nya.

Ada 5 fase utama dalam GA, yaitu *Initial Population, Fitness Function, Selection, Crossover,* dan *Mutation.*

- ***Initial Population***

Fase pertama dalam GA adalah *initial population. Initial population* adalah kumpulan dari individu yang merupakan solusi terhadap masalah yang ingin diselesaikan. Sebuah individu memiliki berbagai macam parameter yang dikenal sebagai *genes*. *Genes* dapat digabungkan untuk membentuk sebuah *chromosome*.

Dalam GA, *genes* pada sebuah individu direpresentasikan menggunakan alfabet dalam bentuk *string.* Biasanya, genes akan di*encode* dengan dengan nilai biner (kumpulan dari 1 dan 0).




Gambar 2.8 Ilustrasi gene, *chromosome*, dan *population* pada GA

Sumber: https://miro.medium.com/max/695/1*vIrsxg12DSltpdWoO561yA.png

- ***Fitness Function***

*Fitness function* adalah sebuah fungsi yang berguna untuk menentukan tinggat *fitness* suatu individu dalam populasi. *Fitness function* memberikan sebuah nilai kepada masing masing individu, yang nanti akan diseleksi untuk melakukan reproduksi berdasarkan nilai *fitness*nya

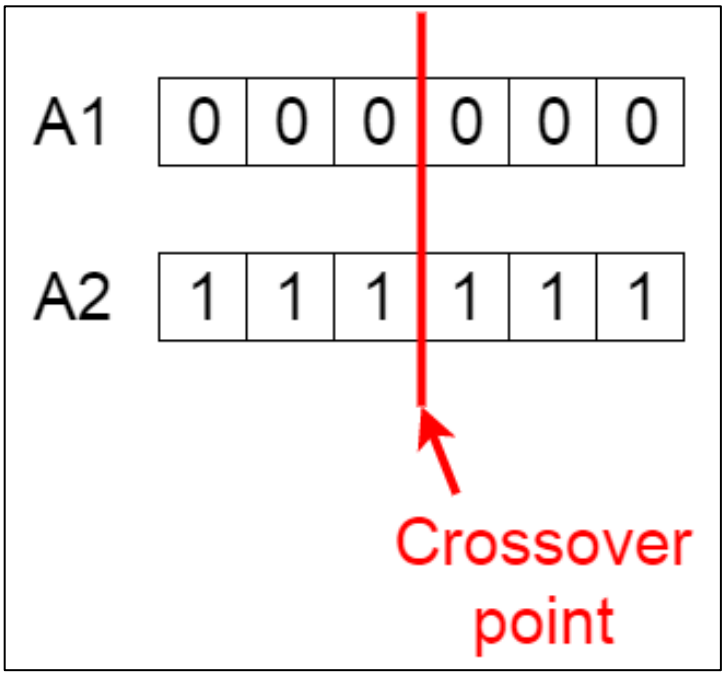
- ***Selection***

Setelah semua individu pada populasi memiliki nilai *fitness*nya, akan dilakukan seleksi untuk menentukan individu mana yang akan melakukan reproduksi. Ada banyak cara untuk melakukan seleksi, salah satunya adalah metode *elitist*. Pada metode *elitist*, 2 individu terbaik akan dipilih untuk melakukan reproduksi yang bertujuan untuk menciptakan sebuah populasi baru.

- ***Crossover***

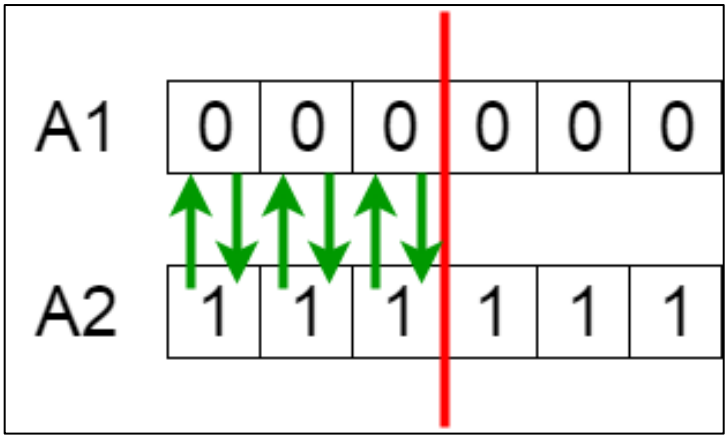
Crossover merupakan bagian terpenting dalam GA. Tujuan dari *crossover* terciptanya keturunan atau *offspring* yang memiliki karakteristik yang mirip dengan *parents*nya. Setelah dipilih 2 individu yang akan berperan sebagai *parents*, akan dipilih titik *crossover* pada *genes* secara acak.

Offstring diciptakan dengan cara melakukan pertukaran *genes* dari kedua *genes* yang dimiliki oleh *parents.*



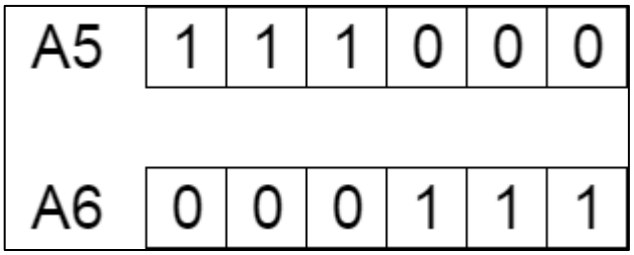

Gambar 2.9 Ilustrasi proses *crossover* pada GA

Sumber: https://miro.medium.com/max/409/1*Wi6ou9jyMHdxrF2dgczz7g.png




Gambar 2.10 Ilustrasi proses *crossover* pada GA

Sumber: https://miro.medium.com/max/389/1*eQxFezBtdfdLxHsvSvBNGQ.png

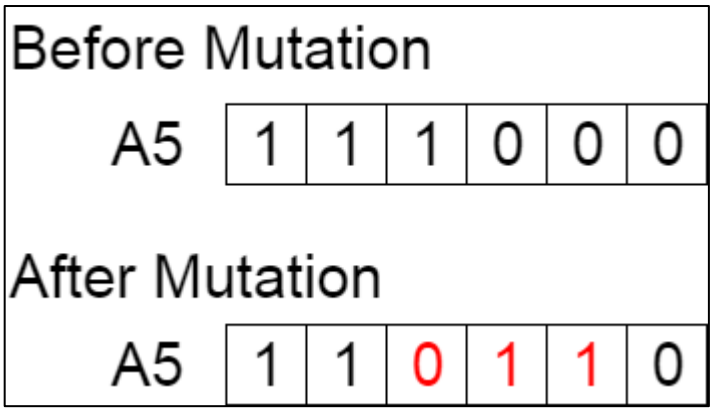



Gambar 2.11 Ilustrasi proses *crossover* pada GA

Sumber: https://miro.medium.com/max/389/1*_Dl6Hwkay-UU24DJ_oVrLw.png

- ***Mutation***

Setelah *genes offsprings* tercipta, akan dilakukan 1 proses lagi sebelum menambah *offsprings* tersebut kedalam generasi baru, yaitu *mutation.* Mutation pada GA terjadi secara acak, dengan probabilitas yang biasanya kecil. Tujuan dilakukannya *mutation* untuk menjaga tingkat variasi didalam sebuah populasi.




Gambar 2.12 Ilustrasi proses *mutation* pada GA

Sumber: https://miro.medium.com/max/439/1*CGt_UhRqCjIDb7dqycmOAg.png

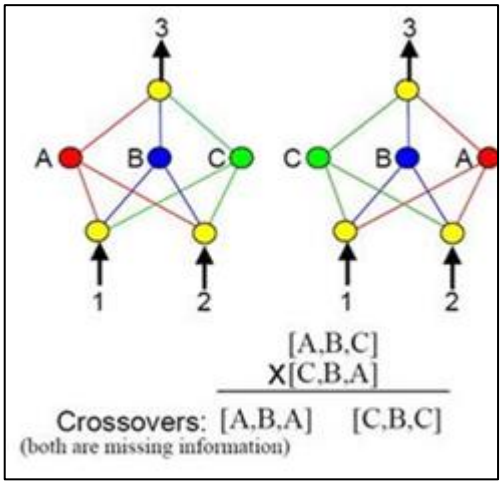
Kelima proses diatas akan dijalankan secara iteratif hingga solusi ditemukan atau mengalami kondisi yang dinamakan *converged*, yaitu suatu kondisi dimana *offsprings* yang dihasilkan tidak memiliki perbedaan yang signifikan dengan generasi sebelumnya.

## 2.5 Neuroevolution of Augmenting Topologies (NEAT)

*Neuroevolution* (NE) merupakan sebuah algoritma penggabungan antara *Genetic Algorithm* (GA) dan *Artificial Neural Network* (ANN). Dalam tradisional

NE, akan dilakukan pemilihan topologi dari sebuah ANN sebelum eksperimen dimulai. Biasanya, topologi dari ANN berupa 1 lapisan tersembunyi yang terhubung ke semua lapisan input dan lapisan *output.* Pencarian bobot-bobot atau *weights* kemudian akan dilakukan menggunakan GA, seperti *crossover* dan *mutation*. Maka, tujuan dari tradisional NE atau *Fixed-Topology Neuroevolution* adalah melakukan optimasi pada bobot-bobot sehingga dapat menemukan ANN yang fungsional (Stanley & Miikkulainen, 2002).

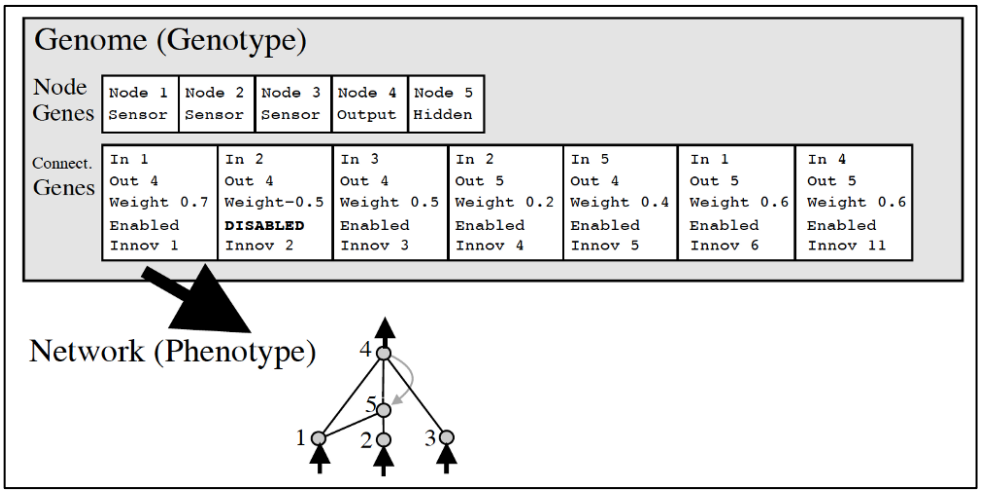
Namun, bobot-bobot yang ada pada ANN bukan penentu satu-satunya terhadap *behaviour* dari sebuah ANN. Topologi atau struktur dari ANN itu sendiri juga mempengaruhi bagaimana ANN bekerja (Chen, et al., 1993). Selain itu, ketika terjadi *crossover* pada tradisional NE, adanya kemungkinan informasi akan hilang, sehingga menciptakan *offspring* yang "cacat".




Gambar 2.13 Hilangnya informasi ketika terjadi crossover antar ANN

Sumber: Stanley & Miikkulainen. Evolving Neural Networks through Augmenting Topologies. 2002

Pada ilustradi diatas, dapat dilihat bahwa hilangnya informasi pada *offspring* yang dihasilkan oleh hasil *crossover* antara jaringan ABC dan CBA. Neuroevolution of Augmenting Topologies (NEAT) mencoba menjawab permasalahan ini dengan memberikan *historical marking* dengan *innovation number* pada setiap *connection genes* yang ada didalam jaringan.




Gambar 2.14 *Encoding* dan *innovation number* pada NEAT

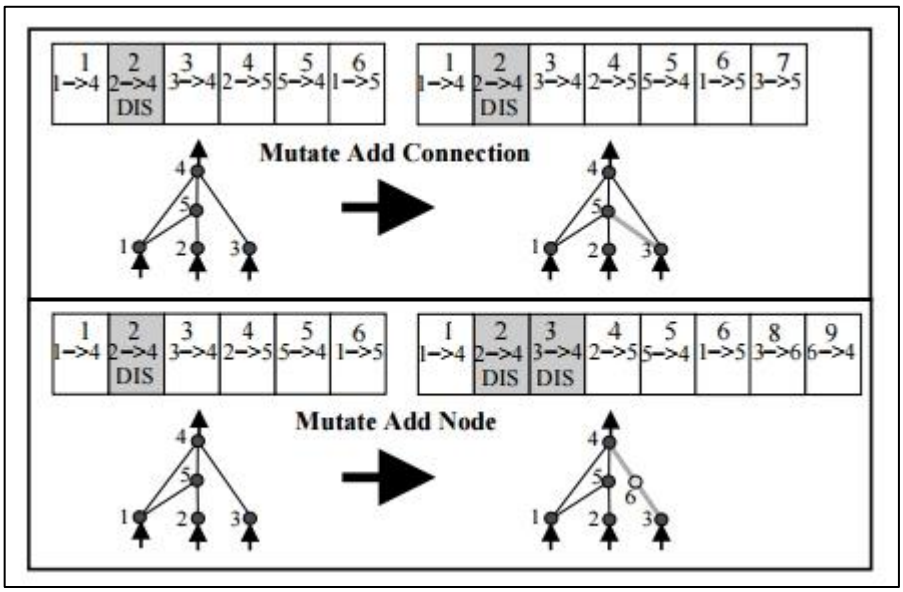
Sumber: Stanley & Miikkulainen. Evolving Neural Networks through Augmenting Topologies. 2002

### 2.5.1 *Genetic Encoding*

*Genetic Encoding* adalah bentuk representasi linear dari koneksi yang ada pada jaringan (Stanley & Miikkulainen, 2002) (Gambar 2.13). *Genetic Encoding* pada NEAT didesain agar mudah untuk disejajarkan ketika terjadi *crossover*. Setiap *genome* pada NEAT terdiri dari 2 *genes*, yaitu *node genes* dan *connection genes*. *Connection genes* berisi atas kumpulan koneksi pada jaringan, yang mengacu kepada 2 *node genes* yang masing-masing mempunya informasi tentang *nodes* yang tersedia pada jaringan. *Connection genes* juga berisikan atas informasi tentang *in-node, out-node,* bobot dari koneksi, apakah koneksi tersebut aktif, dan *innovation number*, yang nanti akan berguna ketika terjadinya *crossover*.

*Mutation* pada NEAT dapat mengubah bobot koneksi atau struktur dari jaringan itu sendiri. Perubahan bobot yang terjadi karena *mutation* sama dengan *mutation* yang ada pada NE tradisional, yaitu antara terputusnya koneksi atau tidak (Stanley & Miikkulainen, 2002). Sebaliknya, untuk *structural mutation*, ada 2 kemungkinan yang bisa terjadi, yaitu *add connection mutation* dan *add node mutation*. Pada *add connection mutation*, satu koneksi baru dengan bobot acak akan muncul untuk menyambungkan 2 *node* yang sebelumnya tidak tersambung. Selanjutnya, pada *add node mutation*, koneksi antara 2 *nodes* akan terbagi dan *node* baru akan muncul diantara 2 *nodes* yang bersambungan tersebut. Koneksi lama

yang menyambungkan 2 *nodes* awal akan mati, dan 2 koneksi baru akan ditambahkan ke *connection genes*. Koneksi pertama akan menyambungkan *node-in* awal ke *node* baru, kemudian koneksi kedua akan menyambungkan *node* baru kepada *node-out* awal (Gambar 2.14).




Gambar 2.15 *Structural mutation* pada NEAT

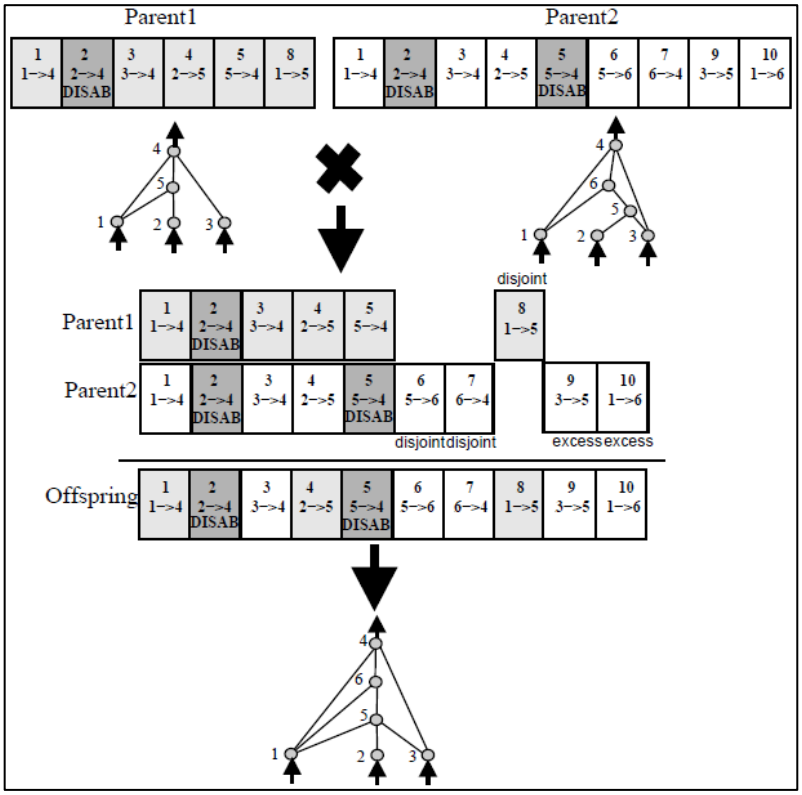
Sumber: Stanley & Miikkulainen. Evolving Neural Networks through Augmenting Topologies. 2002

### 2.5.2 *Historical Marking*

Tanpa adanya *historical marking*, akan sulit untuk mengetahui tentang kecocokan suatu *gene* dengan *gene* lainnya dalam populasi yang luas (Stanley & Miikkulainen, 2002). Dua *genes* yang memiliki kesamaan *historical origins* harus memiliki struktur yang sama (dengan adanya kemungkinan untuk memiliki bobot yang berbeda), karena mereka sama-sama berasal dari satu *ancestral gene* pada masa lalu.

NEAT mencoba menangkal masalah ini dengan memberikan innovation number pada setiap *connection genes*. Ketika muncul *connection genes* baru, *global innovation number* bertambah dan diberikan kepada *connection genes* baru tersebut. Jadi, *Innovation number* disini dapat mewakilkan sebuah kronologi dari kemunculan suatu *gene* dalam sistem. Sebagai contoh, anggaplah *mutation* pada Gambar 2.14 terjadi secara berurutan. *Connection gene* yang tercipta pada *mutation*

pertama mendapat 7 sebagai *innovation number*. Sedangkan pada *mutation* kedua, dua *connection genes* yang tercipta mendapat 8 dan 9 sebagai *innovation number*nya. Dimasa depan, ketika dua *genomes* ini melakukan *crossover*, *offspring* yang dihasilkan akan mewarisi *innovation number* yang sama. *Innovation number* ini tidak akan pernah berubah. Dengan demikian, *historical origin* dari setiap gene akan tetap terjaga.




Gambar 2.16 *Crossover* pada NEAT

Sumber: Stanley & Miikkulainen. Evolving Neural Networks through Augmenting Topologies. 2002

*Historical marking* ini memberi NEAT suatu kemampuan yang powerful. Dengan *historical marking*, NEAT dapat mengetahui dengan tepat tentang kecocokan suatu *gene* dengan *gene* lainnya. Ketika terjadi *crossover*, *connection genes* akan disejajarkan untuk mencocokan *innovation number* pada kedua *genes*. *Genes* yang memiliki *innovation number* yang sama disebut sebagai *matching genes*. Sedangkan *genes* yang tidak memiliki kecocokan disebut sebagai *disjoint* atau *excess genes*, tergantung apakah mereka muncul dalam *range innovation number* yang dimiliki oleh *parent* lainnya. *Disjoint* dan *excess genes* disini

mewakilkan struktur yang tidak dimiliki oleh salah satu *genome* saat terjadinya *crossover*. Dalam pembuatan sebuah *offspring*, *matching genes* akan dipilih salah satu secara acak dari kedua *parents genome*. Sedangakan *disjoint* dan *excess genes* akan diturunkan oleh *parent* yang memiliki nilai *fitness* yang lebih baik.

*Historical marking* memberikan kemudahan dalam melakukan analisa pada jaringan sebelum dilakukannya *crossover*. Dengan adanya *historical marking* ini, dapat tercipta populasi yang memiliki keragaman yang luas. Tetapi, dengan beberagaman yang luas ini suatu populasi akan mengalami kesulitan untuk menjaga sebuah inovasi, dikarenkan struktur jaringan yang kecil membutuhkan waktu lebih sedikit untuk berkembang daripada struktur jaringan yang besar. Masalah ini juga muncul ketika terjadi *crossover* atau *structural mutation*, individu baru yang terbentuk biasanya tidak memiliki kemampuan sebaik *parents*nya, walaupun adanya kemungkinan dari mereka untuk melampaui *parents*nya dimasa depan, mereka dapat mengalami kepunahan dini sebelum memiliki waktu untuk berkembang. NEAT juga menawarkan solusi untuk masalah ini dengan menghadirkan *speciation*.

### 2.5.3 *Speciation*

Membagi populasi kedalam beberapa spesies memungkinkan suatu individu untuk berkompetisi dengan individu sejenisnya. Dengan cara ini, inovasi yang terbentuk dapat terlindungi dalam sebuah komunitas dimana mereka memiliki waktu untuk berkembang. Ide utama dari *speciation* adalah mengelompokkan individu dalam populasi ke beberapa spesies berdasarkan topologinya. Pengelompokkan berdasarkan topologi dapat diselesaikan menggunakan *historical marking*.

Jumlah dari *excess* dan *disjoint genes* dari sepasang *genomes* menjadi tolak ukur apakah kedua *genomes* memiliki kecocokan. Semakin *disjoint* kedua *genomes*, maka semakin sedikit pula mereka memiliki kesamaan sejarah dalam evolusinya, yang berefek pada semakin sedikitnya kecocokan yang mereka miliki. Maka dari itu, kecocokan (*compability distance* $\delta$) *genomes* dalam NEAT dapat diukur dengan persamaan berikut:

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \times \overline{W} \tag{2.9}$$

dimana *E* dan *D* merukapan *excess* dan *disjoint* genes, dan $\overline{W}$ adalah rata-rata perbedaan bobot dari *matching genes*. Koefiesien $c_1, c_2, c_3$ berguna untuk mengatur tingkat kepentingan dari masing-masing *E, D, dan* $\overline{W}$, sedangkan N merupakan jumlah *genes* pada *genome* yang lebih besar, yang dinormalisasi untuk ukuran *genome* (*N* dapat diatur menjadi 1 jika kedua *genomes* berukuran kecil, seperti kurang dari 20 *genes*).

Dari *compability distance* $\delta$, dapat dilakukan pengelompokkan menggunakan *compability threshold* $\delta_t$, dimana genome dianggap satu spesies jika *compability distance*nya berada dibawah *compability threshold* yang telah ditetapkan. Pada setiap generasi, *genomes* akan dikelompokkan secara berurutan kedalam spesies. Setiap spesies yang tersedia diwakilkan oleh sebuah *genome* yang dipilih secara acak dari spesies tersebut pada generasi sebelumnya. Contohnya, sebuah *genome* $g$ pada generasi yang sedang berlangsung akan dikelompokkan pada spesies pertama yang diwakilkan oleh suatu *genome* dari generasi sebelummua, yang memiliki kecocokan dengan $g$. Dengan cara ini, tidak akan terjadi *overlap* pada species. Jika $g$ tidak memiliki kecocokan dengan spesies yang tersedia, maka spesies baru akan muncul.

Mekanisme reproduksi pada NEAT menggunakan *explicit fitness sharing* (Stanley & Miikkulainen, 2002), dimana setiap organisme pada species akan memilik nilai *fitness* yang sama dengan *niche*nya. Jadi, sebuah spesies tidak akan menjadi terlalu besar walaupun banyak anggotanya memiliki performa yang baik. Dengan begini, kecil kemungkinan untuk satu species mengambil alih populasi yang mana akan menjadi bagian krusial dalam evolusi.

Penyesuaian *fitness* $f_i'$ untuk organisme $i$ dihitung berdasarkan jarak $\delta$ dari setiap organisme *j* pada populasi dengan persamaan berikut:

$$f_i' = \frac{f_i}{\sum_{j=1}^{n} sh(\delta\ (i,j))} \tag{2.10}$$

*sharing function* $sh$ akan diset ke 0 jika $\delta\ (i,j)$ melebihi *threshold* $\delta_t$ yang telah ditentukan, jika tidak, $sh$ akan diset ke 1 (Stanley & Miikkulainen, 2002). Dengan

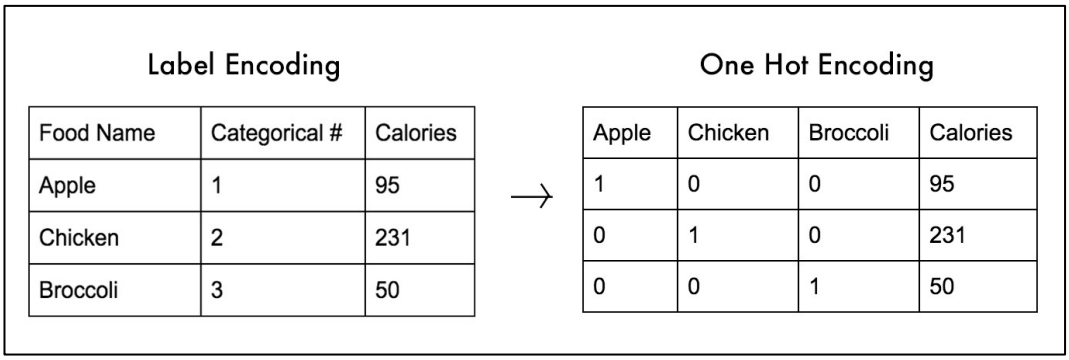
begitu, $\sum_{j=1}^{n} sh(\delta\ (i,j))$ akan mengurangi jumlah dari organisme pasa spesies yang sama menjadi organisme $i$. Pengurangan ini bersifat natural karena sebelumnya spesies telah dikelompokkan berdasarkan *compability threshold* $\delta_t$. Setiap spesies dapat menciptakan jumlah *offsprings* yang kemungkinan berbeda-beda berdasarkan jumlah total dari penyesuaian *fitness* $f_i'$. Spesies kemudian akan melakukan reproduksi, dimulai dengan mengeliminasi anggota yang memiliki performa terburuk. Hasil dari reproduksi inilah yang akan mengisi populasi yang baru.

## 2.6 *Encoding*

*Pre-processing* adalah metode untuk menyiapkan data sebelum diberikan kepada suatu model. *Encoding* merupakan bagian dari *pre-processing* yang bertujuan untuk merepresentasikan data agar dapat dimengerti oleh model.

*Machine Learning* tidak bisa bekerja dengan data bertipe kategori secara langsung, data bertipe kategori harus diubah terlebih dahulu kedalam bentuk angka (Brownlee, 2017). Pada penelitian ini, ada data yang berbentuk kategori, yaitu posisi setiap pemain. Karena model tidak dapat melakukan analisa terhadap posisi pemain yang berbentuk kategori, seperti *striker, midfielder,* dan *defender,* data ini harus di*encode* terlebih dahulu.

Ada 2 metode *encoding* yang dapat dilakukan untuk mengubah data berbentuk kategori ke angka, yaitu label *encoding* dan *one hot encoding.*




Gambar 2.17 *Label encoding* dan *one hot encoding*

Sumber: https://miro.medium.com/max/2736/0*T5jaa2othYfXZX9W

*Label encoding* merepresentasikan setiap kategori pada data kedalam suatu *integer* sehingga dapat dipahami oleh model. Namun, karena *output* yang

dihasilkan metode ini berupa *integer,* maka *output* yang dihasilkan akan memiliki *natural ordered relationship,* yang berarti komputer secara otomatis akan memberikan bobot yang lebih besar kepada kategori yang memiliki nilai *categorical* yang lebih besar, ini akan menjadi masalah pada data yang tidak memiliki *ordinal relationship*.

Berbeda dengan *label encoding* yang menghasilkan nilai *categorical* berupa *integer, one hot encoding* melakukan *binarization* pada data. Metode ini merupakan metode yang tepat untuk data yang tidak memiliki *ordinal relationship*, seperti posisi pemain pada pertandingan sepak bola. Misalnya, jika menggunakan label encoding, akan ada posisi tertentu yang memiliki bobot lebih tinggi dari posisi lainnya, padahal setiap posisi memiliki perannya masing-masing dan tidak memiliki *ordinal relationship*.

# 3. DESAIN SISTEM

Pada bab ini, akan dibahas desain sistem yang akan digunakan untuk melakukan prediksi skor pertandingan sepakbola dengan menggunakan *Neuroevolution of Augmenting Topologies* dan *Backpropagation*.

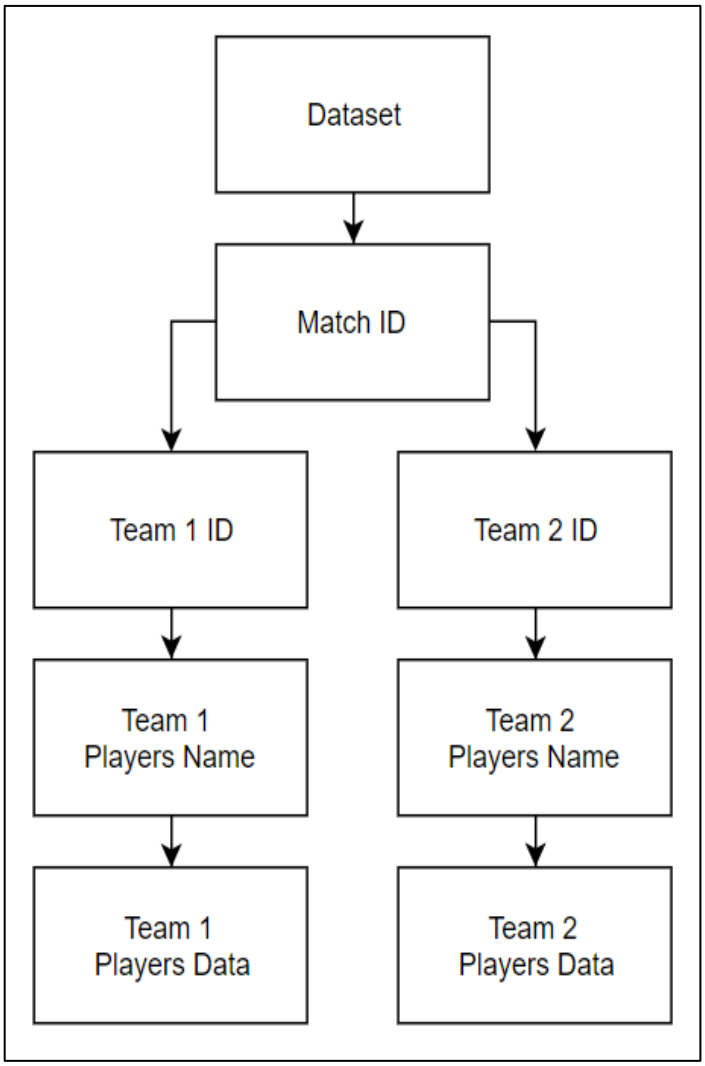
## *3.1 Dataset*

Untuk melakukan prediksi, sebuah *machine learning* model membutuhkan data untuk dipelajari terlebih dahulu. Setelah proses *learning* selesai dilakukan, barulah model mampu membuat prediksi berdasarkan data yang yang telah dipelajari.

Dalam penelitian ini, model akan mempelajari data pertandingan sepak bola, dan melakukan prediksi skor pertandingan yang akan datang berdasarkan data tersebut.


```
"Player_stats":{
    "Wojciech Szczesny":{
        "player_details":{
            "player_id":"73379",
            "player_name":"Wojciech Szczesny",
            "player_position_value":"1",
            "player_position_info":"GK",
            "player_rating":"5.81"
        },
        "Match_stats":{
            "touches":"20",
            "saves":"1",
            "total_pass":"13",
            "aerial_won":"1",
            "formation_place":"1",
            "accurate_pass":"11"
        }
    },
    "Calum Chambers":{
        "player_details":{
            "player_id":"124316",
            "player_name":"Calum Chambers",
            "player_position_value":"2",
            "player_position_info":"DC",
            "player_rating":"7.33"
        },
        "Match_stats":{
            "touches":"111",
            "total_tackle":"4",
            "aerial_lost":"2",
            "total_pass":"93",
            "fouls":"1",
            "aerial_won":"2",
            "formation_place":"5",
            "accurate_pass":"86",
            "yellow_card":"1"
        }
    },
    "Mathieu Debuchy":{
        "player_details":{
            "player_id":"11367",
            "player_name":"Mathieu Debuchy",
            "player_position_value":"2",
            "player_position_info":"DR",
```

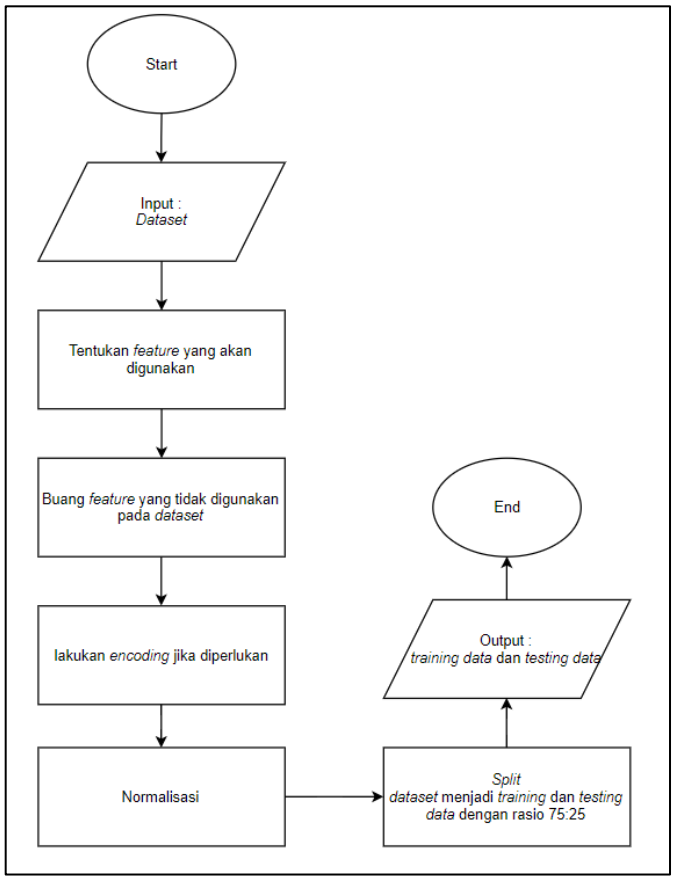

Gambar 3.1 Sebagian dataset yang akan digunakan pada penelitian ini




Gambar 3.2 Struktur dataset yang akan digunakan pada penelitian ini

*Dataset* yang akan digunakan merupakan data pertandingan dari *English Premier League* yang diperoleh dari situs statistik pertandingan sepak bola, *whoscored.com. Player rating, player position, dan team rating* akan menjadi inputan, sedangkan skor akhir pertandingan akan menjadi *output.* Proses *learning* akan menggunakan data dari 3 musim pertandingan, yaitu dari musim 14/15 sampaimusim 16/17. Seterlah proses *learning* dilakukuan, akan dilakukan validasi

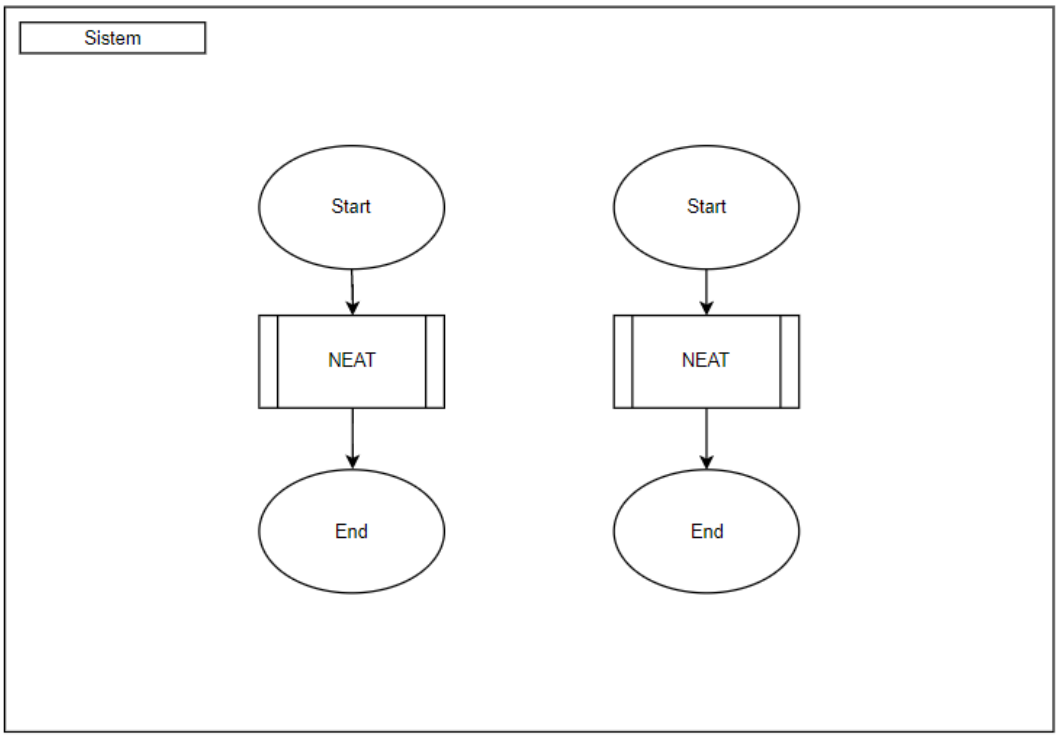
akurasi yang dihasilkan model menggunakan data dari pertandingan pada musim 17/18.




Gambar 3.3 *Flowchart* dari pemrosesan *dataset*

Sebelum data dapat digunakan dalam proses NEAT dan *backpropagation*, akan dilakukan *preprocessing* terlebih dahulu. Seperti yang dapat dilihat pada Gambar 3.1., pada awalnya data yang digunakan berformat *json*. Dari data tersebut kemudian akan ditentukan *feature* apa saja yang akan digunakan untuk melakukan prediksi pertandingan sepak bola. Setelah semua feature yang tidak diperlukan dibuang, selanjutnya akan dilakukan one hot encoding jika ada data yang berbentuk kategori. Kemudian, data akan di*split* menjadi *training* dan *testing* data dengan rasio 75:25. Setelah proses *preprocessing* ini, data siap digunakan.

## 3.2 Desain Sistem

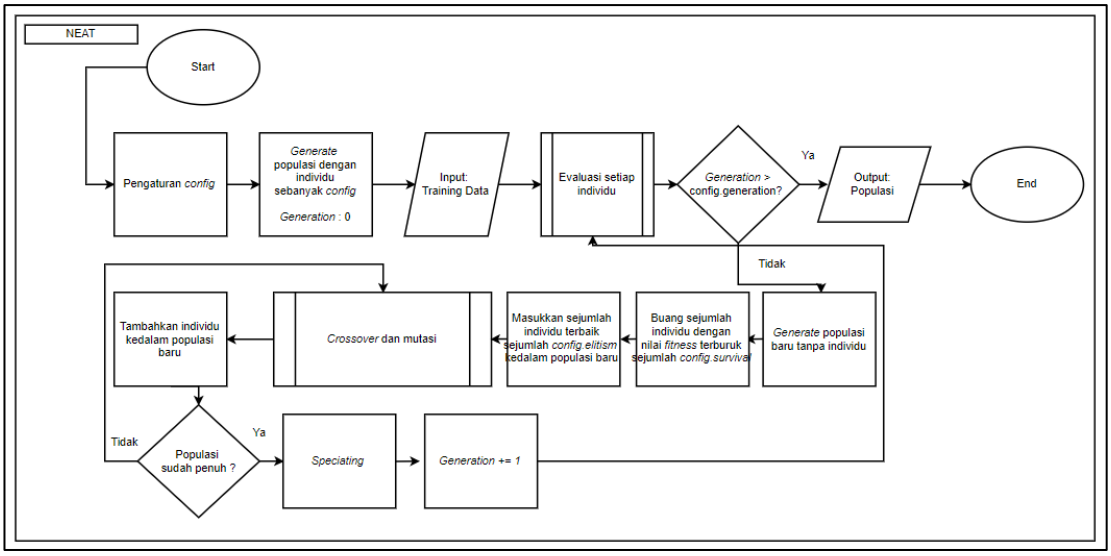



Gambar 3.4 *Flowchart* sistem secara umum

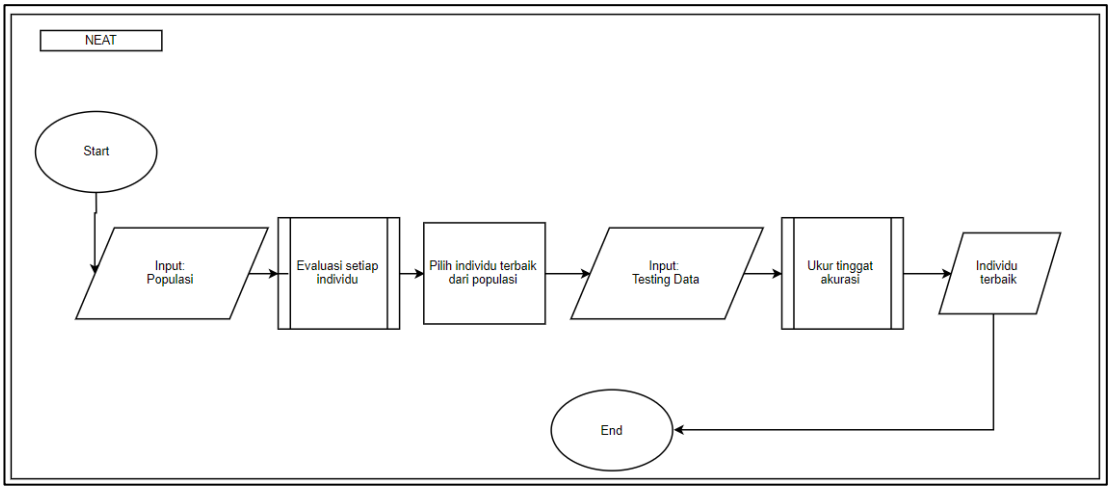
Dapat dilihat pada gambar 3.4., akan ada 2 proses utama pada penelitian ini, yaitu NEAT dan *Backpropagation*, yang masing-masing akan memiliki proses *training* dan *testing*.

Proses NEAT akan mendapat input berupa *training* dan *testing* data. *Training* data digunakan agar model dapat melakukan analisa data sehingga bisa menghasilkan prediksi. *Testing* data, yang berisi data yang tidak dikenali oleh model, akan digunakan untuk mengukur tingkat akurasi dari model. Model yang dihasilkan oleh NEAT kemudian akan dioptimasi oleh *backpropagation* yang bertujuan untuk meningkatkan hasil akurasi dengan menyesuaikan bobot atau *weight* yang ada pada model.

### 3.2.1 *Neuroevolution of Augmenting Topologies* (NEAT)




Gambar 3.5 *Flowchart* dari proses *training* pada NEAT



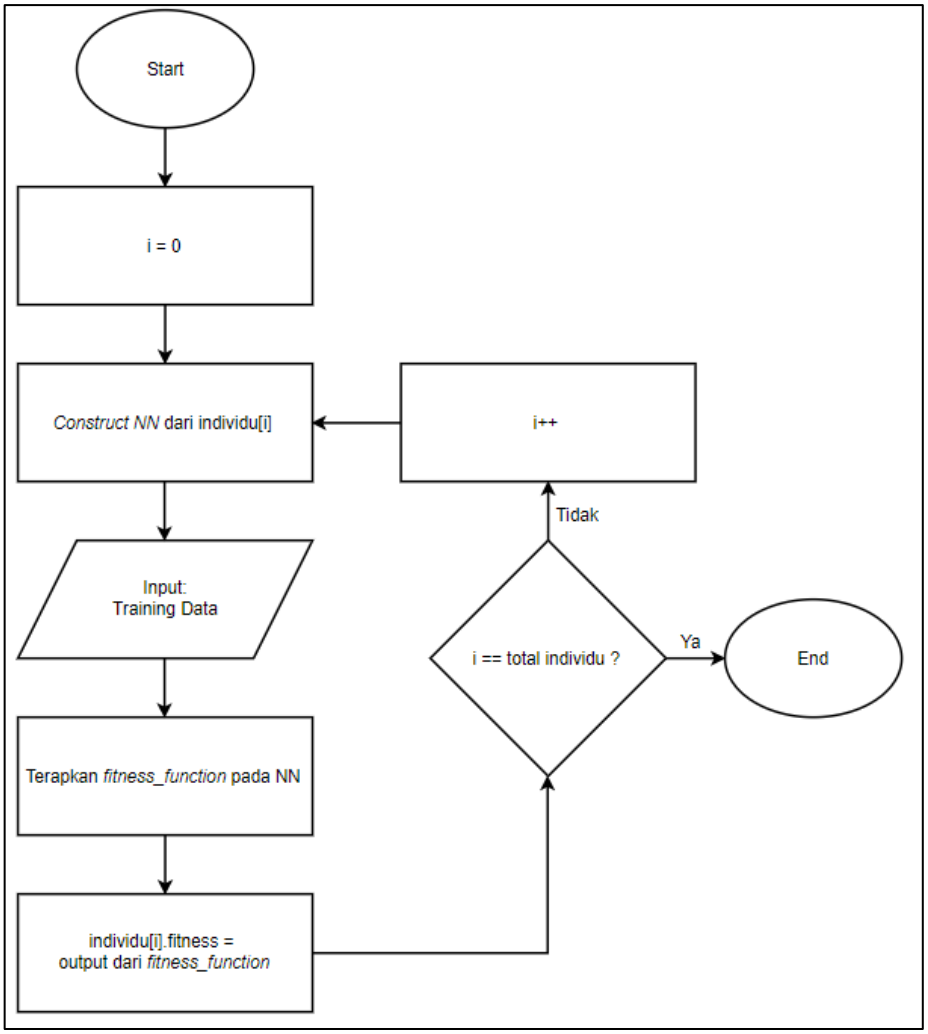

Gambar 3.6 *Flowchart* dari proses *testing* pada NEAT

Proses *training* pada NEAT dimulai dengan melakukan pengaturan *config*, yang merupakan kumpulan dari aturan-aturan yang berlaku selama proses *training* berlangsung, seperti jumlah individu dalam populasi, jumlah generasi yang diinginkan, tingkat kemungkinan terjadinya mutasi, dan lain lain. Pada tahap selanjutnya, NEAT akan membuat sebuah populasi baru dengan jumlah individu yang sudah ditentukan pada *config*. Kemudian, setiap individu akan dievaluasi menggunakan *fitness function* berdasarkan *training data* yang diinputkan untuk

menentukan nilai *fitness* dari masing-masing individu. Selanjutnya, akan dilakukan pengecekan *stopping condition*, yaitu sebuah kondisi untuk menentukan apakah proses NEAT akan berhenti atau tidak. *Stopping condition* disini adalah jumlah generasi. Jika jumlah generasi sudah melebihi jumlah generasi yang ditentukan pada *config*, Maka populasi akan di*export* dan proses *training* berhenti sampai disini. Jika *stopping condition* tidak terpenuhi, NEAT akan membuat sebuah populasi baru yang tidak memiliki individu, populasi baru ini nantinya akan diisi oleh individu-individu baru hasil dari *crossover* dan mutasi. Setelah itu, akan dilakukan seleksi, sejumlah individu pada populasi lama yang memiliki nilai *fitness* terburuk akan dibuang, sehingga ketika terjadinya proses *crossover*, individu-individu tersebut tidak dapat terpilih. Proses selanjutnya pada NEAT adalah *crossover* dan mutasi. Hasil dari proses *crossover* dan mutasi adalah sebuah individu baru yang akan ditambahkan ke dalam populasi baru. Proses *crossover* dan mutasi ini akan terjadi terus menurus hingga populasi baru memiliki individu sebanyak *pop size* pada *config.* Ketika populasi baru sudah penuh, maka akan dilakukan *speciating*, yaitu pembagian individu-individu yang ada pada populasi baru kedalam beberapa spesies. Setelah *speciating*, generasi akan bertambah dan populasi baru akan menggantikan populasi lama. Proses ini akan terus berulang hingga *stopping condition* terpenuhi.

Pada proses *testing*, NEAT akan menerima input berupa populasi yang dihasilkan oleh proses *training*. Setelah itu, akan dilakukan evaluasi yang sama seperti pada proses *training* dan akan dipilih satu individu dengan nilai *fitness* tertinggi. Individu terbaik tersebut kemudian akan diukur tingkat akurasinya dan diekstrak untuk dioptimasi menggunakan *backpropagation* pada proses selanjutnuya.

#### 3.2.1.1 Evaluasi pada NEAT



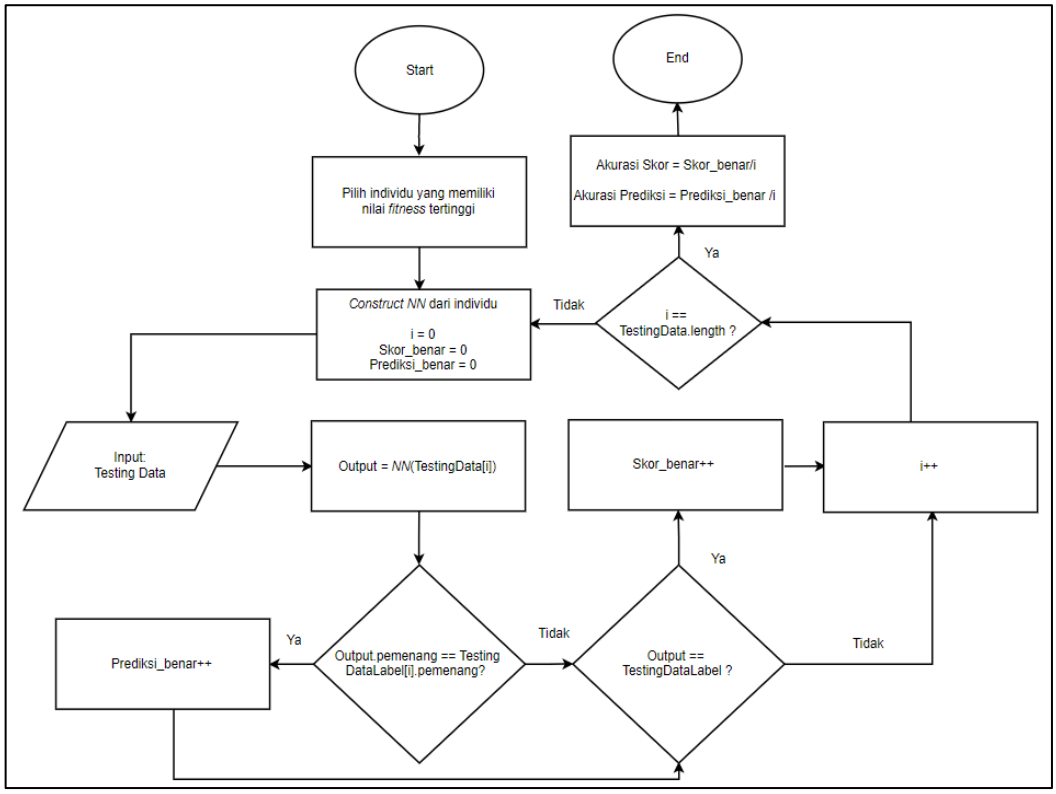

Gambar 3.7 *Flowchart* dari proses evaluasi pada NEAT

Evaluasi pada NEAT merupakan suatu proses yang sangat penting. Tujuan dari proses ini adalah untuk menentukan nilai *fitness*, yang merupakan indikator untuk mengukur seberapa baik sebuah individu dalam populasi.

Langkah pertama dari proses evaluasi adalah memilih individu pada sebuah populasi. Pada setiap individu, terdapat *genome* yang berisikan informasi tentang individu tersebut. Dari informasi yang berasal dari *genome*, dapat dibangun sebuah *Neural Network*, yang merupakan *phenotype* dari individu tersebut. *Neural Network* akan mendapat input berupa *training data*. Kemudian, akan diaplikasikan sebuah *fitness function* kepada *neural network* tersebut. *Output* dari *fitness function* inilah yang akan menjadi nilai *fitness* pada individu tersebut.

Proses Evaluasi ini akan terus dijalankan secara iteratif hingga semua individu pada populasi memiliki nilai *fitness*.

### 3.2.1.2 Pengukuran Tingkat Akurasi pada NEAT




Gambar 3.8 *Flowchart* dari proses pengukuran akurasi pada NEAT

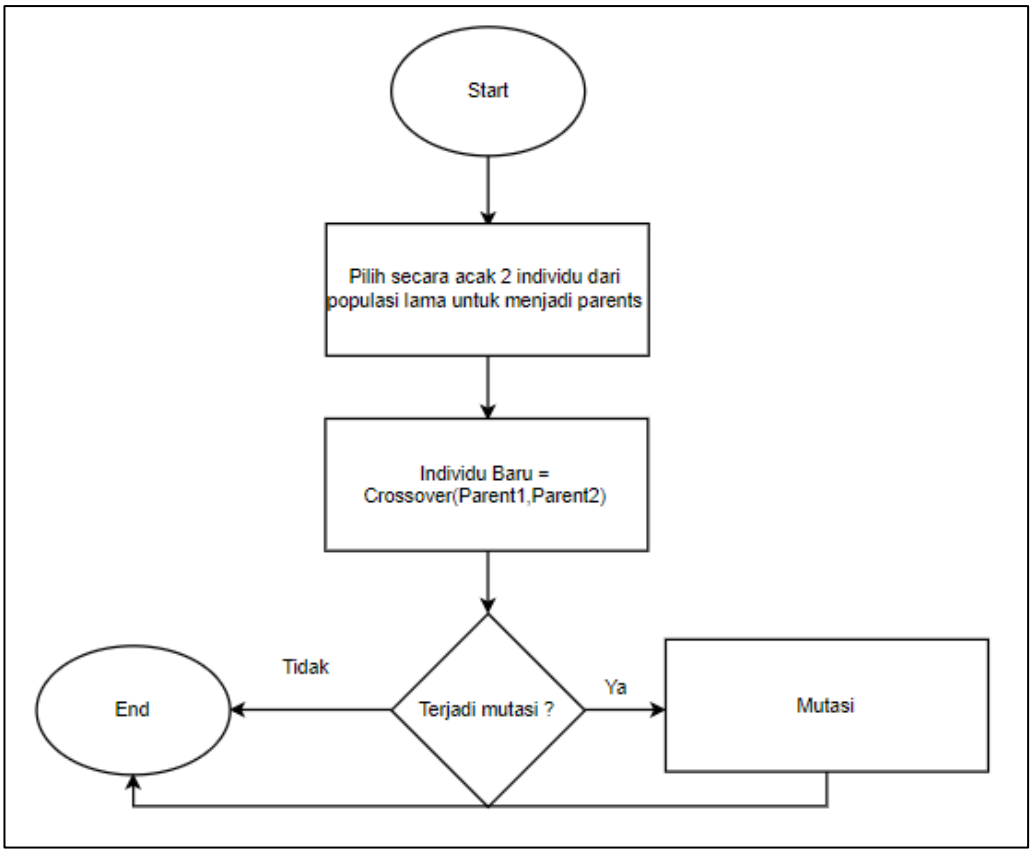
Ketika *stopping condition* pada NEAT terpenuhi, akan dilakukan pengukuran tingkat akurasi pada individu yang memiliki nilai *fitness* terbaik, dan individu terbaik tersebut akan diekstrak untuk dioptimasi menggunakan *backpropagation*. Pengukuran tingkat akurasi pada NEAT bertujuan sebagai pembanding seberapa besar pengaruh *backpropagation* pada individu yang dihasilkan oleh NEAT. Ada 2 jenis akurasi yang dihasilkan pada proses ini, yaitu akurasi skor dan akurasi prediksi. Pada akurasi skor, individu harus bisa melakukan prediksi skor secara tepat untuk dikatakan akurat, sedangkan pada akurasi prediksi, individu hanya perlu menebak tim mana yang keluar sebagai pemenang.

Proses pengukuran dimulai dengan memilih individu dengan nilai *fitness* terbaik dari populasi. Sama seperti proses evaluasi, dari individu tersebut akan

dibangun sebuah *neural network*. Tetapi, pada proses ini data yang digunakan adalah data *testing*, yang merupakan data yang tidak pernah diproses oleh model sebelumnya.

*Output* dari *neural network* yang berupa skor pertandingan kemudian akan dibandingkan dengan label yang tersedia pada *testing data*. Setelah semua *testing* data diproses, dapat dihitung tingkat akurasi dari individu tersebut.

### 3.1.2.3 *Crossover* dan Mutasi pada NEAT



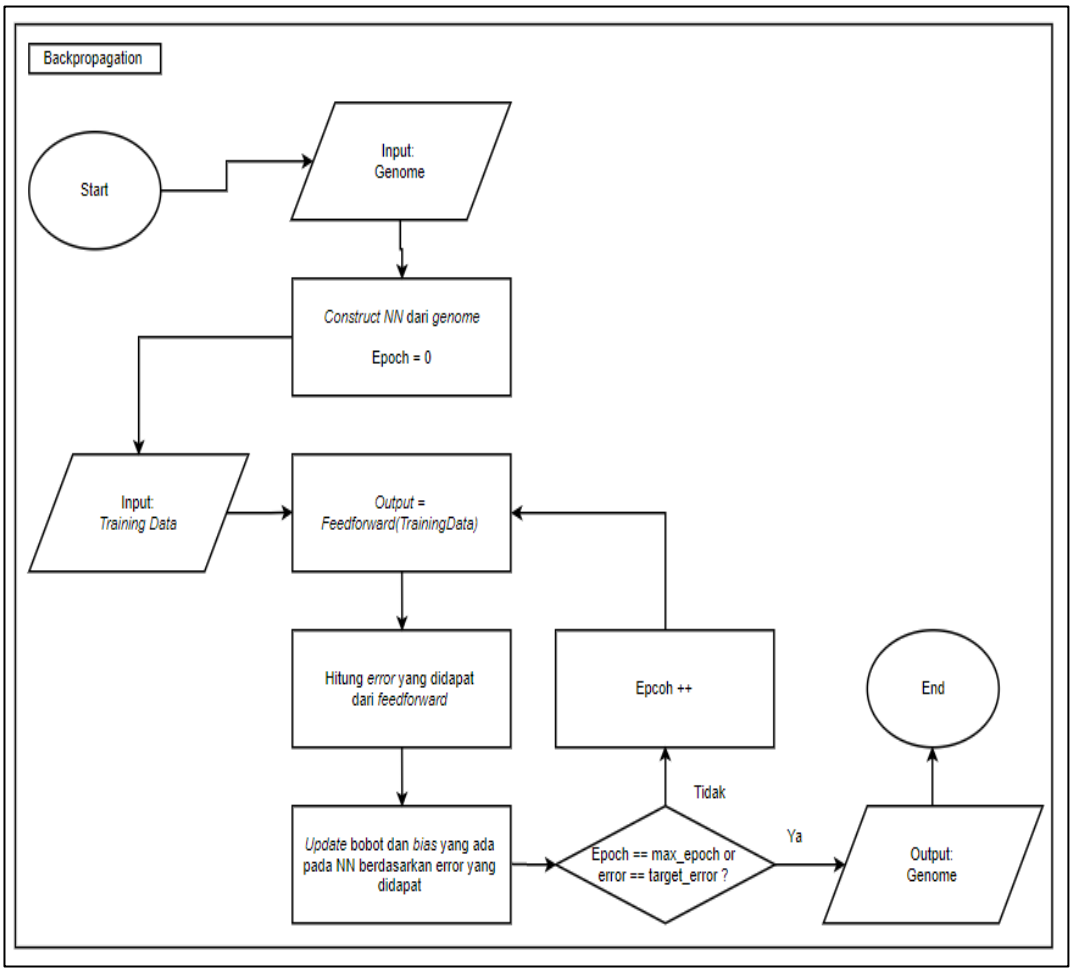

Gambar 3.9 *Flowchart* dari proses *crossover* dan mutasi pada NEAT

*Crossover* dan mutasi adalah proses terpenting pada NEAT. Proses ini memungkinkan terciptanya suatu inovasi yang memiliki potensi untuk mengungguli individu lama dalam nilai *fitness*.

Sebelum terjadinya *crossover*, 2 individu dalam populasi lama akan dipilih secara acak untuk menjadi *parents*. *Parents* tersebut kemudian akan melakukan *crossover*, yang akan menghasilkan 1 individu baru. Individu baru ini kemudian

akan memiliki kemungkinan untuk melakukan mutasi. Kemungkinan terjadinya mutasi ini ditentukan oleh *mutation rate* yang ada pada *config*. Setelah proses *crossover* dan mutasi selesai, individu baru tersebut akan ditambahkan kepopulasi baru. Proses *crossover* dan mutasi akan terjadi terus menerus hingga populasi baru terpenuhi.

### 3.2.2 *Backpropagation*

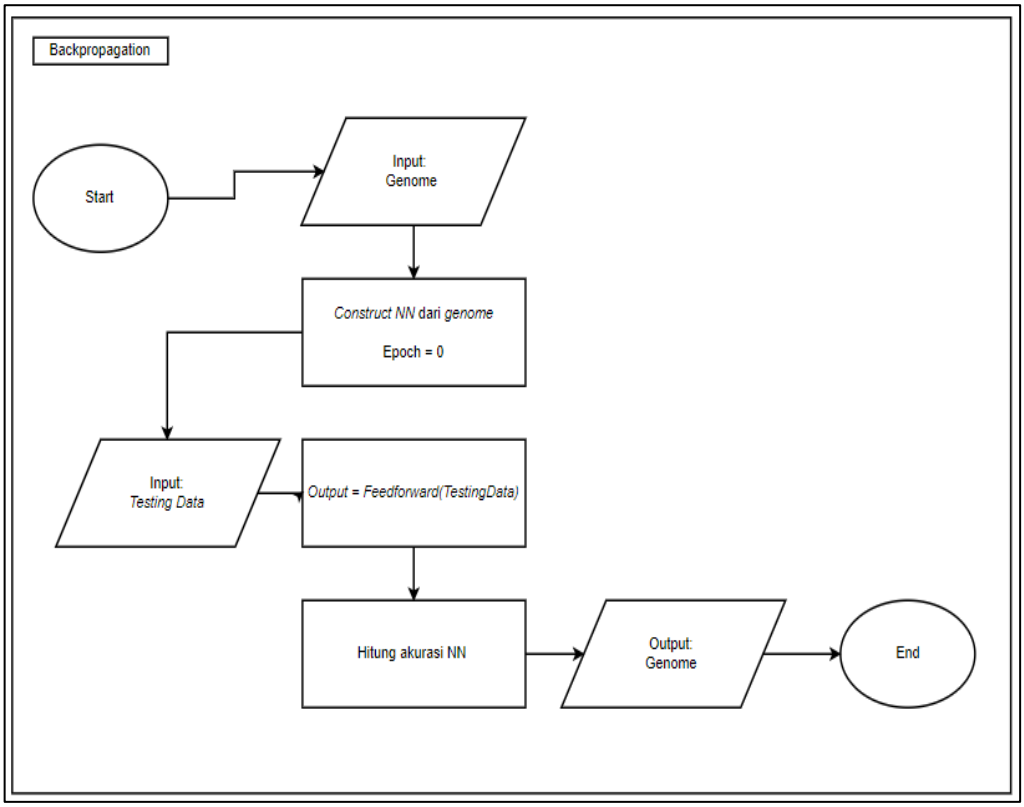



Gambar 3.10 *Flowchart* dari proses *training backpropagation*

Ketika proses NEAT selesai, yang menghasilkan *output* berupa sebuah *genome* dengan nilai *fitness* terbaik dari populasi, proses selanjutnya adalah melakukan optimasi pada *genome* tersebut dengan *backpropagation*.

Proses *training* pada *backpropagation* dimulai dengan membangun sebuah *neural network* dari *genome*. *Neural network* kemudian akan mendapat input

berupa *training* data. Berdasarkan *training* data yang diterima, *neural network* akan mengeluarkan output berupa prediksi skor akhir dari sebuah pertandingan. Hasil prediksi yang dihasilkan oleh *neural network* kemudian akan dibandingkan dengan label yang ada pata *training* data, yang merupakan skor yang benar dari pertandingan tersebut, untuk mendapatkan nilai *error*. Nilai *error* ini akan digunakan untuk menyesuaikan *weight* atau bobot dan *bias* yang ada pada *neural network.*

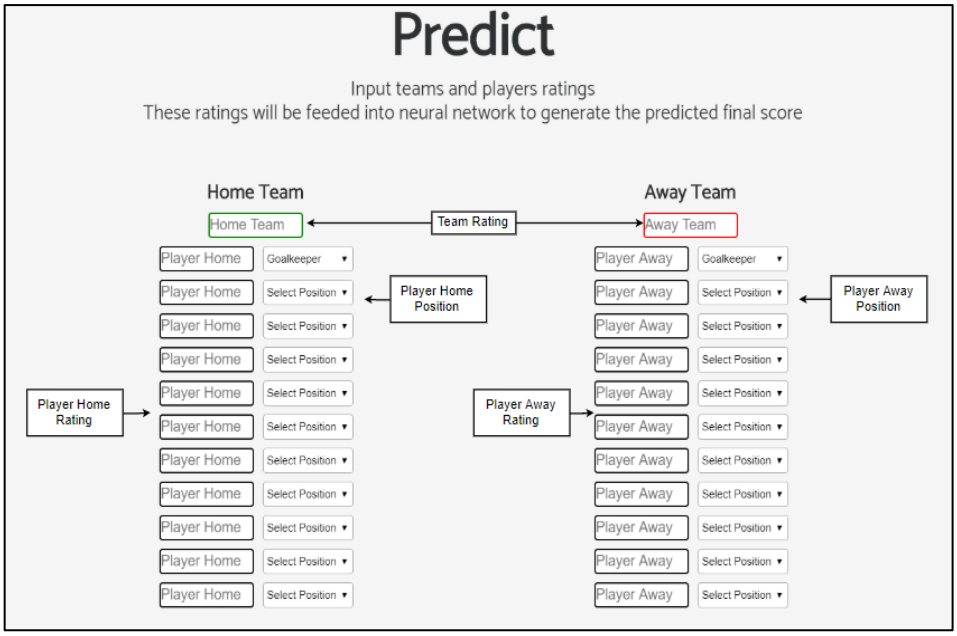



Gambar 3.11 *Flowchart* dari proses *testing backpropagation*

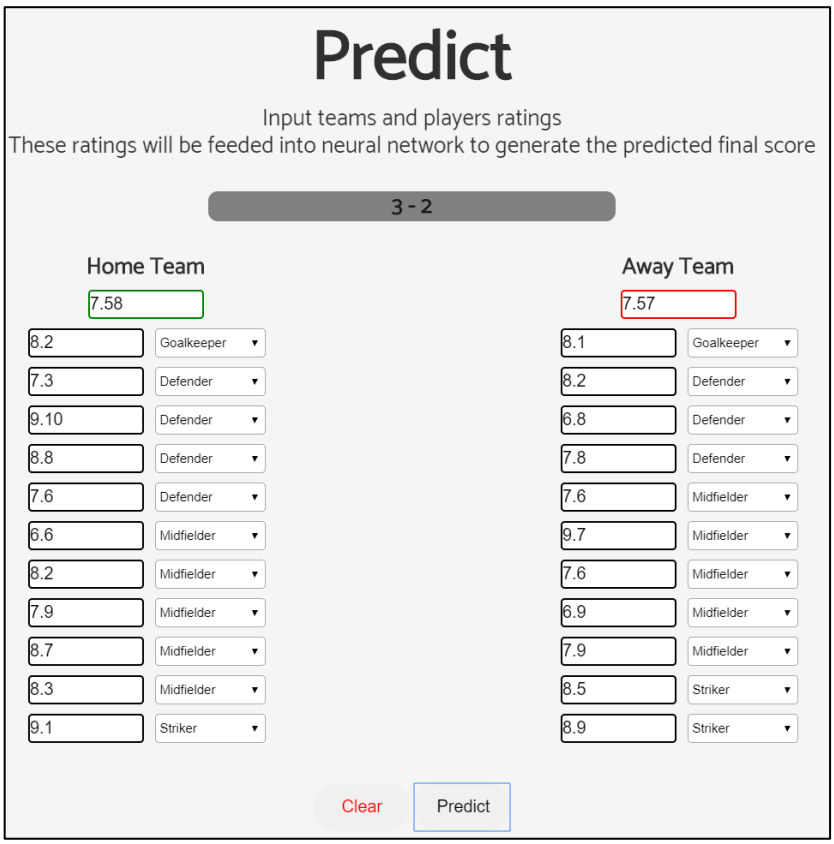
Proses *backpropagation* akan dijalankan terus menurus hingga target *epoch* tercapai, yang ditentukan sebelum proses dimulai, atau jika nilai *error* sudah mencapai target *error*, yang juga ditentukan sebelum proses dimulai.

Setelah proses *backpropagation* selesai, akan dilakukan proses testing dan pengukuran akurasi dari *neural network* yang telah dioptimasi untuk menganalisa seberapa besar pengaruh yang dihasikan oleh *backpropagation* terhadap tingkat akurasi dari sebuah *genome* yang dihasilkan oleh NEAT.

## 3.3 Desain Aplikasi




Gambar 3.12 Desain halaman utama untuk melakukan prediksi




Gambar 3.13 Halaman untuk menampilkan hasil prediksi

*User* harus mengisi *rating* dari kedua team pada kolom *Home Team* dan *Away Team*. Setelah itu, *user* juga harus mengisi *rating* dan posisi pemain dari masing-masing *team*. Ada 3 posisi yang dapat dipilih, yaitu *defender*, *midfielder*, dan *striker*. Setelah semua kolom terisi, *user* dapat menekan tombol *predict* untuk melakukan prediksi.

Tombol *predict* akan mengarahkan *user* kehalaman hasil, yaitu halaman yang berfungsi untuk menampilkan hasil prediksi dari data yang telah dimasukkan oleh *user*. Jika *user* menekan tombol *predict again*, user akan diarahkan kehalaman utama untuk melakukan prediksi lagi.

# 4. IMPLEMENTASI SISTEM

Pada bab ini, akan dibahas implementasi sistem sesuai analisa dan desain sistem pada bab sebelumnya.

Tabel 4.1 Daftar Segmen Program dan Flowchart

| SEGMEN PROGRAM | GAMBAR (FLOWCHART) |
|---|---|
| **4.1** | 3.3 |
| **4.2, 4.3** | 3.5, 3.6 |
| **4.4, 4.5** | 3.7 |
| **4.6** | 3.8 |
| **4.8, 4.9, 4.10** | 3.10 |
| **4.11** | 3.11 |

## 4.1 Instalasi *Open Source Library*

Dalam implementasi sistem ini, digunakan dua *open source library*, yaitu *NEAT-Python* dan *Neataptic.js.*

### 4.1.1 Instalasi NEAT-Python

*NEAT-Python* merupakan sebuah *open source library* dari bahasa pemrograman *python*. *Library* ini nantinya akan digunakan untuk menjalankan proses NEAT.

Untuk meng-*install NEAT-Python*, dibutuhkan *pip*, yang merupakan *package installer* untuk bahasa pemrograman *python*. Jika *pip* sudah ter-*install*, cukup ketikkan **`pip install neat-python`** pada *terminal* jika menggunakan *linux* atau *macOS*, atau *command prompt* jika menggunakan *windows.*

### 4.1.2 Instalasi *Neataptic.js*

Untuk melakukan *backpropagation*, dibutuhkan *open source library* lainnya. Pada penelitian ini, library yang digunakan untuk melakukan

*backpropagation* adalah *Neataptic.js*, yang merupakan *library* dari bahasa pemrograman *javascript* yang pada penelitian ini akan berjalan diatas *nodejs*.

Sama seperti *NEAT-Python*, *Neataptic.js* juga membutuhkan *npm*, yaitu sebuah *package installer* pada bahasa pemrograman *javascript*, untuk melakukan instalasi. Jika *npm* sudah terinstall, cukup ketikkan **`npm install neataptic`** pada *terminal* jika menggunakan *linux* atau *macOS*, atau *command prompt* jika menggunakan windows

### *4.2 Data Pre-processing*

Sebelum data dapat digunakan, harus dilakukan *pre-processing* terlebih dahulu untuk mengekstrak *features* yang akan digunakan. Seperti yang sudah dijelaskan pada Bab 3, pada awalnya format data adalah *json*. Proses ini menggunakan bahasa pemrograman *php*. *Output* dari proses *pre-processing* ini berbentuk *.txt* yang nanti akan digunakan pada proses NEAT.

Segmen Program 4.1 Pre-processing pada data dengan feature berupa player ratings dan team ratings

```
$data = json_decode(file_get_contents("dataset/datafile/season14-
15/season_stats.json"),true);
$fInput = array();
$fOutput = array();
$readCount = 0;

foreach($data as $key => $values){
    $input = array();
    $output = array();
    $readCount++;
    foreach($values as $key2 => $values2){
        foreach($values2 as $key3 => $values3){
            if($key3 == 'team_details'){
                foreach($values3 as $key4 => $values4){
                    if($key4 == 'team_rating'){
                        array_push($input,$values4/10);
                    }
                }
            }
            if($key3 == 'aggregate_stats'){
                $noGoal = false;
                foreach($values3 as $key4 => $values4){
                    if($key4 == 'goals'){
```

```
                        array_push($output,$values4);
                        $noGoal = true;
                    }

                }
                if($noGoal == false){
                    array_push($output,0);
                }
            }
            if($key3 == 'Player_stats'){
                foreach($values3 as $key4 => $values4){
                    foreach($values4 as $key5 => $values5){
                        if($key5 == 'player_details'){
                            foreach($values5 as $key6 => $values6){
                                if($key6 == 'player_rating' ){
                                    if($values6 == 'Sub'){
                                        break;
                                    }else if($key6 == 'player_rating'){
                                        array_push($input,$values6/10);
                                    }
                                }
                            }
                        }
                    }
                }
            }

        }
    }
    array_push($fInput,$input);
    array_push($fOutput,$output);
}

$input_str = '[';
$inputCount = 0;
$outputCount = 0;

foreach($fInput as $key => $values){
    $inputCount++;
    $input_str =  $input_str.'(';
    foreach($values as $key2 => $values2){
        $input_str = $input_str.$values2.', ';
    }
    $input_str =  $input_str.'),';

}

$input_str =  $input_str.']';
```

```
$output_str = '[';

foreach($fOutput as $key => $values){
    $outputCount++;
    $output_str =  $output_str.'(';
    foreach($values as $key2 => $values2){
        $output_str = $output_str.$values2.', ';
    }
    $output_str =  $output_str.'),';

}

$output_str =  $output_str.']';
$myfile = fopen("inputTestWithTeam.txt", "w") or die("Unable to open file!");
fwrite($myfile, $input_str);
$myfile = fopen("OutputTest.txt", "w") or die("Unable to open file!");
fwrite($myfile, $output_str);
```

Setelah proses *pre-processing* selesai, barulah data siap digunakan untuk proses selanjutnya, yaitu NEAT.

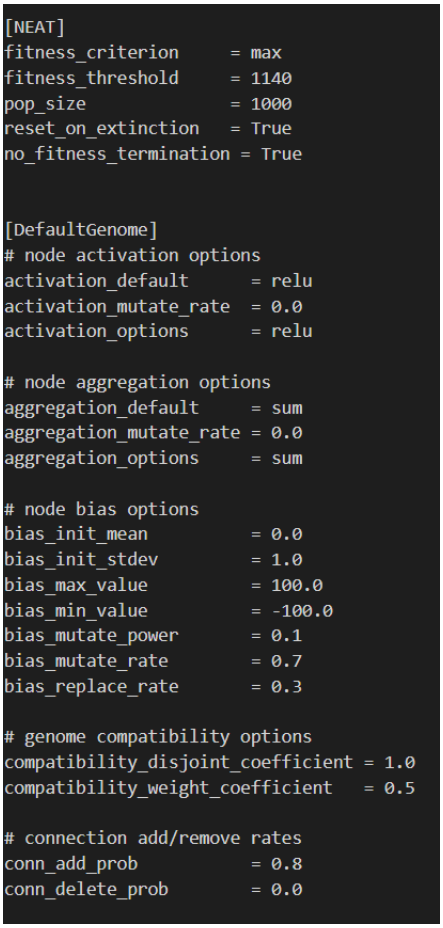
## 4.3 Implementasi NEAT

Proses NEAT dimulai dengan melakukan pengaturan *config*, yang merupakan aturan-aturan yang berlaku selama proses NEAT berlangsung, seperti kemungkinan terjadinya mutasi, jumlah populasi, dan lain-lain. Contoh *config* pada NEAT dapat dilihat pada Gambar 4.1.

Setelah melakukan pengaturan *config*, akan dijalankan fungsi `load_data()` untuk menginputkan data yang telah diproses melalui proses *pre-processing*.

Akan ada 4 data yang akan digunakan, yaitu data untuk *input* dan *output* untuk proses *training* dan data *input* dan *output* untuk proses *testing*.

Jika data sudah diterima dan *config* sudah ditetapkan, proses neat akan berjalan dengan menjalankan fungsi `start()`, yang menerima *config* dan mengaplikasikannya kepada sistem.



```
[NEAT]
fitness_criterion     = max
fitness_threshold     = 1140
pop_size              = 1000
reset_on_extinction   = True
no_fitness_termination = True


[DefaultGenome]
# node activation options
activation_default      = relu
activation_mutate_rate  = 0.0
activation_options      = relu

# node aggregation options
aggregation_default     = sum
aggregation_mutate_rate = 0.0
aggregation_options     = sum

# node bias options
bias_init_mean          = 0.0
bias_init_stdev         = 1.0
bias_max_value          = 100.0
bias_min_value          = -100.0
bias_mutate_power       = 0.1
bias_mutate_rate        = 0.7
bias_replace_rate       = 0.3

# genome compatibility options
compatibility_disjoint_coefficient = 1.0
compatibility_weight_coefficient   = 0.5

# connection add/remove rates
conn_add_prob           = 0.8
conn_delete_prob        = 0.0
```

Gambar 4.1 Contoh *file config* pada NEAT

Segmen Program 4.2 Fungsi `load_data()`

```
def load_data():
    train_inputs = f=open("inputTestWithTeam.txt","r")
    if(f.mode == 'r'):
        xor_inputs = eval(f.read())
    train_outputs = f=open("outputTest.txt","r")
    if(f.mode == 'r'):
        xor_outputs = eval(f.read())
    valid_inputs = f=open("validInputWithTeam.txt","r")
    if(f.mode == 'r'):
        valid_inputs = eval(f.read())
    valid_outputs = f=open("validOutputTest.txt","r")
    if(f.mode == 'r'):
        valid_outputs = eval(f.read())
```

Segmen Program 4.3 Fungsi `start()` untuk menjalankan proses NEAT

```
def start(config_file):
    config = neat.Config(neat.DefaultGenome, neat.DefaultReproduction,
                         neat.DefaultSpeciesSet, neat.DefaultStagnation,
                         config_file)
    p = neat.Population(config)
    p.add_reporter(neat.StdOutReporter(True))
    stats = neat.StatisticsReporter()
    p.add_reporter(stats)
    p.add_reporter(neat.Checkpointer(50))
    pe = neat.ParallelEvaluator(4,fitness_function2)
    winner = p.run(pe.evaluate,5000)
    print('\nBest genome:\n{!s}'.format(winner))
    print('\nOutput:')
    calculate_winner(winner,config)
    node_names = {-1:'A', -2: 'B', 0:'A XOR B'}
    visualize.extract_net(config,winner)
    visualize.draw_net(config, winner, view=False, node_names=node_names)
    visualize.plot_stats(stats, ylog=False, view=False)
    visualize.plot_species(stats, view=False)
```

Fungsi `neat.ParallelEvaluator()` menerima sebuah *fitness function* yang akan digunakan untuk mengevaluasi populasi pada NEAT. Akan ada 2 *fitness function* yang akan digunakan pada penelitian ini.

Segmen Program 4.4 *Fitness function* pertama

```
def eval_genome1(genome, config):
    fitness = 1140
    net = neat.nn.FeedForwardNetwork.create(genome, config)
    for xi, xo in zip(xor_inputs, xor_outputs):
        output = net.activate(xi)
        output[0] = round((output[0]*10))
        output[1] = round((output[1]*10))
        if(xo[0]*10 == output[0] and xo[1]*10 == output[1] ):
            genome.fitness += 1
        else :
            if(xo[0]*10 > xo[1]*10 and output[0] > output[1] ):
                genome.fitness += 0.5
            if(xo[0]*10 < xo[1]*10 and output[0] < output[1] ):
                genome.fitness += 0.5
            if(xo[0]*10 == xo[1]*10 and output[0] == output[1] ):
                genome.fitness += 0.5
```

```
    return fitness
```

Segmen Program 4.5 *Fitness function* kedua

```
def eval_genome2(genome, config):
    fitness = 1140
    net = neat.nn.FeedForwardNetwork.create(genome, config)
    for xi, xo in zip(xor_inputs, xor_outputs):
        output = net.activate(xi)
        fitness -= (output[0] - xo[0]) ** 2
        fitness -= (output[1] - xo[1]) ** 2
    return fitness
```

Pada *fitness function* pertama, masing masing *genome* akan diberikan nilai fitness sebesar 1 jika skor yang dihasilkan benar, sedangkan jika skor salah tetapi pemenang yang diprediksi benar, nilai fitness yang diberikan adalah 0.5. Sedangkan pada *fitness function* kedua, nilai *fitness* dihitung berdasarkan *sum of squared errors*.

Fungsi `neat.parallelEvaluator()` kemudian akan di*passing*kan kepada fungsi `run()` untuk menjalankan proses NEAT. Fungsi `run()` juga menerima parameter berupa jumlah iterasi yang ingin dijalankan.

Setelah fungsi `run()` selesai dijalankan, yang menandakan proses NEAT telah selesai, fungsi ini akan meng*outputkan* genome terbaik dengan nilai *fitness* terbaik. Dari genome terbaik ini kemudian akan di*construct* sebuah *neural network* yang akan di*passing*kan ke fungsi `calculate_winner()`

Fungsi `calculate_winner()` bertujuan untuk melakukan proses *testing* terhadap *genome* terbaik yang dihasilkan oleh NEAT.

Segmen Program 4.6 Fungsi `calculate_winner()`

```
def calculate_winner(winner,config):
    winner_net = neat.nn.FeedForwardNetwork.create(winner,config)
    correct_predict = 0
    correct_winner = 0
    for xi, xo in zip(valid_inputs, valid_outputs):
        output = winner_net.activate(xi)
        output[0] = round((output[0]))
        output[1] = round((output[1]))
        if(xo[0] == output[0] and xo[1] == output[1] ):
```

```
            correct_predict += 1
        if(xo[0] > xo[1] and output[0] > output[1] ):
            correct_winner += 1
        if(xo[0] < xo[1] and output[0] < output[1] ):
            correct_winner += 1
        if(xo[0] == xo[1] and output[0] == output[1] ):
            correct_winner += 1
        print("input {!r}, expected output {!r}, got {!r}".format(xi, xo, output
))
    print("Prediction accuracy = {!r} ".format(correct_predict))
    print("Winner accuracy = {!r} ".format(correct_winner))
```

Setelah semua proses pada NEAT selesai, genome terbaik akan diekstrak untuk dioptimasi menggunakan backpropagation. Proses ekstraksi ini terjadi pada fungsi `visualise.extract_net()`

Segmen Program 4.7 Fungsi `extract_net()`

```
def extract_net(genome,config):
    node_dict = dict()
    node_obj = []

    for k in config.genome_config.input_keys:
        node_dict = collections.defaultdict(dict)
        node_dict['key'] = k
        node_dict['bias'] = 0
        node_dict['activation'] = ''
        node_dict['type'] = 'input'
        node_obj.append(node_dict)
    for i in genome.nodes:
        node_dict = collections.defaultdict(dict)
        if genome.nodes[i].key > 1 :
            node_dict['key'] = genome.nodes[i].key
            node_dict['bias'] = genome.nodes[i].bias
            node_dict['activation'] = genome.nodes[i].activation
            node_dict['type'] = 'hidden'
            node_obj.append(node_dict)
    for i in genome.nodes:
        node_dict = collections.defaultdict(dict)
        if genome.nodes[i].key == 1 or genome.nodes[i].key == 0:
            node_dict['key'] = genome.nodes[i].key
            node_dict['bias'] = genome.nodes[i].bias
            node_dict['activation'] = genome.nodes[i].activation
            node_dict['type'] = 'output'
            node_obj.append(node_dict)

    with open('node_data.json', 'w') as outfile:
```

```
        json.dump(node_obj, outfile)
    conn_dict = dict()
    conn_obj = []
    for i in genome.connections:
        if genome.connections[i].enabled == True:
            conn_dict = collections.defaultdict(dict)
            conn_dict['from'] = genome.connections[i].key[0]
            conn_dict['to'] = genome.connections[i].key[1]
            conn_dict['weight'] = genome.connections[i].weight
            conn_obj.append(conn_dict)
    with open('conn_data.json', 'w') as outfile:
        json.dump(conn_obj, outfile)
```

Fungsi `extract_net()` akan menghasilkan 2 *output* berformat *json*, yaitu *node_data.json* yang berisi informasi dari *node* dan *conn_data.json* yang berisi informasi koneksi yang menghubungkan satu *node* dengan *node* lainnya.

## 4.4 Implementasi *Backpropagation*

Setelah semua proses NEAT selesai, proses selanjutnya adalah *backpropagation*. Langkah pertama pada prose *backpropagation* ialah menyiapkan data untuk *training* dan *testing*, serta melakukan konsturksi *network* berdasarkan output dari `extract_net()` pada NEAT. Semua proses *backpropagation* menggunakan bahasa pemrograman *javascript*

Segmen Program 4.8 fungsi `prepare_data()` untuk menyiapkan data berdasarkan tipe *feature* yang digunakan

```
prepareData(type){
  if(type == 1){
      this.trainingData = fs.readFileSync('./dataset/inputTestPlayerOnly.json');
      this.validData = fs.readFileSync('./dataset/validTestPlayerOnly.json');
   }else if(type == 2){
      this.trainingData = fs.readFileSync('./dataset/inputTestWithTeam.json');
      this.validData = fs.readFileSync('./dataset/validTestWithTeam.json');
   }else if(type == 3){
      this.trainingData = fs.readFileSync('./dataset/inputTestWithTeamAndPos.jso
n');
      this.validData = fs.readFileSync('./dataset/validTestWithTeamAndPos.json')
;
        }else{
            console.log("unknown type");
        }
```

```
    }
```

Segmen Program 4.9 fungsi `build()` untuk mengkonstuksi *network*

```
build(node_json , conn_json){
        let contents = fs.readFileSync(node_json);
        this.node_data = JSON.parse(contents);

        this.node_count = 0;
        for(let i in this.node_data){
            this.node_count++;
        }

        this.nodes = Array(this.node_count);
        for(let i = 0 ; i < this.nodes.length ; i++){
            this.nodes[i] = new Node(this.node_data[i].type , this.node_data[i].
key , this.node_data[i].bias)
        }
        contents = fs.readFileSync(conn_json);
        this.conn_data = JSON.parse(contents);

        for(let i in this.conn_data){
            for(let j = 0 ; j < this.nodes.length ; j++){
                if(this.conn_data[i].from == this.nodes[j].key){
                    for(let h = 0 ; h < this.nodes.length ; h++){
                        if(this.conn_data[i].to == this.nodes[h].key){
                            if(j != h){
                                this.nodes[j].connect(this.nodes[h] , this.conn_
data[i].weight);
                            }

                        }
                    }
                }
            }
        }
        this.network = architect.Construct(this.nodes);
    }
```

Setelah *network* berhasil dikonsturksi dan data telah disiapkan, proses *training* dan proses *testing* dapat dijalankan.

Segmen Program 4.10 fungsi `backprop()` untuk menjalankan proses *training*

```
backprop(){
        let backprop = true;
        let input = JSON.parse(this.trainingData)
        let opt = {
            log: 100,
            error: 0,
            iterations: 100000,
            rate: 0.001,
        }
        this.network.train(input,opt,propagate);
}
```

Pada segmen ini, `backprop` diatur ke `true` sebagai tanda untuk melakukan proses *training*. Object `opt` berisi konfirgurasi pada proses *training*. Selanjutnya, fungsi `network.train()` yang berasal dari *library neataptic.js* akan dijalankan untuk memulai proses *training*.

Setelah proses *training* selesai dijalankan, akan dilakukan proses *testing* menggunakan fungsi `calculate_accuracy()` yang bertujuan untuk mengukur tingkat akurasi dari *network*. Fungsi ini menggunakan data *testing* yang tidak dikenali oleh *network*.

Segmen Program 4.11 fungsi `calculateAccuracy()` untuk mengukur akurasi dari *network*

```
calculateAccuracy(){
        let set = JSON.parse(this.testingData);
        for(let i = 0 ; i < set.length ; i++){
            let input = set[i].input;
            let target = set[i].output;
            let output = this.network.activate(input, true);
            let temOutput = output;

            temOutput[0] = Math.round(temOutput[0]);
            temOutput[1] = Math.round(temOutput[1]);
            console.log('output = '+temOutput+' target = '+target);
            if(Math.round(output[0]) == target[0] && Math.round(output[1]) == ta
rget[1]){
                correct++;
            }
            if(Math.round(output[0]) > Math.round(output[1]) && target[0] > targ
et[1]){
                win++;
```

```
            }
            if(Math.round(output[0]) < Math.round(output[1]) && target[0] < targ
et[1]){
                win++;
            }
            if(Math.round(output[0]) == Math.round(output[1]) && target[0] == ta
rget[1]){
                win++;
            }
        }

        console.log("Prediction correct = "+correct)
        console.log("winner correct = "+win)
    }
```

# 5. PENGUJIAN SISTEM

Pada bab ini, akan dibahas pengujian terhadap sistem NEAT dan *backpropagation* yang telah dibuat.

## 5.1 Sistem Pengujian

Waktu pengujian pada bab ini sangat ditentukan oleh spesifikasi yang digunakan. Semua pengujian yang dilakukan pada bab ini dilakukan menggunakan *notebook asus A412DA* dengan spesifikasi sebagai berikut:

Tabel 5.1 Spesifikasi sitem pengujian

| | |
|---|---|
| *Processor* | *AMD Ryzen 5 3500u* |
| *RAM* | *8GB DDR4* |
| *Operating System* | *Windows 10* |

## 5.2 Pengujian NEAT

Pengujian NEAT pada penelitian ini akan dibagi kedalam 3 tahap berdasarkan *feature* yang digunakan. Pembagian ini bertujuan untuk melihat pengaruh dari *feature* yang digunakan terhadap akurasi dari NEAT. Pada setiap tahap, akan dicoba berbagai konfigurasi untuk mendapatkan hasil yang optimal. Pengujian yang dilakukan meliputi proses *training*, yang bertujuan untuk mencari individu terbaik, dan proses *testing*, untuk mengukur akurasi dari individu yang dihasilkan oleh proses *training*.

Data yang digunakan berasal dari situs *whosocred*.com, yang merupakan situs penyedia data pertandingan sepak bola. Pada proses *training*, data berasal dari pertandingan liga inggris pada musim 2014/2015, 2015/2016, dan 2016/2017. Sedangkan untuk proses *testing*, data yang digunakan berasal dari pertandingan liga inggris pada musim 2017/2018. Pada setiap musim, terdapat 380 pertandingan.

### 5.2.1 Tahap 1

Pada tahap ini, *feature* yang digunakan adalah *rating* pemain dari kedua *team*, sehingga ada 22 *feature* yang terdiri dari 11 pemain dari masing-masing *team*.

### 5.2.1.1 Tahap 1 Pengujian 1

```
[NEAT]
fitness_criterion     = max
fitness_threshold     = 1140
pop_size              = 1000
reset_on_extinction   = True
no_fitness_termination = True

[DefaultGenome]
# node activation options
activation_default      = relu
activation_mutate_rate  = 0.0
activation_options      = relu

# node aggregation options
aggregation_default     = sum
aggregation_mutate_rate = 0.0
aggregation_options     = sum

# node bias options
bias_init_mean          = 0.0
bias_init_stdev         = 1.0
bias_max_value          = 100.0
bias_min_value          = -100.0
bias_mutate_power       = 0.1
bias_mutate_rate        = 0.7
bias_replace_rate       = 0.3

# genome compatibility options
compatibility_disjoint_coefficient = 1.0
compatibility_weight_coefficient   = 0.5

# connection add/remove rates
conn_add_prob           = 0.8
conn_delete_prob        = 0.5

# connection enable options
enabled_default         = True
enabled_mutate_rate     = 0.4

feed_forward            = True
initial_connection      = partial_direct 0.7

# node add/remove rates
node_add_prob           = 0.8
node_delete_prob        = 0.5

# network parameters
num_hidden              = 1
num_inputs              = 22
num_outputs             = 2

# node response options
response_init_mean      = 1.0
response_init_stdev     = 0.0
response_max_value      = 3000.0
response_min_value      = -3000.0
response_mutate_power   = 0.0
response_mutate_rate    = 0.0
response_replace_rate   = 0.0

single_structural_mutation = False
structural_mutation_surer = False

# connection weight options
weight_init_mean        = 0.0
weight_init_stdev       = 1.0
weight_max_value        = 1000
weight_min_value        = -1000
weight_mutate_power     = 0.1
weight_mutate_rate      = 0.8
weight_replace_rate     = 0.4

[DefaultSpeciesSet]
compatibility_threshold = 3.0

[DefaultStagnation]
species_fitness_func = max
max_stagnation       = 1000
species_elitism      = 5

[DefaultReproduction]
elitism            = 40
survival_threshold = 0.5
```

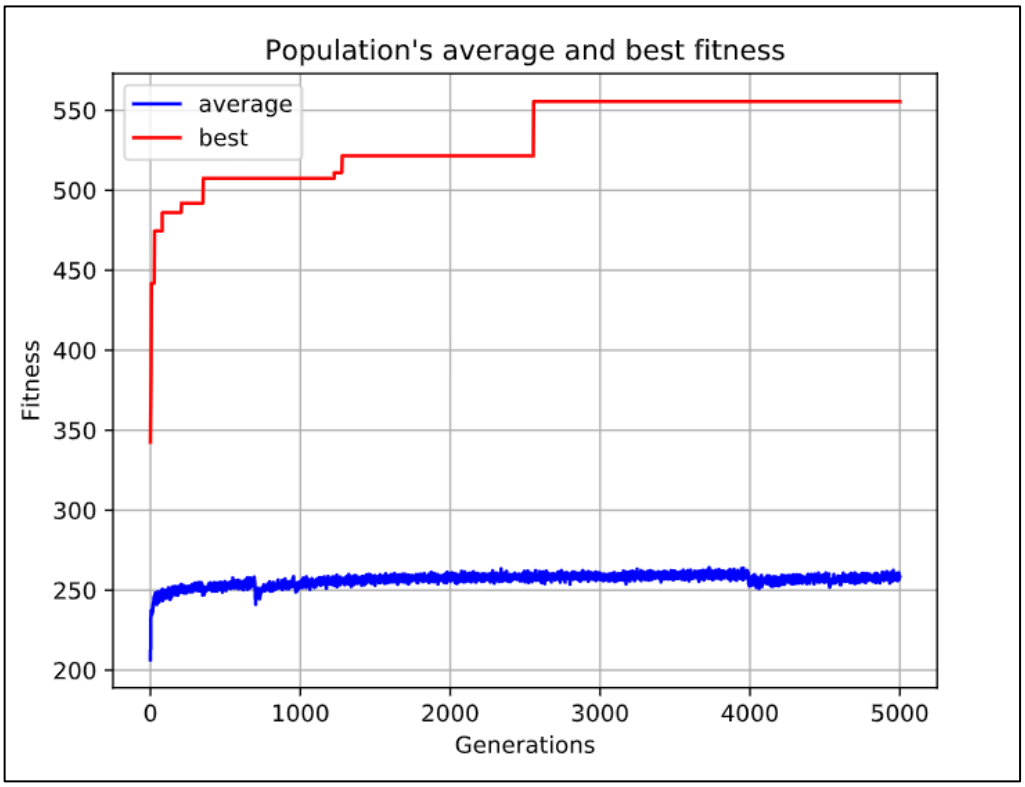
Gambar 5.1 Konfigurasi yang digunakan pada Tahap 1 Pengujian 1

Seperti yang sudah dijelaskan pada Bab 3 dan 4, tahap pertama pada *training* NEAT adalah pengaturan konfigurasi. Konfigurasi yang dilakukan pada pengujian ini dapat dilihat pada Gambar 5.1. Setelah konfigurasi, proses selanjutnya adalah pemilihan *fitness function*. Untuk pengujian pertama ini, *fitness function* yang digunakan adalah *fitness function* pertama, yang dapat dilihat pada Segmen Program 4.4.

```
Population's average fitness: 259.39311 stdev: 97.52384
Best fitness: 555.50000 - size: (11, 23) - species 2 - id 2249411
Average adjusted fitness: 0.360
Mean genetic distance 2.179, standard deviation 0.435
Population of 1000 members in 3 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
     1  4965   255    515.5    0.367  2410
     2  4965   431    555.5    0.347  2381
     3  4965   314    549.5    0.365  1273
Total extinctions: 0
Generation time: 6.861 sec
```

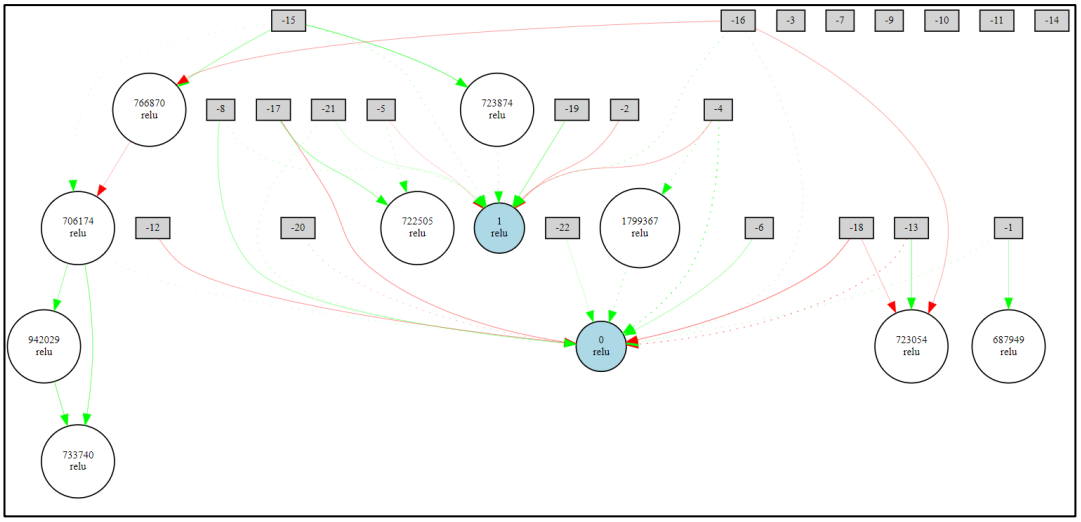
Gambar 5.2 Hasil training dari tahap 1 pengujian 1

Setelah dijalankan sebanyak 5000 generasi dengan waktu rata-rata 6.8 detik per generasi, total waktu yang dibutuhkan NEAT untuk menjalankan proses *training* dengan tahap 1 pengujian 1 adalah 571.75 menit. Hasil pengujian dapat dilihat pada Gambar 5.2. Dengan konfigurasi dan *fitness function* yang digunakan, individu terbaik yang dihasilkan NEAT memiliki nilai fitness sebesar 555.5, dengan rata-rata *fitness* pada populasi sebesar 259.39311. Grafik *fitness* terbaik selama pengujian dan rata-rata *fitness* dalam populasi dapat dilihat pada Gambar 5.3.




Gambar 5.3 Grafik nilai fitness terbaik dan nilai fitness rata-rata dalam populasi selama proses training pada Tahap 1 Pengujian 1

Individu terbaik yang dihasilkan oleh NEAT dapat dilihat pada Gambar 5.4. Individu ini memiliki kompleksitas jaringan berupa 22 *input nodes*, 9 *hidden nodes*, dan 2 *output nodes* dengan 23 koneksi aktif. Dimana *nodes* -1 sampai -22 merupakan *input nodes*, 0 dan 1 merupakan *output nodes*, dan sisanya merupakan *hidden nodes*. Garis merah dan hijau menandakan koneksi antar *nodes*, yang berarti *weight* positif untuk garis hijau dan negatif untuk garis merah. Sedangkan garis putus-putus menandakan koneksi yang tidak aktif.




Gambar 5.4 Network terbaik yang dihasilkan NEAT pada Tahap 1 Pengujain 1

Setelah *training* selesai, proses selanjutnya adalah *testing* yang bertujuan untuk mengukur tingkat akurasi dari *network* terbaik yang dihasilkan NEAT. Namun setelah dilakukannya proses *testing*, tingkat akurasi dari *network* terbaik ini masih sangat buruk. *Network* hanya mampu memprediksi hasil pertandingan dengan benar sebanyak 107 pertandingan, dimana 32 diantaranya dapat diprediksi dengan skor yang tepat. Ringkasan proses *training* dan proses *testing* dapat dilihat pada Tabel 5.2.

Tabel 5.2 Rangkuman proses training dan testing Tahap 1 Pengujian 1

| | |
|---|---|
| Generasi | 5000 |
| Rata-rata waktu per generasi | 6.861 detik |
| Total waktu | 571.75 menit |
| *Fitness Function* | *Fitness Function 1* (Segmen Program 4.4) |
| Rata-rata nilai *fitness* dalam populasi | 259.39311 |
| Nilai *fitness* terbaik dalam populasi | 555.5 |
| Kompleksitas *network* pada individu dengan nilai *fitness* terbaik | 33 *nodes* dan 23 koneksi aktif |

| | |
|---|---|
| Skor pertandingan benar | 32 (8%) |
| Hasil pertandingan benar | 107 (28%) |

#### 5.2.1.2 Tahap 1 Pengujian 2

Karena hasil Tahap 1 Pengujian 1 kurang memuaskan, pada pengujian ini *fitness function* yang digunakan diganti menjadi *fitness function 2*, yang dapat dilihat pada Segmen Program 4.2.

Sedangkan untuk konfigurasi yang digunakan, masih sama dengan konfigurasi yang digunakan pada Tahap 1 Pengujian 1.

```
[NEAT]
fitness_criterion     = max
fitness_threshold     = 1140
pop_size              = 1000
reset_on_extinction   = True
no_fitness_termination = True

[DefaultGenome]
# node activation options
activation_default      = relu
activation_mutate_rate  = 0.0
activation_options      = relu

# node aggregation options
aggregation_default     = sum
aggregation_mutate_rate = 0.0
aggregation_options     = sum

# node bias options
bias_init_mean          = 0.0
bias_init_stdev         = 1.0
bias_max_value          = 100.0
bias_min_value          = -100.0
bias_mutate_power       = 0.1
bias_mutate_rate        = 0.7
bias_replace_rate       = 0.3

# genome compatibility options
compatibility_disjoint_coefficient = 1.0
compatibility_weight_coefficient   = 0.5

# connection add/remove rates
conn_add_prob           = 0.8
conn_delete_prob        = 0.5

# connection enable options
enabled_default         = True
enabled_mutate_rate     = 0.4

feed_forward            = True
initial_connection      = partial_direct 0.7

# node add/remove rates
node_add_prob           = 0.8
node_delete_prob        = 0.5

# network parameters
num_hidden              = 1
num_inputs              = 22
num_outputs             = 2

# node response options
response_init_mean      = 1.0
response_init_stdev     = 0.0
response_max_value      = 3000.0
response_min_value      = -3000.0
response_mutate_power   = 0.0
response_mutate_rate    = 0.0
response_replace_rate   = 0.0

single_structural_mutation = False
structural_mutation_surer = False

# connection weight options
weight_init_mean        = 0.0
weight_init_stdev       = 1.0
weight_max_value        = 1000
weight_min_value        = -1000
weight_mutate_power     = 0.1
weight_mutate_rate      = 0.8
weight_replace_rate     = 0.4

[DefaultSpeciesSet]
compatibility_threshold = 3.0

[DefaultStagnation]
species_fitness_func = max
max_stagnation       = 1000
species_elitism      = 5

[DefaultReproduction]
elitism            = 40
survival_threshold = 0.5
```

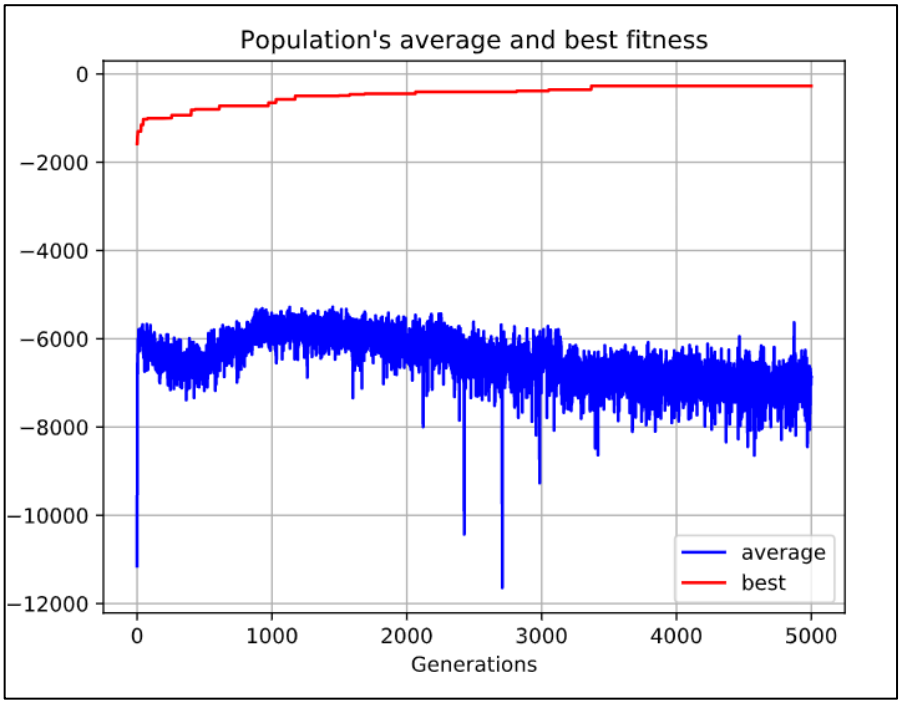
Gambar 5.5 Konfigurasi yang digunakan pada Tahap 1 Pengujian 2

Proses *training* pada pengujian ini juga dijalankan sebanyak 5000 generasi. Namun, pengujian ini memakan waktu yang lebih lama dibanding Tahap 1 Pengujian 1. Rata-rata waktu per generasi yang dibutuhkan pada pengujian ini adalah 8.124 detik, sehingga total waktu yang dibutuhkan untuk menyelesaikan proses *training* adalah 677 menit. Hasil *training* dapat dilihat pada Gambar 5.6

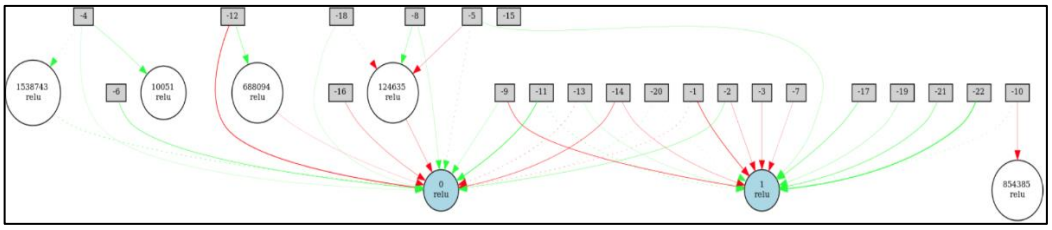
```
Population's average fitness: -7509.48097 stdev: 11127.63908
Best fitness: -272.27539 - size: (7, 29) - species 5 - id 2554238
Average adjusted fitness: 0.963
Mean genetic distance 2.429, standard deviation 0.498
Population of 1000 members in 5 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
     3  4990   164   -691.2    0.963  3414
     4  4990   346   -328.9    0.962   988
     5  4388   149   -272.3    0.971  1622
     6  4378   165   -372.7    0.961   153
     7  4181   176   -381.0    0.960  2175
Total extinctions: 0
Generation time: 8.124 sec
```

Gambar 5.6 Hasil training dari Tahap 1 Pengujian 2

Setelah proses *training* selesai, individu yang keluar sebagai pemenang memiliki nilai *fitness* sebesar -272.275. Perlu diingat bahwa *fitness function* yang digunakan pada penelitain ini berbeda dengan Tahap 1 Pengujian 1, sehingga nilai *fitness* pada penelitian ini tidak bisa dibandingkan secara langsung dengan nilai *fitness* terbaik yang ada pada Tahap 1 Pengujian 1. Grafik *fitness* terbaik selama pengujian dan rata-rata *fitness* dalam populasi dapat dilihat pada Gambar 5.7.




Gambar 5.7 Grafik nilai fitness terbaik dan nilai fitness rata-rata dalam populasi selama proses training pada Tahap 1 Pengujian 2




Gambar 5.8 Network terbaik yang dihasilkan NEAT pada Tahap 1 Pengujain 2

Setelah dilakukan proses testing, *network* yang dihasilkan NEAT ternyata memiliki tingkat akurasi yang jauh lebih tinggi jika dibandingkan dengan Tahap 1 Pengujian 1. *Network* pada pengujian ini mampu memprediksi hasil pertandingan sebanyak 273 dari 380 pertandingan, dimana 85 diantaranya dapat diprediksi dengan skor yang benar. Hal ini membuktikan bahwa *fitness function 2* mampu menilai sebuah *network* lebih baik daripada *fitness function 1* karena konfigurasi yang digunakan pada pengujian ini sama dengan konfigurasi yang digunakan pada Tahap 1 Pengujian 1. Rangkuman Tahap 1 Pengujian 2 dapat dilihat pada Tabel 5.3.

Tabel 5.3 Rangkuman proses training dan testing Tahap 1 Pengujian 2

| | |
|---|---|
| Generasi | 5000 |
| Rata-rata waktu per generasi | 8.124 detik |
| Total waktu | 677 menit |
| *Fitness Function* | *Fitness Function 2* (Segmen Program 4.5) |
| Rata-rata nilai *fitness* dalam populasi | -7509.48 |
| Nilai *fitness* terbaik dalam populasi | -272.275 |
| Kompleksitas *network* pada individu dengan nilai *fitness* terbaik | 29 *nodes* dan 29 koneksi aktif |
| Skor pertandingan benar | 85 (22%) |
| Hasil pertandingan benar | 273 (71%) |

#### 5.2.1.3 Tahap 1 Pengujian 3

Berdasarkan hasil dari Tahap 1 Pengujian 2, *fitness function* yang akan digunakan pada pengujian ini adalah *fitness function 2* karena sudah terbukti lebih baik daripada *fitness function 1*.

Konfigurasi pada pengujian ini sedikit berbeda dengan 2 pengujian sebelumnya. Pada pengujian ini, `conn_delete_prop` dan `node_delete_prob` yang mengatur kemungkinan untuk hilangnya koneksi dan *nodes* dalam proses *mutation* diubah menjadi 0 dari 0.5 seperti pada 2 pengujian sebelumnya. Dengan dilakukannya perubahan konfigurasi ini, diharapkan individu dapat berkembang lebih cepat sehingga mampu menghasilkan akurasi yang lebih baik.

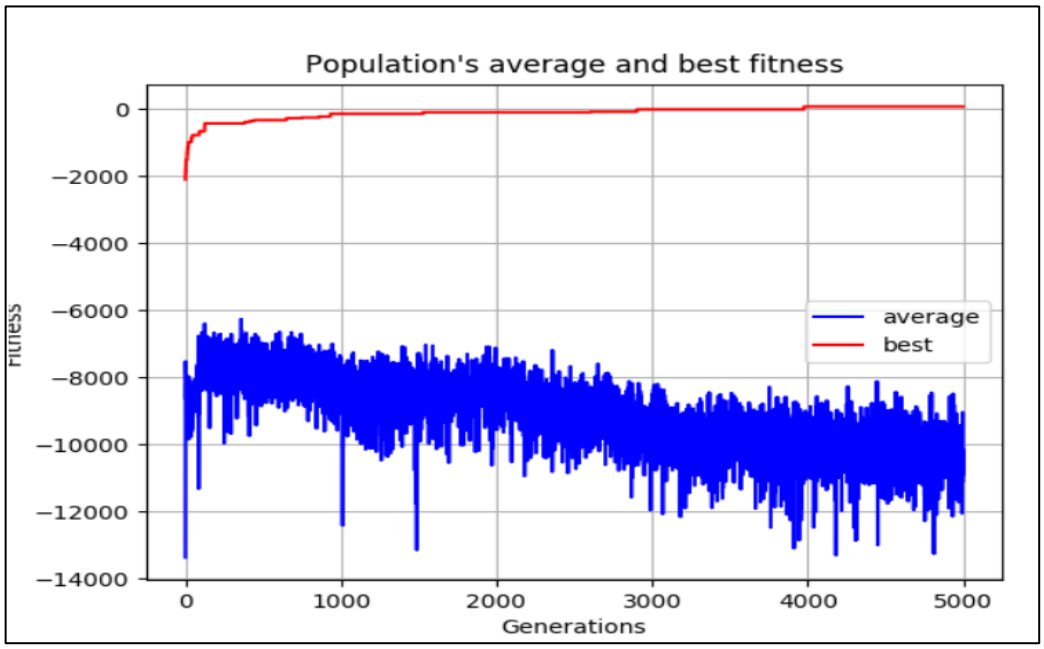
```
[NEAT]
fitness_criterion     = max
fitness_threshold     = 1140
pop_size              = 1000
reset_on_extinction   = True
no_fitness_termination = True

[DefaultGenome]
# node activation options
activation_default      = relu
activation_mutate_rate  = 0.0
activation_options      = relu

# node aggregation options
aggregation_default     = sum
aggregation_mutate_rate = 0.0
aggregation_options     = sum

# node bias options
bias_init_mean          = 0.0
bias_init_stdev         = 1.0
bias_max_value          = 100.0
bias_min_value          = -100.0
bias_mutate_power       = 0.1
bias_mutate_rate        = 0.7
bias_replace_rate       = 0.3

# genome compatibility options
compatibility_disjoint_coefficient = 1.0
compatibility_weight_coefficient   = 0.5

# connection add/remove rates
conn_add_prob           = 0.5
conn_delete_prob        = 0.0

# connection enable options
enabled_default         = True
enabled_mutate_rate     = 0.0

feed_forward            = True
initial_connection      = partial_direct 0.7

# node add/remove rates
node_add_prob           = 0.5
node_delete_prob        = 0.0

# network parameters
num_hidden              = 1
num_inputs              = 22
num_outputs             = 2

# node response options
response_init_mean      = 1.0
response_init_stdev     = 0.0
response_max_value      = 3000.0
response_min_value      = -3000.0
response_mutate_power   = 0.0
response_mutate_rate    = 0.0
response_replace_rate   = 0.0

single_structural_mutation = False
structural_mutation_surer = False

# connection weight options
weight_init_mean        = 0.0
weight_init_stdev       = 1.0
weight_max_value        = 1000
weight_min_value        = -1000
weight_mutate_power     = 0.1
weight_mutate_rate      = 0.8
weight_replace_rate     = 0.4

[DefaultSpeciesSet]
compatibility_threshold = 3.0

[DefaultStagnation]
species_fitness_func = max
max_stagnation       = 1000
species_elitism      = 5

[DefaultReproduction]
elitism              = 40
survival_threshold = 0.5
```

Gambar 5.9 Konfigurasi yang digunakan pada Tahap 1 Pengujian 3

```
Population's average fitness: -12051.08154 stdev: 31720.80446
Best fitness: 75.79604 - size: (14, 70) - species 6 - id 2718253
Average adjusted fitness: 0.981
Mean genetic distance 2.847, standard deviation 0.700
Population of 1000 members in 6 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
     1  4991   167     34.0    0.975    25
     5  4915   166    -78.6    0.977  2386
     6  4915   167     75.8    0.985  1015
     7  4915   166      3.8    0.985   405
     8  4915   167   -100.9    0.979   755
     9  4915   167      4.2    0.985   615
Total extinctions: 0
Generation time: 18.803 sec
```

Gambar 5.10 Hasil training dari Tahap 1 Pengujian 3

Efek perubahan konfigurasi dapat dilihat pada hasil *training* pengujian ini pada Gambar 5.10. Dengan mengatur kemungkinan hilangnya koneksi dan *nodes* menjadi 0, koneksi yang dimiliki *network* terbaik pada pengujian ini menjadi jauh lebih banyak dan nilai *fitness* terbaik juga lebih tinggi dari 2 pengujian sebelumnya. Grafik *fitness* terbaik selama pengujian dan rata-rata *fitness* dalam populasi dapat dilihat pada Gambar 5.3.




Gambar 5.11 Grafik nilai fitness terbaik dan nilai fitness rata-rata dalam populasi selama proses training pada Tahap 1 Pengujian 1

Tidak hanya efek positif, perubahan konfigurasi ini juga disertai efek negatif. Pada 2 pengujian sebelumnya, 1 generasi dapat diselesaikan dalam waktu kurang dari 10 detik. Tetapi pada pengujian ini, diibutuhkan rata-rata 18.8 detik untuk menyelesaikan 1 generasi. Sehingga total waktu yang dibutuhkan untuk menyelesaikan 5000 generasi adalah sekitar 1566 menit, jauh lebih lama dari 2 pengujian sebelumnya.

Efek positif yang ditemukan pada proses *training* juga berpengaruh pada proses *testing*. Tingkat akurasi dari *network* yang dihasilkan pada pengujian ini mampu melebihi tingkat akurasi pada 2 pengujian sebelumnya. *Network* pada pengujian ini mampu memprediksi 300 dari 380 pertandingan dengan benar, dimana 125 diantaranya dapat diprediksi dengan skor yang benar. Rangkuman proses *training* dan *testing* dapat dilihat pada Tabel 5.4.

Tabel 5.4 Rangkuman proses training dan testing pada Tahap 1 Pengujian 3

| | |
|---|---|
| Generasi | 5000 |
| Rata-rata waktu per generasi | 18.803 detik |
| Total waktu | 1566 menit |
| *Fitness Function* | *Fitness Function 2* (Segmen Program 4.5) |
| Rata-rata nilai *fitness* dalam populasi | -12051.081 |
| Nilai *fitness* terbaik dalam populasi | 75.796 |
| Kompleksitas *network* pada individu dengan nilai *fitness* terbaik | 36 *nodes* dan 70 koneksi aktif |
| Skor pertandingan benar | 125 (30%) |
| Hasil pertandingan benar | 300 (78%) |

### 5.2.2 Tahap 2

Pada tahap ini, *feature* yang digunakan adalah *rating* pemain dari kedua *team*, dan *rating* kedua team itu sendiri. Sehingga ada total 24 *feature* yang terdiri dari 11 pemain dari masing-masing *team* dan *rating* dari kedua team.

### 5.2.2.1 Tahap 2 Pengujian 1

Pada pengujian ini, konfigurasi dan *fitness function* yang digunakan sama seperti Tahap 1 Pengujian 3, hanya saja `num_inputs` berubah menjadi 24 dari yang sebelumnya 22 karena adanya penambahan 2 *feature* baru.

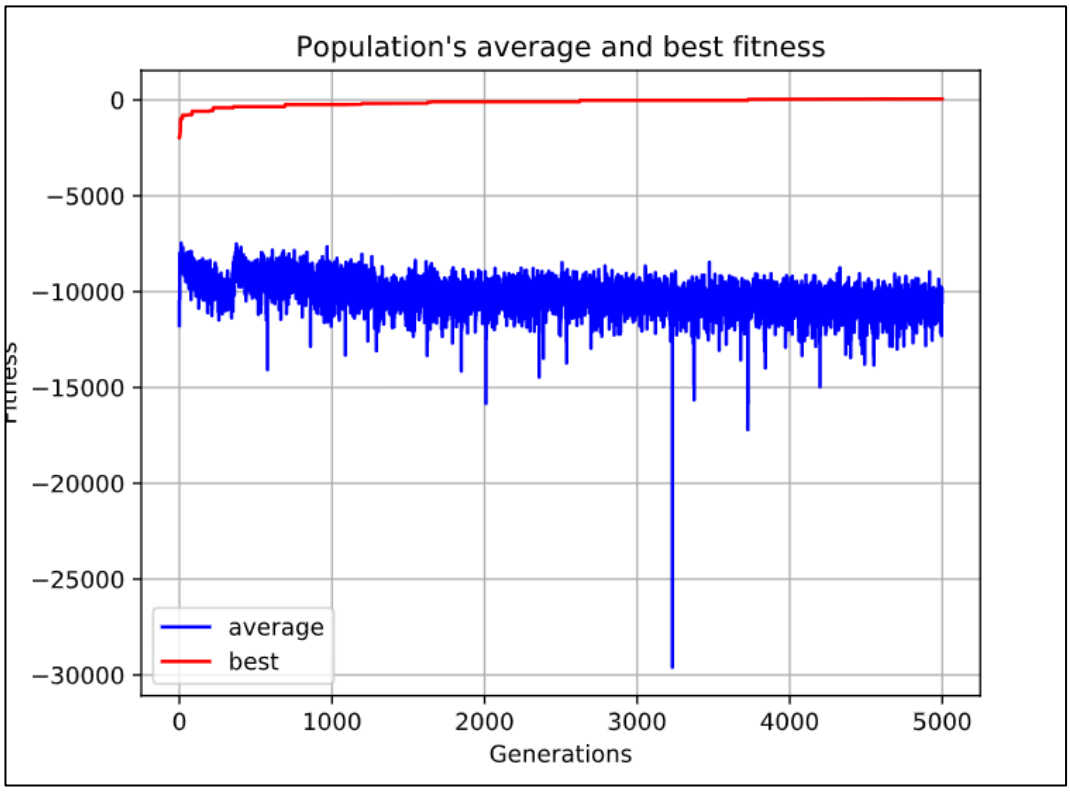
```
[NEAT]
fitness_criterion     = max
fitness_threshold     = 1140
pop_size              = 1000
reset_on_extinction   = True
no_fitness_termination = True

[DefaultGenome]
# node activation options
activation_default      = relu
activation_mutate_rate  = 0.0
activation_options      = relu

# node aggregation options
aggregation_default     = sum
aggregation_mutate_rate = 0.0
aggregation_options     = sum

# node bias options
bias_init_mean          = 0.0
bias_init_stdev         = 1.0
bias_max_value          = 100.0
bias_min_value          = -100.0
bias_mutate_power       = 0.1
bias_mutate_rate        = 0.7
bias_replace_rate       = 0.3

# genome compatibility options
compatibility_disjoint_coefficient = 1.0
compatibility_weight_coefficient   = 0.5

# connection add/remove rates
conn_add_prob           = 0.5
conn_delete_prob        = 0.0

# connection enable options
enabled_default         = True
enabled_mutate_rate     = 0.0

feed_forward            = True
initial_connection      = partial_direct 0.7

# node add/remove rates
node_add_prob           = 0.5
node_delete_prob        = 0.0

# network parameters
num_hidden              = 1
num_inputs              = 24
num_outputs             = 2

# node response options
response_init_mean      = 1.0
response_init_stdev     = 0.0
response_max_value      = 3000.0
response_min_value      = -3000.0
response_mutate_power   = 0.0
response_mutate_rate    = 0.0
response_replace_rate   = 0.0

single_structural_mutation = False
structural_mutation_surer = False

# connection weight options
weight_init_mean        = 0.0
weight_init_stdev       = 1.0
weight_max_value        = 1000
weight_min_value        = -1000
weight_mutate_power     = 0.1
weight_mutate_rate      = 0.8
weight_replace_rate     = 0.4

[DefaultSpeciesSet]
compatibility_threshold = 3.0

[DefaultStagnation]
species_fitness_func = max
max_stagnation       = 1000
species_elitism      = 5

[DefaultReproduction]
elitism            = 40
survival_threshold = 0.5
```

Gambar 5.12 Konfigurasi yang digunakan pada Tahap 2 Pengujian 1

```
Population's average fitness: -11662.11922 stdev: 24262.91578
Best fitness: 50.41736 - size: (9, 61) - species 6 - id 3614032
Average adjusted fitness: 0.971
Mean genetic distance 2.773, standard deviation 0.650
Population of 1000 members in 5 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
     1  4999   189   -155.7    0.983    48
     3  4999   199    -25.2    0.967  2375
     5  4975   200   -107.9    0.970  3372
     6  4653   213     50.4    0.974   267
     8  4635   199   -211.3    0.962  2201
Total extinctions: 0
Generation time: 16.769 sec
```

Gambar 5.13 Hasil training dari Tahap 2 Pengujian 1

Sama seperti proses *training* pada pengujian sebelumnya, *training* pada pengujian ini juga dijalankan selama 5000 generasi. Penambahan *feature* baru pada pengujian ini ternyata tidak memiliki banyak efek positif pada proses *training*. Dapat dilihat pada Gambar 5.13., nilai *fitness* terbaik yang dihasilkan ternyata lebih kecil daripada nilai *fitness* yang dihasilkan oleh Tahap 1 Pengujian 3. Grafik *fitness* terbaik dan rata-rata *fitness* dalam populasi dapat dilihat pada Gambar 5.14.




Gambar 5.14 Grafik nilai fitness terbaik dan nilai fitness rata-rata dalam populasi selama proses training pada Tahap 2 Pengujian 1

Efek positif penambahan 2 *feature* pada pengujian ini baru terlihat pada setelah proses *testing* selesai dijalankan. Tingkat akurasi dari *network* yang dihasilkan pengujian ini sedikit lebih tinggi daripada Tahap 1 Pengujian 3, yang merupakan *network* dengan tingkat akurasi sejauh ini. *Network* yang dihasilkan pada pengujian mampu memprediksi pertandingan dengan benar sebanyak 302 dari 380 pertandingan, dimana 136 diantaranya dapat diprediksi dengan skor yang tepat.

*Network* pada pengujian ini juga memiliki tingkat kompleksitas sedikit lebih rendah dari Tahap 1 Pengujian 3. Rangkuman pengujian ini dapat dilihat pada Tabel 5.5.

Tabel 5.5 Rangkuman proses training dan testing Tahap 2 Pengujian 1

| | |
|---|---|
| Generasi | 5000 |
| Rata-rata waktu per generasi | 16.769 detik |
| Total waktu | 1397 menit |
| *Fitness Function* | *Fitness Function 2* (Segmen Program 4.5) |
| Rata-rata nilai *fitness* dalam populasi | -11662.11922 |
| Nilai *fitness* terbaik dalam populasi | 50.41736 |
| Kompleksitas *network* pada individu dengan nilai *fitness* terbaik | 33 *nodes* dan 61 koneksi aktif |
| Skor pertandingan benar | 136 (35%) |
| Hasil pertandingan benar | 302 (79%) |

### 5.2.2.2 Tahap 2 Pengujian 2

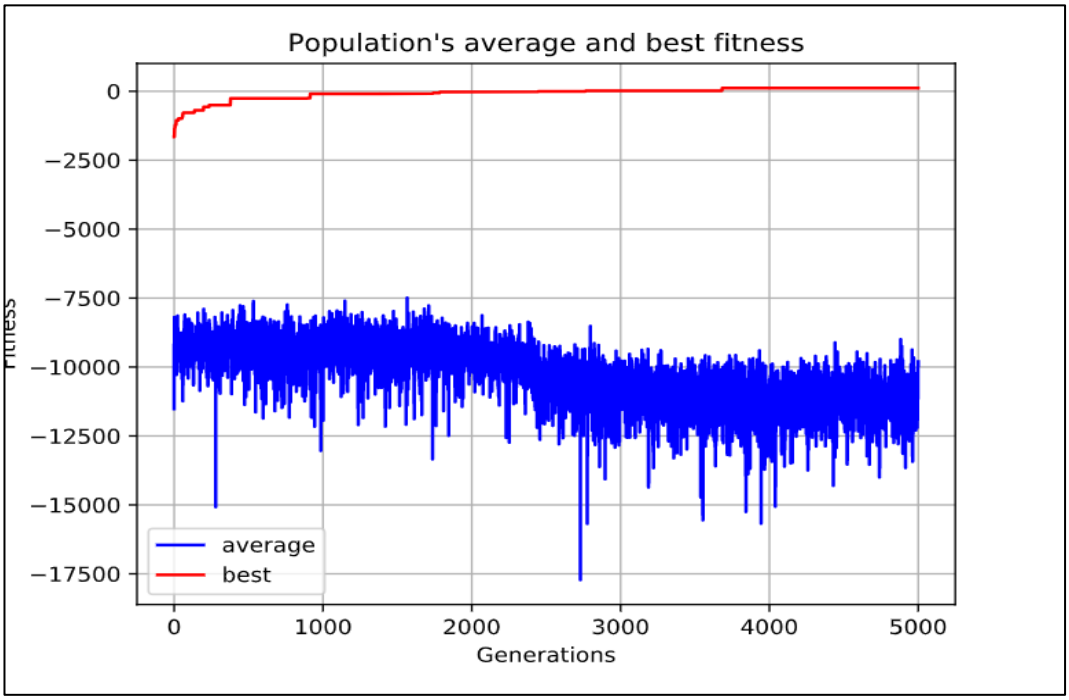
```
[NEAT]
fitness_criterion     = max
fitness_threshold     = 1140
pop_size              = 1000
reset_on_extinction   = True
no_fitness_termination = True

[DefaultGenome]
# node activation options
activation_default      = relu
activation_mutate_rate  = 0.0
activation_options      = relu

# node aggregation options
aggregation_default     = sum
aggregation_mutate_rate = 0.0
aggregation_options     = sum

# node bias options
bias_init_mean          = 0.0
bias_init_stdev         = 1.0
bias_max_value          = 100.0
bias_min_value          = -100.0
bias_mutate_power       = 0.1
bias_mutate_rate        = 0.7
bias_replace_rate       = 0.3

# genome compatibility options
compatibility_disjoint_coefficient = 1.0
compatibility_weight_coefficient   = 0.5

# connection add/remove rates
conn_add_prob           = 0.8
conn_delete_prob        = 0.0

# connection enable options
enabled_default         = True
enabled_mutate_rate     = 0.0

feed_forward            = True
initial_connection      = partial_direct 0.7

# node add/remove rates
node_add_prob           = 0.8
node_delete_prob        = 0.0

# network parameters
num_hidden              = 1
num_inputs              = 24
num_outputs             = 2

# node response options
response_init_mean      = 1.0
response_init_stdev     = 0.0
response_max_value      = 3000.0
response_min_value      = -3000.0
response_mutate_power   = 0.0
response_mutate_rate    = 0.0
response_replace_rate   = 0.0

single_structural_mutation = False
structural_mutation_surer = False

# connection weight options
weight_init_mean        = 0.0
weight_init_stdev       = 1.0
weight_max_value        = 1000
weight_min_value        = -1000
weight_mutate_power     = 0.1
weight_mutate_rate      = 0.8
weight_replace_rate     = 0.4

[DefaultSpeciesSet]
compatibility_threshold = 3.0

[DefaultStagnation]
species_fitness_func = max
max_stagnation       = 1000
species_elitism      = 5

[DefaultReproduction]
elitism           = 40
survival_threshold = 0.5
```

Gambar 5.15 Konfigurasi yang digunakan pada Tahap 2 Pengujian 2

Pada 2 pengujian sebelumnya, menghilangkan kemampuan individu untuk kehilangan *nodes* dan koneksi pada saat terjadinya *mutation* terbukti memberikan efek positif pada tingkat akurasi yang dihasilkan *network*. Pada pengujian ini, `node_add_prob` dan `conn_add_prop` yang mengatur kemungkinan terjadinya penambahan koneksi *nodes* pada proses *mutation* ditingkatkan dari 0.5 menjadi 0.8, dengan harapan *network* dapat berkembang lebih cepat lagi.

```
Population's average fitness: -11531.72723 stdev: 32019.12033
Best fitness: 116.81931 - size: (20, 77) - species 8 - id 2696919
Average adjusted fitness: 0.981
Mean genetic distance 2.868, standard deviation 0.614
Population of 1000 members in 5 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
     1  4987   201    -49.5    0.983   433
     5  4951   200     14.5    0.975  1190
     6  4920   199     34.8    0.981   359
     7  4672   199    -69.8    0.978  1669
     8  4562   201    116.8    0.986  1305
Total extinctions: 0
Generation time: 18.802 sec
```

Gambar 5.16 Hasil training dari Tahap 2 Pengujian 2




Gambar 5.17 Grafik nilai fitness terbaik dan nilai fitness rata-rata dalam populasi selama proses training pada Tahap 2 Pengujian 2

Dan ternyata benar, dapat dilihat pada Gambar 5.16., peningkatan kemungkinan terjadinya penambahan koneksi dan *nodes* saat *mutation* memberikan hasil *training* yang lebih baik. Nilai *fitness* yang dihasilkan pada pengujian ini mampu mengungguli nilai *fitness* yang dihasilkan Tahap 2 Pengujian 1 dan Tahap 1 Pengujian 3, yang merupakan 2 *network* terbaik dalam hal nilai *fitness*. Grafik

nilai *fitness* terbaik dan rata-rata *fitness* dalam populasi selama proses *training* dapat dilihat pada Gambar 5.17.

Tidak hanya pada proses *training*, efek positif dari konfigurasi yang digunakan juga dapat lihat pada hasil proses *testing*. *Network* yang dihasilkan pengujian ini kembali mampu menghasilkan tingkat akurasi yang lebih tinggi dari pengujian-pengujian sebelumnya. *Network* yang dihasilkan pada pengujian ini mampu memprediksi hasil pertandingan dengan benar sebanyak 308 dari 380 pertandingan, dimana 136 diantarnya dapat diprediksi dengan skor yang tepat.

Tentu saja dengan meningkatkan kemungkinan terjadinya penambahan koneksi dan *nodes* pada saat *mutation* membuat kompleksitas *network* yang dihasilkan menjadi lebih tinggi. Rangkuman proses *training* dan *testing* pada pengujian ini dapat dilihat pada Tabel 5.6

Tabel 5.6 Rangkuman proses training dan testing dari Tahap 2 Pengujian 2

| | |
|---|---|
| Generasi | 5000 |
| Rata-rata waktu per generasi | 18.802 detik |
| Total waktu | 1566 menit |
| *Fitness Function* | *Fitness Function 2* (Segmen Program 4.5) |
| Rata-rata nilai *fitness* dalam populasi | -11531.72723 |
| Nilai *fitness* terbaik dalam populasi | 116.81931 |
| Kompleksitas *network* pada individu dengan nilai *fitness* terbaik | 44 *nodes* dan 71 koneksi aktif |
| Skor pertandingan benar | 136 (35%) |
| Hasil pertandingan benar | 302(80%) |

### 5.2.3 Tahap 3

Pada tahap ini, *feature* yang digunakan kembali ditambah. Selain *rating* dari masing-masing *team* dan pemain, posisi pemain juga digunakan sebagai *feature*. Terdapat 16 posisi pemain pada *dataset* yang digunakan, dimana masing-masing

pemain memiliki 1 posisi. Karena posisi pemain tidak berbentuk angka, harus dilakukan *encoding* terlebih dulu sebelum posisi dapat digunakan. Banyaknya *feature* yang digunakan pada tahap ini bergantung pada jenis *encoding* yang digunakan.

### 5.2.3.1 Tahap 3 Pengujian 1

Pada pengujian pertama dalam tahap 3, jenis *encoding* yang digunakan adalah label *encoding*. Label *encoding* merubah posisi pemain menjadi sebuah nilai tunggal. Dengan menggunakan *encoding* jenis ini, jumlah *feature* yang digunakan adalah 46, yang terdiri dari 22 *rating* pemain, 22 posisi yang telah di*encoding*, dan 2 *rating* dari masing-masing *team*.

```
[NEAT]
fitness_criterion     = max
fitness_threshold     = 1140
pop_size              = 1000
reset_on_extinction   = True
no_fitness_termination = True

[DefaultGenome]
# node activation options
activation_default      = relu
activation_mutate_rate  = 0.0
activation_options      = relu

# node aggregation options
aggregation_default     = sum
aggregation_mutate_rate = 0.0
aggregation_options     = sum

# node bias options
bias_init_mean          = 0.0
bias_init_stdev         = 1.0
bias_max_value          = 100.0
bias_min_value          = -100.0
bias_mutate_power       = 0.1
bias_mutate_rate        = 0.7
bias_replace_rate       = 0.3

# genome compatibility options
compatibility_disjoint_coefficient = 1.0
compatibility_weight_coefficient   = 0.5

# connection add/remove rates
conn_add_prob           = 0.8
conn_delete_prob        = 0.0

# connection enable options
enabled_default         = True
enabled_mutate_rate     = 0.0

feed_forward            = True
initial_connection      = partial_direct 0.7

# node add/remove rates
node_add_prob           = 0.8
node_delete_prob        = 0.0

# network parameters
num_hidden              = 1
num_inputs              = 46
num_outputs             = 2

# node response options
response_init_mean      = 1.0
response_init_stdev     = 0.0
response_max_value      = 3000.0
response_min_value      = -3000.0
response_mutate_power   = 0.0
response_mutate_rate    = 0.0
response_replace_rate   = 0.0

single_structural_mutation = False
structural_mutation_surer = False

# connection weight options
weight_init_mean        = 0.0
weight_init_stdev       = 1.0
weight_max_value        = 1000
weight_min_value        = -1000
weight_mutate_power     = 0.1
weight_mutate_rate      = 0.8
weight_replace_rate     = 0.4

[DefaultSpeciesSet]
compatibility_threshold = 3.0

[DefaultStagnation]
species_fitness_func = max
max_stagnation       = 1000
species_elitism      = 5

[DefaultReproduction]
elitism            = 40
survival_threshold = 0.5
```

Gambar 5.18 Konfigurasi yang digunakan pada Tahap 3 Pengujian 1

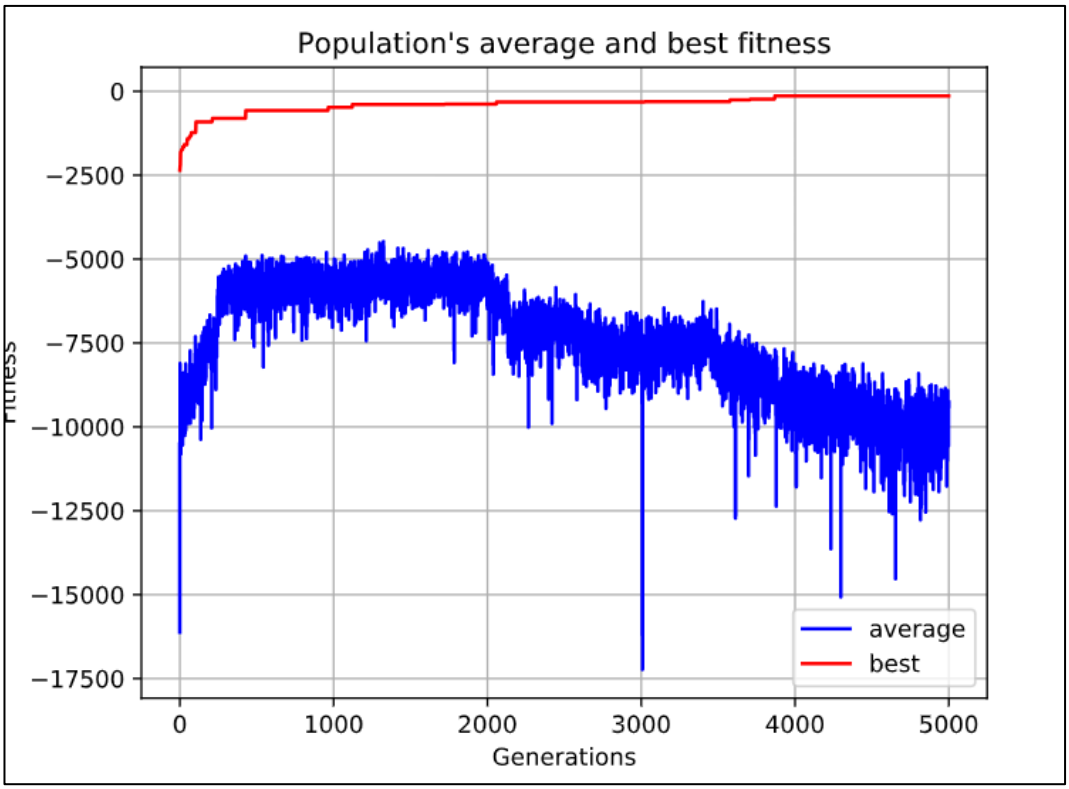
Konfigurasi yang digunakan pada pengujian ini masih sama seperti yang digunakan pada Tahap 2 Pengujian 3, yang merupakan pengujian dengan tingkat akurasi terbaik sejauh ini. Hanya saja, `num_inputs` berubah menjadi 46 dari yang sebelumnya 24. Konfigurasi pada pengujian ini dapat dilihat pada Gambar 5.18.

```
Population's average fitness: -11009.71893 stdev: 24462.22540
Best fitness: -138.71074 - size: (25, 137) - species 8 - id 1892881
Average adjusted fitness: 0.974
Mean genetic distance 2.969, standard deviation 0.608
Population of 1001 members in 7 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
     2  4988   142   -480.1    0.970   119
     4  4980   142   -232.8    0.974  1252
     7  4974   142   -405.4    0.969   214
     8  4915   144   -138.7    0.976  1121
     9  4895   144   -192.6    0.979    33
    10  4855   143   -307.6    0.970  1966
    12  4751   144   -227.4    0.979   426
Total extinctions: 0
Generation time: 26.809 sec
```

Gambar 5.19 Hasil training dari Tahap 3 Pengujian 1

Penambahan *feature* pada pengujian ini membawa banyak efek negatif. Seperti yang dapat dilihat pada Gambar 5.19., waktu yang dibutuhkan untuk menyelesaikan 1 generasi melebihi 20 detik, yang mana tidak pernah terjadi pada penelitian sebelumnya. Untuk menyelesaikan proses *training* sebanyak 5000 generasi, dibutuhkan kurang lebih 2234.08333 menit.

Efek negatif juga dirasakan pada nilai *fitness* yang dihasilkan. Nilai *fitness* terbaik yang dihasilkan pada proses *training* sangat rendah, hanya -138.71074. Hal ini menjadikan pengujian ini sebagai pengujian dengan nilai *fitness* terendah kedua sejauh ini yang menggunakan *fitness function 2*, hanya Tahap 1 Pengujian 2 yang memiliki nilai *fitness* lebih rendah. Belum lagi, kompleksitas *network* juga menjadi tinggi. Grafik nilai *fitness* dan rata-rata nilai *fitness* dalam populasi pada pengujian ini dapat dilihat pada Gambar 5.20




Gambar 5.20 Grafik nilai fitness terbaik dan nilai fitness rata-rata dalam populasi selama proses training pada Tahap 3 Pengujian 1

Efek negatif juga masih terasa pada saat proses testing. Network yang dihasilkan pada pengujian ini hanya mampu memprediksi hasil pertandingan dengan benar sebanyak 260 dari 380 pertandingan, dimana hanya 90 pertandingan yang skornya dapat ditebak dengan benar. Hasil yang kurang memuaskan pada pengujian ini dapat disebabkan oleh *encoding* yang tidak cocok, atau posisi pemain sulit diproses oleh network sehingga akurasi prediksi menjadi rendah. Rangkuman proses *training* dan *testing* dari pengujian ini dapat dilihat pada Tabel 5.7.

Tabel 5.7 Rangkuman proses training dan testing pada Tahap 3 Pengujian 1

| | |
|---|---|
| Generasi | 5000 |
| Rata-rata waktu per generasi | 26.809 detik |
| Total waktu | 2234.08333 menit |
| Jenis *Encoding* | Label *Encoding* |

| *Fitness Function* | *Fitness Function 2* (Segmen Program 4.5) |
|---|---|
| Rata-rata nilai *fitness* dalam populasi | -11009.71893 |
| Nilai *fitness* terbaik dalam populasi | -138.71074 |
| Kompleksitas *network* pada individu dengan nilai *fitness* terbaik | 71 *nodes* dan 137 koneksi aktif |
| Skor pertandingan benar | 90 (23.6%) |
| Hasil pertandingan benar | 260 (68.4%) |

#### 5.2.3.2 Tahap 3 Pengujian 2

Karena hasil yang tidak memuaskan pada Tahap 3 Pengujian 2, pada pengujian ini jenis *encoding* akan diganti. Jenis *encoding* yang akan digunakan pada pengujian ini adalah *onehot encoding*. Terdapat 16 posisi berbeda yang dapat ditempati pemain, yang menjadikan jumlah *feature* pada pengujian ini menjadi sangat banyak. Total jumlah *feature* yang digunakan pada pengujian ini adalah 376, yang terdiri dari 22 *rating* pemain, 2 *rating team*, dan 22 posisi pemain yang setelah di*encode* menjadi 352 (22 * 16).

```
[NEAT]
fitness_criterion     = max
fitness_threshold     = 1140
pop_size              = 700
reset_on_extinction   = True
no_fitness_termination = True

[DefaultGenome]
# node activation options
activation_default      = relu
activation_mutate_rate  = 0.0
activation_options      = relu

# node aggregation options
aggregation_default     = sum
aggregation_mutate_rate = 0.0
aggregation_options     = sum

# node bias options
bias_init_mean          = 0.0
bias_init_stdev         = 1.0
bias_max_value          = 100.0
bias_min_value          = -100.0
bias_mutate_power       = 0.1
bias_mutate_rate        = 0.7
bias_replace_rate       = 0.3

# genome compatibility options
compatibility_disjoint_coefficient = 1.0
compatibility_weight_coefficient   = 0.5

# connection add/remove rates
conn_add_prob           = 0.8
conn_delete_prob        = 0.0

# connection enable options
enabled_default         = True
enabled_mutate_rate     = 0.0

feed_forward            = True
initial_connection      = partial_direct 0.4

# node add/remove rates
node_add_prob           = 0.8
node_delete_prob        = 0.0

# network parameters
num_hidden              = 1
num_inputs              = 376
num_outputs             = 2

# node response options
response_init_mean      = 1.0
response_init_stdev     = 0.0
response_max_value      = 3000.0
response_min_value      = -3000.0
response_mutate_power   = 0.0
response_mutate_rate    = 0.0
response_replace_rate   = 0.0

single_structural_mutation = False
structural_mutation_surer = False

# connection weight options
weight_init_mean        = 0.0
weight_init_stdev       = 1.0
weight_max_value        = 1000
weight_min_value        = -1000
weight_mutate_power     = 0.1
weight_mutate_rate      = 0.8
weight_replace_rate     = 0.4

[DefaultSpeciesSet]
compatibility_threshold = 3.0

[DefaultStagnation]
species_fitness_func = max
max_stagnation       = 1000
species_elitism      = 5

[DefaultReproduction]
elitism              = 40
survival_threshold = 0.5
```

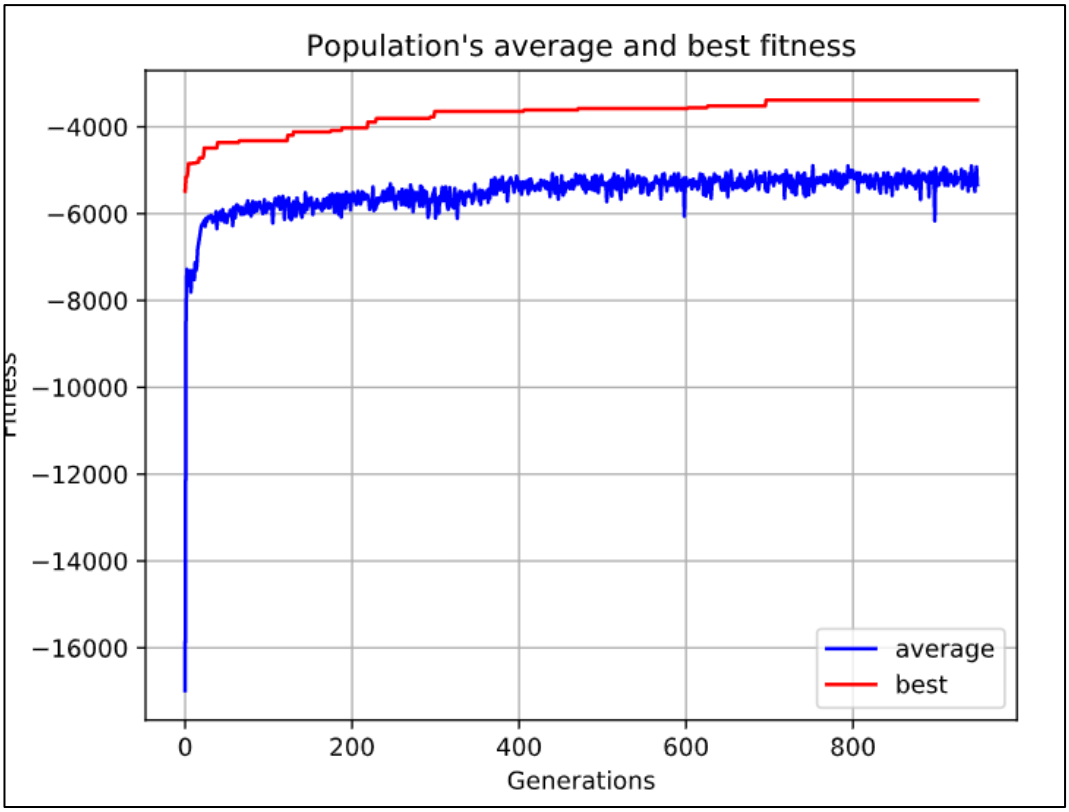
Gambar 5.21 Konfigurasi yang digunakan pada Tahap 3 Pengujian 2

Konfigurasi yang digunakan juga mengalami penyesuaian karena jumlah *feature* yang terlalu besar.Seperti `pop_size` yang mengatur besarnya jumlah populasi diturunkan menjadi 700 dari 1000, dan `initial_connection partial_direct`, yang mengatur koneksi awal pada masing-masing individu juga diturunkan menjadi 0.4 dari yang sebelumnya 0.7.

```
Population's average fitness: -5281.77021 stdev: 4672.45793
Best fitness: -3384.64161 - size: (17, 478) - species 10 - id 117370
Average adjusted fitness: 0.968
Mean genetic distance 3.034, standard deviation 0.530
Population of 698 members in 14 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
     1  995    50  -4192.2    0.973   364
     2  995    50  -3839.4    0.972    57
     3  995    50  -3667.3    0.972   560
     4  993    50  -3798.0    0.973    15
     5  993    49  -4206.4    0.963   147
     6  993    50  -3519.2    0.973   369
     7  984    51  -3869.0    0.970   157
     8  984    50  -3709.3    0.974   353
     9  984    49  -3663.6    0.953   167
    10  983    50  -3384.6    0.972   299
    11  983    51  -3557.6    0.982   392
    12  982    48  -4040.4    0.949    70
    13  982    50  -3842.8    0.955   125
    14  639    50  -3895.3    0.975   205
Total extinctions: 0
Generation time: 55.592 sec
```

Gambar 5.22 Hasil training dari Tahap 3 Pengujian 2

Berbeda dari pengujian-pengujian sebelumnya, proses *training* pada pengujian ini dihentikan pada 1000 generasi. Pengujian dihentikan karena *progress* dari nilai *fitness* dan rata-rata nilai *fitness* dalam populasi sangat lambat, dan waktu yang dibutuhkan untuk menjalankan 1 generasi sangat lama. Dapat dilihat pada Gambar 5.22., nilai *fitness* terbaik yang dihasilkan sangat rendah, yaitu -3384.64161 dan waktu yang untuk menyelesaikan 1 generasi lebih dari 50 detik. Untuk menjalankan 1000 generasi saja, total waktu yang dibutuhkan adalah 926.53333 menit. Selain itu, kompleksitas *network*, terutama koneksi menjadi sangat banyak karena banyaknya *feature* yang digunakan. Grafik nilai *fitness* terbaik dan nilai *fitness* rata-rata dalam populasi dapat dilihat pada Gambar 5.23.




Gambar 5.23 Grafik nilai fitness terbaik dan nilai fitness rata-rata dalam populasi selama proses training pada Tahap 3 Pengujian 2

Selaras dengan proses training, tingkat akurasi yang dihasilkan proses testing juga tidak memuaskan. Network hanya mampu memprediksi hasil pertandingan dengan benar sebanyak 194 dari 380 pertandingan, dimana 47 diantarnya dapat diprediksi dengan skor yang benar. Rangkuman proses *training* dan *testing* pada pengujian ini dapat dilihat pada Tabel

Tabel 5.8 Rangkuman proses training dan testing pada Tahap 3 Pengujian 2

| | |
|---|---|
| Generasi | 1000 |
| Rata-rata waktu per generasi | 55.592 detik |
| Total waktu | 926.53333 menit |
| Jenis *Encoding* | *OneHot Encoding* |

| *Fitness Function* | *Fitness Function 2* (Segmen Program 4.5) |
|---|---|
| Rata-rata nilai *fitness* dalam populasi | -5281.77021 |
| Nilai *fitness* terbaik dalam populasi | -3384.64161 |
| Kompleksitas *network* pada individu dengan nilai *fitness* terbaik | 393 *nodes* dan 478 koneksi aktif |
| Skor pertandingan benar | 47 (12 %) |
| Hasil pertandingan benar | 194 (51 %) |

#### 5.2.3.3 Tahap 3 Pengujian 3

Karena hasil dari Tahap 3 Pengujian 2 masih belum memuaskan juga, pada pengujian kali ini jenis *encoding* akan kembali diganti. Jenis *encoding* yang akan digunakan pada pengujian ini adalah *binary encoding*. *Encoding* jenis ini akan merubah merubah nilai posisi pemain menjadi sebuah nilai *binary*. Keuntungan *encoding* jenis ini dibanding *onehot encoding* adalah dimensi *output* yang dihasilkan lebih sedikit. Karena ada 16 posisi pemain, maka *output* dari *binary encoding* hanya memiliki 4 dimensi, dibandingkan 16 dimensi dari *output* yang dihasilkan *onehot encoding*. Total ada 112 *feature* yang digunakan pada pengujian ini, yang terdiri dari 22 *rating* pemain, 2 *rating* team, dan 88 nilai posisi pemain (22 * 4).

```
[NEAT]
fitness_criterion     = max
fitness_threshold     = 1140
pop_size              = 1000
reset_on_extinction   = True
no_fitness_termination = True

[DefaultGenome]
# node activation options
activation_default      = relu
activation_mutate_rate  = 0.0
activation_options      = relu

# node aggregation options
aggregation_default     = sum
aggregation_mutate_rate = 0.0
aggregation_options     = sum

# node bias options
bias_init_mean          = 0.0
bias_init_stdev         = 1.0
bias_max_value          = 100.0
bias_min_value          = -100.0
bias_mutate_power       = 0.1
bias_mutate_rate        = 0.7
bias_replace_rate       = 0.3

# genome compatibility options
compatibility_disjoint_coefficient = 1.0
compatibility_weight_coefficient   = 0.5

# connection add/remove rates
conn_add_prob           = 0.8
conn_delete_prob        = 0.4

# connection enable options
enabled_default         = True
enabled_mutate_rate     = 0.0

feed_forward            = True
initial_connection      = partial_direct 0.7

# node add/remove rates
node_add_prob           = 0.8
node_delete_prob        = 0.4

# network parameters
num_hidden              = 1
num_inputs              = 112
num_outputs             = 2

# node response options
response_init_mean      = 1.0
response_init_stdev     = 0.0
response_max_value      = 3000.0
response_min_value      = -3000.0
response_mutate_power   = 0.0
response_mutate_rate    = 0.0
response_replace_rate   = 0.0

single_structural_mutation = False
structural_mutation_surer = False

# connection weight options
weight_init_mean        = 0.0
weight_init_stdev       = 1.0
weight_max_value        = 1000
weight_min_value        = -1000
weight_mutate_power     = 0.1
weight_mutate_rate      = 0.8
weight_replace_rate     = 0.4

[DefaultSpeciesSet]
compatibility_threshold = 3.0

[DefaultStagnation]
species_fitness_func = max
max_stagnation       = 1000
species_elitism      = 5

[DefaultReproduction]
elitism            = 40
survival_threshold = 0.5
```

Gambar 5.24 Konfigurasi yang digunakan pada Tahap 3 Pengujian 3

Penyesuaian konfigurasi juga kembali dilakukan pada penelitian ini. Selain `num_inputs` yang berganti menjadi 112, `pop_size` yang mengatur jumlah populasi juga kembali ditingkatkan menjadi 1000 karena jumlah *feature* tidak sebanyak Tahap 3 Pengujian 2. Kemampuan *network* untuk kehilangan koneksi dan *nodes* yang diatur oleh `conn_delete_prob` dan `nodes_delete_prop` juga dikembali diaktifkan, dan diatur ke 0.4.

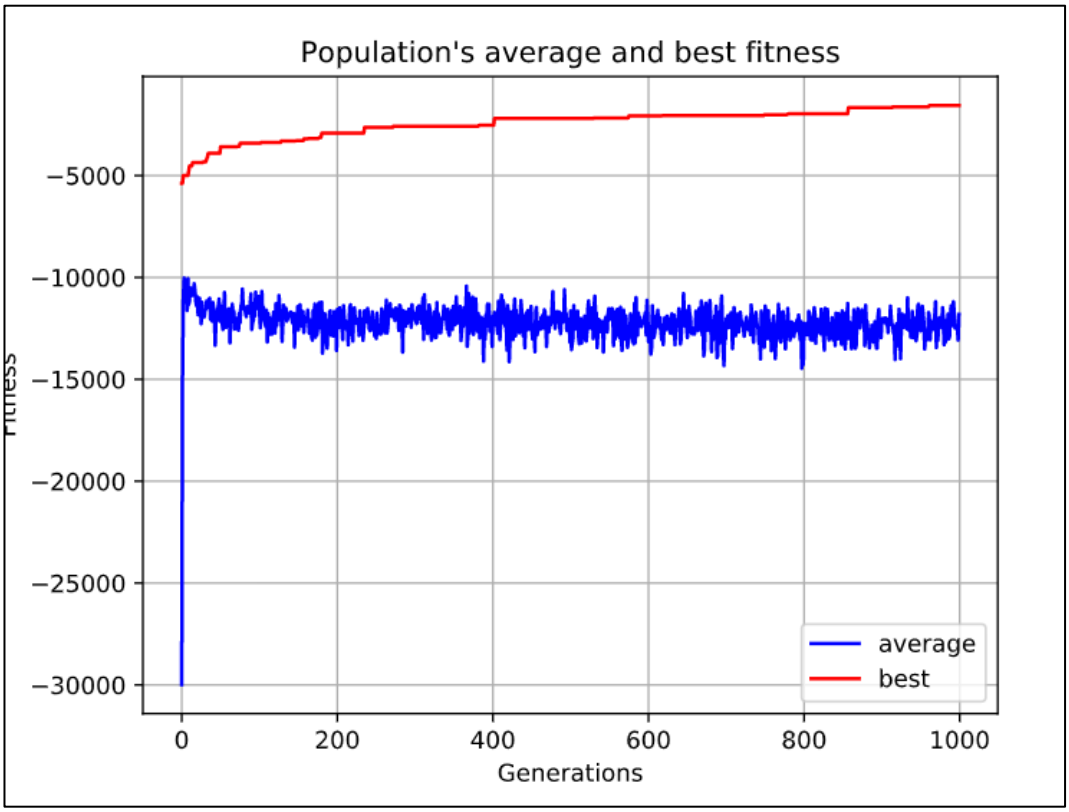
```
Population's average fitness: -12364.50885 stdev: 18992.98633
Best fitness: -1560.54523 - size: (16, 114) - species 3 - id 860255
Average adjusted fitness: 0.945
Mean genetic distance 2.747, standard deviation 0.443
Population of 999 members in 3 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
     1  995   332  -1954.1    0.942    41
     2  949   335  -1836.1    0.947   123
     3  697   332  -1560.5    0.947    34
Total extinctions: 0
Generation time: 29.952 sec
```

Gambar 5.25 Hasil training dari Tahap 3 Pengujian 3

Sama seperti pengujian sebelumnya, proses training pengujian ini juga dihentikan pada 1000 generasi. Alasan dihentikannya proses *training* ini juga sama, yaitu waktu yang terlalu lama disertai *progress* dari nilai *fitness* sangat lambat.

Waktu yang dibutuhkan untuk menyelesaikan 1 generasi pada proses *training* ini sekitar 29 detik. Nilai fitness yang dihasilkan proses *training* pada pengujian ini juga terbilang kecil, hanya -1560.5453 dengan rata-rata nilai *fitness* pada populasi sebesar -12364.50885. Grafik *fitness* terbaik selama pengujian dan rata-rata *fitness* dalam populasi dapat dilihat pada Gambar 5.26.

Hasil dari proses *testing* pada pengujian ini juga kurang memuaskan. *Network* hanya mampu menebak hasil pertandingan dengan benar sebanyak 223 dari 380 pertandingan, dimana 46 pertandingan diantaranya dapat ditebak dengan skor yang benar. Tingkat akurasi yang dihasilkan pengujian ini hanya sedikit lebih baik daripada Tahap 3 Pengujian 2 dan masih tertinggal cukup jauh daripada pengujian dengan tingkat akurasi terbaik yang pernah dilakukan, yaitu pada Tahap 2 Pengujian 2. Rangkuman proses *training* dan *testing* pada pengujian ini dapat dilihat pada Tabel 5.9.




Gambar 5.26 Grafik nilai fitness terbaik dan nilai fitness rata-rata dalam populasi selama proses training pada Tahap 3 Pengujian 3

Tabel 5.9 Rangkuman proses training dan testing pada Tahap 3 Pengujian 3

| | |
|---|---|
| Generasi | 1000 |
| Rata-rata waktu per generasi | 28.952 detik |
| Total waktu | 482.53333 |
| Jenis *Encoding* | *Binary Encoding* |
| *Fitness Function* | *Fitness Function 2* (Segmen Program 4.5) |
| Rata-rata nilai *fitness* dalam populasi | -12364.50885 |
| Nilai *fitness* terbaik dalam populasi | -1560.54523 |
| Kompleksitas *network* pada individu dengan nilai *fitness* terbaik | 128 *nodes* dan 114 koneksi aktif |
| Skor pertandingan benar | 46 (12 %) |
| Hasil pertandingan benar | 223 (58 %) |

#### 5.2.3.4 Tahap 3 Pengujian 4

Berbeda dengan pengujian lainnya yang dilakukan pada Tahap 3., posisi setiap pemain pada pengujain ini tidak di*encode* satu persatu. Jumlah dari masing-masing *defender*, *midfielder*, dan *striker*lah yang menjadi parameter tambahan. Dengan begitu, jumlah *features* yang digunakan pada pengujian ini hanya 30, yang terdiri dari 11 *rating* pemain, 1 *ratimg team*, 1 jumlah *defender*, 1 jumlah *midfielder*, 1 jumlah *striker* dari masing masing *team*.

```
[NEAT]
fitness_criterion     = max
fitness_threshold     = 1140
pop_size              = 1000
reset_on_extinction   = True
no_fitness_termination = True

[DefaultGenome]
# node activation options
activation_default      = relu
activation_mutate_rate  = 0.0
activation_options      = relu

# node aggregation options
aggregation_default     = sum
aggregation_mutate_rate = 0.0
aggregation_options     = sum

# node bias options
bias_init_mean          = 0.0
bias_init_stdev         = 1.0
bias_max_value          = 100.0
bias_min_value          = -100.0
bias_mutate_power       = 0.1
bias_mutate_rate        = 0.7
bias_replace_rate       = 0.3

# genome compatibility options
compatibility_disjoint_coefficient = 1.0
compatibility_weight_coefficient   = 0.5

# connection add/remove rates
conn_add_prob           = 0.8
conn_delete_prob        = 0.0

# connection enable options
enabled_default         = True
enabled_mutate_rate     = 0.0

feed_forward            = True
initial_connection      = partial_direct 0.8

# node add/remove rates
node_add_prob           = 0.8
node_delete_prob        = 0.0

# network parameters
num_hidden              = 2
num_inputs              = 30
num_outputs             = 2

# node response options
response_init_mean      = 1.0
response_init_stdev     = 0.0
response_max_value      = 3000.0
response_min_value      = -3000.0
response_mutate_power   = 0.0
response_mutate_rate    = 0.0
response_replace_rate   = 0.0

single_structural_mutation = False
structural_mutation_surer = False

# connection weight options
weight_init_mean        = 0.0
weight_init_stdev       = 1.0
weight_max_value        = 1000
weight_min_value        = -1000
weight_mutate_power     = 0.1
weight_mutate_rate      = 0.8
weight_replace_rate     = 0.4

[DefaultSpeciesSet]
compatibility_threshold = 3.0

[DefaultStagnation]
species_fitness_func = max
max_stagnation       = 1000
species_elitism      = 5

[DefaultReproduction]
elitism            = 40
survival_threshold = 0.5
```

Gambar 5.27 Konfigurasi yang digunakan pada Tahap 3 Pengujian 4

Konfigurasi yang digunakan pada pengujian ini masih sama dengan konfigurasi yang digunakan pada Tahap 3 Pengujian 3., hanya saja jumlah input berkurang menjadi 30, yang menjadikan pengujian ini pengujian dengan jumlah *features* paling sedikit pada Tahap 3. Konfigurasi dapat dilihat pada Gambar 5.27.

```
Population's average fitness: -9819.25758 stdev: 25635.21339
Best fitness: 27.67263 - size: (12, 114) - species 9 - id 895896
Average adjusted fitness: 0.981
Mean genetic distance 2.942, standard deviation 0.576
Population of 1000 members in 8 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
     2  4990   125   -122.0    0.976  1586
     3  4990   124   -229.2    0.980   586
     8  4971   125    -66.7    0.985  1209
     9  4969   125     27.7    0.980  2673
    10  4969   126    -43.8    0.983   920
    11  4957   125   -184.7    0.984   722
    14  4857   125   -186.2    0.976   108
    16  4752   125    -38.1    0.984   620
Total extinctions: 0
Generation time: 25.464 sec
```

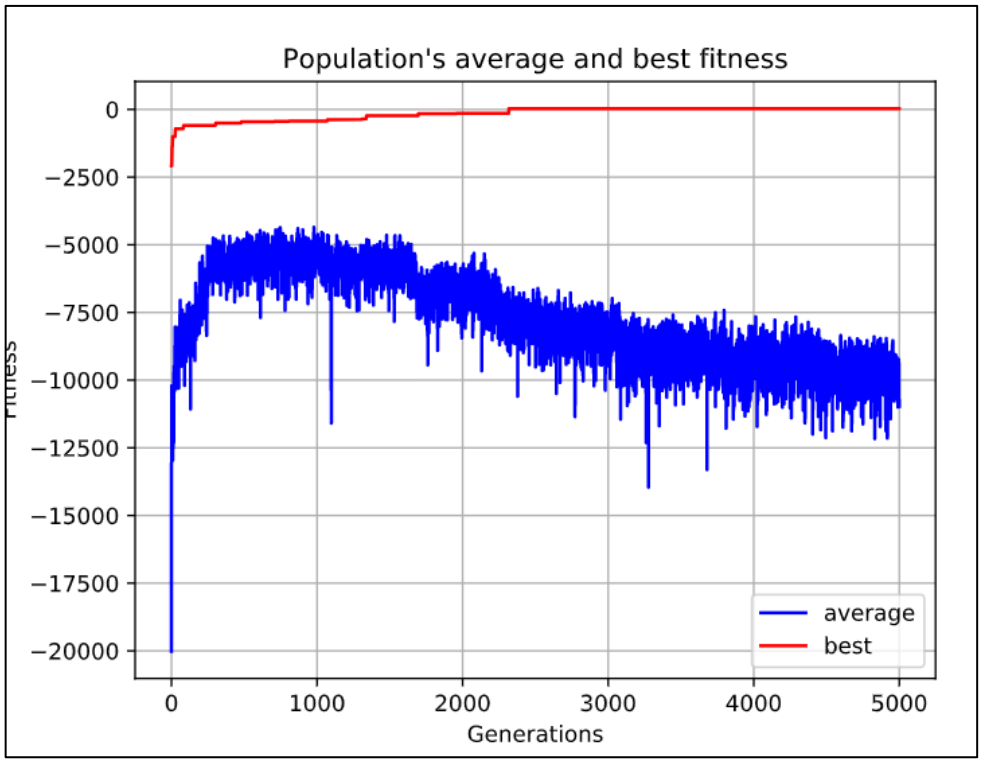
Gambar 5.28 Hasil training dari Tahap 3 Pengujian 4

Ternyata, penggunaan *encoding* posisi pemain secara keseluruhan membawa hasil positif pada proses *training*. Selain lebih singkatnya waktu yang dibutuhkan untuk menyelesaikan satu generasi, nilai fitness terbaik yang dihasilkan, 27.67263, juga melebihi seluruh nilai fitness terbaik yang dihasilan oleh pengujian pada Tahap 3. Proses training ini dijalankan sebanyak 5000 generasi dengan dengan rata-rata waktu yang dibutuhkan sebanyak 25 detik pergenerasi. Grafik *fitness* terbaik selama pengujian dan rata-rata *fitness* dalam populasi dapat dilihat pada Gambar 5.29.

Hasil positif juga berlanjut pada proses *testing*. Akurasi yang dihasilkan pengujian ini juga melampaui semua akurasi yang dihasilkan oleh pengujian lain pada tahap ini, baik dalam akurasi hasil maupun akurasi skor pertandingan. *Network* yang dihasilkan oleh pengujian ini mampu menebak 299 dari 380 pertandingan dengan benar, dimana 131 diantaranya dapat ditebak dengan skor yang tepat.

Tabel 5.10 Rangkuman proses training dan testing pada Tahap 3 Pengujian 4

| | |
|---|---|
| Generasi | 5000 |
| Rata-rata waktu per generasi | 25.464 |
| Total waktu | 2122 |
| *Fitness Function* | *Fitness Function 2* (Segmen Program 4.5) |
| Rata-rata nilai *fitness* dalam populasi | -9819.25758 |
| Nilai *fitness* terbaik dalam populasi | 27.67263 |
| Kompleksitas *network* pada individu dengan nilai *fitness* terbaik | 42 *nodes* dan 114 koneksi aktif |
| Skor pertandingan benar | 131 (34 %) |
| Hasil pertandingan benar | 299 (78.6 %) |




Gambar 5.29 Grafik nilai fitness terbaik dan nilai fitness rata-rata dalam populasi selama proses training pada Tahap 3 Pengujian 4

### 5.2.4 Kesimpulan Pengujian NEAT

Selain *feature*, *fitness function* juga memiliki peranan yang sangat penting pada NEAT. Dapat dilihat pada Tahap 1 Pengujian 1 dan Tahap 1 Pengujian 2, terjadi peningkatan akurasi yang signifikan. Pengujian dilakukan dengan *feature* dan konfigurasi yang sama, namun *fitness function* yang digunakan berbeda. Tingkat akurasi hasil pertandingan dan akurasi skor yang dihasilkan Tahap 1 Pengujian 1, yaitu 28% dan 8% meningkat menjadi 71% dan 22% pada Tahap 1 Pengujian 2.

Selain *fitness function* yang digunakan, konfigurasi juga memiliki peranan penting ketika proses NEAT berjalan. Pentingnya konfigurasi dapat dilihat dari Tahap 1 Pengujian 2 dan Tahap 1 Pengujian 3. Dengan menggunakan *feature* yang sama, yaitu *rating* pemain, terjadi peningkatan akurasi dari 71% untuk akurasi prediksi hasil pertandingan dan 22% untuk akurasi prediksi skor yang dihasilkan

oleh Tahap 1 Pengujian 2, menjadi 78% dan 32% yang dihasilkan oleh Tahap 1 Pengujian 3.

Penambahan 2 *feature* baru, yaitu *rating team* juga ikut meningkatkan tingkat akurasi dari sebuah *network*, walaupun tidak signifikan. Dengan menggukana konfigurasi dan *fitness function* yang sama, akurasi prediksi hasil pertandingan dan akurasi prediksi skor meningkat dari 78% dan 32% yang dihasilkan oleh Tahap 1 Pengujian 3, menjadi 79% dan 35% pada Tahap 2 dan Pengujian 1.

Tingkat akurasi terbaik pada pengujian NEAT dihasilkan pada Tahap 2 Pengujian 2. Dengan sedikit melakukan perubahan konfigurasi yang digunakan pada Tahap 2 Pengujian 1, Tahap 2 Pengujian 2 mampu menghasilkan akurasi yang lebih baik, yaitu sebesar 81% untuk akurasi prediksi hasil pertandingan, dan 42% untuk akurasi prediksi skor.

Pada Tahap 3, posisi pemain ikut menjadi *feature* pada proses NEAT. Dari 3 pengujian yang mengaplikasikan *encoding* kepada posisi masing-masing pemain dan 1 pengujian yang menggunakan total dari posisi pemain pada setiap sektor, hasil terbaik diperoleh pada Tahap 3 Pengujian 4 yang menggunakan total posisi pemain pada tiap sektor. Pengujian ini menghasilkan tingkat akurasi sebesar 78% untuk akurasi hasil pertandingan dan 34% untuk akurasi skor.

Selain tingkat akurasi yang dihasilkan tidak sebaik Tahap 2, waktu yang dibutuhkan untuk menyelesaikan proses *training* juga jauh lebih lama. Pada Tahap 2, waktu yang dibutuhkan untuk menyelesaikan 1 generasi dalam proses *training* tidak pernah melebihi 20 detik. Sedangkan pada Tahap 3, waktu tercepat yang dibutuhkan untuk menyelesaikan 1 generasi adalah sekitar 25 detik.

Berdasarkan beberapa pengujian yang dilakukan dengan berbagai *features*, *fitness function*, dan konfigurasi yang berbeda, dapat disimpulkan bahwa pengujian yang menghasilkan tingkat akurasi terbaik adalah Tahap 2 Pengujian 3 dengan *features* berupa *rating* pemain dan *rating team*. Rangkuman pengujian dapat dilihat pada Tabel 5.11.

Tabel 5.11 Rangkuman seluruh pengujian NEAT

| Pengujian | Akurasi Hasil Pertandingan | Akurasi Skor Pertandingan |
|---|---|---|
| Tahap 1 Pengujian 1 | 107 / 380 (28%) | 32 / 380 (8%) |
| Tahap 1 Pengujian 2 | 273 / 380 (71%) | 85 / 380 (22%) |
| Tahap 1 Pengujian 3 | 300 / 380 (78%) | 125 / 300 (32%) |
| Tahap 2 Pengujian 1 | 302 / 380 (79%) | 136 / 380 (35%) |
| Tahap 2 Pengujian 2 | 305 / 380 (80%) | 136 / 380 (35%) |
| Tahap 3 Pengujian 1 | 260 / 380 (68%) | 90 / 380 (23%) |
| Tahap 3 Pengujian 2 | 194 / 380 (51%) | 47 / 380 (12%) |
| Tahap 3 Pengujian 3 | 223 / 380 (52 %) | 46 / 380 (12%) |
| Tahap 3 Pengujian 4 | 299 / 380 (78.6 %) | 131380 (34 %) |

## 5.3 Pengujian *Backpropagation*

Setelah proses NEAT selesai dijalankan, proses selanjutnya adalah *backpropagation*. Proses *backpropagation* bertujuan untuk mengoptimasi *weight* dari *network* yang dihasilkan oleh NEAT. Hasil dari proses *backpropagation* dapat dilihat pada Tabel 5.12.

Tabel 5.12 Hasil pengujian backpropagation

| Pengujian | *Error* Terakhir | *Epoch* | Akurasi Hasil Pertandingan | Akurasi Skor Pertandingan | Total Waktu |
|---|---|---|---|---|---|
| Tahap 1 Pengujian 1 | 759.55 | 100.000 | 291 / 380 (76%) | 94 / 380 (24%) | 264 detik |
| Tahap 1 Pengujian 2 | 460.110 | 100.000 | 289 / 380 (76%) | 134 / 380 (35%) | 234 detik |
| Tahap 1 Pengujian 3 | 433.55 | 44.000 | 301 / 380 (79%) | 155 / 380 (40 %) | 181 detik |

| | | | | | |
|---|---|---|---|---|---|
| Tahap 2 Pengujian 1 | 351.01 | 100.000 | 303 / 380 (79 %) | 167 / 380 (44 %) | 355 detik |
| Tahap 2 Pengujian 2 | 395.55 | 100.000 | 308 / 380 (81%) | 161 / 380 (42%) | 482 detik |
| Tahap 3 Pengujian 1 | 386.29 | 100.000 | 299 / 380 (78%) | 153 / 380 (40%) | 789 detik |
| Tahap 3 Pengujian 2 | 492.47 | 100.000 | 306 / 380 (80%) | 122 / 380 (32%) | 2269 detik |
| Tahap 3 Pengujian 3 | 469.00 | 100.000 | 304 / 380 (80%) | 162 / 380 (42%) | 1199 detik |
| Tahap 3 Pengujian 4 | 326.57 | 100.000 | 310 / 380 (81.5%) | 185 / 380 (48%) | 656 detik |

Semua pengujian *backpropagation* dijalankan dengan *learning rate* sebesar 0.0001 dan selama 100.000 *epoch*, kecuali untuk Tahap 1 Pengujian 3 karena terjadinya *overfitting* pada *epoch* 44.000 – 45.000 yang menyebabkan proses *backpropagation* terpaksa dihentikan.

Berdasarkan pengujian-pengujian yang telah dilakukan, proses *backpropagation* terbukti mampu meningkatkan tingkat akurasi pada *network* yang dihasilkan oleh NEAT pada penelitian ini dengan rata-rata peningkatan akurasi sebesar 15,25% pada akurasi hasil pertandingan, dan 12,625% pada akurasi skor pertandingan.

Peningkatan paling signifikan pada *network* setelah dilakukannya *backpropagation* terjadi pada Tahap 1 Pengujian 1. *Network* yang dihasilkan oleh Tahap 1 Pengujian 1 pada awalnya hanya memiliki akurasi sebesar 28% dan 8%. Namun setelah proses *backpropagation* diaplikasikan pada pengujian tersebut, tingkat akurasi mengalami kenaikkan menjadi 76% dan 24%. Hal ini membuktikan bahwa walaupun tingkat akurasi yang dihasilkan oleh *network* dari proses NEAT rendah, struktur yang dimiliki *network* itu sudah cukup baik.

Perbandingan tingkat akurasi sebelum dan sesudah dilakukannya *backpropagation* dapat dilihat pada Tabel 5.13.

Tabel 5.13 Perbandingan akurasi dari network sebelum dan sesudah backpropagation

| Pengujian | Sebelum *Backpropagation* | | Sesudah *Backpropagation* | |
|---|---|---|---|---|
| | Akurasi Hasil | Akurasi Skor | Akurasi Hasil | Akurasi Skor |
| Tahap 1 Pengujian 1 | 107 / 380 (28%) | 32 / 380 (8%) | 291 / 380 (76%) | 94 / 380 (24%) |
| Tahap 1 Pengujian 2 | 273 / 380 (71%) | 85 / 380 (22%) | 289 / 380 (76%) | 134 / 380 (35%) |
| Tahap 1 Pengujian 3 | 300 / 380 (78%) | 125 / 300 (32%) | 301 / 380 (79%) | 155 / 380 (40 %) |
| Tahap 2 Pengujian 1 | 302 / 380 (79%) | 136 / 380 (35%) | 303 / 380 (79 %) | 167 / 380 (44 %) |
| Tahap 2 Pengujian 2 | 305 / 380 (80%) | 136 / 380 (35%) | 308 / 380 (81%) | 161 / 380 (42%) |
| Tahap 3 Pengujian 1 | 260 / 380 (68%) | 90 / 380 (23%) | 299 / 380 (78%) | 153 / 380 (40%) |
| Tahap 3 Pengujian 2 | 194 / 380 (51%) | 47 / 380 (12%) | 306 / 380 (80%) | 122 / 380 (32%) |
| Tahap 3 Pengujian 3 | 223 / 380 (52 %) | 46 / 380 (12%) | 304 / 380 (80%) | 162 / 360 (42%) |
| Tahap 3 Pengujian 4 | 299 / 380 (78.6 %) | 131 / 380 (34 %) | 310 / 380 (81.5%) | 185 / 380 (48%) |

Setelah *backpropagation* selesai dijalankan, pengujian yang menghasilkan *network* dengan akurasi terbaik berubah. Tahap 3 Pengujian 4 mengalahkan Tahap 2 Pengujian 2 yang sebelumnya menjadi pengujian dengan akurasi terbaik.

Akurasi yang dihasilkan oleh Tahap 3 Pengujian 4 meningkat dari yang sebelummya 78% untuk akurasi hasil pertandingan dan 34% akurasi skor

pertandingan, menjadi 81% dan 48%. Akurasi yang dihasilkan oleh pengujian ini mengalahkan semua akurasi dari semua pengujian yang telah dilakukan.

## 5.4 Pengujian *Real Life*

Pada data pertandingan *real*, tidak diketahui berapa *rating* pemain yang akan bermain pada pertandingan tersebut. Untuk memprediksi *rating* pemain yang akan bermain, akan dicoba beberapa cara, yaitu rata-rata 5 pertandingan terakhir, rata-rata 10 pertandingan terakhir.

Untuk rata-rata 5 dan 10 pertandingan terakhir, akan diambil *rating* dari pemain dan *team* pada 5 dan 10 pertandingan terakhir sebelum pertandingan akan dicoba untuk diprediksi. Sebagai contoh, pada pertandingan ke 20, akan dihitung rata-rata pada pertandingan ke-14 sampai ke-19 atau pertandingan ke-9 sampai pertandingan ke-19. Jika data yang tersedia kurang dari 5 atau 10, akan dihitung rata-rata dari jumlah pertandingan yang tersedia dalam rentang 5 atau 10 pertandingan terakhir. Jika seorang pemain tidak pernah bermain pada 5 atau 10 pertandingan terakhir, akan diberi *rating* 0.55.

Selain itu, akan diuji juga rata-rata 5 dan 10 pertandingan terakhir yang sudah dinormalisasi. Normalisasi dilakukan dengan cara mengalikan *rating* pemain dengan *rating team* lawan, lalu dibagi dengan *rating team* dari pemain itu sendiri.

Agar tersedianya cukup data, pengujian *real life* dimulai dengan pertandingan dari pertandingan ke-10 untuk rata-rata 5 pertandingan terakhir, dan pertandingan ke-15 untuk rata-rata 10 pertandingan terakhir.

Pengujian *real life* dilakukan dengan *network* hasil pengujain *backpropagation* yang memiliki tingkat akurasi tertinggi, yaitu *network* dari Tahap 3 Pengujian 4.

Tabel 5.14 Hasil pengujian real life

| Metode Prediksi Rating | Akurasi Hasil Pertandingan | Akurasi Skor Pertandingan |
|---|---|---|
| Rata-rata 5 pertandingan terakhir | 124 / 280 (44%) | 28 / 280 (10%) |

| | | |
|---|---|---|
| Rata-rata 5 pertandingan terakhir dengan normalisasi | 78 / 280 (27.8%) | 25 / 280 (8.9%) |
| Rata-rata 10 pertandingan terakhir | 92 / 230 (40%) | 21 / 230 (9%) |
| Rata-rata 10 pertandingan terakhir dengan normalisasi | 63 / 230 (26.5%) | 20 / 230 (8.6%) |

Seperti yang dapat dilihat pada Tabel 5.14., hasil pengujian *real life* sangat memiliki akurasi yang sangat rendah jika dibandingkan dengan hasil pengujian *backpropagation* maupun hasil pengujian NEAT yang menggunakan *rating* yang sebenarnya. Kedua metode yang digunakan untuk melakukan pengujian ini memperoleh tingkat akurasi yang hampir sama, yaitu 44% dan 10% untul rata-rata 5 pertandingan terkahir, 27.8% dan 8.9% jika dilakukan dengan normalisasi dan 40% dan 9% untuk rata-rata 10 pertandingan terakhir, 26.5% dan 8.6% jika dilakukan dengan normalisasi. Hal ini disebabkan oleh *rating* pemain yang tidak konsisten pada setiap pertandingan sehingga sulit untuk diprediksi menggunakan rata-rata.

## 5.5 Kesimpulan Pengujian

Setelah semua pengujian selesai dijalankan, dapat disimpulkan bahwa *features* terbaik pada pengujian NEAT adalah *rating* pemain disertai dengan *rating team* yang terjadi pada Tahap 2 Pengujian 2 dengan tingkat akurasi sebesar 80% untuk akurasi hasil pertandingan, dan 35% untuk akurasi skor. Penggunaan rating pemain dan *rating* team sedikit meningkatkan tingkat akurasi jika *feature* yang digunakan hanya *rating* pemain.

Penambahan posisi pemain sebagai *features* pada NEAT justru menimbulkan efek negatif. Selain tingkat akurasi yang lebih rendah jika dibandingkan dengan penggunaan *features* lainnya, waktu *training* yang dibutuhkan juga jauh lebih lama.

Dalam beberapa kasus, struktur yang dihasilkan oleh NEAT sudah cukup baik, tetapi *weight* yang tidak optimal menyebabkan tingkat akurasi pada pengujian menjadi rendah. Hal ini dibuktikan dengan peningkatan akurasi yang signifikan pada *network* yang memiliki tingkat akurasi rendah ketika proses *backpropagation* dijalankan.

*Backpropagation* terbukti mampu meningkatkan tingkat akurasi pada semua *network* yang dihasilkan oleh NEAT. Pada *network* yang sudah memiliki tingkat akurasi tinggi, memang optimasi *backpropagation* tidak terlalu signifikan. Tetapi pada *network* yang memiliki akurasi rendah, *backpropagation* mampu mengoptimasi *weight* sehingga terjadi peningkatan akurasi yang signifikan. Peningkatan akurasi terbesar terjadi pada Tahap 1 Pengujian 1, dimana akurasi hasil pertandingan meningkat dari 28% menjadi 76%.

Setelah proses *backpropagation* selesai, Tahap 3 Pengujian 4 yang menambahkan total posisi tiap pemain pada berbagai sektor keluar sebagai pengujian yang menghasilkan tingkat akurasi terbaik, mengalahkan Tahap 2 Pengujian 2 yang sebelumnya memiliki tingkat akurasi terbaik. Hal ini kembali membuktikan bahwa adanya kemungkinan *weight* yang dihasilkan NEAT tidak optimal.

Sayangnya, akurasi yang didapat pada pengujian *real life* tidak sebaik akurasi yang dihasilkan pada proses *testing*. Karena pada pengujian *real life rating* pemain maupun *rating team* tidak diketahui, sehingga harus diprediksi terlebih dulu. Penggunaan metode rata-rata untuk menentukan *rating* pemain dan *rating* team gagal menghasilkan akurasi yang baik. Inkonsistensi dari pemain pada satu pertandingan ke pertandingan lainnya diyakini menjadi penyebab rendahnya akurasi yang dihasilkan.

# 6. KESIMPULAN DAN SARAN

Pada bab ini, akan dijabarkan kesimpulan yang diperoleh dari penelitian untuk melakukan prediksi pertandingan sepak bola menggunakan *neuroevolution of augmenting topologies* beserta saran untuk pengembangan skripsi lebih lanjut.

## 6.1 Kesimpulan

Dari hasil seluruh pengujian yang telah selesai, dapat diambil kesimpulan sebagai berikut, antara lain:

- *Features* terbaik pada proses NEAT adalah *rating* pemain dengan *rating team*.
- Penambahan *rating* team sebagai *features* sedikit meningkatkan tingkat akurasi yang dihasilkan oleh NEAT jika dibandingkan *rating* pemain saja yang menjadi *feature.*
- Akurasi tertinggi yang dihasilkan oleh NEAT didapapat pada Tahap 2 Pengujain 2 dengan tingkat akurasi mencapai 80% pada prediksi hasil pertandingan dan 35% pada prediksi skor pertandingan.
- Penambahan posisi pemain sebagai *features* membuat proses *training* pada NEAT menjadi jauh lebih lama dan tingkat akurasi yang dihasilkan juga lebih rendah jika dibandingkan dengan *rating* pemain atau *rating* pemain dan *rating team* yang digunakan sebagai *features*.
- Penggunaan *encoding* pada posisi setiap pemain pada Tahap 3 justru memberikan efek negatif, yaitu akurasi yang lebih rendah dan waktu *training* yang lebih lama.
- Penggunaan jumlah posisi pemain pada tiap sektor yang dilakukan pada Tahap 3 Pengujian 4 menghasilkan tingkat akurasi yang lebih baik dan waktu *training* yang lebih singkat daripada melakukan *encoding* pada masing-masing posisi pemain.
- Dalam beberapapa kasus, struktur *network* yang dihasilkan NEAT sudah cukup baik, namun *weight* yang tidak optimal menyebabkan tingkat akurasi yang rendah. Hal ini dibuktikan oleh proses *backpropagation.*

- *Backpropagation* mampu meningkatkan akurasi pada semua pengujian yang dilakukan.
- Akurasi tertinggi yang dihasilkan setelah *backpropagation* berasal dari Tahap 3 Pengujian 4, dengan tingkat akurasi sebesar 81.5% pada prediksi hasil pertandingan, dan 48% pada prediksi skor pertandingan.
- Tahap 3 Pengujian 4, yang keluar sebagai pengujian dengan akurasi terbaik, menggunakan *rating team*, *rating* pemain, dan jumlah dari posisi pemain dari setiap sektor (*defender*, *midfielder*, dan *striker*).
- Peningkatan akurasi terbaik pada setelah proses *backpropagation* dilakukan terjadi pada Pengujian 1 Tahap 1, dimana akurasi hasil pertandingan meningkat dari 28% menjadi 76%.
- *Rating* pemain yang tidak konsisten menyebabkan pengujian *real life* yang menggunakan metode pencarian *rating* dengan rata-rata menghasilkan tingkat akurasi yang rendah.

## 6.2 Saran

Saran yang dapat diberikan untuk mengembangkan skripsi ini lebih lanjut antara lain:

- Penambahan dataset dari berbagai liga agar akurasi yang didapat bisa lebih tinggi lagi.
- Gunakan metode lain untuk memprediksi *rating* pemain, seperti *linear regression*.

## DAFTAR REFERENSI


Aggarwal, C. C. (2018). *Neural networks and deep learning: a textbook.* Cham: Springer Nature.

Boudway, I. (2018). *Soccer Is the World's Most Popular Sport and Still Growing.* Retrieved September 3, 2019, from Bloomberg: https://www.bloomberg.com/news/articles/2018-06-12/soccer-is-the-world-s-most-popular-sport-and-still-growing

Brownlee, J. (2017, July 12). *How to One Hot Encode Sequence Data in Python*. Retrieved September 9, 2019, from Machine Learning Mastery: https://machinelearningmastery.com/how-to-one-hot-encode-sequence-data-in-python/

Burkov, A. (2019). *The hundred-page machine learning book.*

Chen, D., Giles, C., Sun, G., Chen, H., Lee, Y., & Goudreau, M. (1993). Constructive learning of recurrent neural networks. *IEEE International Conference on Neural Networks.* IEEE. doi:10.1109/ICNN.1993.298727

Hurwitz, J., & Kirsch, D. (2018). *Machine Learning for Dummies, IBM Limited Edition.* John Wiley & Sons, Inc.

Igiri, C. (2015). Support Vector Machine–Based Prediction System for a Football Match. *IOSR-JCE*.

Mallawaarachchi, V. (2017). *Introduction to Genetic Algorithms — Including Example Code*. Retrieved September 7, 2019, from Towards Data Science: https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3

Morse, G., & Stanley, K. (2016). Simple Evolutionary Optimization Can Rival Stochastic Gradient Descent in Neural Networks. *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference - GECCO '16*. doi:10.1145/2908812.2908916

Negnevitsky, M. (2005). *Artificial Intelligence A Guide to Intelligent System.* Addison-Wesley Publishing Company, Inc.

Pappalardo, L., & Cintia, P. (2018). Quantifying the relation between performance and success in soccer. *Advances in Complex Systems, 21*(03n04). doi:10.1142/s021952591750014x

Simeone, O. (2018). A Very Brief Introduction to Machine Learning With Applications to Communication Systems. *IEEE Transactions on Cognitive Communications and Networking*, 648-664. doi:10.1109/tccn.2018.2881442

Stanley, K. O., & Miikkulainen, R. (2002). Evolving Neural Networks through Augmenting Topologies. *Evolutionary Computation, 10*, 99-127. doi:10.1162/106365602320169811

Tierney, J. (2014). *Soccer, a Beautiful Game of Chance*. Retrieved September 3, 2019, from New York Times: https://www.nytimes.com/2014/07/08/science/soccer-a-beautiful-game-of-chance.html

Zhang, A., Lipton, Z. C., & Mu Li, A. J. (2019). *Dive into Deep Learning.*