BAB II

TINJAUAN PUSTAKA

2.1 Time Series

Time series atau runtun waktu merupakan suatu pengamatan terhadap variabel dari waktu lampau dan dicatat secara beruntut sesuai urutan waktu dengan periode waktu yang tetap (Hanke dan Winchern, 2004). Pada umumnya pencatatan ini dilakukan dalam periode tertentu misalnya harian, bulanan, tahunan dan sebagainya, metode time series juga merupakan metode peramalan dengan menggunakan analisa hubungan anatara variabel yang akan diperkirakan dengan variabel waktu. Peramalan suatu data times series perlu memperhatikan tipe atau pola data. Terdapat empat macam pola data time series yaitu horizontal, musiman, siklis dan trend (Makridakis *et al*, 1995). Pola horizontal terjadi bilamana nilai data fluktuasi disekitar nilai rata-rata yang konstan. Pola musiman terjadi bilamana suatu deret dipengaruhi oleh faktor musiman (misalnya kuartal tahun tertentu, bulanan, atau hari-hari pada minggu tertentu). Pola siklis terjadi bilamana datanya dipengaruhi oleh fluktuasi jangka panjang. Sedangkan pola *trend* terjadi bilamana terdapat kenaikan atau penurunan sekuler jangka panjang dalam data.

2.2 *White Noise*

Sebuah proses $[e_t]$ disebut *white noise* jika merupakan serangkaian variabel random yang tidak berkorelasi dan berdistribusi tertentu dengan rata-rata tetap E

http://repository.unimus.ac.id

(e_t) = μ biasanya benilai 0, variansi konstanta $Var\ (e_t)$= σ^2 dan $Cov\ (e_t, e_t+k)$ = 0 untuk semua $k \neq 0$ (Wei, 2006). Dengan demikian proses dari white noise e_t adalah stasioner dengan fungsi autokovariansi:

$$\gamma_k = \begin{cases} \sigma_t^2 & k = 0 \\ 0 & k \neq 0 \end{cases} \quad (1)$$

fungsi *autokorelasi*

$$\rho_k = \begin{cases} 1 & k = 0 \\ 0 & k \neq 0 \end{cases} \quad (2)$$

dan fungsi *autokorelasi parsial*

$$\phi_{kk} = \begin{cases} 1 & k = 0 \\ 0 & k \neq 0 \end{cases} \quad (3)$$

Proses *white noise* dapat diketahui melalui uji autokorelasi residual pada analisis *error*-nya. Uji korelasi residual digunakan untuk mendeteksi ada atau tidak korelasi residual antar *lag*. Langkah-langkah pengujian korelasi residual, yaitu:

H_0 : $\rho_1 = \rho_2 = \rho_3 = \ldots = \rho_k = 0$ (Tidak ada korelasi residual antar *lag*)

H_1 : paling sedikit ada satu *$\rho_k \neq 0$,* (Ada korelasi residual antar *lag*)

Dengan k = 1,2,3,…., k

Taraf signifikansi α = 5%

Statistik uji dengan menggunakan *Ljung Box-Pierce*. Rumus *Ljung Box-Pierce* (Wei, 2006):

$$Q_k = T(T+2)\sum_{k=1}^{K} \frac{\rho_k^2}{T-K} \quad (4)$$

dengan,

T : banyaknya data

http://repository.unimus.ac.id

K : banyaknya *lag* yang diuji
ρ_k : dugaan autokorelasi residual periode k

kriteria keputusan yaitu tolak H_0 jika $Q_{hitung} > X^2$ tabel, dengan derajat kebebasan k dikurangi banyaknya parameter pada model atau *p-value* < α, artinya e_t adalah barisan yang tidak memiliki nilai korelasi.

2.3 *Neural Network* (NN)

Jaringan syaraf tiruan (JST) atau sering dikenal dengan istilah *neural network* adalah sistem pemroses informasi yang memiliki karakteristik mirip dengan jaringan syaraf biologi manusia (Siang, 2005). *Neural network* telah diaplikasikan dalam berbagai bidang diantaranya *pattern recognition, medical diagnostic, signal processing*, dan peramalan. Pada dasarnya, *neural network* merupakan kumpulan dari elemen-elemen pemroses yang saling berhubungan, yang disebut dengan *unit-unit* atau syaraf-syaraf (Suhartono, 2007). *Neural network* bekerja berdasarkan pola yang terbentuk pada *input*nya. Setiap *neuron* dihubungkan dengan *neuron* lainnya dengan suatu *connection link*, yang direpresentasikan dengan *weight*/bobot. Metode untuk menentukan nilai *weight* disebut dengan *training*, learning, atau *algoritma*. Setiap *neuron* menggunakan fungsi aktivasi pada net *input* untuk menentukan prediksi *output*.

Neural network termasuk dalam salah satu bentuk *time series* nonlinier dan memiliki bentuk fungsional yang fleksibel sehingga *neural network* tidak dapat diintrepretasikan atau tidak memiliki asumsi-asumsi yang harus dipenuhi. *Neuron-*

http://repository.unimus.ac.id

neuron dalam *neural network* disusun dalam grup, yang disebut dengan *layer* (lapis). Secara garis besar pada *Neural Network* (NN) memiliki dua tahapan dalam *system* pemrosesan informasi, yaitu:

a) Tahap pelatihan (*training*)

Tahapan ini dimulai dengan memasukan data latih ke dalam jaringan (warsito, 2009). Dengan menggunakan data latih, jaringan akan mengubah-ubah bobot yang menjadi penghubung antar node. Pada setiap evaluasi dilakukan evaluasi terhadap *output* jaringan. Tahapan ini berlangsung pada beberapa iterasi dan berhenti setelah menemukan bobot yang sesuai dimana nilai *error* yang diinginkan telah tercapai atau jumlah iterasi telah mencapai nilai maksimal yang ditetapkan, selanjutnya bobot ini akan menjadi dasar pengetahuan pada tahapan pengenalan.

b) Tahap pengujian (*testing*)

Pengujian dilakukan dengan memasukan suatu pola yang belum pernah dilatih sebelumnya (data uji) dengan menggunakan bobot hasil tahap pelatihan. Diharapkan bobot-bobot hasil pelatihan yang telah menghasilkan *error* minimal juga akan menghasil *error* minimal pada tahap pengujian (Warsito, 2009).

Data *training* digunakan dalam proses pembentukan arsitektur terbaik dalam proses pelatihan, sedangkan data *testing* digunakan dalam pengujian keakuratan dari arsitektur yang telah terbentuk. Menurut (Hota, 2013) pilihan

http://repository.unimus.ac.id

ukuran untuk data *training* dan *testing* yang dapat digunakan adalah sebagai berikut:

Tabel 2.1 Pembagian Data *training* dan *testing*

Data *Training* (%)	Data *Testing* (%)
60	40
75	25
80	20

2.4 Komponen *Neural Network*

Neural network memilik karakteristik yang sama dengan karakteristik syaraf manusia, berdasarkan karakteristiknya *neural network* dibangun dengan komponen-komponen yang sama dengan manusia, diantaranya:

1. *Neuron/Node*

   Bertugas memproses semua informasi yang diterima, sama dengan fungsi *neuron* pada otak manusia, dimana semua proses perhitungan dilakukan disini (Kusumdewi, 2004).

2. *Input*

   Informasi yang diproses dalam *neuron input* berasal dari lingkungan ataupun dari node lain (Pandjaitan, 2007). *Neural network* hanya dapat memproses data masukan berupa data numerik sehingga apabila ada masalah yang melibatkan data kualitatif seperti grafik, gambar, sinyal atau suara harus ditranformasikan terlebih dahulu kedalam data numerik.

3. Fungsi aktivasi

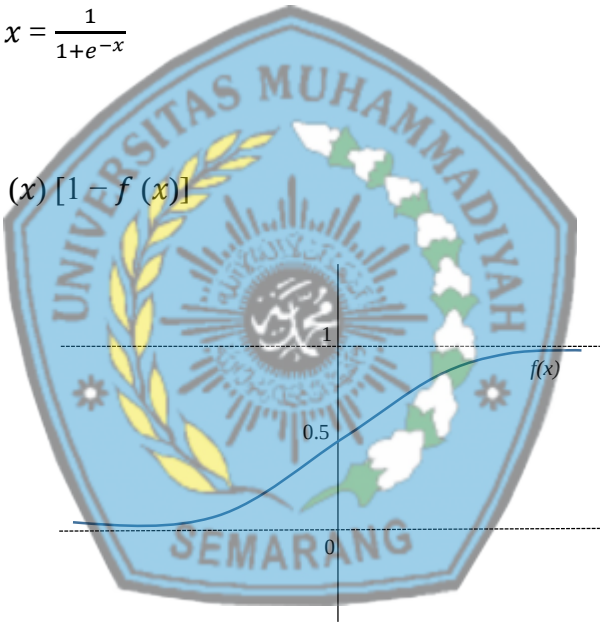
http://repository.unimus.ac.id

Fungsi aktivasi yang akan menentukan apakah sinyal dari *input neuron* akan diteruskan atau tidak (Siang, 2005). Fungsi aktivasi jaringan syaraf tiruan yang dilatih menggunakan metode *backpropagation* adalah fungsi sigmoid biner. Fungsi sigmoid biner mempunyai interval range 0 sampai 1, sehingga fungsi ini digunakan untuk jaringan syaraf tiruan yang memiliki nilai *output* yang terletak pada interval 0 sampai 1. Fungsi sigmoid biner dirumuskan sebagai :

$$y = f\ x = \frac{1}{1+e^{-x}} \quad (5)$$

dengan,

$$y'\ x = (x)\ [1 - f\ (x)]$$



Gambar 2.1 Fungsi Aktivasi Sigmoid Biner

Fungsi sigmoid memiliki nilai maksimum = 1. Maka untuk pola yang targetnya > 1, pola *input* dan *output* harus terlebih dahulu ditransformasi sehingga semua polanya memiliki range yang sama seperti fungsi sigmoid yang dipakai.

4. Bias dan *threshold*

http://repository.unimus.ac.id

Bias dapat dipandang sebagai *input* yang bernilai 1. Bias berfungsi untuk mengubah nilai *threshold* menjadi 0. Sedangkan *threshold* berperan sebagai penimbang dalam suatu hubungan dari sebuah *unit* tertentu (Siang, 2005).

5. Bobot

   Bobot merupakan suatu nilai yang menunjukan kekuatan antar node (Siang, 2005). Jika nilai bobot diatur berbeda, maka *output* yang dihasilkan akan berbeda. Bobot-bobot yang optimal akan memungkin kan system menterjemahkan data masukan secara benar dan menghasilkan keluaran yang diinginkan.

6. *Output*

   Menurut (Pandjaitan, 2007) Suatu nilai yang dihasilkan dari fungsi aktivasi, yang bisa berupa *output* dari jaringan atau menjadi *input* bagi node lain disebut sebagai *output*. Data *output* merupakan data numerik.

2.5 *Artificial Neural Network* (ANN)

Arsitektur jaringan dalam NN merupakan susunan dari *neuron-neuron* dalam lapisan *input*, *hidden* dan *output* yang terhubung dengan bobot, fungsi aktivasi dan fungsi pembelajaran. Arsitektur ini merupakan salah satu karakteristik penting yang membedakan *neural network*. Secara umum ada tiga lapis yang membentuk *neural network*:

1) Lapis *input*

http://repository.unimus.ac.id

Unit-unit di lapisan *input* disebut *unit-unit input*. *Unit-unit input* tersebut menerima pola *input*an dari luar yang menggambarkan suatu permasalahan. banyak node atau *neuron* dalam lapis *input* tergantung pada banyaknya *input* dalam model dan setiap *input* menentukan satu *neuron*.

2) Lapis tersembunyi (hidden *layer*)

Unit-unit dalam lapisan tersembunyi disebut *unit-unit* tersembunyi, di mana *output*nya tidak dapat diamati secara langsung. Lapis tersembunyi terletak di antara lapis *input* dan lapis *output*, yang dapat terdiri atas beberapa lapis tersembunyi.

3) Lapis *output*

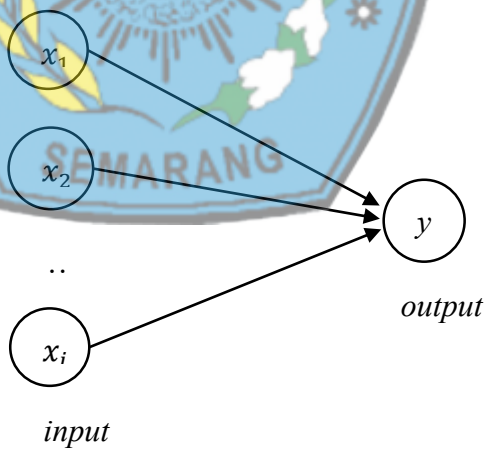
Unit-unit dalam lapisan *output* disebut *unit-unit output*. *Output* dari lapisan ini merupakan solusi *Neural Network* terhadap suatu permasalahan. Setelah melalui proses *training, network* merespon *input* baru untuk menghasilkan *output* yang merupakan hasil peramalan.

Berdasarkan jumlah *layer* yang dimiliki, *neural network* dibedakan menjadi:

a. Jaringan lapisan tunggal (*Single Layer*)

http://repository.unimus.ac.id

Jaringan lapisan tunggal atau *single layer* hanya memiliki dua lapisan yakni lapisan *input* dan lapisan *output*, dimana lapisan *input* berperan dalam menerima sinyal data *input* sedangkan lapisan *output* berperan sebagai media dalam memberikan hasil *output*. *Layer input* disusun oleh beberapa *neuron* yang dihubungkan oleh bobot menuju *layer output* dalam satu alur maju dan tidak sebaliknya. Itulah sebabnya arsitektur ini disebut sebagai arsitektur umpan maju (*feedforward)*. Walaupun arsitektur ini terdiri dari dua *layer*, namun arsitektur ini dikategorikan sebagai arsitektur *layer* tunggal karena *layer output* secara tunggal melakukan proses komputasi tanpa melibatkan *layer* lain diantara *layer input* dan *output*. Arsitektur dari jaringan *single layer* digambarkan sebagai berikut:



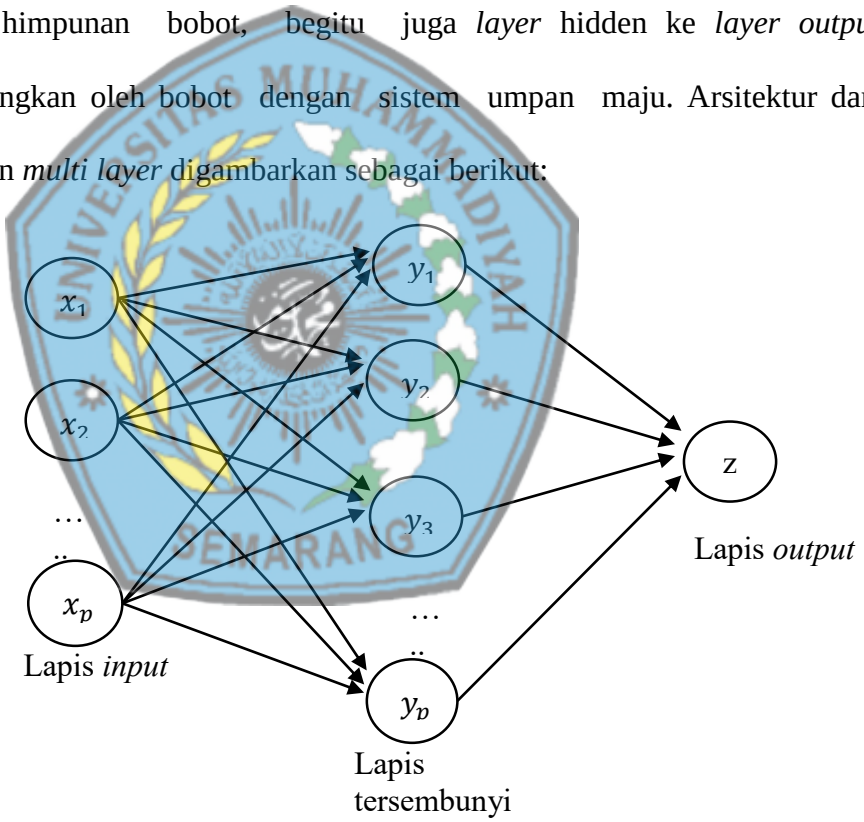
Gambar 2.2 Jaringan *Single Layer*

b. Jaringan multi lapis (*Multi Layer)*

Jaringan *multi layer* merupakan perluasan dari jaringan *single layer*, dimana pada jaringan ini terdapat *layer* tambahan yang terletak diantara

http://repository.unimus.ac.id

lapisan *input* dan lapisan *output*, *layer* tersebut dikenal dengan *hidden layer*/lapisan tersembunyi. *Layer* hidden terdiri dari *neuron* hidden yang melakukan perhitungan dari *layer input* untuk kemudian dilanjutkan kepada *layer output*. Dalam satu arsitektur multi*layer* jumlah *layer* hidden yang digunakan boleh lebih dari satu, sesuai dengan kasus ataupun masalah yang akan diselesaikan. *Layer input* dihubungkan ke *layer* hidden oleh himpunan bobot, begitu juga *layer* hidden ke *layer output* dihubungkan oleh bobot dengan sistem umpan maju. Arsitektur dari jaringan *multi layer* digambarkan sebagai berikut:



Gambar 2.3 Jaringan *Multi Layer*

Dalam penelitian ini, digunakan jaringan *multi layer* dengan model *feedforward neural network* (FFNN). FFNN adalah bentuk khusus jaringan *multi layer* dengan satu lapisan tersembunyi.

http://repository.unimus.ac.id

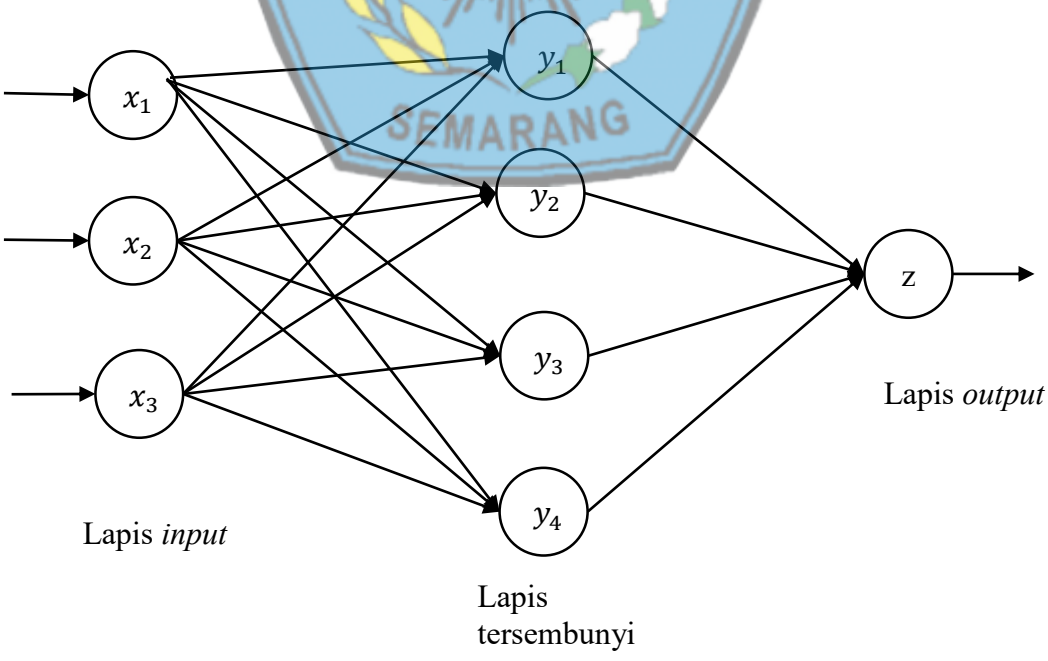
Pada pemodelan FFNN untuk data *time series, input* model adalah data nilai tukar rupiah terhadap dollar dan targetnya adalah pembukaan nilai tukar atau nilai tukar yang diharapkan pada bulan tersebut (Fadila, 2017).

Bentuk umum persamaan FFNN untuk data *time series* dituliskan dalam persamaan berikut:

$$X_t = \varphi_o\{w_{k0} + \sum_{j=1}^{H} w_{kj}\, \varphi_j(v_{j0} + \sum_{i=1}^{p} v_{ji}\, X_i)\} \quad (6)$$

Dengan;

φ_o : fungsi aktivasi yang digunakan pada lapisan *output*
φ_j : fungsi aktivasi yang digunakan pada lapisan tersembunyi
v_{ji} : bobot *neuron* ke-I pada lapisan *input* menuju *neuron* ke-j pada lapisan tersembunyi
v_{j0} : bobot bias pada lapisan *input* menuju *neuron* ke-j pada lapisan tersembunyi
w_{kj} : bobot *neuron* ke-j pada lapisan tersembunyi menuju lapisan *output*
w_{k0} : bobot bias pada lapisan tersembunyi menuju lapisan *output*



Gambar 2.4 Ilustrasi *Feedforward Neural Network*

http://repository.unimus.ac.id

Metode pelatihan jaringan merupakan suatu proses atau prosedur pelatihan jaringan yang merupakan urutan langkah-langkah *algoritma* untuk memodifikasi nilai-nilai bobot dan bias dengan tujuan agar suatu jaringan mendapat nilai-nilai bobot dan bias yang sesuai sehingga dapat menghasilkan *output* jaringan yang diinginkan. Jika kesalahan pada *output* jaringan sangat kecil, maka dapat dikatakan telah diperoleh nilai-nilai bobot dan bias yang sesuai dan jaringan tersebut telah mencapai jaringan yang baik. Dalam penelitian ini digunakan nilai MAPE (*Mean Absolute Persentage Error*) untuk mengukur kesalahan nilai dugaan model yang dinyatakan dalam bentuk rata-rata persentase absolute residual. Nilai MAPE dapat dirumuskan sebagai berikut:

$$MAPE=\frac{1}{n}\sum_{t=1}^{n}\frac{y_i-\hat{y}_i}{y_i}\ x\ 100\% \qquad (7)$$

dimana,

n : banyaknya data yang diprediksi
y_i : data aktual period eke-i
$\hat{y}$: data hasil prediksi period eke-i

Interpretasi dari perhitungan MAPE menurut (Chen *et al*, 2007) adalah sebagai berikut:

1. nilai MAPE < 10% sangat baik untuk melakukan prediksi
2. nilai MAPE berada diantara 10% - 20% menghasilkan nilai prediksi yang baik

http://repository.unimus.ac.id

3. nilai MAPE berada diantara 20% - 50% menghasilkan prediksi yang proporsional/wajar artinya masih dapat digunakan dalam memprediksi
4. nilai MAPE > 50% tidak dapat digunakan dalam memprediksi

2.6 Metode Pelatihan

Jaringan *neural network* mempunyai karakteristik yang penting selain arsitektur, yakni metode pengaturan nilai bobot / *Training* (Fausset, 1994). Metode pelatihan pada *neural network* dibagi menjadi dua jenis, yakni:

1. Pelatihan Terawasi

   Pelatihan ini dilakukan dengan adanya urutan vektor pelatihan, atau pola masing-masing terkait dengan target *output*, kemudian bobot disesuaikan untuk *algoritma* pembelajaran.

2. Pelatihan Tak Terawasi

   Pada pelatihan ini *neural network* mengatur segala kinerja dirinya sendiri, mulai dari masukan target hingga menggunakan data *training* untuk melakukan pembelajaran .

   Pada penelitian ini, yang digunakan sebagai metode untuk mengatur nilai bobot dalam *backpropagation* adalah metode pelatihan terawasi (Kusumadewi, 2004).

2.7 *Feedforwad Neural Network*

Feedforward Neural Network (FFNN) merupakan bentuk arsitektur *neural network* yang paling sering digunakan karena FFNN sangat fleksibel terhadap

http://repository.unimus.ac.id

fungsi-fungsi nonlinier. Secara umum FFNN bekerja dengan menerima suatu vektor dari *input* **x** dan kemudian menghitung suatu respon atau *output* $\hat{y}(\mathbf{x})$ dengan memproses **x** melalui elemen-elemen proses yang saling terkait. Elemen-elemen proses tersusun dalam beberapa lapisan dan data *input*, **x**, mengalir dari satu lapis ke lapis berikutnya secara berurutan. Dalam tiap-tiap lapis *input* ditransformasikan kedalam lapis secara nonlinier oleh elemen-elemen proses dan kemudian diproses maju ke lapisan berikutnya. Akhirnya nilai *output* $\hat{y}$, yang dapat berupa nilai-nilai vektor dihitung pada lapis *output*.

$$\hat{y}_{(k)} = f^0\left[\sum_{j=1}^{q}\left[w_{kj} f_j^h\left(\sum_{i=1}^{p} v_{ji} x_{i(k)} + v_{j0}\right) + w_{k0}\right]\right] \quad (8)$$

Dengan,

$\hat{y}_{(k)}$: nilai dugaan dari variabel *output*

f^0 : fungsi aktivasi pada *neuron* lapis *output*

w_{kj} : bobot dari *neuron* ke- *j* di lapis tersembunyi yang menuju *neuron* pada lapis *output*

f_j^h : fungsi aktifasi di *neuron* ke- *j* pada lapis tersembunyi, (*j*= 1,2,…,*q*)

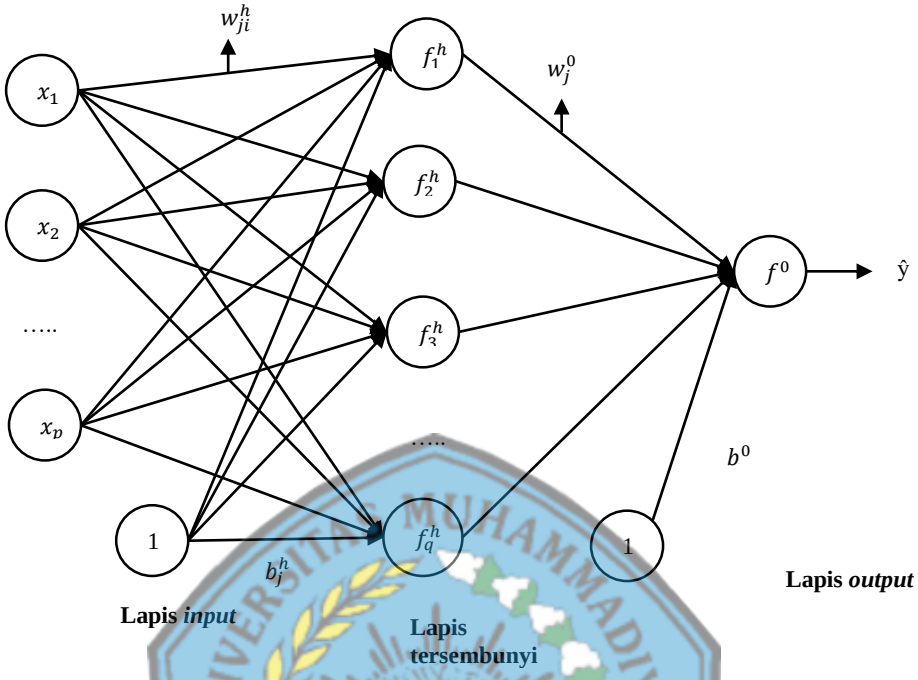
v_{ji} : bobot dari *input* ke-*i* yang menuju *neuron* ke- *j* pada lapis tersembunyi, (*j*= 1,2,…,*q*)

$x_{i(k)}$: variabel *input* sebanyak *p*, (*i* = 1,2,…,*p*)

v_{j0} : bias pada *neuron* ke- *j* pada lapis tersembunyi, (*j*= 1,2,…,*q*)

w_{k0} : bias pada *neuron* di lapis *output*

http://repository.unimus.ac.id



Gambar 2.5 Arsitektur FFNN dengan satu lapis tersembunyi, *p unit input*, *q unit neuron* dilapis tersembunyi dan satu *neuron output*.

2.6 Algoritma Backpropagation

Keberadaan pendekatan suatu fungsi adalah tidak ada gunanya apabila tidak diketahui cara untuk menemukan pendekatan tersebut (Ripley, 1996), hal tersebut yang mendorong dilakukannya penelitian tentang NN selama bertahun-tahun. Pembelajaran (Rumelhart dan McClelland, 1986) yang mengawali ide dasar tentang pendekatan NN yang digunakan untuk melakukan *fitting* terhadap parameter-parameter fungsi dengan metode *least squares*. Misalkan pada penelitian ini mempunyai beberapa pasang sampel *input* dan target $(x_{(k)},y_{(k)})$ dan *output* dari *network* adalah $\hat{y} = f(x,w)$. Maka, vektor parameter **w** dipilih dengan cara meminimumkan:

http://repository.unimus.ac.id

$$E(w) = \sum_{k=1}^{n}[y_{(k)} - f(x_{(k)}; w)]^2 \quad (9)$$

Tujuan utama dari pelatihan *algoritma backpropagation* adalah menemukan suatu penyelesaian **w** pada permasalahan optimasi, dimana **w** adalah indeks dari bobot yang optimal.

Pelatihan *algoritma backpropagation* terdiri dari tiga fase antara lain:

Fase I : *Propagasi* Maju

Propagasi maju artinya, sinyal yang diterima dari *input* (*xi*) dipropagasi menuju lapisan tersembunyi menggunakan fungsi aktivasi yang telah ditentukan. Keluaran dari setiap *unit* lapisan tersembunyi (*zj*), kemudian dipropagasi maju lagi ke lapisan tersembunyi selanjutnya menggunakan fungsi aktivasi yang telah ditentukan. Dilakukan berulang-ulang hingga menghasilkan keluaran jarring/*Output* (*yk*).

Berikutnya, *output* jaringan dibandingkan dengan target yang harus dicapai (*t*). Selisih *t-yk* merupakan kesalahan yang terjadi. Jika kesalahan ini lebih kecil dari toleransi yang ditentukan, maka iterasi dihentikan. Akan tetapi apabila kesalahan masih lebih besar dari batas toleransinya, maka bobot setiap garis dalam haringan akan dimodifikasi untuk mengurangi kesalahan yang terjadi.

Fase II: *Propagasi* Mundur

Berdasarkan kesalahan *t-yk*, dihitung factor δ_k (k= 1,2,3,…,m) yang dipakai untuk mendistribusikan kesalahan di *unit yk* ke semua *unit* tersembunyi yang terhubung langsung dengan *yk*. δ_k dipakai untuk mengubah bobot garis yang

http://repository.unimus.ac.id

menghubungkan langsung dengan bobot keluaran/*output*. Dengan cara yang sama, dihitung bobot pada setiap lapisan tersembunyi sebagai dasar perubahan bobot semua garis yang berasal dari *neuron* di lapisan terbawah. Demikian seterusnya hingga factor δ pada *neuron* tersembunyi yang berhubungan langsung dengan *neuron input* dihitung.

Fase III: Perubahan Bobot

Setelah semua factor δ dihitung, kemudian semua bobot garis dimodifikasi secara bersamaan. Perubahan bobot suatu garis didasarkan atas factor δ *neuron* di lapisan atasnya. Sebagai contoh, perubahan bobot garis yang menuju ke lapisan keluaran didasarkan atas dasar δ_k yang ada pada *neuron* keluaran.

Ketiga fase diulang terus menerus sampai kondisi penghentian terpenuhi. Umumnya kondisi penghentian yang dipakai adalah jumlah iterasi atau kesalahan. Iterasi akan dihentikan ketika jumlah iterasi yang dilakukan telah mencapai jumlah maksimum iterasi yang ditetapkan, akan tetapi jika kesalahan yang terjadi kurang dari batas toleransi yang diijinkan.

Algoritma Pelatihan *Backpropagation* terdiri dari dua proses, *feedforward* dan *backpropagation* dari errornya. Berikut merupakan *algoritma* pelatihan *Backpropagation* dengan satu lapisan tersembunyi (dengan fungsi aktivasi sigmoid biner):

a. Langkah 0 : Inisialisasi bobot (ambil bobot awal dengan nilai random yang cukup kecil).

http://repository.unimus.ac.id

b. Langkah 1 : Jika kondisi penghentian belum terpenuhi, lakukan langkah 2 – 9.

c. Langkah 2 : untuk setiap pasang data pelatihan, lakukan langkah 3 – 8.

d. Fase I : Feedforward

Langkah 3 : tiap *unit* masukan (x_i, $i = 1,2,3,\ldots,n$) menerima sinyal dan meneruskannya ke *unit* selanjutnya yaitu lapisan tersembunyi

Langkah 4 : hitung semua keluaran pada lapisan tersembunyi (z_j, $j = 1,2,3,\ldots.,p$)

$$Z\_net_j = v_{jo} + \sum_{i=1}^{n} x_i\, v_{ji} \quad (10)$$

Dengan:

Z_net_j : *input* untuk *unit* tersembunyi z_j
vjo : bias pada *unit* tersembunyi
x_i : *unit input* (data sudah dinormalisasi)

digunakan fungsi aktivasi sigmoid biner untuk menghitung sinyal *output*nya:

$$Zj = f\,(Z\_netj) = \frac{1}{1+e^{-z\_netj}} \quad (11)$$

Dengan:

Zj : sinyal keluaran yang merupakan hasil dari aktivasi Z_inj
e^{-z_inj} : eksponensial dari Z_inj

Dan kirimkan sinyal tersebut ke semua *unit* lapisan atasnya (*unit-unit output*).

Langkah ini dilakukan sebanyak jumlah lapisan tersembunyi.

Langkah 5 : hitung semua keluaran jaringan di lapisan *output* (y_k, $k = 1,2,\ldots,m$)

http://repository.unimus.ac.id

$$Y\_net_k = w_{k0} + \sum_{j=1}^{p} z_j\, w_{kj} \quad (12)$$

Dengan:

Y_in_k : masukan untuk *unit* keluaran *yk*
w_{k0} : bias pada *unit* keluaran *k* atau bias akhir
w_{kj} : bobot antara lapisan *output* dengan lapisan masukan yang telah disesuaikan

digunakan fungsi aktivasi sigmoid biner untuk menghitung sinyal *output*nya:

$$Y_k = (y\_net_k) = \frac{1}{1+e^{-y\_netk'}} \quad (13)$$

Dengan:

Y_k : sinyal keluaran yang merupakan hasil dari aktivasi
$e^{-y\_netk'}$: eksponensial y_net_k

e. Fase II : *Backpropagation*

Langkah 6 : hitung faktor δ *unit* keluaran berdasarkan kesalahan di setiap *unit* keluaran$(y_k, k = 1,2,\dots,m)$

$$\delta_k = t_k - y_k\, f'(y\_in_k) \quad (14)$$

δ merupakan *unit* kesalahan yang akan dipakai dalam perubahan bobot layar di bawahnya (langkah 7)

Kemudian hitung koreksi bobot (yang nantinya akan digunakan untuk memperbaiki w_{kj}) dengan laju percepatan α.

$$\Delta w_{kj} = \alpha.\delta_k.z_j \quad (15)$$

Kemudian hitung juga koreksi bias (yang nantinya akan digunakan untuk memperbaiki nilai w_{k0})

$$\Delta w_{k0} = \alpha\delta_k \quad (16)$$

http://repository.unimus.ac.id

Langkah 7 : Hitung faktor δ *unit* tersembunyi berdasarkan kesalahan di setiap *unit* tersembunyi (z_j, = 1,2,… ,p)

$$\delta\_net_j = \sum_{k=1}^{m} \delta_k \, w_{kj} \quad (17)$$

Faktor δ *unit* tersembunyi :

$$\delta_j = \delta\_net_j \, f' \, z\_net_j \quad (18)$$

Kemudian hitung koreksi bobot (yang nantinya akan digunakan untuk memperbaiki nilai v_{ji})

$$\Delta v_{ji} = \alpha.\delta_j.x_i \quad (19)$$

Kemudian hitung juga koreksi bias (yang nantinya akan digunakan untuk memperbaiki nilai v_{j0})

$$\Delta v_{j0} = \alpha.\delta_j \quad (20)$$

f. Fase III : Perubahan bobot

Langkah 8 : Tiap-tiap *unit output* (Y_K, k = *1,2,… ,m*) memperbaiki bobotnya (j = 0,1,2,...,p)

$$w_{kj} \; baru = w_{kj} \; lama + \Delta w_{kj} \quad (21)$$

Tiap-tiap *unit* tersembunyi (Z_j, j = *1,2,3,...,p*) memperbaiki bobotnya (j=0,1,2,3,...,n)

$$v_{ji} \; baru = v_{ji} \; lama + \Delta v_{ji} \quad (22)$$

Langkah 9 : Kondisi pelatihan berhenti Model FFNN *algoritma backpropagation* secara sistematis dapat dituliskan sebagai berikut :

$$Y_k = \sum_{j=1}^{p} w_{kj}.f[v_{jo} + \sum_{i=1}^{n} x_i \, v_{ji}] + w_{k0} \quad (23)$$

http://repository.unimus.ac.id

Pengujian dilakukan melalui feedforward dengan langkah-langkah sebagai berikut :

0. Inisialisasi bobot (hasil pelatihan);
1. Untuk setiap vektor *input*, kerjakan langkah 2 – 4;
2. Untuk i=1,…,n : set aktivasi *unit input* X_1.
3. Untuk j=1,…,p :

$$Z\_net_j = v_{jo} + \sum_{i=1}^{n} x_i \, v_{ji} \quad (24)$$

$$Z_j = f\,(Z\_in_j) \quad (25)$$

4. Untuk k-1,…,p :

$$Y\_in_k = w_{k0} + \sum_{j=1}^{p} z_j \, w_{kj} \quad (26)$$

$$Y_k = f\,(y\_in_k) \quad (27)$$

http://repository.unimus.ac.id