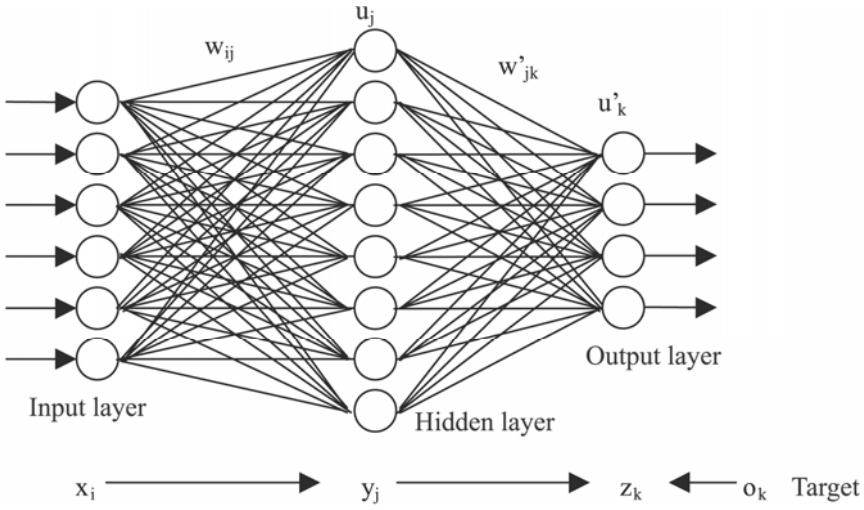
Lets Study Backpropagation Neural Network

Algoritma pelatihan Backpropagation Neural Network (BPNN) pertama kali dirumuskan oleh Werbos dan dipopulerkan oleh Rumelhart & Mc.Clelland. *Backpropagation neural network* merupakan tipe jaringan saraf tiruan yang menggunakan metode pembelajaran terbimbing (*supervised learning*). Pada *supervised learning* terdapat pasangan data *input* dan *output* yang dipakai untuk melatih JST hingga diperoleh bobot penimbang (*weight*) yang diinginkan. Penimbang itu sendiri adalah sambungan antar lapis dalam JST. Algoritma ini memiliki proses pelatihan yang didasarkan pada interkoneksi yang sederhana, yaitu apabila keluaran memberikan hasil yang salah, maka penimbang dikoreksi agar galat dapat diperkecil dan tanggapan JST selanjutnya diharapkan dapat mendekati nilai yang benar. BPNN juga berkemampuan juga berkemampuan untuk memperbaiki penimbang pada lapis tersembunyi (*hidden layer*).

Secara garis besar BPNN terdiri atas tiga lapis (*layer*) yaitu lapis masukan (*input layer*) x_i, lapis tersembunyi (*hidden layer*) y_j, dan lapis keluaran (*output layer*) z_k. Lapis masukan dan lapis tersembunyi dihubungkan dengan penimbang w_{ij} dan antara lapis tersembunyi dan lapis keluaran dihubungkan oleh penimbang w'_{jk}. Pada pelatihan BPNN, ketika JST diberi pola masukan sebagai pola pelatihan maka pola tersebut akan menuju ke unit pada lapis tersembunyi untuk diterusan pada unit yang berada pada lapis keluaran. Keluaran sementara pada lapis tersembunyi u_j akan diteruskan pada lapis keluaran dan lapis keluaran akan memberikan tanggapan yang disebut sebagai keluaran sementara u'_k. Ketika $u'_k \neq o_k$ dimana o_k adalah keluaran yang diharapkan, maka selisih (*error*) keluaran sementara u'_k akan disebarkan mundur (*backward*) pada lapis tersembunyi dan diteruskan ke unit pada lapis masukan. Oleh karena itu proses tersebut disebut propagasi balik (*backpropagation*) dimana tahap pelatihan dilakukan dengan merubah penimbang yang menghubungkan unit dalam lapis JST ketika diberi umpan maju dan umpan balik. Untuk mempercepat proses pelatihan digunakan parameter laju pelatihan (*learning rate*) yang nilainya berada pada kisaran 0-1. Selain parameter laju pelatihan, untuk mempercepat proses pelatihan dapat digunakan parameter tambahan berupa momentum yang nilainya dijaga antara 0.5-0.9. Ketika proses pelatihan selesai dan JST dapat digunakan untuk menyelesaikan masalah, tahap tersebut disebut sebagai tahap penggunaan yang disebut *mapping* atau pemetaan.



Gambar lapis dan aliran sinyal dalam algoritma BPNN

Algoritma pelatihan BPNN terdiri dari dua tahap, yaitu *feed forward propagation* dan *feed backward propagation*. Secara umum langkah dalam pelatihan JST menggunakan BPNN yang dilengkapi bias dan momentum adalah sebagai berikut :

1. Menentukan jumlah *input* (pola masukan), *hidden layer*, dan *output* (target pelatihan).

2. Memberi nilai awal secara random bagi seluruh *weight* antara *input-hidden layer* dan *hidden layer-output*.
3. Melakukan langkah 3-11 secara berulang hingga diperoleh nilai *error* minimal yang memungkinkan bagi JST untuk belajar dengan baik.

   {*FEED FORWARD PROPAGATION*}
4. Tiap unit *input* (Xi) menerima sinyal *input* dan sinyal tersebut dikirimkan pada seluruh unit *hidden layer*.
5. Tiap unit *hidden layer* (Zin_j) ditambah dengan *input* (Xi) yang dikali dengan *weight* (Vij) dan dijumlah dengan bias bagian *input*;

```
{Unit Input*Weight(Input->Hidden)}
Zin_j[j]:=Zin_j[j]+Xi[i]*Vij[i,j];
{Ditambah Bias}
Zin_j[j]:=Zin_j[j]+Vij[0][j];
{Dihitung dalam Fungsi Pengaktif}
Zj[j]:=f(Zin_j[j]);
```

   Fungsi pengaktif neuron yang digunakan pada seluruh bagian pelatihan harus sama. Fungsi pengaktif neuron yang umum digunakan terdapat beberapa macam, yang paling umum adalah fungsi sigmoid baik yang bipolar (-0.5 - +0.5) maupun unipolar (0 – 1) seperti berikut :

```
{sigmoid bipolar}
fbipolar:=(2/(1+exp(-1* Zin_j[j])))-1;
{sigmoid unipolar}
funipolar:=1/(1+exp(-Zin_j[j]));
```

6. Tiap unit *output* (Yink) ditambah dengan nilai keluaran *hidden layer* (Zj) yang dikali *weight* (Wjk) dan dijumlah dengan bias bagian *hidden layer*. Untuk mendapatkan keluaran JST, maka Yink dihitung dalam fungsi pengaktif menjadi Yk.

```
{Unit Keluaran*Weight(Hidden->Keluaran)}
Yink[k]:=Yink[k]+Zj[j]*Wjk[j,k];
{Ditambah Bias}
Yink[k]:=Yink[k]+Wjk[0,k];
{Dihitung dalam Fungsi Pengaktif}
Yk[k]:=f(Yink[k]);
```

   {FEED BACKWARD PROPAGATION}
7. Tiap *output* dibandingkan dengan target yang diinginkan, untuk memperoleh *error* global digunakan metode *Sum Squared Error* (SSE).

```
Error:=Error+(((O_target[k]-Yk[k])*(O_target[k]-Yk[k]))*0.5);
```

8. Tiap unit *output* menerima pola target sesuai dengan pola masukan saat pelatihan dan dihitung nilai *error*-nya dan diperbaiki nilai *weight*-nya.

```
{Perhitungan Error dalam turunan Fungsi Pengaktif}
delta_k[k]:=(O_target[k]-Yk[k])*f'(Yink[k]);
```

   Perbaikan *weight output-hidden layer* dilakukan dengan memperhitungkan laju pelatihan dan momentum, laju pelatihan dijaga pada nilai kecil antara 0-1 dan momentum pada nilai 0.5-0.9.

```
{Perbaikan weight antara hidden layer-output}
update_Wjk[j,k]:=eLaju*delta_k[k]*Zj[j]+(update_Wjk[j,k]*eMomentum);
{Perbaikan weight bias antara hidden layer-output}
```

update_Wjk[0,k]:=eLaju*delta_k[k];

9. Tiap *weight* yang menghubungkan unit *output* dengan unit *hidden layer* dikali selisih *error* (delta_k) dan dijumlahkan sebagai masukan unit berikutnya.

   {Perhitungan *Error**Bobot Keluaran}
   delta_in_j[j]:=delta_in_j[j]+delta_k[k]*Wjk[j,k];
   {Perhitungan *Error* dalam turunan Fungsi Pengaktif}
   delta_j[j]:=delta_in_j[j]*f'(Zin_j[j]);

   Perbaikan *weight hidden layer-input* dilakukan dengan memperhitungkan laju pelatihan dan momentum, laju pelatihan dijaga pada nilai kecil antara 0-1 dan momentum pada nilai 0.5-0.9.

   {Perbaikan *weight* antara masukan dan *hidden layer*}
   update_Vij[i,j]:=eLaju*delta_j[j]*Xi[i]+(Vij[i,j]*eMomentum);
   { Perbaikan *weight* bias antara masukan}
   update_Vij[0,j]:=eLaju*delta_j[j];

10. Tiap *weight* dan bias yang ada pada JST diperbaiki.

    {Penambahan Nilai Perbaikan Bobot *Hidden layer*-Keluaran}
    Wjk[j,k]:=Wjk[j,k]+update_Wjk[j,k];
    {Penambahan Nilai Perbaikan Bobot Masukan-*Hidden layer*}
    Vij[i,j]:=Vij[i,j]+update_Vij[i,j];

11. Uji kondisi pemberhentian pelatihan.

Pada kondisi dimana JST telah selesai dilatih, maka JST tersebut dapat diujicoba sebelum pada akhirnya JST tersebut digunakan untuk menyelesaikan suatu masalah. Maka untuk menggunakan hasil pelatihan tersebut digunakan weight yang telah diperoleh dari proses pelatihan untuk memperoleh hasil target yang telah dilatihkan. Pada BPNN yang telah dibahas di atas, algoritma ujicoba JST yang dapat digunakan adalah sebagai berikut :

1. Tiap unit *input* (Xi) menerima sinyal *input* dan sinyal tersebut dikirimkan pada seluruh unit *hidden layer*.
2. Tiap unit *hidden layer* (Zin_j) ditambah dengan *input* (Xi) yang dikali dengan *weight* (Vij) yang diperoleh dari proses pelatihan dan dijumlah dengan bias bagian *input*;

   {Unit *Input**Weight*(*Input*->*Hidden*)}
   Zin_j[j]:=Zin_j[j]+Xi[i]*Vij[i,j];
   {Ditambah Bias}
   Zin_j[j]:=Zin_j[j]+Vij[0][j];
   {Dihitung dalam Fungsi Pengaktif}
   Zj[j]:=*f*(Zin_j[j]);

   Fungsi pengaktif neuron yang digunakan pada seluruh bagian ujicoba harus sama. Fungsi pengaktif neuron yang umum digunakan terdapat beberapa macam, yang paling umum adalah fungsi sigmoid baik yang bipolar (-0.5 - +0.5) maupun unipolar (0 – 1) seperti berikut :

   {sigmoid bipolar}
   fbipolar:=(2/(1+exp(-1* Zin_j[j])))-1;
   {sigmoid unipolar}
   funipolar:=1/(1+exp(-Zin_j[j]));

3. Tiap unit *output* (Yink) ditambah dengan nilai keluaran *hidden layer* (Zj) yang dikali *weight* (Wjk) yang diperoleh dari proses pelatihan dan dijumlah dengan bias bagian *hidden layer*. Untuk mendapatkan keluaran JST, maka Yink dihitung dalam fungsi pengaktif menjadi Yk.

```
{Unit Keluaran*Weight(Hidden->Output)}
Yink[k]:=Yink[k]+Zj[j]*Wjk[j,k];
{Ditambah Bias}
Yink[k]:=Yink[k]+Wjk[0,k];
{Dihitung dalam Fungsi Pengaktif}
Yk[k]:=f(Yink[k]);
```

4. Untuk mengetahui keandalan JST yang digunakan, hasil target pelatihan dibandingkan dengan keluaran yang diperoleh ketika dilakukan ujicoba.

Daftar Pustaka :

- Mauridhi Hery P; Agus Kurniawan, *Supervised Neural Networks dan Aplikasinya*, Graha Ilmu, Yogyakarta, 2006.
- Saludin Muis, *Teknik Jaringan Saraf Tiruan*, Graha Ilmu, Yogyakarta, 2006.
- Jong Jek Siang, *Jaringan Saraf Tiruan dan Pemrogramannya Menggunakan Matlab*, Andi, Yogyakarta, 2005.