BAB 2

LANDASAN TEORI

2.1 Teori-teori Dasar / Umum

2.1.1 Genetic Algorithm

Genetic algorithm adalah suatu algoritma yang biasanya digunakan untuk mencari solusi-solusi yang optimal untuk berbagai masalah yang sulit (Matic, 2010), misalnya masalah optimasi, *traveling salesperson problem*, dan *learning*. Algoritma ini menggunakan mekanisme seleksi alamiah dan genetika alamiah yang dikenal dalam dunia ilmu biologi, yaitu teori "Survival of the Fittest" yang dicetuskan oleh Charles Darwin. Dari kedua hal tersebut muncul istilah-istilah seperti gen, kromosom, populasi, *crossover*, mutasi, seleksi, dan *fitness* (Malhotra, 2011).

Dalam *genetic algorithm*, setiap solusi yang ada dari masalah akan dikonversikan atau dikodekan menjadi gen-gen yang membentuk sebuah kromosom. Jika di dalam ilmu biologi dikenal kode-kode A, C, T, dan G untuk merepresentasikan gen, maka dalam algoritma ini bisa digunakan bilangan biner, bilangan diskrit desimal, ataupun bilangan real sebagai kodenya. Misalkan untuk masalah *traveling salesperson problem* yang memiliki solusi berupa urutan kota yang harus dikunjungi, maka kromosomnya bisa berupa deretan bilangan diskrit desimal yang menyatakan jarak dari kota yang satu ke kota yang lainnya.

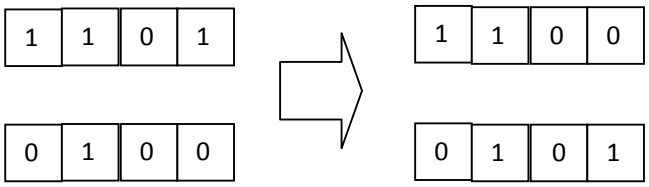
Setelah kromosom-kromosom terbentuk, maka sekarang terbentuk jugalah populasi, yaitu kumpulan dari kromosom-kromosom yang ada.

Selanjutnya dilakukan proses iterasi untuk menemukan solusi global yang optimal, menggunakan pseudocode sebagai berikut:

```
Bangkitkan populasi awal, P individu
Loop untuk P individu
    Dekodekan individu
    Evaluasi Individu
End
Loop sampai kondisi berhenti
    Pilih dua individu sebagai parent1 dan parent2
    If perlu crossover then
        Offspring = crossover(parent1, parent2)
    End
    If perlu mutasi then
        Offspring = mutation(offspring)
    End
    If offspring lebih baik then
        Population = replacement(population, offspring)
    End
End
```

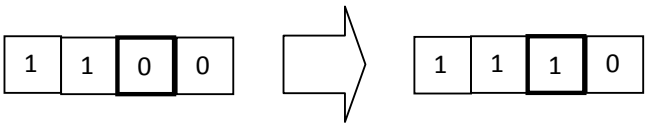
Genetic algorithm dipercaya hampir selalu dapat menemukan solusi yang optimal karena ada proses seleksi solusi yang optimal dimulai dari perkawinan silang (*crossover*) antara individu-individunya, biasanya individu-individu terbaik yang dipilih menjadi orang tua, sehingga menghasilkan anak (*offspring*) yang juga memiliki kualitas yang baik. Dari dua orang tua, misalnya 1101 dan 0100, yang melakukan *crossover* pada titik

tertentu, misalnya titik tengah, akan menghasilkan *offspring*, misalnya 1100 dan 0101.



Gambar 2.1 Proses *Crossover*

Jika diperlukan bisa juga dilakukan mutasi, misalnya pada deret ketiga dari anak pertama, sehingga menjadi 1110.



Gambar 2.2 Proses Mutasi

Kemudian *offspring* siap dimasukkan ke dalam populasi dan menggantikan dua anggota populasi yang nilai *fitness*-nya lebih kecil dibandingkan *offspring*. Nilai *fitness* adalah nilai yang didapat dari suatu perhitungan, cara perhitungan disesuaikan dengan masalahnya, yang menentukan kualitas keoptimalan dari sebuah solusi. Selanjutnya dilakukan iterasi proses seleksi ini sampai kondisi berhenti yang diinginkan dan dianggap sudah mencapai solusi yang optimal, misalnya berdasarkan jumlah iterasi, batasan waktu, atau ada tidaknya penggantian individu di populasi (Suyanto, 2007: 205).

Ketika komponen-komponen dari *genetic algorithm* dipilih dengan baik, proses reproduksi akan meningkatkan kualitas populasi secara berlanjut,

yang pada akhirnya akan menjadi suatu solusi yang mendekati *global optimum*. *Genetic algorithm* dapat dengan efisien mencari dalam skala besar dan rumit (memiliki banyak *local optima*) untuk menemukan solusi yang mendekati *global optima*.

Genetic algorithm tidak memiliki masalah yang sama dengan skala seperti *back propagation*. Salah satu alasannya adalah *Genetic algorithm* selalu mengimprovisasi kandidat terbaik sementara mereka secara monoton. Mereka melakukan ini dengan menyimpan kandidat terbaik sementara mereka di dalam populasi dan secara bersamaan juga mencari kandidat yang lebih baik (Montana, 2005).

2.1.2 Neural Network

2.1.2.1 Pengertian Neural Network

Neural Network atau lebih sering disebut dengan *Artificial Neural Network* (Jaringan Saraf Tiruan) adalah sebuah sistem pemrosesan informasi yang memiliki karakteristik serupa dengan jaringan neural biologis manusia. Sistem ini memproses informasi setelah dilakukan penyesuaian *weight* melalui pelatihan dari data yang sudah ada.

Jaringan Saraf Tiruan memiliki fase pelatihan untuk memperbaiki *weight* koneksi akan disesuaikan dengan data-data yang sudah ada. Dengan kata lain, JST ‘belajar’ dari contoh-contoh seperti seorang anak kecil yang belajar membedakan anjing dan kucing dari kumpulan contoh anjing dan kucing. Jika dilatih dengan baik, maka JST dapat menampilkan kemampuan generalisasi yang melebihi data pelatihan untuk menghasilkan keluaran yang tepat untuk kasus baru yang tidak terdapat pada saat pelatihan.

Haykin (1994: 24) mengatakan bahwa sebuah Jaringan Saraf Tiruan adalah sebuah prosesor yang terdistribusi secara massal yang memiliki kecenderungan alami untuk menyimpan *experiential knowledge* dan membuatnya dapat digunakan kembali. Hal ini mencerminkan otak manusia dalam dua hal:

1. *Knowledge* didapatkan dari hasil proses pembelajaran.
2. Kekuatan dari koneksi inter-neuron atau disebut juga *weight* sinapsis, digunakan untuk menyimpan *knowledge*.

2.1.2.2 Sejarah Neural Network

Artificial Neural Network lahir setelah McCulloc dan W.H.Pitts, pada tahun 1943, memperkenalkan pemodelan matematis neuron. Lalu pada tahun 1949, Hebb mencoba mengkaji proses pembelajaran yang dilakukan oleh neuron yang dikenal sebagai Hebbian Law. Tahun 1958, Rosenblatt memperkenalkan konsep perseptron, suatu jaringan yang terdiri dari beberapa lapisan yang saling berhubungan melalui umpan maju (*feed forward*). Pada tahun 1962, beliau membuktikan bahwa bila setiap perseptron dapat memilah-milah dua buah pola yang berbeda maka siklus pelatihannya dapat dilakukan dalam jumlah yang terbatas. Teorema ini dikenal juga sebagai *perceptron convergence theorem.*

Pada tahun 1960, Widrow dan Hodd menemukan ADALINE (*Adaptive Linear Neuron*). ADALINE mampu beradaptasi dan beroperasi secara linier. Penemuan ini telah memperluas aplikasi jaringan syaraf tiruan, tidak hanya untuk pemilihan pola saja, namun juga untuk pengiriman sinyal khususnya dalam bidang *adaptive filtering*.

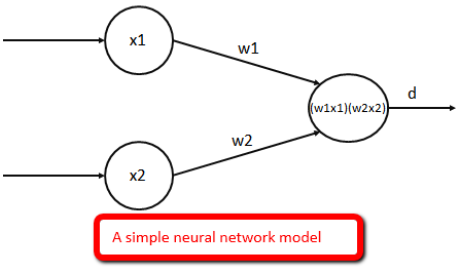
Pada tahun 1969, konsep perseptron yang diperkenalkan oleh Rosenblatt mendapat kritikan dari Minsky dan Papert atas kelemahannya dalam memilah pola-pola yang tidak linier. Sejak saat itu, penelitian mengenai jaringan syaraf tiruan terhenti selama satu dasawarsa.

Hopfield, pada tahun 1982, berhasil memperhitungkan fungsi energi ke dalam jaringan syaraf agar dapat mengingat atau memperhitungkan suatu obyek dengan obyek yang telah dikenal / diingat sebelumnya (*associative memory*). Hasil penelitian Hopfield ini sering disebut sebagai *recurrent network* atau Hopfield *Net*.

Pada tahun 1986 Rumelhart, Hinton, dan William menciptakan suatu algoritma pembelajaran yang disebut dengan propagasi balik (*back propagation*). Metode ini dapat menyelesaikan masalah yang telah dilontarkan oleh Minsky dan Papert apabila algoritma ini diterapkan dalam perseptron dengan banyak lapisan (*multi-layer perceptron*) (Rahajaan, 2011).

2.1.2.3 Arsitektur Jaringan

Di dalam JST, istilah simpul (*node*) sering digunakan untuk menggantikan *neuron*, dimana setiap simpul pada jaringan menerima atau mengirim sinyal dari / ke simpul-simpul lainnya. Pengiriman sinyal disampaikan melalui penghubung. Kekuatan hubungan yang terjadi antara setiap simpul yang saling terhubung dikenal dengan nama bobot atau *weight*.

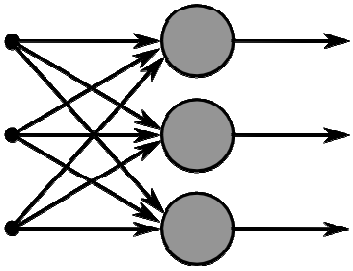


Gambar 2.3 *Artificial Neural Network* sederhana

Berdasarkan arsitekturnya, model JST terbagi menjadi 3, yaitu:

a. Jaringan Layar Tunggal (*Single Layer Network*)

Pada jaringan ini, sekumpulan *input* neuron dihubungkan langsung dengan sekumpulan *output*-nya. Sinyal mengalir searah dari lapisan *input* sampai ke lapisan *output*. Setiap simpul dihubungkan dengan simpul lainnya yang berada di atasnya dan di bawahnya, tetapi tidak dengan simpul yang berada pada lapisan yang sama. Contoh: ADALINE, *Hopfield*, *Perceptron*, LVQ, dan lain-lain.

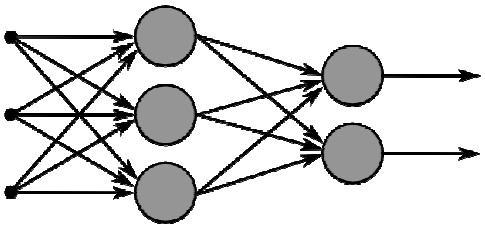


Gambar 2.4 Jaringan Layar Tunggal

b. Jaringan Layar Jamak (*Multiple Layer Network*)

Model ini merupakan pengembangan dari *single layer network*. Dalam jaringan ini, selain unit *input* dan *output*, terdapat pula unit-unit lain yang sering disebut dengan layar tersembunyi (*hidden layer*). Layar

tersembunyi ini tidak pasti satu. Bisa saja sebuah *multiple layer network* memiliki lebih dari satu *hidden layer*. Contoh: *back propagation* dan MADALINE.



Gambar 2.5 Jaringan Layar Jamak

c. Jaringan *Recurrent*

Model jaringan *recurrent* mirip dengan jaringan layar tunggal maupun jamak. Hanya saja, pada jaringan ini terdapat simpul keluaran yang memberikan sinyal pada unit masukan, sering juga disebut dengan *feedback loop*. Dengan kata lain, pada model ini sinyal berjalan dua arah, yaitu maju dan mundur. Contoh: *Hopfield network*, *Jordan network*, dan *Elmal network*.

2.1.2.4 Fungsi Aktivasi

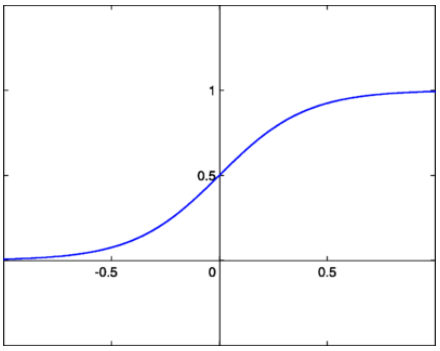
Dalam *Neural Network*, bagian yang paling penting adalah fungsi aktivasinya atau seringkali disebut juga dengan *threshold function* maupun *transfer function*. Fungsi aktivasi digunakan untuk membatasi *output* keluaran yang dihasilkan oleh neuron. Beberapa contoh fungsi aktivasi yang sering digunakan adalah: fungsi sigmoid biner dan fungsi sigmoid bipolar (Karlik & Olgac, 2011).

1. Sigmoid Biner

Fungsi aktivasi sigmoid biner:

$$g(x) = \frac{1}{1 + e^{-x}} \quad \text{.....(1)}$$

Fungsi ini sangat berguna untuk digunakan dalam *Neural Network* yang melakukan proses pelatihan dengan metode *Back propagation* karena mudah dibedakan dan mengurangi kapasitas yang diperlukan.



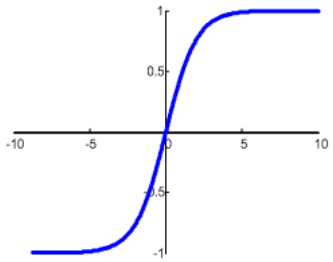
Gambar 2.6 Grafik Fungsi Aktivasi Sigmoid Biner

2. Sigmoid Bipolar

Fungsi aktivasi sigmoid bipolar:

$$g(x) = \frac{1 - e^{-x}}{1 + e^{-x}} \quad \text{.....(2)}$$

Fungsi aktivasi ini mirip dengan fungsi aktivasi sigmoid biner. Fungsi ini bekerja dengan baik untuk aplikasi yang memproduksi nilai dalam jarak [-1, 1].

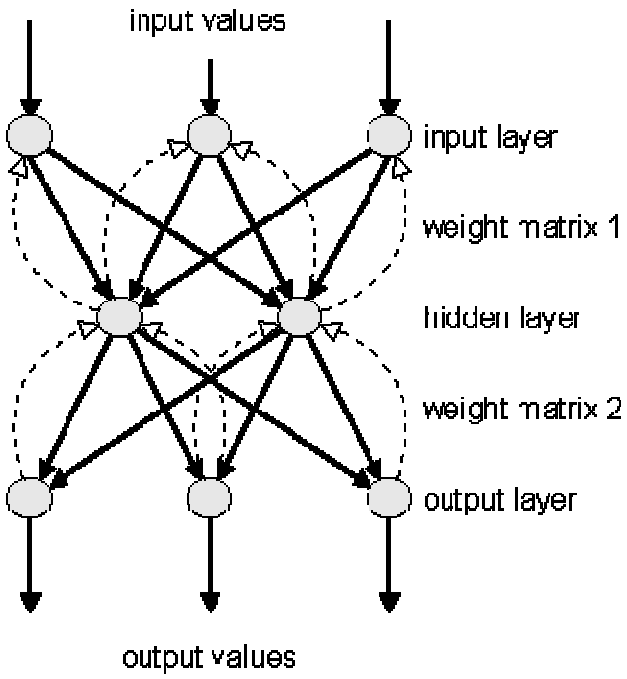


Gambar 2.7 Grafik Fungsi Aktivasi Sigmoid Bipolar

2.1.2.5 Back Propagation Network

Back Propagation Network (BPN) merupakan salah satu terapan dari *Multi Layered Network*. BPN sendiri sebenarnya lebih merupakan algoritma pelatihan daripada jaringan itu sendiri.

Metode pelatihan BPN adalah *supervised training algorithm* untuk jaringan layar jamak. Dikarenakan metode yang digunakan adalah pelatihan terpantau, maka baik *input* maupun *target output* telah disediakan untuk melatih jaringan. Kesalahan pada data di layer output dihitung menggunakan *network output* dan *target output*. Kemudian kesalahan itu di propagasikan kembali ke *hidden layer*, memberikan perubahan *weight* pada *synapses* yang menuju ke layer tersebut. (Pinjare, 2012)



Gambar 2.8 *Back Propagation Network*

Pada dasarnya, proses *back propagation* berjalan melalui dua tahap, yaitu tahap *feed forward* dan tahap *backward*.

1. Tahap *Feedforward*

Model *Neural Network* ini adalah yang paling sederhana dibandingkan dengan model-model *Neural Network* lainnya. Di sini, informasi hanya berjalan satu arah, dari lapisan *input*, menuju ke lapisan tersembunyi (jika ada), lalu ke lapisan *output*. Di model ini tidak terdapat perulangan maupun siklus.

Secara matematis, untuk setiap neuron pada JST, mendapat / menerima input Xi yang kemudian dimodulasikan oleh suatu *weight* Wi, sehingga jumlah total input adalah:

$$\sum_{i-1}^{n} x_i \, v_{ij} \quad \text{.....(3)}$$

Berikut ini merupakan algoritma dari *feedforward network* menurut Sutojo (2011: 326):

1. Tentukan arsitektur jaringan yang terdiri dari *input layer* (x_i; i = 1, 2, 3, ..., n), *hidden layer* (z_j; j = 1, 2, 3, ..., p), dan *output layer* (y_k; k = 1, 2, 3, ..., m). *Input layer* dan *hidden layer* memiliki node bias yang biasanya bernilai 1.
2. Inisialisasi nilai bobot (*weight*) dari *input layer* ke *hidden layer* (v_{ij}; i = 0, 1, 2, 3, ..., n; j = 1, 2, 3, ..., p) dan dari *hidden layer* ke *output layer* (w_{jk}; j = 0, 1, 2, 3, ..., p; k = 1, 2, 3, ..., m) dengan nilai random yang cukup kecil, misalnya dari -0,5 sampai 0,5.
3. Setiap node pada *input layer* (x_i; i = 1, 2, 3, ..., n) menerima sinyal x_i dan meneruskan sinyal tersebut ke semua node pada *hidden layer*.
4. Setiap node pada *hidden layer* (z_j; j = 1, 2, 3, ..., p) menjumlahkan *weight* sinyal *input* dengan persamaan berikut,

$$z_in_j = v_{0j} + \sum_{i=1}^{n} x_i \, v_{ij} \quad \text{.....(4)}$$

dan menerapkan fungsi aktivasi untuk menghitung sinyal *output*-nya:

$$z_j = f(z_in_j) \quad (5)$$

kemudian mengirimkan sinyal tersebut ke semua node pada *output layer*.

5. Setiap node pada output layer (y_k; k = 1, 2, 3, ..., m) menjumlahkan *weight* sinyal *input*

$$y_in_k = w_{0k} + \sum_{j=1}^{p} z_j \, w_{jk} \quad (6)$$

dan menerapkan fungsi aktivasi untuk menghitung sinyal output-nya:

$$y_k = f(y_in_k) \quad (7)$$

2. Tahap *Backward*

Pada tahap mundur, perubahan *weight* dari *synapses* dilakukan. Besarnya perubahan *weight* yang dilakukan dipengaruhi oleh besarnya kesalahan atau selisih di antara *network output* dan *target output*.

Berikut adalah algoritma dari tahap *backward* menurut Rahajaan (2011):

1. Hitung factor δ unit keluaran berdasarkan kesalahan di setiap unit keluaran y_k (k = 1, 2, 3, …, m).

$$\delta_k = (t_k - y_k) f'(y_{in_k}) = (t_k - y_k)\, y_k\, (1 - y_k) \quad \text{.....(8)}$$

t_k = target keluaran

δ_k = merupakan unit kesalahan yang akan dipakai dalam perubahan *weight* layer dibawahnya.

Hitung perubahan *weight* w_{kj} dengan laju pemahaman α.

$$\Delta w_{kj} = \alpha \delta_k z_j \quad \text{.....(9)}$$

k = 1, 2, …, m; j = 0, 1, …, p

2. Hitung factor δ unit keluaran berdasarkan kesalahan di setiap unit tersembunyi z_j (j = 1, 2, …, p).

$$\delta_j = \delta_in_j\ f'(z_in_j) = \delta_in_j\ z_j(1 - z_i) \quad \text{.....(10)}$$

Hitung suku perubahan *weight* v_{ji}

$$\Delta v_{ji} = \alpha \delta_j x_i \quad \text{.....(11)}$$

j = 1, 2, …, p; i = 1,2, …, n

3. Hitung semua perubahan *weight*. Perubahan *weight* garis yang menuju ke unit keluaran, yaitu:

$$w_{kj}(baru) = w_{kj}(lama) + \Delta w_{kj} \quad \text{.....(12)}$$

k = 1, 2, …, m; j = 0, 1, …, p

Perubahan *weight* yang menuju ke unit tersembunyi, yaitu:

$$v_{ji}(baru) = v_{ji}(lama) + \Delta v_{ji} \quad \text{.....(13)}$$

j = 1, 2, …, p; i = 1,2, …, n

Parameter α merupakan laju pemahaman yang menentukan kecepatan iterasi. Nilai α terletak antara 0 dan 1 ($0 \leq \alpha \leq 1$). Semakin besar nilai α, semakin sedikit iterasi yang terjadi. Namun apabila nilai α terlalu besar, maka pola yang sudah benar dapat menjadi rusak dan memperlambat proses pembelajaran.

Pemilihan *weight* awal sangat mempengaruhi JST dalam kecepatan pelatihan menuju kekonvergenan. Apabila *weight* awal terlalu besar, maka *input* ke setiap lapisan tersembunyi atau lapisan *output* akan jatuh pada daerah dimana turunan fungsi sigmoidnya akan sangat kecil, begitu pula jika *weight* awal terlalu kecil. Hal ini menyebabkan proses pelatihan menjadi sangat lambat. Biasanya *weight* awal diinisialisasi secara acak dengan nilai antara -0,5 sampai 0,5 (atau -1 sampai 1 atau interval yang lainnya). Setelah pelatihan selesai dilakukan, jaringan dapat dipakai untuk pengenalan pola. Dalam hal ini, yang dipakai hanyalah tahap maju (*feedforward*) saja untuk menentukan *network output* (Rahajaan, 2011).

2.1.2.6 Back Propagation Momentum

Menurut Istook (2002), proses pelatihan *back propagation* sering kali memakan waktu yang sangat lama. Sering kali perubahan *weight* yang terjadi sangatlah kecil sehingga memerlukan proses iterasi yang banyak. Hal tersebut bisa diakali dengan menaikkan *learning rate*-nya, yang menghasilkan

perubahan *weight* lebih besar, namun dengan naiknya *learning rate* maka tingkat konvergensinya justru menurun. Dari hal tersebut maka digunakanlah momentum.

Menurut Rahajaan (2011) pada BPN standar, perubahan *weight* didasarkan atas gradien yang terjadi untuk pola yang dimasukkan saat itu. Modifikasi yang bisa dilakukan adalah melakukan perubahan *weight* yang didasarkan atas arah gradien pola terakhir dan pola sebelumnya (disebut momentum) yang dimasukkan sehingga tidak hanya pola terakhir saja yang diperhitungkan.

Penambahan momentum dimaksudkan untuk menghindari perubahan *weight* yang mencolok akibat adanya data yang sangat berbeda dari data yang lain. Apabila beberapa pelatihan terakhir telah memiliki pola serupa (berarti arah gradien sudah benar), maka perubahan *weight* dilakukan secara cepat. Namun apabila terdapat pola yang berbeda dari yang lain, maka perubahan *weight* dilakukan secara lambat.

Dengan penambahan momentum, *weight* baru pada waktu ke (t+1) didasarkan atas *weight* pada waktu t dan (t-1). Jika μ adalah konstanta momentum ($0 \leq \mu \leq 1$) maka *weight* baru dihitung berdasarkan persamaan sebagai berikut (Rahajaan, 2011):

$$w_{kj}(t+1) = w_{kj} + \alpha\delta_k z_j + \mu\left(w_{kj}(t) - w_{kj}(t-1)\right) \quad \text{.....(14)}$$

dengan,

$w_{kj}(t)$ = *weight* awal pola kedua (hasil iterasi pola pertama)

$w_{kj}(t-1)$ = *weight* awal pada iterasi pola pertama

dan

$$v_{ji}(t+1) = v_{ji} + \alpha\delta_j x_i + \mu\left(v_{ji}(t) - v_{ji}(t-1)\right) \quad \text{.....(15)}$$

dengan,

$v_{ji}(t)$ = *weight* awal pola kedua (hasil iterasi pola pertama)

$v_{ji}(t-1)$ = *weight* awal pada iterasi pola pertama

Momentum di BPN tidak berbeda dengan momentum pada fisika.

Dalam bukunya yang berjudul *Machine Learning* oleh Tom Mitchell [18, p.100], momentum dideskripsikan sebagai berikut:

> "Too see the effect of this momentum term, consider that the gradient descent search trajectory is analogous to that of a (momentumless) ball rolling down the error surface. The effect of μ is to add momentum that tends to keep the ball rolling down the error surface."

2.1.3 Optical Character Recognition

Optical Character Recognition (OCR) adalah proses merekognisi atau mengenali sebuah karakter oleh komputer yang discan secara optikal dan halaman text yang telah diubah ke dalam bentuk digital. OCR merupakan salah satu area yang paling menarik dan menantang dari rekognisi pola. OCR dapat memberikan kontribusi yang besar kepada perkembangan proses automatisasi dan dapat meningkatkan kualitas antarmuka antara manusia dan mesin dalam berbagai macam aplikasi (Kannan, R. J. & Prabhakar, R., 2009).

2.2 Teori-teori Khusus yang Berhubungan dengan Topik yang Dibahas

2.2.1 Kombinasi Neural Network dan Genetic Algorithm

Salah satu algoritma yang paling populer untuk melakukan pelatihan pada *neural network* adalah algoritma *Back propagation*. Algoritma *Back propagation* berjalan dengan membuat modifikasi pada nilai *weight* dimulai dari *output layer* kemudian mundur ke belakang melalui *hidden layer*. Salah satu faktor utama yang paling berpengaruh dalam proses pelatihan *Back propagation* adalah penentuan nilai *weight* awal. Karena kurangnya bukti pasti, penentuan nilai awal *weight* dari algoritma *Back propagation* biasanya dilakukan secara sembarang. Oleh karena itu tidak jarang kecepatan untuk mencapai kondisi konvergen sangat pelan, bahkan terkadang tidak pernah mencapai kondisi konvergen. Dikarenakan kelemahan dari *Back propagation* tersebut, maka perlu dilakukan optimisasi dan improvisasi.

Untuk melakukan optimasi dan improvisasi dapat digunakan *genetic algorithm*. Tidak seperti algoritma pencarian yang lainnya yang melakukan pencarian secara lokal, *Genetic algorithm* melakukan pencarian secara global. *Genetic algorithm* dapat digunakan untuk mengoptimalisasi berbagai macam parameter *Neural Network* seperti arsitektur *Neural Network*, *weight*, pemilihan *input*, fungsi aktivasi, tipe *Neural Network*, algoritma pelatihan, jumlah pengulangan, dan rasio pemisahan dataset.

Dalam kasus ini digunakan *genetic algorithm* untuk menentukan *weight* awal dari arsitektur *Neural Network* dan menentukan arsitektur *hidden layer* yang optimal yang nantinya akan digunakan untuk pelatihan.

Untuk menentukan *weight* awal yang optimal, maka terlebih dahulu *weight* awal ditentukan secara sembarang, langkah kedua adalah mengevaluasi nilai *fitness* dari semua *weight*, yang ketiga adalah

mengaplikasikan proses evolusi seperti operasi seleksi, *crossover*, dan mutasi sesuai dengan nilai *fitness* yang ada. Proses evolusi berhenti ketika nilai *fitness* lebih besar dari nilai yang ditentukan (contohnya pelatihan *error* lebih kecil dari suatu nilai yang ditentukan) atau populasi menjadi konvergen. (Karegowda, A. G., Manjunath, A. S., & Jayaram, M. A., 2011)

Menurut Fiszelew (2007), untuk menentukan arsitektur *hidden layer* yang optimal harus mengikuti langkah-langkah berikut ini :

- Membuat populasi awal dari tiap individu (*neural networks*) dengan arsitektur yang acak. Latih tiap individu dengan menggunakan metode *Genetic algorithm* (seperti di atas)
- Pilih pasangan induk dari populasi
- Lakukan reproduksi untuk mendapatkan 2 anak
- Lakukan mutasi kepada ke 2 anak secara acak
- Latih tiap anak dengan menggunakan *genetic algorithm* (seperti di atas)
- Letakkan anak-anak tersebut ke dalam populasi untuk menggantikan anggota populasi yang memiliki nilai terburuk
- Ulangi dari langkah kedua sampai dengan jumlah yang diinginkan

2.2.2 Optical Character Recognition dengan Kombinasi Neural Network dan Genetic algorithm

Menurut riset yang terdahulu dengan menggunakan Teknik Kombinasi *Neural Network* dan *Genetic algorithm* akan menghasilkan rekognisi dengan efisiensi yang lebih baik dibandingkan algoritma *back propagation* (Kaur, R. & Baljit, S., 2011). Hasil penelitian dengan

menggunakan Neural Network yang sudah dioptimasi dengan *Genetic algorithm* pengenalan terhadap karakter ini diprediksi akan lebih cepat dan tepat karena hasil pelatihan yang diperoleh adalah *global optima* dan arsitektur yang telah dioptimalisasi dengan menggunakan *genetic algorithm* akan memperoleh efisiensi yang lebih tinggi.