BAB 2
LANDASAN TEORI

2.1 Pengantar *Forecasting*

Forecasting adalah proses untuk membuat pernyataan atas suatu kejadian dimana kejadian tersebut belum diketahui atau diobservasi (www.wikipedia.org). Hal yang biasanya dilakukan dalam *forecasting* adalah mengestimasi *expected value* suatu variabel yang akan diteliti di masa mendatang. *Forecating* dapat dilakukan menggunakan metode statistik misalnya menggunakan data *time series* dan data *cross section,* ataupun bisa juga dengan menggunakan metode sederhana seperti *judgement* dari pembuatnya.

Forecasting berperan sangat penting dalam bisnis. Kemampuan untuk memprediksi secara akurat kejadian di masa depan menjadi dasar dalam pengambilan keputusan. Kemampuan *forecasting* banyak dipakai di bidang marketing, produksi, pengendalian inventori, dan banyak aktivitas bisnis lainnya.

Dalam bidang ekonomi dan pasar modal, *forecasting* yang akurat sangat penting bagi para *investor* dalam melakukan pengambilan keputusan. *Forecast* yang akurat mampu memberikan pengaruh yang signifikan terhadap *investor* seperti peningkatan return, pengurangan risiko, dan keputusan-keputusan *investor* lainnya. Hal tersebut mendorong sejumlah peneliti untuk melakukan penelitian mengenai peningkatan keakuratan model seperti yang dilakukan oleh Tjung (2010, p.7), Granger (2001, p.14), dan Soderlind (2008, p.8).

Penelitian telah banyak dilakukan untuk mendapatkan model *forecast* yang paling baik, karena dalam *forecasting* selalu mengandung unsur kesalahan / *error*. Secara statistik, *error* dikategorikan ke dalam *error* yang dapat diterima, dan *error* yang tidak dapat diterima. *Error* yang dapat diterima disebut juga *random error,* yang mana kejadiannnya terjadi secara acak. *Error* tidak dapat diterima jika kemunculannya sering dapat dipastikan, yang disebut *systematic error.* Model *forecast* tergolong sebagai model yang baik jika model *forecast* tersebut dapat menghilangkan *systematic error* (Sianturi, 1996, p.67).

2.2 *Forecasting* Harga Saham

Karakteristik umum harga saham adalah memiliki tingkat ketidakpastian yang menimbulkan risiko bagi investasi yang dilakukan *investor*. Untuk menghadapi ketidakpastian pergerakan harga, *investor* dapat melakukan prediksi dengan melakukan prediksi (*forecasting*) harga saham (Van den Goorbergh, 1999, p.8).

Fama (1970) menyatakan dalam Teori *Efficient Market Hypothesis* bahwa memprediksi saham adalah suatu kegiatan yang tidak mungkin, karena segala informasi yang digunakan untuk memprediksi harga saham telah tercermin seluruhnya di harga saham tersebut (Tjung, 2010, p.7). Para peneliti berusaha membuktikan bahwa meskipun sulit, memprediksi harga saham tetap dapat dilakukan. Hal ini dilakukan dengan cara mengurangi *error* dari prediksi tersebut. *Error* dari prediksi dapat dikurangi dengan membuat model *forecast* yang baik, seperti yang dilakukan oleh (Tjung, 2010, p.7) menggunakan *Ordinary Least Square* dan *Artificial Neural Network* (ANN). Model *forecast* harga saham menggunakan (*Autoregressive Moving Average /* ARIMA) dibandingkan dengan ANN telah dilakukan oleh Portugal (1995, p.2), Lawrence (1997), dan Zhang (2004b, p.6).

2.3 Harga Saham

Harga saham pada umumnya berubah setiap waktu. Jika harga saham naik, berarti terdapat *capital gain* bagi para *investor*, begitu pula sebaliknya, jika harga saham turun, maka *investor* akan menderita *capital loss*. Harga saham ditentukan oleh dua hal, yakni nilai intrinsik dan nilai ekspektasi (Sianturi, 1996, p.67). Nilai intrinsik dihitung dari pendapatan perusahaan, *asset* perusahaan, dan dividen. Nilai ekspektasi dilihat dari pengharapan *investor* tentang harga saham tersebut di masa depan. Jika para *investor* beranggapan bahwa harga saham tersebut di masa depan akan turun, maka akan berpengaruh kepada turunnya harga saham saat ini, begitu pula kondisi sebaliknya.

Harga saham intrinsik dapat dihitung menurut analisis teori *fundamental*, sedangkan nilai ekspektasi didapat dari analisis teori teknikal. Kedua perhitungan ini disebutkan oleh Tweles sebagai teori konvensional dan teori *confidence* seperti dikutip oleh (Sianturi, 1996, p.15). Teori konvensional (*The Conventional Theory*

of Stock Price) menjelaskan bahwa harga saham bergerak sebagai antisipasi terhadap perubahan pendapatan perusahaan. Kondisi ini disebut kondisi *fundamental* perusahaan. Teori kedua, teori *confidence* (*the confidence theory of stock price*), menjelaskan bahwa faktor-faktor yang menyebabkan pergerakan harga saham adalah pengharapan para *investor* bahwa harga saham, pendapatan, dan dividen di masa mendatang akan bergerak naik atau turun. Teori ini mengikutsertakan sentimen pasar dalam perhitungannya. Teori ini disebut juga teori teknikal. Karya akhir ini memfokuskan pada analisis teknikal.

2.4 Indeks Harga Saham

Indeks harga saham merupakan ukuran untuk mengetahui perubahan harga kumpulan saham tertentu berdasarkan tahun dasarnya (Sianturi, 1996, p.19). Indeks saham sering dipakai oleh *investor* untuk mengamati perubahan yang terjadi pada suatu kumpulan perusahaan. Pada dasarnya, indeks harga saham adalah pengukuran statistik yang dilakukan untuk menunjukkan peruhahan-perubahan harga-harga saham pada suatu waktu tertentu yang relatif terhadap suatu tanggal dasar tertentu. Biasanya, indeks dihitung atas sekumpulan saham.

Di Indonesia, indeks saham ada banyak macamnya, contohnya IHSG (Indeks Harga Saham Gabungan), LQ45, Kompas100, Bisnis27, dan beberapa lainnya seperti perkebunan, manufaktur, dan pertambangan. Di luar Indonesia terdapat indeks bursa saham negara lain seperti KLSE (*Kuala Lumpur Stock Exchange /* indeks Bursa Kuala Lumpur), STI (*Strait Times Index /* indeks bursa Singapura), ataupun di Amerika terdapat DJI (*Dow Jones Industrial*) yang diperdagangkan di Bursa New York.

IHSG (Indeks Harga Saham Gabungan) adalah indeks dari semua saham yang diperdagangkan di Bursa Efek Indonesia. Perhitungan IHSG dapat dilakukan menggunakan rumus berikut. Rumus dibagi menjadi dua, yakni rumus jika tidak ada *listing* harga saham baru dan rumus jika ada *listing* saham baru di Bursa. Rumus jika tidak ada saham baru yang *listing* (Wicaksono, 2006, p.8):

$$Indeks\_H\arg a\_Individual = \frac{H\arg a\_Hari\_In}{H\arg a\_Perdana} \tag{2.1}$$

$$IHSG = \frac{Total\_Nilai\_Pasar}{Total\_Nilai\_Dasar} \tag{2.2}$$

Rumus jika terdapat saham baru yang *listing*:

$$Nilai\_Dasar\_t = \frac{Nilai\_Pasar\_t-1 + Nilai\_Pasar\_Perdana}{Nilai\_Pasar\_Perdana} * Nilai\_Dasar\_t-1 \quad (2.3)$$

$$IHSG = \frac{Nilai\_Pasar}{Nilai\_Dasar\_Baru} \quad (2.4)$$

Kekurangan dari IHSG adalah masih memperhitungkan saham-saham yang tidak aktif (Sianturi, 1996, p.22).

Penelitian ini memfokuskan kepada indeks LQ45 sebagai representasi harga saham yang diperdagangkan di Bursa Efek Indonesia. Indeks LQ45 dipilih dengan pertimbangan bahwa saham-saham tersebut masuk ke dalam kategori saham-saham yang sangat likuid. Kriteria likuid sangat penting dalam proses *forecasting* data *time series* karena semakin likuid maka berarti saham–saham tersebut ditransaksikan secara aktif di pasar, sehingga semakin banyak data harga historis yang dapat digunakan dalam penelitian ini.

Berdasarkan informasi dari Bursa Efek Indonesia (BEI, 2007), indeks LQ45 ini terdiri dari 45 saham yang dipilih setelah melalui beberapa kriteria sehingga indeks ini terdiri dari saham-saham yang mempunyai likuiditas yang tinggi dan juga mempertimbangkan kapitalisasi pasar dari saham-saham tersebut. Untuk masuk dalam pemilihan tersebut, sebuah saham harus memenuhi kriteria sebagai berikut :

a) Masuk dalam *top* 60 dari total transaksi saham di pasar reguler (rata-rata nilai transaksi selama 12 bulan terakhir),
b) Masuk dalam ranking yang didasarkan pada nilai kapitalisasi pasar (rata-rata kapitalisasi pasar selama 12 bulan terakhir),
c) Telah tercatat di BEI sekurang-kurangnya 3 bulan,
d) Kondisi keuangan perusahaan, prospek pertumbuhan perusahaan, frekuensi dan jumlah transaksi di pasar reguler.

2.5 Model *Forecast*

2.5.1 *Time series forecasting*

Ada banyak model dalam *forecasting.* Salah satu yang paling populer adalah *time series forecasting. Time series* adalah sekumpulan data yang diukur berdasarkan waktu yang berturutan dengan pencacahan waktu yang sama. Analisa *time series forecasting* bertujuan untuk memprediksi data di masa depan berdasarkan data-data di masa lalu (Zhang, 2004b, p.2).

Dalam *time series forecasting,* data historis dari variabel-variabel yang digunakan untuk memprediksi terlebih dahuu dikumpulkan dan dianalisa untuk membentuk model yang dapat menggambarkan hubungan-hubungan di antara observasi yang berbeda waktu. Model yang didapat kemudian digunakan untuk memodelkan pergerakan data di masa depan (Peter, 2004, p.5)**.**

Time series forecasting dapat dikelompokkan menjadi 3 golongan, sebagai berikut (Iskandar, 2005, p.10):

a) *Subjective*: *forecast* yang dibuat berdasarkan *judgement,* intuisi, dan pengetahuan sebelumnya.
b) *Univariate* (regresi sederhana): mem-*forecast* nilai di masa depan hanya berdasarkan satu tipe dari nilai-nilai di masa lalu. Contohnya adalah: ekstrapolasi kurva trend, *exponential smoothing.*
c) *Multivariate* (regresi *multivariate*): mem-*forecast* berdasarkan nilai dari satu atau lebih variabel. Contohnya adalah: model *multiple* regresi linear, dan model-model ekonometri.

Menurut Gujarati seperti dikutip oleh Wicaksono (2006, p.28), asumsi yang cukup penting dari suatu *time series* adalah *time series* dibangkitkan dari suatu proses stokastik, artinya rangkaian pengamatan *time series* yang nilainya tidak dapat dirumuskan secara pasti, tetapi dirumuskan dengan pendekatan probabilistik dengan setiap nilai dari suatu rangkaian pengamatan tersebut berasal dari suatu variabel random dengan distribusi tertentu. Secara umum, *time series* pada waktu t_1, t_2, t_3, ..., t_n berupa variabel *random* Z_1, Z_2, Z_3, ..., Z_n dengan fungsi distribusi probabilitas $P(Z_1, Z_2, Z_3, ..., Z_n)$. Urutan variabel random $(Z_1, Z_2, Z_3, ..., Z_n)$ disebut proses stokastik.

2.5.2 *ARIMA (Auto Regressive Integrated Moving Average)*

Tidak seperti model regresi, yang mana Y_t dapat dijelaskan oleh *k* buah *regressor* $X_1, X_2, X_3, ..., X_k$, model ARIMA memungkinkan Y_t dijelaskan oleh nilai *Y* sendiri di masa lalu ditambah dengan *stochastic error terms* (Gujarati, 2003, p. 867). Model ARIMA dapat dibentuk dari dua buah model, yakni model AR(*p*) (*Auto Regressive*) dan model MA(*q*) (*Moving Average*). Model AR berbentuk hubungan antara variabel dependen *Y* dengan variabel dependen *Y* waktu sebelumnya. Model MA menunjukkan hubungan variabel dependen *Y* terhadap nilai-nilai residual pada waktu sebelumnya secara berturutan (Nachrowi, 2006, p. 83). *P* menunjukkan orde *autoregressive,* dan *q* menunjukkan notasi orde dari *moving average.* Proses AR dan MA dapat dimodelkan sebagai berikut (Gujarati, 2003, p. 867):

$$(Y_t-\delta) = \alpha_1(Y_{t-1} - \delta) + \alpha_2(Y_{t-2} - \delta) + ... + \alpha_p(Y_{t-p} - \delta) + \mu_t \quad (2.5)$$

$$Y_t = \mu + \beta_0\mu_t + \beta_1\mu_{t-1} + \beta_2\mu_{t-2} + ... + \beta q\mu_{t-q} \quad (2.6)$$

$$Y_t = \theta + \alpha_p(Y_{t-p} - \delta) + \beta q\mu_{t-q} \quad (2.7)$$

Rumus (2.5) disebut model AR, dan rumus (2.6) disebut model MA. Gabungan keduanya disebut model ARMA. Contoh model ARMA(1,1) dapat dilihat pada rumus (2.7).

Rumus AR dan MA di atas mengasumsikan bahwa data mengikuti proses yang stasioner. Jika data tidak stasioner, maka perlu melakukan *differencing* seperti disebutkan pada sub bab 2.5, menghasilkan ARIMA (*p,d,q*), dimana *d* menunjukkan orde *differencing* sampai didapat data yang stasioner. Model ARIMA *(p,d,q)* dapat dinyatakan dalam Nachrowi (2006, p.83):

$$\phi(B)\Delta^d Y_t = \delta + \theta(B)e_t \quad (2.8)$$

$$\phi(B) = 1 - \phi_1(B) - \phi_2(B)^2 - ... - \phi_{p1}(B)^p \quad (2.9)$$

$$\theta(B) = 1 - \theta_1 B - \theta_2 B^2 - ... - \theta_q B^q \quad (2.10)$$

Dimana (2.8) adalah model ARIMA (*p,d,q*), (2.9) adalah AR(*p*), dan (2.10) adalah MA(*q*).

Model ARIMA(p,d,q) tersebut merupakan *univariate* ARIMA, yang berarti variabel dependen *Y* hanya dipengaruhi oleh pergerakan variabel dependen Y pada waktu sebelumnya. Penelitian ini menggunakan *multivariate* ARIMA, dikarenakan variabel dependen *Y* (harga saham masa depan) juga dipengaruhi oleh banyak variabel independen (X_i) waktu sebelumnya. Model *multivariate* ARIMA dapat dituliskan sebagai berikut (Tsay, 2005, p. 16):

$$Z_t = f(Y_t) \tag{2.11}$$

$$\Delta Z_t = \mu + \sum_{i=i}^{k} \frac{Num_i}{Den_i} \Delta_i f_i(X_{it}) + \frac{AR}{MA} a_t \tag{2.12}$$

dimana,

$$Num_i = (\omega_{i0} - \omega_{i1}B - ... - \omega_{iu}B^u) * (1 - \Omega_{i1}B^s - ... - \omega_{iv}B^{vs})B^b \tag{2.13}$$

$$Den_i = (1 - \delta_{i1}B - ... - \delta_{ir}B^r)(1 - \Delta_{i1}B^s - ...) \tag{2.14}$$

Rumus (2.11) dan (2.12) merupakan rumus *multivariate* ARIMA.

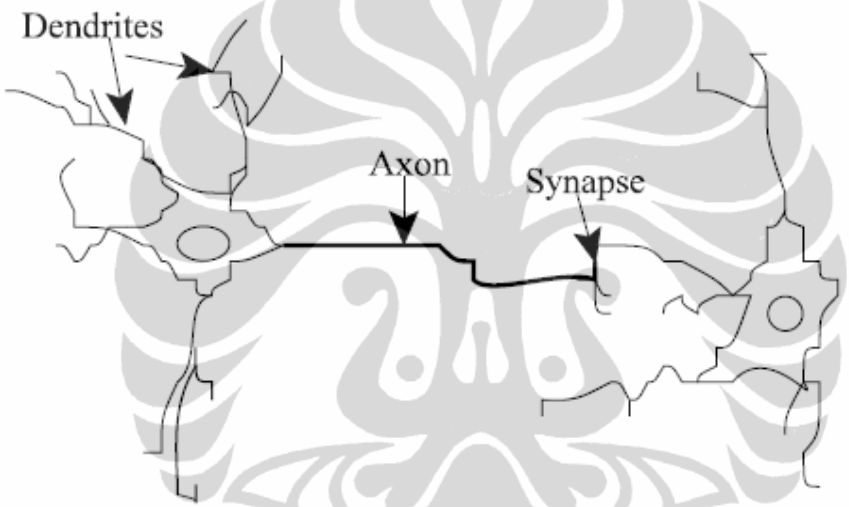
2.5.3 ANN *(Artificial Neural Network)*

Meskipun model *multivariate* ARIMA cukup fleksibel dalam memodelkan sebagaian besar pola *time series*, kekurangannya adalah bahwa ARIMA mengasumsikan model yang linier. Hal ini menyebabkan model ARIMA tidak dapat menangkap pola-pola yang non-linier yang umum terdapat pada *time series* dalam ekonomi dan bisnis. Oleh karena itu, perlu dibuat suatu model yang dapat menangkap pola-pola yang non-linier. Model tersebut adalah model ANN (*Artificial Neural Network*) (Zhang, 2004b, p.6).

Menurut Gershenson (2003, p.2), cara yang efisien untuk memecahkan problem yang kompleks adalah memecahkan permasalahan tersebut ke dalam elemen-elemen yang lebih kecil (*divide and conquer*). Model ANN dapat melakukan kegiatan semacam ini (Gershenson, 2003, p.2).

Model ANN diinspirasi dari sistem saraf makhluk hidup. Saraf menerima sinyal melalui sinapsis yang terletak di dendrit atau membran *neuron*. *Neuron*

adalah satu sel saraf. Ketika sinyal yang disampaikan cukup kuat (melampaui suatu batas / *threshold* tertentu), *neuron* akan diaktivasi dan akan mentransmisikan sinyal melalui akson, dan dapat mengaktivasi *neuron* yang lain (Gershenson, 2003, p. 2). Ketika sejumlah besar *neuron* memproses sinyal dalam waktu bersamaan, maka makhluk hidup dapat memecahkan suatu masalah tertentu yang kompleks. Untuk memecahkan permasalahan yang belum pernah dijumpai dalam hidupnya, makhluk hidup perlu belajar dari pengalaman-pengalamannya di masa lalu, untuk kemudian memutuskan solusi yang tepat.



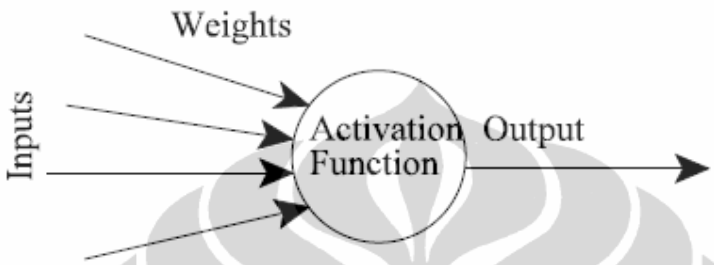
Gambar 2.1 Sistem saraf makhluk hidup

Sumber: Gershenson (2003, p. 2)

Konsep yang sama dimodelkan oleh konsep jaringan saraf tiruan (*Artificial Neural Network*) menggunakan program komputer. Ketika sejumlah besar *artificial neuron* memproses data secara bersamaan, maka permasalahan baik ekonomi, bisnis, ataupun permasalahan sehari-hari yang kompleks dapat dipecahkan dengan mudah. Sama seperti saraf makhluk hidup, *artificial neuron* perlu mendapatkan *training* / pelatihan untuk memecahkan permasalahan yang belum pernah ditemui. Memecahkan masalah – masalah yang belum diketahui ini adalah konsep *forecasting,* dimana *artificial neuron* dilatih menggunakan data-data di masa lalu, untuk kemudian dipakai untuk memecahkan masalah-masalah

di masa depan yang belum pernah diketahui sebelumya. Sejumlah besar *artificial neuron* disebut juga *artificial neural network.*

Konsep *artificial neuron* yang menjadi dasar ANN dapat digambarkan seperti Gambar 2.2 berikut ini:



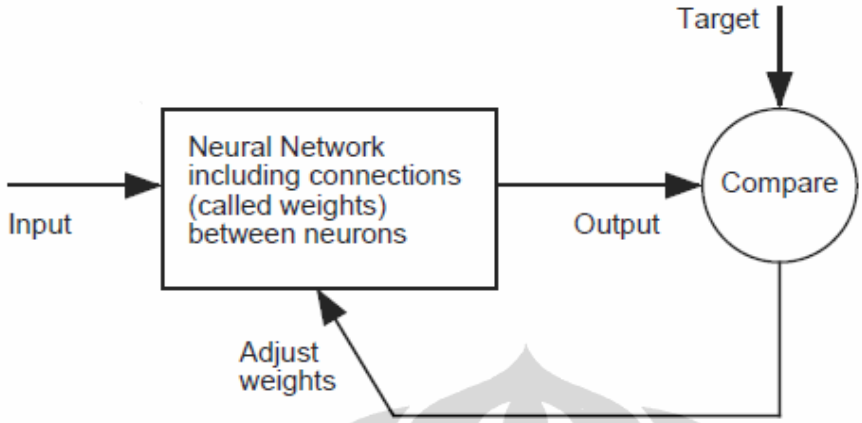
Gambar 2.2 Konsep *Artificial Neuron*

Sumber: Gershenson (2003, p.2)

Pada dasarnya, *artificial neuron* terdiri dari *input* (seperti sinapsis), yang kemudian dimultiplikasi oleh *weights* (kekuatan sinyal) dan kemudian dihitung oleh fungsi matematik yang dilambangkan dengan fungsi aktivasi. Jika hasil perhitungan melampaui batas tertentu, maka hasil perhitungan akan ditransmisikan melalui *output* ke *neuron* yang lain (Gershenson, 2003, p. 3).

Semakin besar *weights*, maka semakin kuat sinyal *input* yang dimasukkan ke dalam *neuron*. Dengan mengatur besarnya *weights*, kita bisa mendapatkan *output* yang diinginkan dengan menggunakan *input* tertentu. Jika terdapat ratusan atau ribuan *neuron,* sulit menemukan perhitungan yang tepat jika menggunakan perhitungan manual. Oleh karena itu digunakan algoritma yang secara otomatis melakukan perhitungan *weights* ini agar didapat *weights* yang dapat memetakan *input* menjadi *output* yang sesuai. Proses perubahan *weights* secara otomatis inilah yang disebut pelatihan / *training* pada *artificial neural network.*

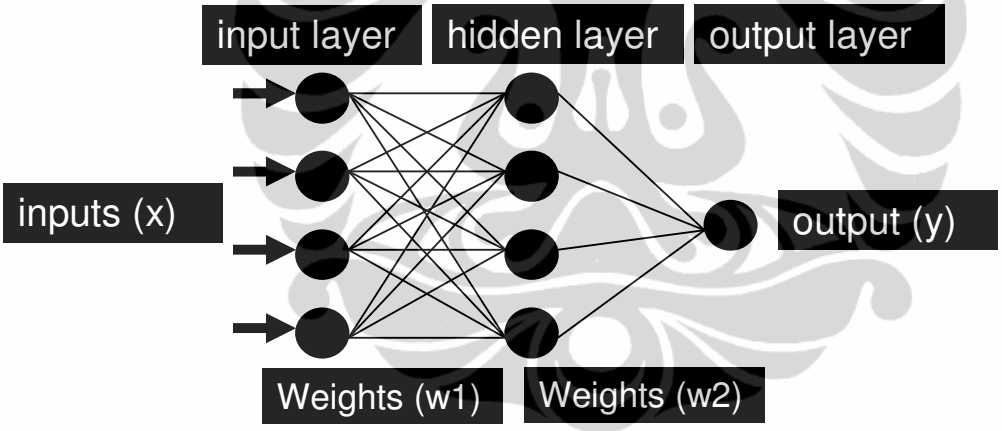
Gambar 2.3 memperlihatkan proses perubahan *weights* yang membandingkan antara *input* dengan *target*, sehingga didapatkan *weights* yang sesuai.



Gambar 2.3 Proses pelatihan ANN

Sumber: Demuth (2009, p. 22)

Pada Gambar 2.3 terdapat abstraksi dari *neural network* yang pada dasarnya dapat memiliki susunan *aritificial neuron* seperti pada Gambar 2.4 berikut:



Gambar 2.4 Detail ANN

Sumber: Zhang, 2004a, p. 3. Telah diolah kembali.

Pada Gambar 2.4 terlihat ANN selain mempunyai *neuron, weights, input*, dan *output*, ANN juga mempunuyai *hidden layer. Hidden layer* berada di antara *input* dan *output*. *Hidden layer* dapat terdiri dari satu atau lebih *neuron*. Model ANN bisa mempunyai satu atau lebih *hidden layer*.

2.5.3.1 *Hidden Layer* dan *Neuron*

Hidden layer berpengaruh terhadap kemampuan model ANN untuk menggeneralisasi / aproksimasi suatu fungsi / pola. Penelitian yang dilakukan oleh Iskandar (2006, p. 21) menghasilkan informasi bahwa cukup diperlukan satu atau dua *hidden layer* saja untuk fungsi aproksimasi ini.

Jumlah *neuron* yang optimal pada *hidden layer* berpengaruh terhadap keakuratan model ANN dalam mem-*forecast* suatu deret data. Jika jumlah *neuron* terlalu banyak, maka model cenderung akan tidak mampu menggeneralisasi (menjadi *overfitting*). Ini berarti model hanya berfungsi baik hanya di ruang sampelnya saja, dan apabila di-*test* menggunakan data yang di luar sampel, maka model ANN akan tidak dapat memprediksi dengan baik. Jika jumlah *neuron* terlalu sedikit, maka yang terjadi adalah model akan tidak dapat memprediksi dengan baik meskipun di-*test* menggunakan data yang berada di dalam sampel.

2.5.3.2 Algoritma *Backpropagation*

Pada algoritma *backpropagation,* vektor *input* dan vektor *target* digunakan untuk melatih ANN sampai ANN dapat mengaproksimasi sebuah deret data. *Error* / selisih yang terjadi antara *target* dengan data yang sebenarnya, dipropagasikan / ditransmisikan kembali kembali ke dalam network melalui *hidden layer* menuju ke *neuron input*. *Weights* kemudian akan dihitung kembali dan masing-masing *neuron* akan mentransmisikan kembali sinyal ke *hidden layer dan neuron output* untuk kemudian akan dihitung kembali *error* / selisih antara *target* dengan data sebenarnya. Proses ini diulang terus menerus sampai *error* yang terjadi berada dalam *range* yang ditentukan di awal (Demuth, 2009, p. 22).

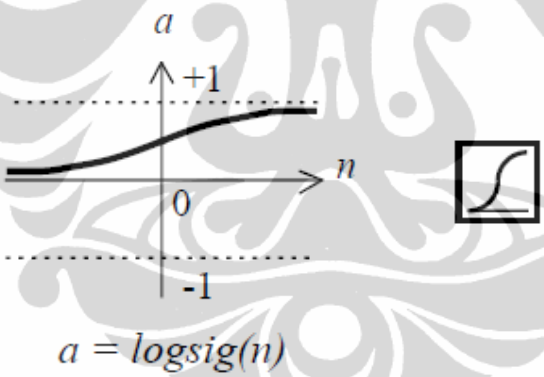
Menurut Siang (2005), algoritma *backpropagation* meliputi tiga fase. Fase pertama adalah fase maju. Pola *input* dihitung maju mulai dari *input* layer hingga *output* layer menggunakan fungsi aktivasi yang ditentukan. Fungsi aktivasi akan dibahas di sub bab berikutnya. Fase kedua adalah fase mundur. Selisih antara *output* dengan *target* adalah *error* yang terjadi. *Error* tersebut dipropagasikan mundur, mulai dari *output* layer, *hidden layer,* sampai ke *input layer.* Fase ketiga adalah modifikasi *weights* untuk menurunkan *error* yang terjadi.

2.5.3.3 Fungsi Aktivasi

Fungsi aktivasi digunakan untuk menentukan keluaran / *output* suatu *neuron*. Parameter fungsi aktivasi adalah kombinasi linier antara *input* dan *weights.* Dalam *backpropagation*, fungsi aktivasi yang dipakai harus memenuhi beberapa syarat, yaitu kontinu, terdiferensial dengan mudah, dan merupakan fungsi yang tidak turun (Siang, 2005).

Menurut Buwana (2006, p.19), fungsi aktivasi pada ANN digunakan untuk memformulasikan *output* dari setiap *neuron*. Pada ANN, terdapat tiga jenis fungsi aktivasi, yakni fungsi aktivasi sigmoid *logistic*, fungsi aktivasi sigmoid *tangent*, dan fungsi aktivasi linier.

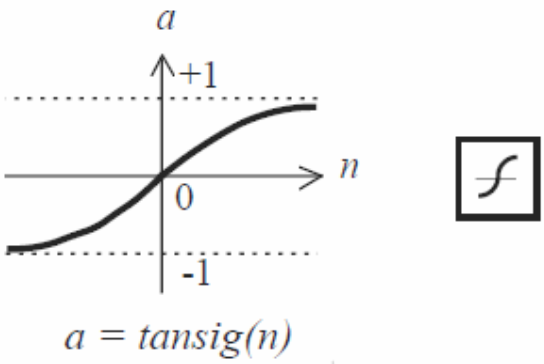
Fungsi aktivasi sigmoid *logistic* memetakan nilai *input* antara 0 dan +1. Nilai yang ditransmisikan / diaktivasi ke *neuron* lain berada pada jangkauan 0 dan +1. Grafik fungsi sigmoid *logistic* dapat dilihat pada Gambar 2.5.



Gambar 2.5 Fungsi aktivasi *sigmoid logistic*

Sumber: Demuth (2009, p. 80)

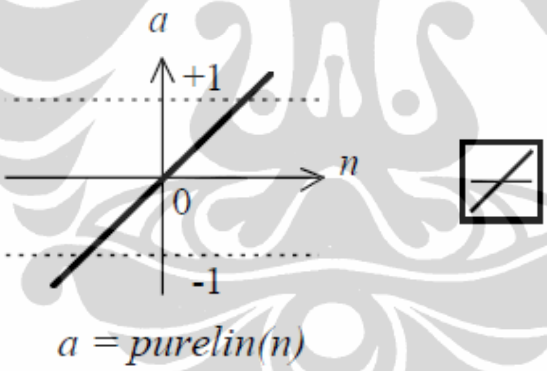
Fungsi aktivasi sigmoid *tangent* memetakan nilai *input* antara +1 dan -1. Nilai yang ditransmisikan / diaktivasi ke *neuron* lain berada pada jangkauan +1 dan -1. Grafik fungsi aktivasi sigmoid *logistic* dapat dilihat pada Gambar 2.6.



Gambar 2.6 Fungsi aktivasi *sigmoid tangent*

Sumber: Demuth (2009, p. 163)

Fungsi aktivasi linier meneruskan nilai *input* ke *neuron* lain dengan tanpa perubahan nilai aktivasi. Grafik fungsi aktivasi linier dapat dilihat pada Gambar 2.7.



Gambar 2.7 Fungsi aktivasi linier

Sumber: Demuth (2009, p. 163)

2.5.3.4 *Preprocessing* dan *Postprocessing*

Pelatihan / *training* pada ANN dapat lebih efisien jika data *input* dan data *target* dilakukan pemrosesan terlebih dahulu. Jika data *input* tidak dilakukan pemrosesan / ditransformasikan, ANN tidak dapat menghasilkan *output* yang akurat (Mendelsohn, 2007, p.3). Data *input* ditransformassikan ke dalam bentuk yang lebih sederhana melalui proses *preprocessing*. Setelah data diproses oleh ANN, *output*nya dikembalikan kembali ke bentuk data semula melalui proses *postprocessing*.

Proses *preprocessing dan postprocessing* yang dilakukan dalam penelitian ini menggunakan transformasi minimum-maksimum, sehingga setiap data *input* ditransformasikan menjadi dalam range -1 sampai dengan +1. menurut (Iskandar, 2005), pemetaan seperti inilah yang berdampak maksimum terhadap keakuratan hasil *forecast* model ANN. Algoritma minumum-maksimum adalah sebagai berikut (Demuth, 2009, p. 584):

$$Y = \frac{(Y\_\max - Y\_Min) * (X - X\_\min)}{(X\_\max - X\_\min)} + Y\_\min \qquad (2.15)$$

Y adalah nilai diatara -1 dan +1. *ymax* = +1, *ymin* = -1, dan *xmax* serta *xmin* bergantung pada nilai maksimum dan minimum data *input*nya. Proses transformasi balik ke data awal dilakukan dengan menggunakan invers dari persamaan (2.15) dengan mencari variabel *X*.

2.5.3.5 *Training* Pada *Backpropagation*

Algoritma *backpropagation* mempunyai bermacam-macam jenis *training*. Macam-macam *training* tersebut diantaranya (Demuth, 2009, p. 806):

a) *Gradient descent backpropagation*

   Pada jenis *training* ini, *weights* dari ANN di-*update* berdasarkan *gradient* negatif dari fungsi perubahan *error*-nya.

b) *Gradient descent backpropagation with momentum*

   Metode ini sama dengan *gradient* descent, hanya saja ditambahkan momentum yang memungkinkan ANN merespon tidak hanya kepada *local gradient error*-nya, tetapi juga trend yang baru saja terjadi pada perubahan *error*-nya.

c) *Conjugate gradient backpropagation with Powell-Beale restarts*

   Conjugate *gradient* mencari arah penurunan yang paling curam dari *gradient* penurunan *error*-nya. Terdapat cara yang lebih efisien untuk meningkatkan efisiensi *training*, yaitu dengan menerapkan reset functions. Algoritma ini melakukan reset jika ditemui ortogonalitas yang kecil di antara *gradient error* saat ini dengan *gradient* iterasi sebelumnya.

d) *Scaled conjugate gradient backpropagation*

Setiap algoritma yang telah disebutkan sebelumnya memerlukan proses pencarian *gradient error* terkecil pada tiap iterasi pelatihannya. Ini memerlukan waktu yang cukup lama, terlebih jika ANN yang dilatih mempunyai ukuran yang besar. Algoritma *scaled conjugate gradient* dapat meminimalkan waktu pencarian tersebut, dimana mengkombinasikan pendekatan *trust-region* seperti yang dilakukan oleh algoritma Levenberg-Marquardt..

e) *BFGS quasi-newton backpropagation*

Algoritma ini merupakan alternatif dari algoritma *conjugate gradient*. Algoritma ini konvergen lebih cepat daripada algoritma *conjugate gradient*, akan tetapi perhitungannya lebih rumit dan membutuhkan kapasitas memori yang besar.

f) *Levenberg-Marquardt backpropagation*

Seperti halnya metode *quasi-newton*, algoritma ini didesain untuk meningkatkan kinerja *training*, tanpa perlu melakukan perhitungan turunan kedua dari perubahan *error*-nya. Algoritma ini menggunakan pendekatan (aproksimasi) untuk menghitung turunan kedua perubahan *error*-nya.

g) *Batch training with weight and bias learning rules*

Algoritma ini melatih ANN menggunakan *weight learning rules* dengan update secara batch. *Weights* di-*update* pada akhir setiap iterasi pelatihan ANN.

h) *Bayesian regulation backpropagation*

Algoritma *training* ini melakukan update *weights* sesuai dengan cara yang dilakukan oleh proses optimisasi Levenberg-Marquardt. Algoritma ini meminimalisasi kombinasi squared *error* dan *weights* lalu kemudian menentukan kombinasinya yang sesuai untuk menghasilkan ANN yang dapat menggeneralisasi dengan baik.

i) *Gradient descent with adaptive learning rate backpropagation*

Algoritma ini melakukan *update* pada *weight* sesuai dengan metode *gradient descent* dengan *adaptive learning rate*. Pada setiap iterasi

training, jika *gradient error* menurun ke arah target *error*, maka laju pembelajaran (*learning rate*) ditingkatkan. Jika *gradient error* tidak menurun ke arah target *error*, maka laju pembelajaran (*learning rate*) tidak akan berubah.

j) *Gradient descent with momentum and adaptive learning rate backpropagation*

Algoritma ini melakukan *update* pada *weight* sesuai dengan metode *gradient* descent dengan *adaptive learning rate* dikombinasikan dengan momentum.

k) *One-step secant backpropagation*

Algoritma BFGS di atas membutuhkan perhitungan yang rumit dan kapasitas memori yang besar dibandingkan algoritma conjugate *gradient*. Algoritma ini menggunakan pendekatan (aproksimasi) secant sehingga perhitungan rumit tersebut bisa lebih cepat dan tidak membutuhkan banyak kapasitas memori. Algoritma ini tidak menyimpan seluruh matriks turunan kedua perubahan *error*nya dengan mengasumsikan matriks turunan kedua pada iterasi sebelumnya adalah matriks identitas. Keuntungannya adalah tidak diperlukan perhitungan matriks invers dalam penentuan *gradient error*nya.

l) *Resilient backpropagation*

Algoritma ini merupakan algoritma backpropagation yang mana perubahan membesar / mengecilnya *weights* ditentukan oleh perubahan tanda (*sign*) dari turunan pertama perubahan *error*-nya.

m) *Sequential order incremental training with learning functions*

Algoritma ini men-*update* nilai tiap *weights* mengacu kepada fungsi pembelajaran / *learning* berdasarkan urutan waktu dan urutan data input.

2.6 Keakuratan *Forecast*

Dalam menentukan apakah suatu model *forecast* akurat atau tidak, ada dua aspek yang harus diperhatikan. Yang pertama perlu diketahui seberapa akurat model tersebut memprediksi harga, dan yang kedua seberapa baik model tersebut memprediksi harga jika dibandingkan dengan model lain (Kunst, 2004, p. 2). Ukuran yang umum dipakai dalam menentukan keakuratan hasil *forecast* adalah

RMSE (*Root Mean Squared Error*), MAE (*Mean Absolute Error*), dan MAPE (*Mean Absolute Percentage Error*), seperti dilakukan dalam penelitian yang dilakukan oleh Cuaresma (2004, p. 7), Iskandar (2005, p. 48), dan Buwana (2006, p. 49).

RMSE merupakan akar kuadrat rata-rata dari selisih antara *output* model dengan data yang sebenarnya. Rumus MSE adalah sebagai berikut (Buwana, 2006, p. 49):

$$RMSE = \sqrt{\frac{\sum (n_f - n_a)^2}{m}} \tag{2.16}$$

MAE merupakan hasil nilai absolut dari selisih antara nilai *output* model dengan data sebenarnya. Rumus MAE adalah sebagai berikut (Buwana, 2006, p. 49):

$$MAE = \frac{\sum |n_f - n_a|}{m} \tag{2.17}$$

Sedangkan MAPE adalah perhitungan MAE yang hasilnya dalam bentuk persentase, seperti rumus berikut (Buwana, 2006, p. 49):

$$MAPE = \frac{1}{m} * \frac{\sum |n_f - n_a|}{n_a} * 100\% \tag{2.18}$$

Penjelasan rumus tersebut di atas adalah sebagai berikut:

n_f = harga saham *forecast*

n_a = harga saham *aktual*

m = jumlah data

Untuk mengetahui apakah keakuratan *forecast* model yang berbeda, signifikan secara statistik atau tidak, maka perlu dilakukan uji keakuratan *forecast*. Uji yang dilakukan adalah uji Diebold-Mariano. Beberapa penelitian, seperti Cuaresma (2004, p. 8), Chung (2006, p. 22), dan McNellis (2005, p. 96) menggunakan uji Diebold-Mariano dalam mengukur apakah suatu model akurat memprediksi suatu data dibandingkan dengan model lain. Uji Diebold-Mariano menggunakan *loss criterion* dalam perhitungannya.

Jika ada suatu deret data aktual dan dua hasil prediksi, *loss criterion* dapat dihitung berdasarkan (MSE / *Mean Squared Error*) untuk menghitung ukuran keakuratan prediksi, yang dihipotesiskan sebagai berikut (Diebold, 1995, p. 253):

- Ho: Keakuratan *forecast* tidak signifikan lebih akurat

- H_1: Keakuratan *forecast* signifikan lebih akurat

Kriteria uji: tolak Ho jika nilai signifikansi statistik p-*value* < 0.05.

Detail metode Diebold-Mariano adalah sebagai berikut (Diebold, 1995, p. 253):

Misalkan *forecast error* dari kedua model adalah

$$\varepsilon^1_{t+h|t} = y_{t+h} - y^1_{t+h|t} \tag{2.19}$$

$$\varepsilon^2_{t+h|t} = y_{t+h} - y^2_{t+h|t} \tag{2.20}$$

h adalah *forecast h* langkah ke depan, uji Diebold-Mariano dibuat berdasarkan *loss differential* sebagai berikut:

$$d_t = L(\varepsilon^1_{t+h|t}) - L(\varepsilon^1_{t+h|t}) \tag{2.21}$$

Diebold-Mariano statistik:

$$S = \frac{\overline{d}}{(a\,\overline{\mathrm{var}(d)})^{0.5}} = \frac{\overline{d}}{(LRV_{\overline{d}}/T)^{0.5}} \tag{2.31}$$

$$\overline{d} = \frac{1}{T_0}\sum_{t=t_0}^{T} d_t \tag{2.22}$$

$$LRV_{\overline{s}} = \gamma_0 + 2\sum_{j=1} \gamma_j, \gamma_j = \mathrm{cov}(d_t, d_{t-j}) \tag{2.23}$$