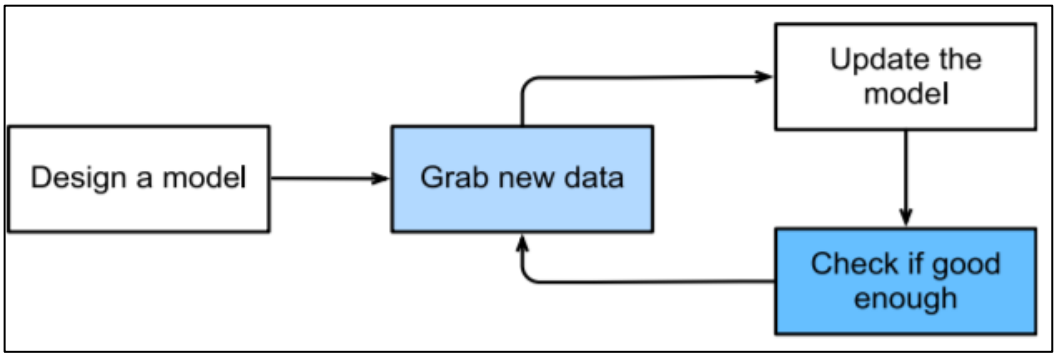
2. LANDASAN TEORI

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

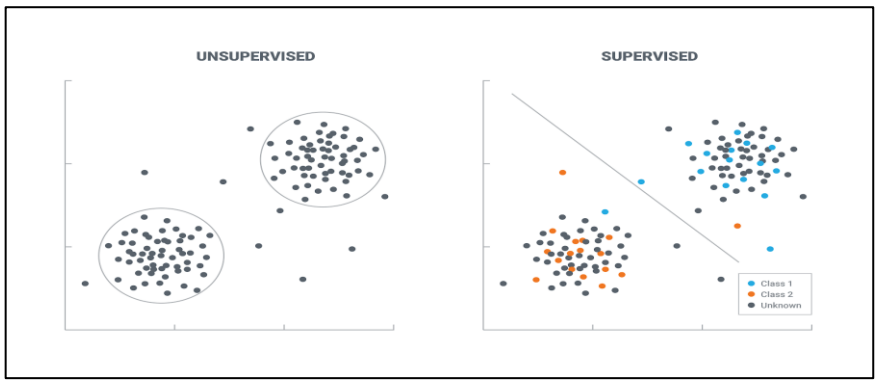
2.1 *Machine Learning*



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 (Burkov, 2019). 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.



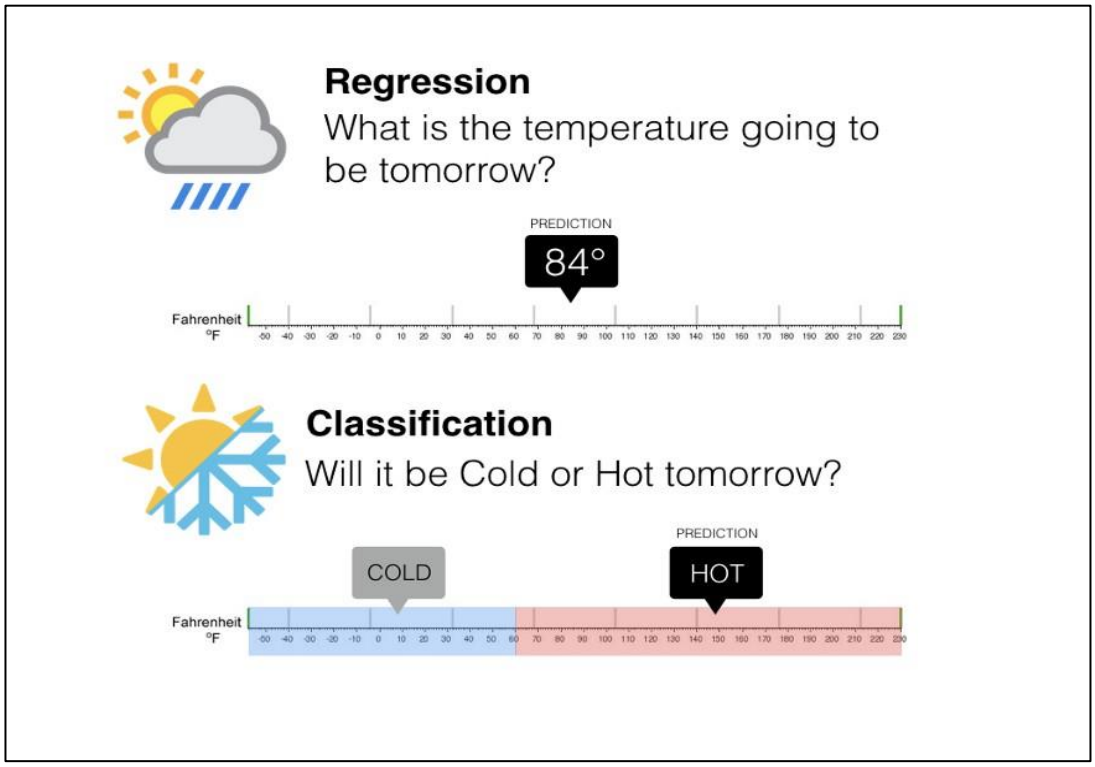
Gambar 2.2 Ilustrasi supervised learning dan unsupervised learning

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/

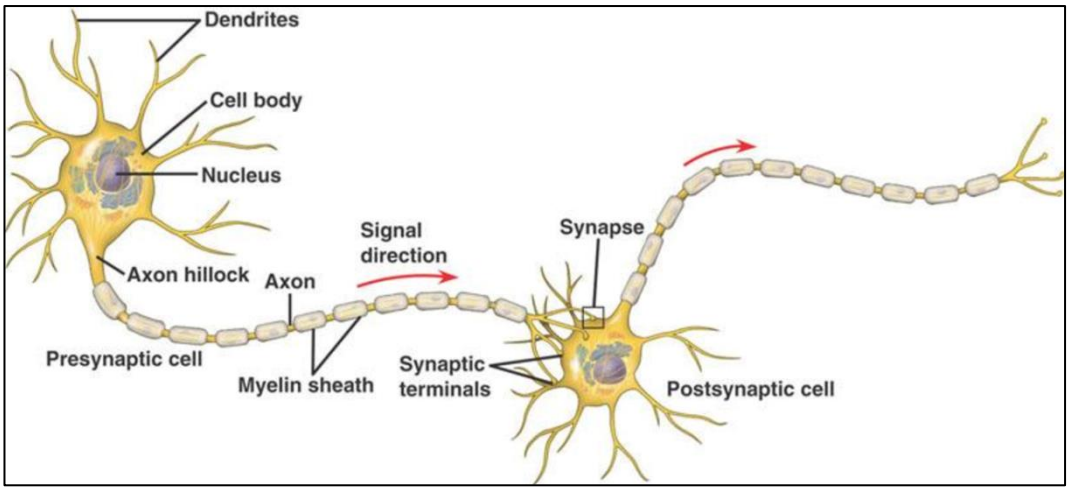
Supervised Learning memiliki tujuan untuk melakukan sebuah prediksi dari input data yang diberikan. Hasil prediksi dari *supervised learning*, yang biasa disebut label, dapat dinotasikan sebagai y, Sedangkan input data, yang biasa disebut *examples* juga dinotasikan sebagai x. Tujuan dari *supervised 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).

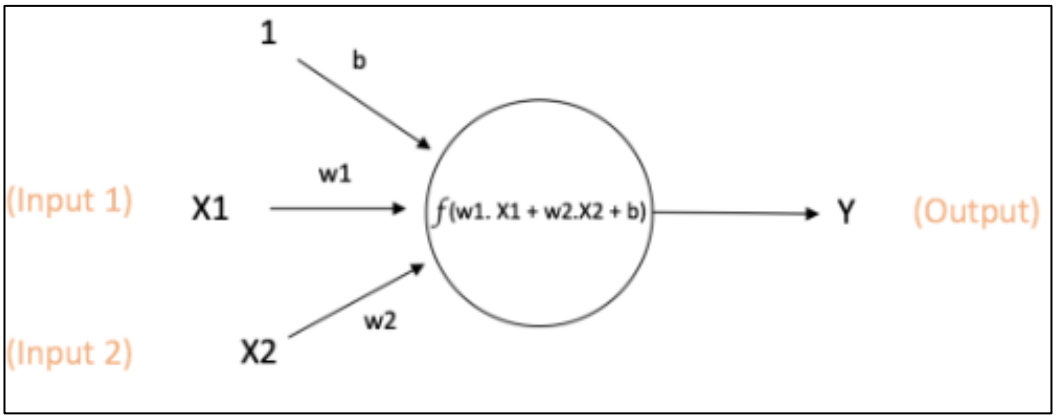
2.2 Jaringan Saraf Tiruan

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

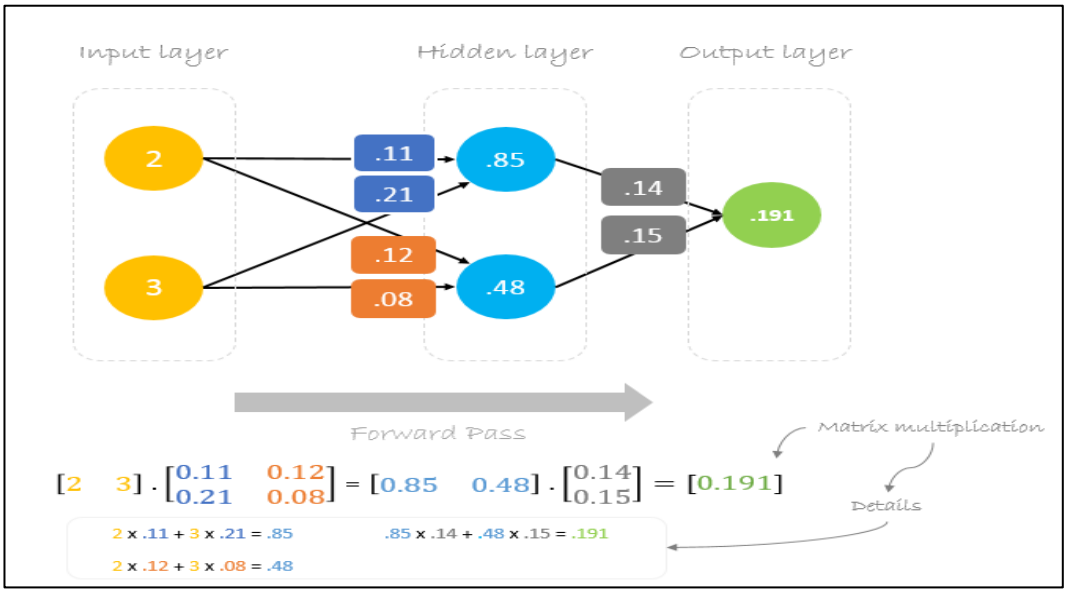
Sumber: https://www.semanticscholar.org/paper/An-Introduction-to-Artificial-Neural-Networks-(-ANN-Heger/7524e25abbdea2f982eb673ca1e60773c118cd66/figure/1



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.*

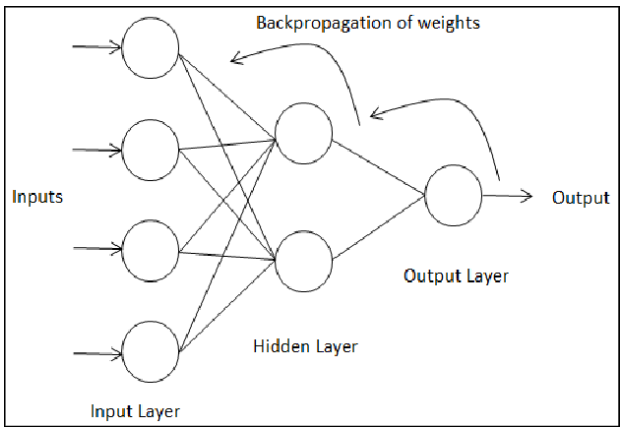


Gambar 2.6 Ilustrasi *forward-pass* pada *artificial neural network*

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 (Aggarwal, 2018).

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, θ 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 α 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).

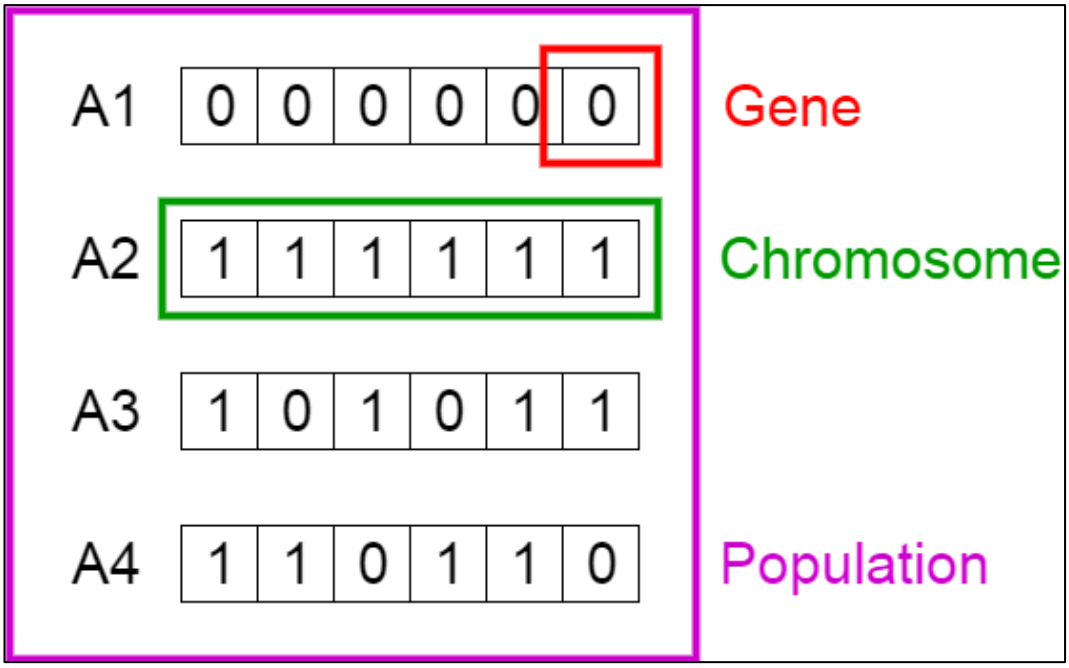
Proses seleksi alam dalam GA dimulai dengan melakukan seleksi dalam sebuah populasi yang bertujuan untuk menemukan individu-individu yang terbaik. Individu-individu terbaik tersebut akan dipilih menjadi *parents* untuk melakukan reproduksi. *Crossover* dilakukan agar terciptanya keterunan yang mewarisi 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

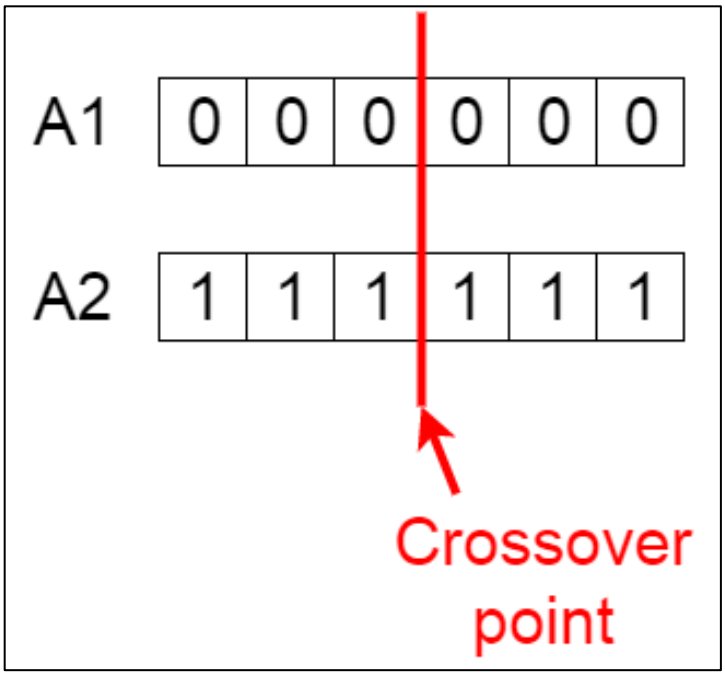
- ***Selection***

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***

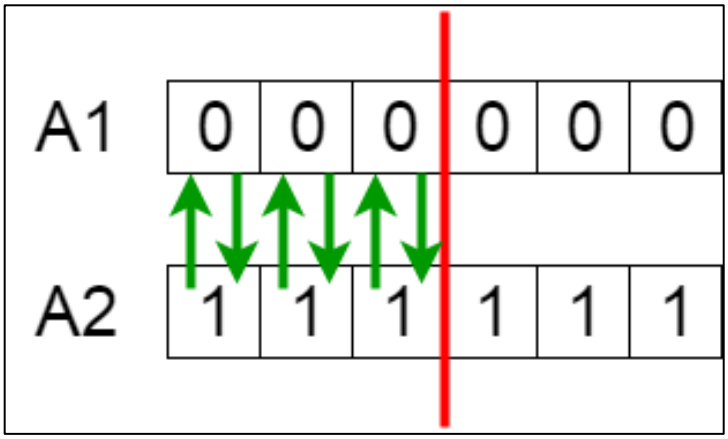
Crossover merupakan bagian terpenting dalam GA. Tujuan dari *crossover* terciptanya keturunan atau *offspring* yang memiliki karakteristik yang mirip dengan *parents*nya. Setelah dipilih 2 individu yang akan berperan sebagai *parents*, akan dipilih titik *crossover* pada *genes* secara acak.

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



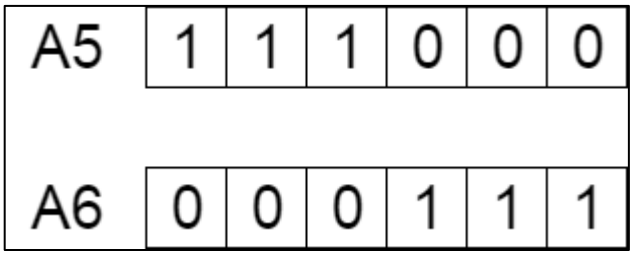
Gambar 2.9 Ilustrasi proses *crossover* pada GA

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

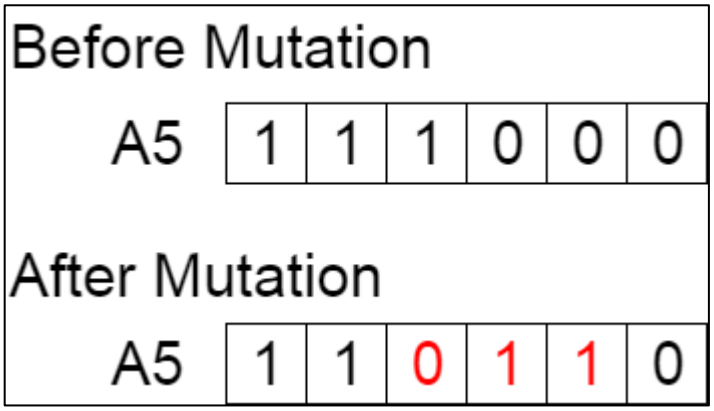


Gambar 2.11 Ilustrasi proses *crossover* pada GA

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

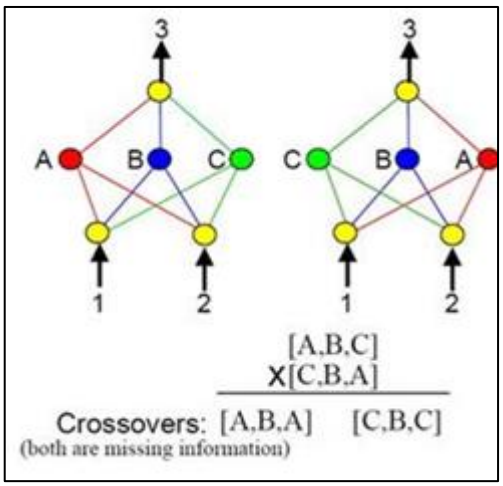
Kelima proses diatas akan dijalankan secara iteratif hingga solusi ditemukan atau mengalami kondisi yang dinamakan *converged*, yaitu suatu kondisi dimana *offsprings* yang dihasilkan tidak memiliki perbedaan yang signifikan dengan generasi sebelumnya.

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).

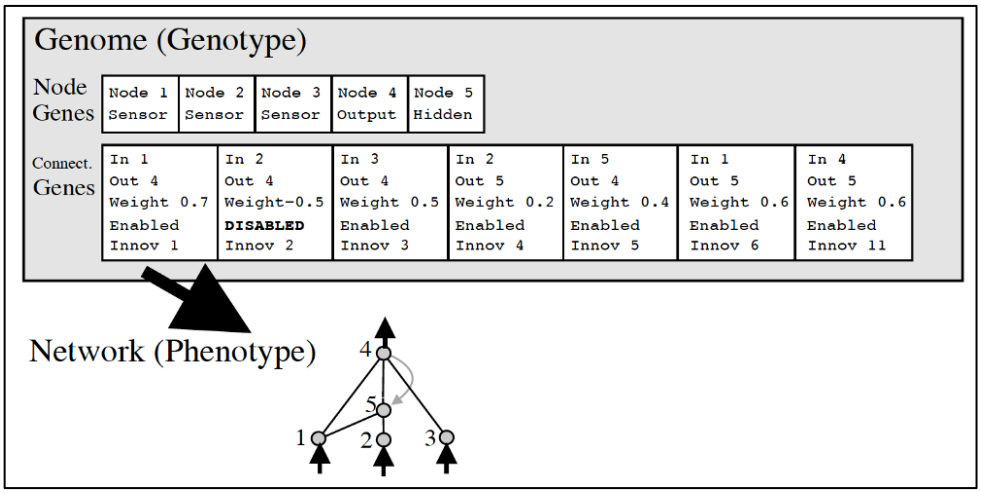
Namun, bobot-bobot yang ada pada ANN bukan penentu satu-satunya terhadap *behaviour* dari sebuah ANN. Topologi atau struktur dari ANN itu sendiri juga mempengaruhi bagaimana ANN bekerja (Chen, et al., 1993). Selain itu, ketika terjadi *crossover* pada tradisional NE, adanya kemungkinan informasi akan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

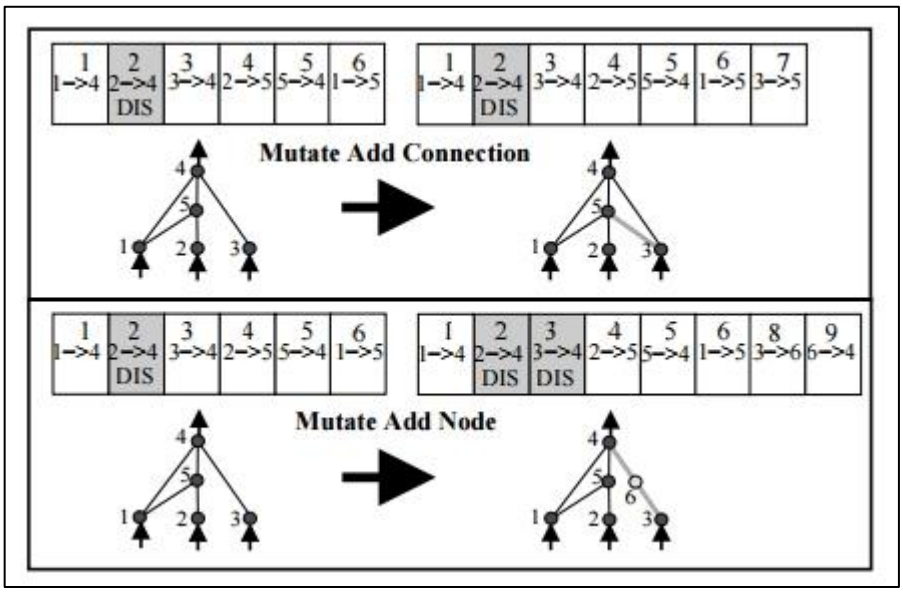
Sumber: Stanley & Miikkulainen. Evolving Neural Networks through Augmenting Topologies. 2002

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

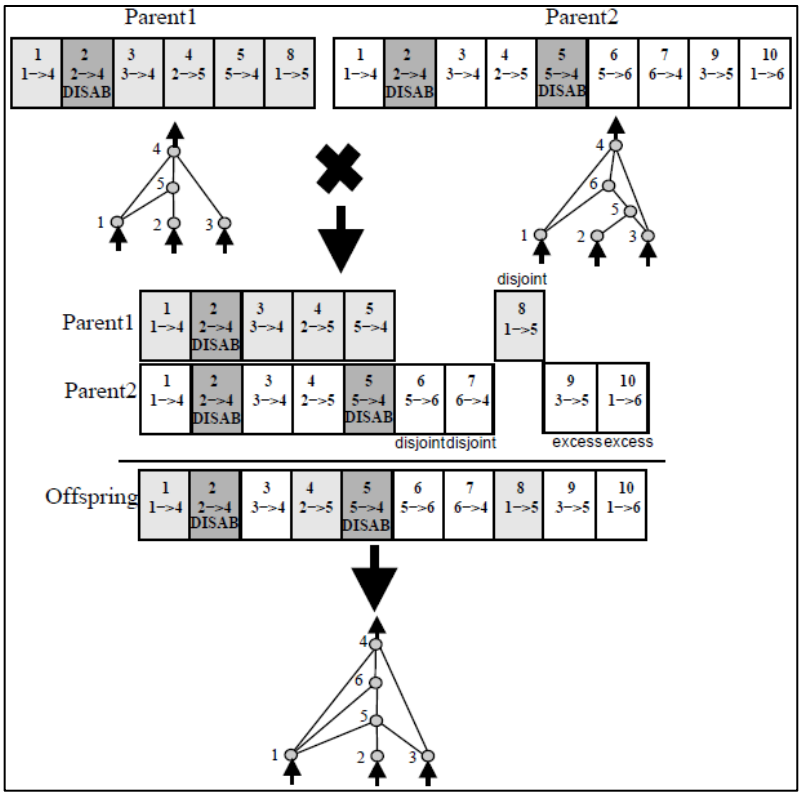
Sumber: Stanley & Miikkulainen. Evolving Neural Networks through Augmenting Topologies. 2002

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* δ) *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* δ, dapat dilakukan pengelompokkan menggunakan *compability threshold* δ_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 δ 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* δ_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* δ_t. Setiap spesies

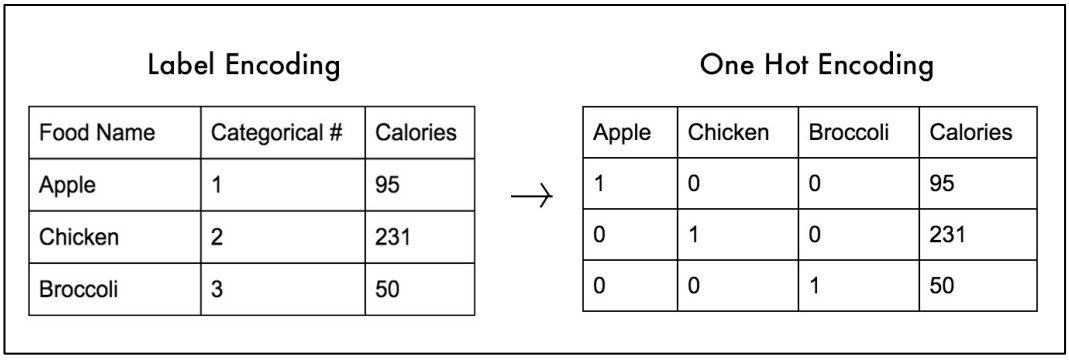
dapat menciptakan jumlah *offsprings* yang kemungkinan berbeda-beda berdasarkan jumlah total dari penyesuaian *fitness* f_i'. Spesies kemudian akan melakukan reproduksi, dimulai dengan mengeliminasi anggota yang memiliki performa terburuk. Hasil dari reproduksi inilah yang akan mengisi populasi yang 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*.