ANDRÉ TERRERI RODRIGUES
GUILHERME ROCHA GOMES
PAULO ROBERTO MOURA CUNHA

APLICAÇÃO DE REDES NEURAIS NA PREVISÃO DE SÉRIES ECONÔMICO-FINANCEIRAS

Projeto de formatura apresentado à disciplina PCS2502 – Laboratório de Projeto de Formatura II, da Escola Politécnica da Universidade de São Paulo.

São Paulo
2004

ANDRÉ TERRERI RODRIGUES
GUILHERME ROCHA GOMES
PAULO ROBERTO MOURA CUNHA

APLICAÇÃO DE REDES NEURAIS NA PREVISÃO DE SÉRIES ECONÔMICO-FINANCEIRAS

Projeto de formatura apresentado à disciplina PCS2502 – Laboratório de Projeto de Formatura II, da Escola Politécnica da Universidade de São Paulo.

Área de Concentração:
Engenharia da Computação

Orientador:
Prof. Doutor
Jorge Kinoshita

São Paulo
2004

"O futuro não é mais incerto do que o presente"

Walt Whitman , poeta americano.

RESUMO

O presente trabalho tem como fim estudar a aplicação de redes neurais na previsão de séries econômico-financeiras, para tanto foram criados modelos econométricos ARIMA-GARCH e modelos baseados em redes neurais, comparando-se os resultados obtidos com as duas abordagens. A preocupação do texto está centrada na teoria econométrica por trás dos modelos criados, nas arquiteturas de redes neurais e algoritmos de treinamento disponíveis, e na comparação qualitativa dos resultados.

Das diversas séries temporais estudadas, a apresentada neste texto é a de preços de fechamento da ação preferencial da Petrobrás, devido aos resultados obtidos nos modelos, econométricos e baseados em redes neurais artificiais, elaborados.

ABSTRACT

The present work has as its objective to study the application of artificial neural networks in the forecast of economic-financiers time series, for that had been created ARIMA-GARCH econometrical models and models based on neural networks, comparing the results gotten with the two boardings. The concern of the text is centered in the econometrical theory of the models created, in the architectures of neural networks and training algorithms available, and in the qualitative comparison of the results.

Of the diverse studied time series, the presented one in this text is the series of the closing prices of the Petrobra's stock, due to the results gotten in the elaborated econometrical models and based in artificial neural networks.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1 INTRODUÇÃO

O que distingue os milhares de anos de história do que consideramos os tempos modernos? A resposta transcende em muito o progresso da ciência, da tecnologia, do capitalismo e da democracia.

O passado remoto foi repleto de cientistas brilhantes, de matemáticos, de inventores, de tecnólogos e de filósofos políticos. Centenas de anos antes do nascimento de Cristo, os céus haviam sido mapeados, a grande biblioteca de Alexandria fora construída e a geometria de Euclides era ensinada. A demanda por inovações tecnológicas para fins bélicos era tão insaciável quanto atualmente. Carvão, óleo, ferro e cobre estiveram a serviço dos seres humanos por milênios, e as viagens e comunicações marcaram os primórdios da civilização conhecida.

A idéia revolucionaria que define a fronteira entre os tempos modernos e o passado é o domínio do risco: a noção de que o futuro é mais do que um capricho dos deuses e de que os homens e mulheres não são passivos ante a natureza. Até os seres humanos descobrirem como transpor essa fronteira, o futuro era um espelho do passado ou o domínio obscuro de oráculos e adivinhos que detinham o monopólio sobre o conhecimento dos eventos previstos (Bernstein, 1997).

Atualmente, a previsão de séries temporais tem se tornado questão chave em diversas áreas de estudo. Estudando-se o comportamento de uma série dada, tenta-se, através de modelos pré-estabelecidos, prever o comportamento futuro desta.
A área que estuda o comportamento das séries econômicas é chamada de Econometria, e a tarefa do econometrista moderno é desenvolver modelos razoavelmente simples capazes de prever, interpretar, e testar hipóteses referentes a dados econômicos.

Possuindo-se uma grande base de dados passados, através de uma abordagem matemática, é possível decompor uma série em componentes sazonais, cíclicos, irregulares e de tendência. Podendo, então, extrapolar-se no futuro as componentes

conhecidas e prever como a série evolui. Dentre as técnicas disponíveis, destaca-se a abordagem de Box e Jenkins para estimar modelos auto-regressivos integrados de média móvel (ARIMA), e no que concerne as séries econômico-financeiras os modelos de heterocedasticidade autoregressiva condicionada (ARCH).

Outra técnica que vem sendo estudada e aplicada cada vez mais é a de redes neurais artificiais, baseada no funcionamento dos neurônios e que são poderosas ferramentas de modelagem. As características das RNs que as tornam propícias na tentativa de se prever uma série temporal são a capacidade das RNs apreenderem características não-lineares dos dados, serem multivariadas, capacidade de generalização e abstração de ruídos e distorções.

Objetivos e escopo do projeto

O objetivo deste trabalho é a comparação da aplicação das técnicas econométricas convencionais e dos modelos baseados em redes neurais artificiais (RNA) na previsão de séries econômico-financeiras. Nos capítulos seguintes será apresentada tanto a teoria econométrica quanto a de RNAs e os resultados obtidos na utilização de cada abordagem na previsão das séries temporais.

2 MODELOS ECONOMÉTRICOS.

A Econometria refere-se à análise de dados que descrevem os fenômenos econômicos. Esses dados econômicos vêm quase que exclusivamente de origens não experimentais, isto é, os dados são observacionais. Os cientistas sociais geralmente devem aceitar as condições sob as quais seus objetos de estudos estão submetidos e a forma como as respostas ocorrem. Estes pesquisadores não podem especificar ou escolher o nível de um estímulo para depois então registrar um novo resultado podendo apenas observar os experimentos naturais que acontecem (Ruud, 2000). A Econometria pode ser considerada como uma área da matemática que tem intersecção com teoria econômica e estatística. Um processo econométrico básico envolve os seguintes passos (McNelis, 2002):

1- Desenvolvimento de hipóteses testáveis da teoria econômica;
2- Especificação de hipóteses em forma matemática;
3- Especificação de um modelo estatístico ou econométrico;
4- Coleta de dados;
5- Estimação;
6- Previsão ou predição;
7- Análise política ou escolha comportamental.

A análise econométrica começa com um modelo econômico, isto é, existe alguma relação prévia que é assumida, baseada em um modelo teórico, o qual é então testado usando dados do mundo real. A Estatística é então utilizada, por exemplo, para estimar parâmetros e testar estatisticamente os modelos. Com modelos de regressão o objetivo é *predizer* os valores das variáveis dependentes *dentro* do domínio das variáveis independentes. Com modelos de séries temporais o objetivo é *prever* os valores das variáveis dependentes *fora* do domínio das variáveis independentes. Sendo que a variável independente reduz-se ao tempo.

2.1 *Séries Temporais.*

A primeira razão para o interesse na análise de séries temporais é adquirir conhecimento de padrões que se desenvolvem a medida que os eventos acontecem em um período de tempo. Isto é genericamente feito através de observações e análises dos dados passados sobre ocorrência dos eventos. Tipicamente, tais processos envolvem a consolidação das experiências históricas em sistemas matemáticos que descrevem o comportamento dos eventos através da passagem do tempo. Esta tarefa deve ser feita da forma mais concisa possível e pode ser chamada de construção do modelo. Num segundo estágio, estes modelos matemáticos são utilizados para projetar o que é provável de ocorrer durante algum curto período de tempo. Tais processos são usualmente chamados de "previsão", ou seja, uma extrapolação para além do domínio temporal conhecido. A combinação destes dois processos forma essencialmente o que se convencionou chamar de análise de séries temporais. Os resultados dessa análise podem ser utilizados de modos muito diferentes, sendo que sua aplicação é de grande importância nas áreas de Economia e Finanças.

Uma série temporal é um conjunto de observações geradas seqüencialmente no tempo (Box e Jenkins, 1976). Se o conjunto é contínuo, a série temporal é chamada de contínua. Se o conjunto é discreto, a série temporal é chamada de discreta. As observações de uma série temporal feitas nos tempos $t_1, t_2, ..., t_i, ..., t_n$ podem ser denotadas por $y(t_1), y(t_2), ..., y(t_i), ..., y(t_n)$. Tendo n sucessivos valores de uma série disponível para análise também podemos escrever $y_1, y_2, ..., y_i, ..., y_n$ para denotar as observações feitas em intervalos de tempo eqüidistantes $t_0 + h, t_0 + 2h, ..., t_0 + ih, ..., t_0 + nh$. Na prática, muitos dados de séries temporais econômicas e financeiras são coletadas em períodos de tempo discretos. Dessa forma, utilizaremos séries temporais discretas ao invés de contínuas, sendo que y_t representará uma variável aleatória observável da série temporal num dado instante t. Normalmente, as séries econômicas, financeiras ou de algum outro negócio podem possuir cinco características chave: (i) tendência, (ii) sazonalidade, (iii) algum ponto

de influência discrepante, (iv) uma variância que se altera devido as observações passadas - heterocedasticidade condicional e (v) não-linearidade. Tipicamente, uma série temporal econômica apresenta pelo menos duas ou três dessas características (Franses, 1998).

2.2 *Processos Estocásticos.*

Se os valores futuros de uma série temporal são determinados exatamente por alguma função matemática, então esta série é denominada de determinística. Se os valores futuros podem ser descritos apenas em termos de uma distribuição de probabilidade a série temporal é chamada de não-determinística ou simplesmente série temporal estocástica. Um fenômeno estatistico que envolva leis probabilísticas no tempo é chamado de processo estocástico. Neste trabalho as séries temporais serão analisadas como uma realização particular produzidas por um mecanismo de probabilidade subjacente. Ou seja, quando utilizarmos a metodologia de Box e Jenkins para analisar as séries temporais estaremos nos referindo a uma realização particular de um processo estocástico.

Um bloco construtivo básico dos modelos estocásticos discretos de séries temporais é o processo de ruído branco. A seqüência $\{\varepsilon_t\}$ é um processo ruído branco se cada valor da seqüência tiver esperança nula, variância constante e forem serialmente não correlacionados.

Se a notação **E[**x**]** denota o valor esperado teórico de x, a seqüência $\{\varepsilon_t\}$ é um processo de ruído branco se para cada instante de tempo *t* tivermos (Enders, 2003):

$$E[\varepsilon_t] = E[\varepsilon_{t-1}] = \ldots = 0$$
$$E[\varepsilon_t^2] = E[\varepsilon_{t-1}^2] = \ldots = \sigma^2 < \infty$$
$$E[\varepsilon_t, \varepsilon_{t-s}] = E[\varepsilon_{t-j}, \varepsilon_{t-j-s}] = 0, t \neq s$$

2.3 *Modelos Estacionários.*

Formalmente, a definição de estacionariedade é que para uma dada série temporal Yt as condições abaixo deverão ser satisfeitas (Enders, 2003):

$E[y] = \mu$ para todo t = 1,2,...,n (2.1)

$E[(y_t - \mu)^2] = \gamma_0$ para todo t = 1,2,...,n (2.2)

$E[(y_t - \mu)(y_{t-s} - \mu)] = \gamma_s$ para todo t = 1,2,...,n (2.3)

e para todo s = ...,-2,-1,0,1,2...

Sendo que μ, γ_0, γ_s são todos números finitos (Enders, 2003). Ou seja, as equações acima significam que a média, auto-variância e auto-covariância devem ser constantes para que a série seja estacionária. Para uma dada série temporal é usualmente difícil verificar se estas três condições acontecem ao mesmo tempo. Intuitivamente, para verificar (2.1) com um determinado teste estatístico, nós precisaremos de um estimador da variância condicionada de y_t o qual necessariamente deverá obedecer (2.2) que por sua vez depende da validade de (2.1). Na literatura, um processo de covariância estacionária é também referido como possuindo uma estacionariedade fraca. Para que um processo possua estacionariedade estrita (ou forte) é necessário que todos os momentos sejam iguais para uma dada distribuição conjunta das séries $\{y_1, y_2, ..., y_n\}$ e $\{y_{1+h}, y_{2+h}, ..., y_{n+h}\}$, sendo h um número inteiro.

2.4 *Modelos Auto-regressivos de Médias Móveis (ARMA).*

Os modelos auto-regressivos (AR) e de médias móveis (MA) são modelo geral denominado auto-regressivo de médias móveis (ARMA). ARMA é um modelo linear que tem a seguinte forma:

$$y_t = \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} + \varepsilon_0 + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + ... + \theta_q \varepsilon_{t-q} \quad (2.4)$$

que também pode ser escrita como:

$$y_t = \phi_0 + \sum_{i=1}^{p} \phi_i y_{t-i} + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i} \qquad (2.5)$$

sendo que $\{\varepsilon_t\} \sim$ RB.

A abordagem de Box e Jenkins necessita que o modelo seja invertível. Formalmente, $\{y_t\}$ é invertível se puder ser representado por uma ordem finita ou o processo auto-regressivo for convergente. A invertibilidade é importante por causa do uso das funções de autocorrelação (FAC) e auto-correlação parcial (FACP) que implicitamente assumem que a seqüência $\{y_t\}$ pode ser aproximada por um modelo auto-regressivo. Considerando um modelo MA(1) da forma $y_t = \varepsilon_t - \theta_1 \varepsilon_{t-1}$, se $|\theta_1| < 1$ teremos que $y_t \geq (1 - \theta_1 B)\varepsilon_t$ ou $y_t + \theta_1 y_{t-1} + \theta_1^2 y_{t-2} + \theta_1^3 y_{t-3} + \ldots = \varepsilon_t$. Se $|\theta_1| < 1$ este modelo pode ser estimado usando a metodologia de Box e Jenkins. No entanto, se $|\theta_1| \geq 1$, então a seqüência $\{y_t\}$ não pode ser representada por um processo de ordem finita AR e dessa forma não é invertível. Genericamente, para um modelo ARMA ter uma representação AR convergente, as raízes do polinômio $1 + \theta_1 B + \theta_2 B^2 + \ldots + \theta_q B^q$ devem estar fora do círculo unitário. Sendo que B é um operador de atraso unitário definido pela seguinte propriedade:

$B^k y_t = y_{t-k}$ para k=...,-2,-1,0,1,2,...

Conseqüentemente, $B^{-2} y_t$ significa y_{t+2} e $B^0 y_t$ significa y_t. Quando $0 < \alpha < 1$ podemos escrever: $(1 - \alpha B)^{-1} = 1 + \alpha B + \alpha^2 B^2 + + \alpha^3 B^3 + \ldots$, este operador é muito útil para sumarizar modelos complicados de séries temporais.

2.5 Modelos Auto-regressivos (AR).

Supondo que as observações em uma série temporal y_t depende de p de suas observações atrasadas, ou seja, que y_t pode ser descrita pelo um modelo linear:

$$y_t = \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} + \varepsilon_t \qquad t = p+1, p+2, ..., n \; e \; n > p \qquad (2.6)$$

sendo que ϕ_1, ϕ_2, até ϕ_p são parâmetros. Consideramos ε_t como sendo um processo ruído branco. Na prática, as observações sobre ε_t não são diretamente observadas e têm de ser estimadas dos dados, baseado nos pressupostos do modelo y_t. No caso da equação (2.6), y_t pode ser descrita por um modelo de regressão que inclua apenas variáveis y_t atrasadas, e conseqüentemente este modelo (2.6) é usualmente chamado de modelo auto-regressivo de ordem p [AR(p)], ou um modelo de um processo auto-regressivo de ordem p. Uma restrição para que um processo AR(1) seja estacionário é que $|\phi_1| < 1$.

Com o operador de atraso B, a expressão (2.6) pode ser abreviada como:

$$\phi_p(B) y_t = \varepsilon_t \qquad (2.7)$$

$$\phi_p(B) = 1 + \phi_1 B + ... + \phi_p B^p \qquad (2.8)$$

o qual é chamado polinômio-AR em B de ordem p. Os pesos sobre os atrasos são os parâmetros ϕ_1 até ϕ_p e estes expressam em qual extensão y_t depende do seu passado. Desde que a observação em qualquer período de tempo t depende de p observações passadas, então (2.6) assume que de algum modo y_t depende de todas as observações passadas. Para prever y_{n+h} deve-se antes de tudo assegurar que esta dependência do passado seja constante. De fato, se para qualquer tempo t esta dependência do passado diferir, haverá dificuldade na tentativa de prever y_{n+h} uma vez que para qualquer horizonte de tempo futuro h a função de previsão poderá ser viesada. Além disso, para que possamos fazer afirmações sensatas sobre y_{n+h} deverá ser considerado que o passado imediato é mais importante do que o menos recente.

Em outras palavras, se para medir o impacto da observação no tempo t = 10, y_{10}, será necessário considerar observações não tão distantes, ou seja será preferível utilizar informações sobre y_{11} a informações dadas por y_{70}, por exemplo (Franses, 1998).

2.6 *Modelos de Médias Móveis (MA).*

Em alguns casos é conveniente considerar apenas uma variante simples de um modelo ARMA(p,q), ou seja o modelo MA(q) dado por:

$$y_t = \varepsilon_t + \theta_1\varepsilon_{t-1} + \theta_2\varepsilon_{t-2} + ... + \theta_q\varepsilon_{t-q} \qquad (2.9)$$

ou

$$y_t = \theta_q(B)\varepsilon_t \qquad (2.10)$$

sendo que

$$\theta_q(B) = 1 + \theta_1 B + ... + \theta_q B^q \qquad (2.11)$$

Uma característica importante de um modelo MA(q), e conseqüentemente de um modelo ARMA(p,q) é que as variáveis na equação (2.9), ou seja, ε_{t-1} até ε_{t-q}, não são observadas e têm de ser estimadas usando a amostra de dados disponível. Para que isso não cause problemas é usual manter q o tanto menor quanto possível. Na prática, este valor de q é freqüentemente tomado como sendo 0, 1 ou 2. A primeira vista pode parecer que y_t não depende de seu próprio passado quando um modelo MA(q) descreve esta variável. No entanto, de forma similar ao que foi mostrado para o modelo auto-regressivo, a equação (2.10) pode ser escrita como:

$$[\theta_q(B)]^{-1} y_t = \varepsilon_t \qquad (2.12)$$

o que mostra que y_t depende de todos os valores prévios de y_t. Por exemplo, para o modelo MA(1) temos que $y_t = \varepsilon_t + \theta_1\varepsilon_{t-1}$ pode ser derivado da seguinte forma:

$y_1 = \varepsilon_1$

$$y_2 = \varepsilon_2 + \theta_1 \varepsilon_1 = \varepsilon_2 + \theta_1 y_1 \qquad (2.13)$$

$$y_3 = \varepsilon_3 + \theta_1 \varepsilon_2 = \varepsilon_2 + \theta_1 (y_2 - \theta_1 y_1)$$

...

e assim por diante. Similarmente ao conceito de raiz unitária no polinômio AR(P), pode haver uma ou mais raízes unitárias em um polinômio MA(q). Em geral, os modelos MA(q) são invertíveis quando as soluções para:

$$1 + \theta_1 z + ... + \theta_q z^q = 0 \qquad (2.14)$$

estão todas fora do círculo unitário.

2.7 Modelos Não-estacionários.

Quando uma variável apresenta efeitos permanentes dos choques as séries são usualmente transformadas para uma série temporal com efeitos transitórios tomando-se as primeira diferença da série temporal y_t. A motivação é que quando y_t é uma série temporal denominada de passeio aleatório, ou seja apresenta a forma:

$$y_t = y_0 + \sum_{i=0}^{t-1} \varepsilon_{t-i} \qquad t = 1,2,...,n \qquad (2.15)$$

ela não apresenta estacionariedade. Por exemplo, no caso em que:

$$y_t = y_{t-1} + \varepsilon_t \qquad (2.16)$$

todos os choques tem um efeito permanente. No entanto, para a série transformada $z_t = y_t - y_{t-1}$, a qual pode ser descrita por um simples modelo ruído branco:

$$z_t = y_t - y_{t-1} = \varepsilon_t \qquad (2.17)$$

os choques passados ε_t têm apenas efeitos transitórios. Definindo um operador diferença ∇ onde $\nabla = (1 - B)$ temos que:

$$\nabla_j^d = (1 - B^j)^d \qquad (2.18)$$

Quando a série temporal precisa ser diferenciada d vezes, ela é chamada de integrada de ordem d, ou de forma abreviada I(d). Quando y_t é uma série temporal I(d) e depois de diferenciada d vezes ela pode ser modelada usando um modelo AR(p), este modelo pode ser escrito como:

$$\nabla_1^d y_t = \phi_1 \nabla_1^d y_{t-1} + ... + \phi_p \nabla_1^d y_{t-p} + \varepsilon_t \qquad t = p+d, p+d+1, ..., n \qquad (2.19)$$

O modelo representado pela equação (2.19) é normalmente chamado de ARI(p,d).

2.8 *Heterocedasticidade condicionada.*

Em muitas séries econômico-financeiras tem-se que informações adicionais discrepantes e *outliers* provindos de inovações não acontecem com freqüência. No entanto, as séries temporais financeiras são exceção. Uma vez que os dados refletem resultados de negócios entre compradores e vendedores, por exemplo, o mercado acionário, várias fontes de eventos econômicos exógenos podem ter um impacto sobre o padrão dos preços de capital. Dado que algumas notícias podem levar a várias interpretações diferentes e também que certos eventos econômicos específicos, tal como uma crise de petróleo, podem durar por algum tempo, freqüentemente observa-se que grandes quantidades positivas e grandes quantidades negativas de observações em séries temporais financeiras tendem a aparecer em *clusters* (agrupamentos). Uma abordagem para tratar desta situação é explorar o fato de que os *outliers* aparecem em clusters e tentar construir um modelo de série temporal para os próprios *outliers*. Uma vez que a seqüência de *outliers* pode ser considerada como refletindo a volatilidade do período, estes modelos de séries temporais podem

ser utilizados para prever a volatilidade. Ou seja, devido à presença de conjuntos de valores discrepantes, a variância de séries temporais financeiras varia em função do tempo e conseqüentemente os intervalos de previsão para cada nível também deverão variar. A intuição é que em períodos de maior volatilidade existe maior incerteza sobre a próxima observação do que em períodos de menor volatilidade e conseqüentemente em períodos que apresentam maior volatilidade os intervalos de previsão serão maiores.

2.9 Modelos ARCH.

O modelo ARCH - Autoregressive Conditional Heteroskedasticity, ou seja de Heterocedasticidade Autoregressiva Condicionada foi proposto inicialmente por Engle (1982). Desde então, têm aparecido inúmeros estudos com refinamentos e modificações do modelo ARCH básico e também com aplicações empíricas em taxa de câmbio, mercado de ações e muitos outros tipos de ativos (Bollerslev, Engle e Nelson/1994).

Consideremos uma série temporal ε_t a qual pode ser descrita pelo conjunto de equações:

$$\varepsilon_t = \eta_t \sqrt{h_t} \tag{2.20}$$

sendo que:

$$h_t = Var[\varepsilon_t \mid I_{t-1}]$$

$$\eta_t \sim NID(0;1) \tag{2.21}$$

$$h_t = \alpha_0 + \alpha_1 \varepsilon_{t-1}^2 \tag{2.22}$$

sendo que a equação (2.20) descreve o comportamento de ε_t condicionado a I_{t-1} e sendo que NID significa normal e identicamente distribuído. Na prática, a série ε_t pode ser igual a $\left[\phi_p(B)/\theta_q(B)\right]x_t$ onde x_t é a série realmente observada. Usualmente,

x_t é o retomo sobre um ativo financeiro, ou seja, $x_t = \log(z_t) - \log(z_{t-1})$, sendo z_t a série no nível.

Dessa forma, tendo que $\varepsilon_t \mid I_{t-1} = \eta_t \sqrt{h_t}, \eta_t \sim NID(0;1)$ e a série y_t que pode ser escrita como $y_t = (1-B)x_t = \nabla_1^1 x_t$, resulta então que: $y_t = \frac{\theta_q(B)}{\phi_p(B)}\varepsilon_t$, $\{\varepsilon_t\} \sim$ RB, isolando ε_t

teremos $\varepsilon_t = {\phi_p(B)}/{\theta_q(B)} y_t$, portanto:

$$\varepsilon_t = {\phi_p(B)}/{\theta_q(B)} \nabla_1^1 \ln x_t$$

Devido a h_t depender do período atrasado ε_{t-1}^2' a série ε_t é então chamada de modelo ARCH de ordem 1. A expressão (2.22) indica que a variância condicional de ε_t é variante no tempo e é importante notar que não há um termo de erro adicional. A equação (2.21) assume que todas as observações η_t têm as mesmas propriedades distribucionais.

Para analisar as propriedades dos dados do tipo ARCH é conveniente escrever (2.20) e (2.22) como:

$$\varepsilon_t = \eta_t \sqrt{\alpha_0 + \alpha_1 \varepsilon_{t-1}^2} \qquad (2.23)$$

$$\eta_t \sim NID(0;1) \qquad (2.24)$$

Uma vez que $E[\eta_t] = 0$, sendo que $E_t[\bullet]$ denota o operador esperança, pode-se verificar que a esperança condicionada de ε_t é igual a:

$$E_t[\varepsilon_t] = E[\sqrt{h_t}\, E[\eta_t]] = 0 \qquad (2.25)$$

Para a variância condicional de ε_t tem-se da equação (2.23) que:

$$E[\varepsilon_t^2] = h_t = \alpha_0 + \alpha_1 \varepsilon_{t-1}^2 \tag{2.26}$$

Usando a expressão (2.23) tem-se que a variância condicional de ε_t é igual a:

$$E_t[\varepsilon_t^2] = E[h_t] = \alpha_0 + \alpha_1 E[\varepsilon_{t-1}^2]$$

Esta expressão mostra que ε_t^2 imita um processo AR(1) com parâmetro α_1, ou seja, a série temporal quadrática apresenta autocorrelação. Pode-se verificar, dadas as definições anteriores, que esta série não é estritamente um ruído branco. Uma vez que, ε_t pode ser uma série residual de um modelo ARMA para x_t, isto indica a diferença-chave entre séries temporais ARCH e aquelas que não são ARCH. Com $0 < \alpha_1 < 10$, a equação (2.27) pode ser resolvida como:

$$E_t[\varepsilon_t^2] = \frac{\alpha_0}{(1-\alpha_1)} \tag{2.28}$$

A expressão em (2.22) mostra que valores absolutos grandes (ou pequenos) de ε_t, são esperados ser seguidos por valores absolutos grandes (ou pequenos), enquanto houver a igualdade:

$$E[\varepsilon_t \varepsilon_{t-h}] = 0 \tag{2.29}$$

ou seja, a série ε_t é não correlacionada. Conseqüentemente, um modelo ARCH pode descrever uma série temporal com seqüências de dados pontuais que parecem com *outliers*, onde o fato de que estes *outliers* aparecem em *clusters* é causado pela equação de variância e não pelas autocorrelações relevantes no nível da série temporal.

2.10 Modelos GARCH.

O modelo ARCH generalizado, conhecido como GARCH - *Generalized* ARCH foi primeiramente proposto por Bollerslev (1986).

Dado que um modelo AR(p)-ARCH(q) pode ser representado por:

$$\varepsilon_t = \eta_t \sqrt{h_t} \tag{2.30}$$

$$\eta_t \sim NID(0;1) \tag{2.31}$$

$$h_t = \alpha_0 + \alpha_1 \varepsilon_{t-1}^2 + \alpha_2 \varepsilon_{t-2}^2 + ... + \alpha_q \varepsilon_{t-q}^2 \tag{2.32}$$

com $\alpha_0 > 0, \alpha_i \geq 0$ (i = 1,2,...,q) e $\phi_p(1) > 0$ e $\sum_{i=1}^{q} \alpha_i < 1$. Para muitas séries temporais financeiras, o valor de q em (2.32) pode assumir inconvenientemente valores grandes de tal forma que seja necessário estimar muitos parâmetros. Além disso, pode ser inconveniente impor as restrições $\alpha_i \geq 0$ para todos os i em (2.32). Portanto, pode ser útil aproximar o polinômio de q-ésima ordem em (2.32) por um polinômio de ordem (p,q) similar a um modelo padrão ARMA para uma série y_t.

O modelo GARCH é então expresso por:

$$h_t = \alpha_0 + \alpha_1 \varepsilon_{t-1}^2 + ... + \alpha_q \varepsilon_{t-q}^2 - \beta_1 h_{t-1} - ... - \beta_p h_{t-p} \tag{2.33}$$

ou seja, é a equação GARCH(p,q), onde $\alpha_0, \alpha_1, ..., \alpha_q$ e $\beta_1, ..., \beta_p$ excedem zero e $\sum_{j=1}^{q} \alpha_j + \sum_{i=1}^{p} \beta_i < 1$. Na prática, o valor de q em (2.33) é muito menor que em (2.32) e é típico encontrar que o modelo GARCH(1,1) produz uma descrição adequada de muitas séries temporais financeiras (Bollerslev, Chou e Kroner/1992). Expressões explícitas para a curtose e as autocorrelações de uma série ε_t gerada pelas equações de (2.30) a (2.32) podem ser geradas mostrando que um modelo GARCH(p,q) pode descrever séries temporais com seqüências de grandes

observações tanto positivas como negativas. A função de autocorrelação de ε_t^2 pode ser encontrada aplicando as mesmas técnicas descritas anteriormente (Bollerslev/1988). Por exemplo, a equação GARCH(1,1):

$$h_t = \alpha_0 + \alpha_1 \varepsilon_{t-1}^2 + \beta_1 h_{t-1} \tag{2.34}$$

pode ser expressa como:

$$\varepsilon_t^2 = \alpha_0 + (\alpha_1 + \beta_1)\varepsilon_{t-1}^2 + \upsilon_t - \beta_1 \upsilon_{t-1} \tag{2.35}$$

onde $\upsilon_t = \varepsilon_t^2 - h_t$. Uma vez que a série υ_t é não correlacionada com o seu próprio passado, a equação (2.35) indica que o modelo GARCH(1,1) implica em que a FAC de ε_t^2 parece com a FAC de um modelo ARMA(1,1). Sendo que quando $\alpha_1 = 0$, o parâmetro β_1 em (2.35) é não determinado. Conseqüentemente, para qualquer processo GARCH(p,q) o valor de q deverá ser no mínimo igual a 1.

Uma das motivações para considerar modelos do tipo GARCH é que tais modelos permitem a previsão de volatilidade condicional. Por exemplo, com um modelo GARCH(1,1), pode ser gerada a previsão de um passo à frente para h_t (no tempo n) como:

$$h_{n+1} = \alpha_0 + \alpha_1 \varepsilon_n^2 + \beta_1 h_n \tag{2.36}$$

Para comparar a performance de previsão para amostras de vários modelos GARCH rivais deve-se usar:

$$v_{n+1} = (x_{n+1} - \bar{x})^2 \tag{2.37}$$

como uma medida de verdade, mas não observada, volatilidade, onde $\bar{x}$ é alguma média de um intervalo longo selecionado a priori de retornos passados (Day e Lewis, 1992) e (Pagan e Scwert, 1990).

Nesse caso o erro do modelo ARIMA(p,1,q) pode ser escrito como $\varepsilon_t^{ARIMA(p,1,q)} = \frac{\phi_q(B)}{\theta_p(B)}\nabla_1^1 \ln x_t$ o erro do modelo GARCH é dado por $\varepsilon_t = \eta_t \sqrt{Var_{t-1}[\varepsilon_t}$, portanto podemos escrever que $Var_{t-1}\varepsilon_t = \frac{(\varepsilon_t^{ARIMA(p,1,q)})^2}{\eta_t^2}$. Outra importante expressão que pode ser construída para os modelos GARCH em que se visualizam os principais parâmetros do modelo é $\frac{\phi_p^m(B)}{\phi_q^m(B)}\nabla_1^1 \ln x_t = \eta_t \sqrt{\frac{\alpha_q^V(B)}{\beta_p^V(B)}\varepsilon_t}$, isolando x_t temos o valor da série no nível:

$$x_t = \left(\frac{\theta_q^m(B)}{\phi_p^m(B)}\right)\eta_t \sqrt{\frac{\alpha_q^V(B)}{\beta_p^V}\varepsilon_t}$$

2.11 *Modelo GARCH-M.*

Existem muitos tipos diferentes de formas funcionais não-lineares para serem utilizadas como alternativa aos modelos lineares. Muitos modelos não-lineares tentam capturar os processos não-lineares "verdadeiros" ou"subjacentes" através de "suposições paramétricas" com formas funcionais não-lineares específicas. Um exemplo desta abordagem é o modelo GARCH-in-Mean ou GARCH-M. Nesta abordagem a variância do termo de distúrbio afeta diretamente a média da variável dependente e evolui através do tempo como uma função do seu próprio valor passado e do passado do erro quadrático de predição (Engle, Lilien e Robins, 1987). Por esta razão, a variância variando no tempo é chamada de *variância condicionada.* As seguintes equações descrevem um típico modelo paramétrico GARCH-M:

$$\sigma_t^2 = \delta_0 + \delta_1\sigma_{t-1}^2 + \delta_2\varepsilon_{t-1}^2 \qquad (2.38)$$

$$\varepsilon_t^2 \sim N(0;\sigma_t^2)$$

$$y_t = \alpha + \beta\sigma_t + \varepsilon_t \qquad (2.39)$$

sendo que y é a taxa de retorno sobre um ativo, α é a taxa esperada de avaliação e ε_t é um termo de distúrbio normalmente distribuído com média zero e *variância condicionada* σ_t^2. O parâmetro β representa o efeito de prêmio pelo risco sobre o retorno do ativo, enquanto que os parâmetros δ_0, δ_1 e δ_2 definem a evolução da variância condicional.

O modelo GARCH-M é um sistema recursivo estocástico, dadas as condições iniciais σ_0^2 e ε_0^2, bem como as estimativas $\alpha, \beta, \delta_0, \delta_1$ e δ_2. Uma vez que a variância condicional é dada, o choque aleatório é "arrancado" de uma distribuição normal, e o retorno do ativo é completamente determinado como uma função de sua própria média, o choque aleatório e o efeito do prêmio pelo risco determinados por $\beta\sigma_t$.

Uma vez que a distribuição do choque é "normal" pode-se utilizar a estimação de máxima verossimilhança para aproximar as estimativas para $\alpha, \beta, \delta_0, \delta_1$ e δ_2. A função de verossimilhança L é uma função de probabilidade conjunta para $\hat{y}_t = y_t$, para $t = 1,2,...,T$. Para o modelo GARCH - M tem-se a seguinte forma:

$$L_t = \prod_{t=1}^{T} \frac{1}{2\pi\sigma_t} \exp\left[\frac{(y_t - \hat{y}_t)^2}{2\hat{\sigma}_t^2}\right] \qquad (2.40)$$

$$\hat{y}_t = \hat{\alpha} + \hat{\beta}\hat{\sigma}_t$$

$$\hat{\varepsilon}_t = y_t - \hat{y}_t$$

$$\hat{\sigma}_t^2 = \hat{\delta}_0 + \hat{\delta}_1\hat{\sigma}_{t-1}^2 + \hat{\delta}_2\hat{\varepsilon}_{t-1}^2$$

sendo que os símbolos $\hat{\alpha}, \hat{\beta}, \hat{\delta}_0, \hat{\delta}_1$ e $\hat{\delta}_2$ são estimadores dos parâmetros. Este método para obter as estimativas dos parâmetros maximiza a soma do logaritmo da função de verossimilhança sobre toda a amostra de tempo T, t = 1 até t = T, com respeito à escolha da estimativa dos coeficientes, sujeita à restrição de que a variância é maior que zero, dada a condição inicial $\hat{\sigma}_0^2$ e $\hat{\varepsilon}_{t-1}^2$.

A tendência para o uso da abordagem GARCH-M é que ela remete à origem da não-linearidade do processo. A variância condicional é uma transformação não-linear dos valores passados, do mesmo modo que a medida de variância é uma transformação não-linear da predição dos erros passados. A justificativa do uso da *variância condicional* como uma variável afetando a variável dependente é que a *variância condicional* representa um "fator de risco" bem entendido o qual eleva a "taxa de retorno exigida" quando se tem uma previsão dinâmica do preço do ativo.
Uma das maiores desvantagens do método GARCH-M é que a minimização das funções do logaritmo da verossimilhança são freqüentemente difíceis de se alcançar. Se estivermos interessados na significância dos coeficientes estimadores $\hat{\alpha}, \hat{\beta}, \hat{\delta}_0, \hat{\delta}_1$ e $\hat{\delta}_2$ será difícil obter estimativas para os intervalos de confiança. Todas estas dificuldades são comuns nas abordagens por máxima verossimilhança para estimação de parâmetros.

A abordagem paramétrica GARCH-M para a especificação de processos não-lineares é, portanto, restritiva: tem-se um conjunto específico de parâmetros que se quer estimar, os quais tem uma interpretação e um significado muito bem definidos, sabe-se como estimar estes parâmetros e até mesmo se há alguma dificuldade.

A vantagem dos modelos GARCH-M é que eles capturam fenômenos bem observados em séries temporais financeiras, onde períodos de alta volatilidade são seguidos por alta volatilidade e períodos de baixa volatilidade são seguidos por períodos similares.

As limitações do modelo GARCH-M também são suas desvantagens: fica-se limitado a conjuntos de parâmetros bem definidos, distribuições bem definidas, formas funcionais não-lineares específicas e a um método de estimação o qual nem sempre converge para parâmetros estimados que fazem algum sentido. A conclusão é que utilizando-se modelos não-lineares específicos, pode-se, portanto, faltar flexibilidade para se ajustar determinados processos não-lineares alternativos.

Com uma rede neural, utilizada como método de "aproximação", pode-se aproximar processos não-lineares desconhecidos. Não há limite no número de parâmetros e nem, infelizmente, eles têm uma interpretação direta tal como nos modelos GARCH-M.

2.12 Metodologia de Box e Jenkins.

2.12.1 Identificação.

A metodologia de Box e Jenkins consiste basicamente em quatro estágios e é utilizada para propósitos de estimação e previsão de séries temporais univariadas (Enders, 1995, pg. 63). No estágio de identificação, o pesquisador examina visualmente os gráficos da série temporal, as funções de auto-correlação e correlação parcial. A partir do gráfico construído com cada observação da seqüência y_t é possível verificar a presença de valores discrepantes *(outliers)*, valores faltantes e quebra estrutural nos dados. A aplicação da primeira diferença também pode ser realizada após a análise gráfica. As variáveis não-estacionárias podem ter uma tendência pronunciada ou desviar de forma não constante da média e da variância.
Valores faltantes e *outliers* podem ser corrigidos neste ponto da análise. O modelo ARMA tem uma característica importante que o torna distinto de muitos outros modelos econométricos. Este modelo é adequado para descrever determinadas séries temporais que são reconhecidas por características específicas de seus dados. Estas características são chamadas de auto-correlação (FAC) e auto-correlação parcial (FACP). Na fase de identificação a comparação entre amostras de FAC e FACP de vários processos ARMA podem sugerir muitos modelos plausíveis.

2.12.2 Função de Auto-correlação.

A função de auto-correlação FAC é definida pela seguinte expressão:

$$\rho_k = \frac{\gamma_k}{\gamma_0} \tag{2.41}$$

sendo que γ_k é a k-ésima auto-covariância de y_t, ou seja:

$$\gamma_k = E[(y_t - \mu)(y_{t-k} - \mu)] \quad \text{k = ...,-2,-1,0,1,2,...} \tag{2.42}$$

Dada a equação (2.42) é fácil verificar que para as auto-correlações isto implica que $\rho_0 = 1, \rho_{-k} = k$ e que $-1 < \rho_k < 1$.

A função de auto-correlação é útil para caracterizar modelos de séries temporais ARMA. Um simples exemplo é a série ruído branco ε_t para a qual $E[\varepsilon_t] = 0$ e $\rho_k = 0$ para todo $k \neq 0$. Para o modelo AR(1):

$$y_t - \mu = \phi_1(y_{t-1} - \mu) + \varepsilon_t \tag{2.43}$$

nós podemos derivar que:

$$E(y_t) = \mu + \phi_1 E(y_{t-1} - \mu) + E(\varepsilon_t) = (1-\phi_1)\mu + \phi_1 E(y_{t-1}) \tag{2.44}$$

Quando $E(y_{t-1}) = E(y_t)$, que é o caso no qual as séries temporais correspondem aos modelos ARMA sendo que não há componente (1-B) na parte AR, implicando em $|\phi_1| < 1$, a equação (2.33) pode ser escrita como:

$$E(y_t) = (1-L)^{-1}(1-\phi_1)\mu = \mu \tag{2.45}$$

Para calcular a FAC, começamos com:

$$\gamma_0 = E[(y_t - E(y_t))(y_t - E(y_t))] \tag{2.46}$$

Para o modelo AR(1), o lado direito da equação (2.46) é:

$$E[(y_t-\mu)(y_t-\mu)]=E[\phi_1(y_{t-1}-\mu)\phi_1(y_{t-1}-\mu)]+E[\varepsilon_t^2]+2E[\varepsilon_t\phi_1(y_{t-1}-\mu)] \quad (2.47)$$

A covariância de μ coma série temporal é logicamente igual a zero. Considerando que o modelo AR(1) pode ser escrito como:

$$y_t=\varepsilon_t+\theta_1\varepsilon_{t-1}+\theta_2\varepsilon_{t-2}+\ldots+y_0 \quad (2.48)$$

onde os parâmetros são escalonados por θ_0, na versão para um período de atraso temos que:

$$y_{t-1}=\varepsilon_{t-1}+\theta_1\varepsilon_{t-2}+\theta_2\varepsilon_{t-3}+\ldots+y_0 \quad (2.48)$$

fica claro que $E[y_{t-1}\varepsilon_t]=0$. Movendo a equação (2.49) cada vez mais atrás no tempo segue que $E[y_{t-j}\varepsilon_t]=0$ para qualquer j discreto maior que zero. A partir das equações (2.48) e (2.49) fica evidente que $E[y_{t-j}\varepsilon_{t-j}]=E[\varepsilon_t^2]=\sigma^2$. Com estes resultados na equação (2.47) tem-se:

$$\gamma_0=\left(1-\phi_1^2\right)^{-1}\sigma^2, \text{ sujeito a condição } |\phi_1|<1 \quad (2.50)$$

A autocovariância de primeira ordem pr uma série temporal com modelo AR(1) é:

$$\gamma_1=E[(y_t-\mu)(y_{t-1}-\mu)]=E[\phi_1(y_{t-1}-\mu)(y_{t-1}-\mu)]+E[\varepsilon_t(y_{t-1}-\mu)]=\phi_1\gamma_0 \quad (2.51)$$

Conseqüentemente, o coeficiente de correlação ρ_1 par ao modelo AR(1) torna-se simplesmente:

$$\rho_1=\gamma_1/\gamma_0=\phi_1 \quad (2.52)$$

Para calcular ρ_k é conveniente considerar a expressão para o modelo AR(1):

$$E[(y_t - \mu)(y_{t-k} - \mu)] = E[\phi_1 (y_{t-1} - \mu)(y_{t-1} - \mu)] \qquad (2.53)$$

considerando o fato que $E[y_{t-k}\varepsilon_t] = 0$. Dividindo ambos os lados por γ_0 resulta que:

$$\rho_k = \phi_1 \rho_{k-1}, \text{ para k = 1,2,3,...} \qquad (2.54)$$

Por exemplo, quando $\phi_1 = 0{,}8$ as primeira quatro auto-correlações são dadas por 0,8, 0,64, 0,512 e 0,4096. Na prática nós podemos determinar tais correlações para dados reais, e verificar se o padrão se encaixa com esta seqüência. Se isso acontecer, podemos então considerar um modelo AR(1) para realizar a previsão.

Em princípio, a determinação dos coeficientes de autocorrelação para modelos auto-regressivos de ordens maiores segue um procedimento similar ao que foi mostrado no caso do modelo AR(1). Por exemplo, considerando o modelo AR(2):

$$y_t - \phi_1 y_{t-1} - \phi_2 y_{t-2} = \varepsilon_t \qquad (2.55)$$

multiplicando ambos os lados por y_{t-1}, tomando as esperanças e dividindo por γ_0 resulta em:

$$\rho_1 - \phi_1 \rho_0 - \phi_2 \rho_1 = 0 \qquad (2.56)$$

e sendo $\rho_0 = 1$, obtém-se:

$$\rho_1 = \frac{\phi_1}{1 - \phi_2} \qquad (2.57)$$

para determinar uma expressão para ρ_2 operações análogas são aplicadas sobre a equação (2.55), produzindo:

$$\rho_2 - \phi_1\rho_1 - \phi_2\rho_0 = 0 \qquad (2.58)$$

Substituindo (2.57) em (2.58) encontra-se:

$$\rho_1 = \frac{\phi_1^2}{1-\phi_2} + \phi_2 \qquad (2.59)$$

Portanto, analogamente as equações (2.56) e (2.58) tem-se:

$\rho_k = \phi_1\rho_{k-1} + \phi_2\rho_{k-2}$, para k = 2,3,4,... (2.60)

Para encontrar expressões para a FAC para modelos AR(p) com p > 2, nós utilizamos a mesma técnica mostrada acima. Em geral isto implica que a FAC de um processo AR mostra um padrão de decaimento exponencial.

A FAC é mais utilizada para a determinação de modelos MA(q) (Shummway e Stoffer, 2000).

Considere por exemplo, o modelo MA(2):

$$y_t = \varepsilon_t + \phi_1\varepsilon_{t-1} - \phi_2\varepsilon_{t-2} \qquad (2.61)$$

onde para a versão com k atrasos tem-se que:

$$y_{t-k} = \varepsilon_{t-k} + \theta_1\varepsilon_{t-k-1} - \theta_2\varepsilon_{t-k-2} \qquad (2.62)$$

A variância γ_0 é igual a:

$$\gamma_0 = (1 + \theta_1^2 + \theta_2^2)\sigma^2 \qquad (2.63)$$

respeitando a condição que todas as covariâncias entre ε_t e os seus atrasos são iguais a zero. Com as equações (2.61) e (2.62) podemos verificar que:

$$\gamma_1 = E(y_t y_{t-1}) = (\theta_1 + \theta_1\theta_2)\sigma^2 \qquad (2.64)$$

$$\gamma_2 = E(y_t y_{t-2}) = \theta_2\sigma^2 \qquad (2.65)$$

$$\gamma_k = 0 \text{ , para k = 3.,4,...} \qquad (2.66)$$

conseqüentemente, $\rho_k = 0$ para k = 3,4,...

Isto implica que quando na prática a FAC estimada é disponível, sendo que todos os valores são zero depois do q-ésimo lag, nós poderemos decidir analisar um modelo MA(q) para y_t. Isto segue do fato de que para um modelo MA(q) tem-se:

$$\gamma_k = \left(\sum_{i=0}^{q-k} \theta_i\theta_{i+k}\right)\sigma^2 \text{ para k = 0,1,...,q} \qquad (2.67)$$

$$\gamma_k = 0 \qquad \text{para k > q}$$

Para modelo ARMA(p,1) o padrão da FAC é uma mistura do padrão da FAC para modelos AR e MA puros. Por exemplo, considerando o modelo ARMA(1,1):

$$y_t = \phi_1 y_{t-1} + \varepsilon_t + \theta_1\varepsilon_{t-1} \qquad (2.68)$$

pode-se derivar que:

$$\gamma_0 = \phi_1\gamma_1 + \sigma^2 + \theta_1 E(y_t\varepsilon_{t-1}) = \phi_1\gamma_1 + [1 + \theta_1(\phi_1 + \theta_1)]\sigma^2$$

$$\gamma_1 = \phi_1\gamma_0 + \theta_1\sigma^2$$

$$\gamma_1 = \phi_1\gamma_1$$

$$\gamma_k = \phi_1\gamma_{k-1}$$

sendo k = 3, 4, 5..., de tal forma que após alguma álgebra a função de correlação resulta em:

$$\rho_k = \frac{\phi_1^{k-1}(1+\phi_1\theta_1)(\phi_1+\theta_1)}{(1+\phi_1\theta_1+\theta_1^2)}, \quad \text{para k} = 1,2,3,... \tag{2.69}$$

Da expressão (2.69) pode-se verificar que ρ_k pode assumir uma ampla variedade de valores para escolhas distintas de ϕ_1 e θ_1. Isto implica que a identificação de um modelo de série temporal ARMA a partir somente da análise dos padrões de comportamento da FAC é uma tarefa difícil.

2.12.3 Função de Autocorrelação Parcial.

A FAC é utilizada para identificar se um modelo MA (de uma ordem qualquer) pode descrever y_t, mas é menos útil para identificar modelos AR.

A razão para isso é que, por exemplo, para um modelo AR(1) tem-se:

$$y_t = \phi_1^2 y_{t-1} + \varepsilon_t \tag{2.70}$$

o qual pode ser escrito como:

$$y_t = \phi_1^2 y_{t-2} + \varepsilon_t + \phi_1\varepsilon_{t-1} \tag{2.71}$$

a inclusão de y_{t-1} no modelo de regressão para y_t, também faz com que y_t dependa de y_{t-2} (sendo $\phi_1^2 < \phi_1$) como pode ser observado na FAC da equação (2.54). É importante notar que para identificar um modelo AR, adicionando y_{t-2} à regressão em (2.70), poderá não auxiliar para a explicar y_t, ou seja o parâmetro correspondente deverá ser igual a zero. Isso nos leva a construção da função de autocorrelação parcial (FACP). O valor da FACP no *lag* 1, digamos ψ_1 é dado por:

$$y_t = \psi_1 y_{t-1} + u_t \tag{2.72}$$

onde u_t é apenas uma série temporal de erro ruído branco quando o modelo para y_t é de fato um AR(1). Da equação (2.72) segue que sendo ψ_1 igual a γ_1 / γ_0, por construção nós temos que $\psi_1 = \rho_1$ para todos os modelos de série temporal. O segundo valor da FACP resulta do modelo de regressão:

$$y_t = \eta_1 y_{t-1} + \psi_2 y_{t-2} + u_t \tag{2.73}$$

No caso de um AR(1), o ψ_2 é igual a zero. No caso de um AR(2) ou de ordem superior, ψ_2 é diferente de zero. Para um modelo AR(2), acontece que $\psi_3 = 0$ na regressão:

$$y_t = \eta_1 y_{t-1} + \eta_2 y_{t-2} + \psi_3 y_{t-3} + u_t \tag{2.74}$$

conseqüentemente, quando ψ_{p+1} é igual a zero, quando ψ_p não é, pode-se então considerar o modelo como sendo um AR de ordem p.

Uma outra forma de determinar a FACP é através das equações Yule-Walker. Sendo ϕ_{11} tanto a auto-correlação como a auto-correlação parcial entre y_t e y_{t-1}, teremos que a partir das auto-correlações podemos formar as auto-correlações parciais da seguinte forma:

$$\phi_{11} = \rho_1 \tag{2.75}$$

$$\phi_{22} = \left(\rho_2 - \rho_1^2\right) / \left(1 - \rho_1^2\right) \tag{2.76}$$

para *lags* adicionais temos uma recursão dada por:

$$\phi_{kk} = \left(\rho_k - \sum_{j=1}^{k-1} \phi_{k-1,j} \rho_{k-j} \right) \left(1 - \sum_{j=1}^{k-1} \phi_{k-1,j} \rho_k \right)^{-1} \text{, sendo k = 3,4,5,...} \qquad (2.77)$$

2.13 Estimação.

A estratégia de especificação de uma série temporal ARMA começa com a inspeção dos valores da FAC e da FACP estimadas, as quais são dadas por:

$$\hat{\rho}_k = C_k / C_0 \qquad (2.78)$$

sendo que:

$$C_k = \sum_{t=k+1}^{n} (y_t - \bar{y})(y_{t-k} - \bar{y}) \qquad (2.79)$$

onde $\bar{y}$ denota a média amostral relevante de y_t, t = 1,2,3,...n. Dessa forma, $\hat{\rho}_k$ para k = 0,1,2,... forma a FAC estimada.

A FACP pode ser obtida aplicando-se mínimos quadrados ordinários em:

$$y_t - \bar{y} = \hat{\psi}_1 (y_{t-1} - \bar{y}) + ... + \hat{\psi}_k (y_{t-k} - \bar{y}) + \upsilon_t \qquad (2.80)$$

para quaisquer valores de k, onde υ_t não é necessariamente uma série temporal ruído branco.

Para checar quais valores são significantes de tal forma que estruturas de modelos ARMA simples e razoáveis possam ser levados como hipótese, devemos estimar os parâmetros dos vários modelos e investigar se os resíduos estimados podem ser considerados aproximadamente como ruído branco. Esta estratégia pode ser descrita como sendo de identificação, estimação e modificação.

Vamos analisar a estimação de modelos AR. Os parâmetros no modelo AR(p), onde

a escolha de p está baseada nos valores estimados da FACP que são significantes, dados por:

$$y_t = \delta + \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} + \varepsilon_t \qquad t = p+1, p+2, ..., n \qquad (2.81)$$

podem ser estimados pelo método dos mínimos quadrados ordinários (MQO), onde as observações de y_1 até y_p são assumidas como sendo os valores iniciais. Pode ser mostrado que os estimadores de MQO dos parâmetros são consistentes e assintoticamente normal e a estatística t padrão pode ser utilizada para investigar a significância.

Quanto aos modelos ARMA existe uma grande variedade de rotinas de estimação para determiná-los. A principal razão para isto é a importante característica de que as variáveis ε_t atrasadas na parte MA do modelo são variáveis não observadas e conseqüentemente têm que ser estimadas. Por exemplo, para o modelo ARMA(1,1):

$$y_t = \phi_1 y_{t-1} + \varepsilon_t + \theta_1 \varepsilon_{t-1} \qquad (2.82)$$

os parâmetros ϕ_1 e θ_1 na são conhecidos e a variável ε_{t-1} também não é conhecida. O modelo (2.82) pode ser escrito como:

$$(1+\theta_1 B)^{-1} y_t = \phi_1 (1+\theta_1 B)^{-1} y_{t-1} + \varepsilon_t \qquad (2.83)$$

denotando $z_t = (1+\theta_1 B)^{-1} y_t$ tal que:

$$z_t = y_t - \theta_1 y_{t-1} + \theta_1^2 y_{t-2} - \theta_1^3 y_{t-3} + ... \qquad (2.84)$$

tem-se as observações (assumindo que $y_0 = 0$):

$z_1 = y_1$

$$z_2 = y_2 - \theta_1 y_1$$

$$z_3 = y_3 - \theta_1 y_2 + \theta_1^2 y_3$$

...

e assim por diante. Atribuindo um valor para θ_1, pode-se construir a série temporal z_t e estimar ϕ_1 via MQO aplicados em (2.83). Esta regressão fornece a série $\hat{\varepsilon}_t$ estimada, a qual quando atribuímos $\varepsilon_1 = 0$ pode ser usada em (2.82) para atribuir um valor para θ_1 em um segundo passo. Uma rotina de otimização comumente aplicada que produz os estimadores finais para os parâmetros ARMA em um número não muito elevado de iterações é a rotina de Gauss-Newton.

2.14 Diagnóstico

Um requisito óbvio para um modelo de série temporal ARMA é que a série temporal de resíduo estimado seja aproximadamente um ruído branco. Se não for, pode-se não ter alcançado a estrutura dinâmica em y_t que incorpore um modelo ARMA. A FAC estimada dos resíduos é dada por:

$$r_k(\hat{\varepsilon}) = \frac{\sum_{t=k+1}^{n} \hat{\varepsilon}_t \hat{\varepsilon}_{t-k}}{\sum_{t=1}^{n} \hat{\varepsilon}_t^2} \quad \text{para k = 1, 2, 3, ...} \qquad (2.85)$$

Dado um modelo adequado, pode-se mostrar que as populações equivalentes de $r_k(\hat{\varepsilon}_k)$ são assintoticamente não-correlacionados e têm variâncias que podem ser aproximadas por $(n-k)/(n^2+2n) \cong n^{-1}$. Conseqüentemente, sob a suposição adicional de normalidade, uma checagem ao nível de confiança de 95% pode testar se as auto-correlações residuais estimadas estão no intervalo $\pm 2n^{-\frac{1}{2}}$. Ljung e Box (1978) propõem um teste conjunto para a significância das m primeiras auto-correlações residuais, a qual é dada por:

$$LB(m) = n(n+2)\sum_{k=1}^{m}(n-k)^{-1} r_k^2(\hat{\varepsilon}) \qquad (2.86)$$

a qual segue assintoticamente uma distribuição $\chi^2(m-p-q)$ sob as hipóteses de não haver auto-correlação residual, sendo que m/n é um valor pequeno e m é moderadamente grande (>30).

Realizando os estágios de identificação, estimação e diagnóstico resulta em um conjunto de modelos, estes modelos não podem ser rejeitados usando as medidas de diagnóstico mencionadas acima então será necessário adotar alguns critérios de seleção.

Um estudo sobre critérios de seleção de modelos pode ser encontrado em De Gooijer *et al.* (1985).

É sensato assumir que nenhum modelo pode ser preferível *a priori* e que conseqüentemente os modelos devem ser tratados simetricamente (Granger, King e White, 1995). Em geral, isto implica que o modelo final selecionado é aquele que minimiza o valor de uma certa função de critério.

O coeficiente padrão de determinação R^2 é pouco útil para avaliar modelos de séries temporais. Por exemplo, para o modelo AR(1) $y_t = \phi_1 y_{t-1} + \varepsilon_t$, teoricamente segue que:

$$\begin{aligned} R^2 &= 1-\sigma^2/\gamma_0 \\ &= 1-\sigma^2/[\sigma^2/(1-\phi_1^2)] \\ &= \phi_1^2 \end{aligned} \qquad (2.87)$$

Ou seja, o R^2 depende somente do parâmetro AR. Esta expressão mostra que o R^2 no caso em $\phi_1 = 0{,}9$ é muito maior que no caso onde $\phi_1 = 0{,}2$, quando ambos parecem ser adequados (Nelson, 1976). Um critério R^2 modificado pode ser

estudado em Harvey (1989).

Dois critérios freqüentemente utilizados para selecionar modelos de séries temporais são o critério de informação Akaike (1974) e Schwarz (1978). Ambos os critérios avaliam o ajuste versus o número de parâmetros. Quando n denota o número de observações efetivas (as quais são observações necessárias para estimar os parâmetros) e k denota o número de parâmetros ARMA a serem estimados, o critério de informação de Akaike é dado por:

$$AIC(k) = n\log\hat{\sigma}^2_{ML} + 2k$$

onde $\hat{\sigma}^2_{ML} = RSS/n$ e RSS é a soma dos resíduos quadráticos (*residual sum of squares*). O valor de k que minimiza AIC(k) é selecionado. A mesma regra de decisão aplica-se para o critério de Schwarz, o qual é dado por:

$$SIC(k) = n\log\hat{\sigma}^2_{ML} + k\log n$$

Comparando as expressões para AIC e SIC, quando $n \geq 8$, o critério SIC penaliza a inclusão de regressores (e portanto, de parâmetros adicionais) mais que o AIC. Isto significa que a ordem do modelo selecionado com o critério SIC é usualmente menor que a ordem do modelo selecionado com o AIC, ou seja o critério SIC sempre seleciona um modelo mais parcimonioso que o AIC (Enders, 1995, pg. 88).

2.15 Previsão.

Uma vez que um ou mais modelos de séries temporais foram selecionados devemos gerar previsões l-passo ou h-passos à frente de previsão pata y_t (onde y_t denota um variável diferenciada adequadamente) . As previsões referem-se as observações y_{n+h} e conseqüentemente será denotada por $\hat{y}_{n+h}$, com h = 1,2,3,..,m. Em geral, a previsão um passo à frente é definida por $\hat{y}_{n+1}$ a qual é baseada no conjunto de informação y_n. No caso onde m observações foram registradas para avaliar uma previsão um-passo à frente, avança-se o conjunto de informação com cada previsão. Por exemplo,

$\hat{y}_{n+2}$ é baseado em y_{n+1} e $\hat{y}_{n+3}$ é baseado em y_{n+2}. A motivação intuitiva por trás dessa lógica é que não é sempre possível ser capaz de re-estimar os parâmetros do modelo de séries temporais. Neste caso, são mantidos os estimadores para os parâmetros fixados para a amostra t = 1, 2, ..., n enquanto faz-se previsões um-passo à frente para y_{n+1} até y_{n+m}. Se possível, podemos também decidir estimar os parâmetros para cada um dos conjuntos de informação y_{n+h-1}. Além disso, podemos então apagar os primeiros *h* dados pontuais, de tal forma que a amostra de estimação tem o mesmo tamanho para as *m* previsões. Finalmente, uma previsão h-passos (ou dinâmica) denota uma previsão de y_{n+h} (para h = 1,2, ...,m) a qual é baseada em y_n.

Obviamente, quando *h* é maior que a memória do modelo (ou, por exemplo, o comprimento do polinômio AR), a previsão $\hat{y}_{n+h}$ provavelmente inclui previsões para dados pontuais prévios como $\hat{y}_{n+h-1}$ e também os erros de previsão tendem a ser maiores (Enders, 1995).

O princípio de previsão de modelos ARMA é muito simples. Considere o modelo MA(2):

$$y_t = \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} \quad t = 1,2,...,n \tag{2.88}$$

para o qual é possível verificar que:

$$y_{n+1} = \varepsilon_{n+1} + \theta_1 \varepsilon_n + \theta_2 \varepsilon_{n-1}$$

No tempo n, o ε_{n+1} é desconhecido, e desde que para sua expectância no tempo n aconteça que $E[\varepsilon_{n+1}] = 0$, uma previsão não viesada de y_{n+1} é igual a:

$$\hat{y}_{n+1} = \theta_1 \varepsilon_n + \theta_2 \varepsilon_{n-1} \tag{2.90}$$

Na prática, θ_1, θ_2, ε_n e ε_{n+1} devem ser estimados, mas por conveniência assumindo

que são dados, podemos comparar as expressões y_{n+1} e $\hat{y}_{n+1}$, com isso o erro de previsão (ou erro de predição) é dado por:

$$y_{n+1} = \hat{y}_{n+1} - \varepsilon_{n+1} \tag{2.91}$$

e conseqüentemente o erro de predição quadrático (squared prediction error – SPE) é σ^2. Para dois passos à frente, tem-se:

$$y_{n+2} - \hat{y}_{n+2} = (\varepsilon_{n+2} + \theta_1\varepsilon_{n+1} + \theta_2\varepsilon_n) + (\theta_2\varepsilon_n) \tag{2.92}$$

uma vez que no tempo n, ε_{n+2} e ε_{n+1} não são conhecidos, para os quais o SPE é igual a $(1+\theta_1^2)\sigma^2$. Para três passos à frente, temos:

$$y_{n+3} - \hat{y}_{n+3} = (\varepsilon_{n+3} + \theta_1\varepsilon_{n+2} + \theta_2\varepsilon_{n+1}) + (0) \tag{2.93}$$

e conseqüentemente o SPE é igual a $(1+\theta_1^2+\theta_2^2)\sigma^2$. Similarmente, para três passos à frente, não há memória no modelo MA(2) que possa auxiliar a previsão y_t quatro passos à frente, uma vez que:

$$y_{n+4} - \hat{y}_{n+4} = (\varepsilon_{n+4} + \theta_1\varepsilon_{n+3} + \theta_2\varepsilon_{n+2}) + (0) \tag{2.94}$$

o qual novamente produz um SPE de $(1+\theta_1^2+\theta_2^2)\sigma^2$.

Em geral para um modelo MA(q) a previsão h-passos à frente é igual a:

$$\hat{y}_{n+h} = \sum_{i=0}^{q} \theta_{i+h}\varepsilon_{n-i} \tag{2.95}$$

com $\theta_0 = 1$ e $\theta_{i+h} = 1$ para i + h > q, com um erro h-passos:

$$e_{n+h} = y_{n+h} - \hat{y}_{n+h} = \sum_{i=0}^{h-1} \theta_i \varepsilon_{n+h-i} \tag{2.96}$$

Dada a suposição de ruído branco sobre ε_i segue que:

$$E(e_{n+h}) = 0 \quad \text{e} \tag{2.97}$$

$$SPE(h) = E[e_{n+h}^2] = \sigma^2 \sum_{i=0}^{h-1} \theta_i^2 \tag{2.98}$$

Da equação (2.74) pode ser visto que para uma série temporal com média zero que pode ser descrita como um modelo MA(q) isto implica que $\hat{y}_{n+h} = 0$ quando h > q. Assumindo normalidade, um intervalo de previsão de 95% para y_{n+h} é limitado por $\hat{y}_{n+h} - 1{,}95RSPE(h)$ e $\hat{y}_{n+h} + 1{,}95RSPE(h)$, onde RSPE é a raiz quadrada do SPE.

No modelo AR(P) acontece que y_i depende de todas as observações prévias, a previsão h-passos à frente tem propriedades similares ao que foi mostrado anteriormente. Considerando, por exemplo, o modelo AR(2):

$$y_i = \phi_1 y_{i-1} + \phi_2 y_{i-2} + \varepsilon_i \tag{2.99}$$

e a previsão um passo à frente no tempo n, é dada por:

$$\hat{y}_{n+1} = \phi_1 y_n + \phi_2 y_{n-1} \tag{2.100}$$

e uma vez que

$$y_{n+1} = \phi_1 y_n + \phi_2 y_{n-1} + \varepsilon_{n+1} \tag{2.101}$$

o SPE para um passo à frente, denotado por SPE(1) é σ^2. Para dois passos à frente, obtemos:

$$\hat{y}_{n+2} = \phi_1 \hat{y}_{n+1} + \phi_2 y_n = \phi_1(\phi_1 y_n + \phi_2 y_{n-1}) + \phi_2 y_n \quad (2.102)$$

uma vez que:

$$y_{n+2} = \phi_1 y_{n+1} + \phi_2 y_n + \varepsilon_{n+2} = \phi_1(\phi_1 y_n + \phi_2 y_{n-1} + \varepsilon_{n+1}) + \phi_2 y_n + \varepsilon_{n+2} \quad (2.103)$$

implica que:

$$SPE(2) = (1 + \phi_1^2)\sigma^2$$

Para três passos à frente tem-se que:

$$\hat{y}_{n+3} = \phi_1 \hat{y}_{n+2} + \phi_2 y_{n+1} = \phi_1(\phi_1(\phi_1 y_n + \phi_2 y_{n-1}) + \phi_2 y_n) + \phi_2(\phi_1 y_n + \phi_2 y_{n-1}) \quad (2.104)$$

e sendo:

$$y_{n+3} = \phi_1 y_{n+2} + \phi_2 y_{n+1} + \varepsilon_{n+3} = \phi_1(\phi_1(\phi_1 y_n + \phi_2 y_{n-1} + \varepsilon_{n+1}) + \phi_2 y_n + \varepsilon_{n+2}) +$$
$$+ \phi_2(\phi_1 y_n + \phi_2 y_{n-1} + \varepsilon_{n+1}) + \varepsilon_{n+3} \quad (2.105)$$

a variância do erro de previsão é:

$$SPE(3) = (1 + \phi_1^2 + \phi_2^2 + 2\phi_1^2\phi_2 + \phi_1^4)\sigma^2 \quad (2.106)$$

o qual mostra que SPE(3) > SPE(2).

Em geral acontece que SPE(h) > SPE(h-l) para modelos AR(p) (Franses, 1998). A expressão (2.106) mostra que a expressão para os erros de previsão h-passos à frente pode ser notacionalmente pesada. Por isso é mais usual escrever um modelo AR(p) no formato MA e dessa forma usar fórmulas semelhantes a (2.98). Por exemplo, o modelo AR(2) representado por (2.99) pode ser escrito como:

$$y_t = \varepsilon_t + \eta_1 \varepsilon_{t-1} + \eta_2 \varepsilon_{t-2} + \eta_3 \varepsilon_{t-3} + \ldots \quad (2.107)$$

para o qual tem-se:

$$SPE(h) = \sigma^2 \sum_{i=0}^{h-1} \eta_i^2 \text{ , com } \eta_0 = 1 \quad (2.108)$$

Para o modelo AR(3), pode-se verificar que $\eta_1 = \phi_1$ e $\eta_2 = \phi_1^2 + \phi_2$, as quais substituídas em (2.108) produzem o resultado apresentado em (2.106).

A previsão h-passos para frente para modelos ARMA é derivada de forma similar ao que foi realizado para os modelos AR e MA. Por exemplo, para o modelo ARMA(1,1) tem-se que (Hamilton, 1994):

$$\hat{y}_{n+1} = \phi_1 y_n + \theta_1 \varepsilon_n \quad (2.109)$$

sendo que $SPE(1) = \sigma^2$ e:

$$\hat{y}_{n+2} = \phi_1 \hat{y}_{n+1} = \phi_1 (\phi_1 y_n + \theta_1 \varepsilon_n) \quad (2.110)$$

que comparado com:

$$y_{n+2} = \phi_1 y_{n+1} + \varepsilon_{n+2} + \theta_1 \varepsilon_{n+1} = \phi_1 (\phi_1 y_n + \theta_1 \varepsilon_{n+1} + \theta_1 \varepsilon_n) + \varepsilon_{n+2} + \theta_1 \varepsilon_{n+1} \quad (2.111)$$

produz $SPE(2) = (1 + \phi_1^2 + \theta_1^2 + 2\phi_1\theta_1)\sigma^2$. Esta última expressão também pode ser obtida escrevendo-se o modelo ARMA(1,1) como:

$$y_t = (1 + \phi_1 B)^{-1} (1 + \theta_1 B) \varepsilon_t = \varepsilon_t + \eta_1 \varepsilon_{t-1} + \eta_2 \varepsilon_{t-2} \eta_3 \varepsilon_{t-3} + \ldots \quad (2.112)$$

onde $\eta_1 = \phi_1 + \theta_1$.

Um outro ponto importante para se destacar na etapa de previsão concerne ao fato da realização de previsão de w_t quando o modelo é descrito pela transformação $y_t = \log(w_t)$. Considerando novamente o modelo MA(2) e a sua previsão um passo

para a frente:

$$\hat{y}_{n+1} = \theta_1 \varepsilon_n + \theta_2 \varepsilon_{n-1} \tag{2.113}$$

Para prever w_{n+1} usando:

$$\hat{w}_{n+1} = \exp(\hat{y}_{n+1}) \tag{2.114}$$

é fácil verificar que $\hat{w}_{n+1}$ é viesado para w_{n+1} uma vez que:

$$\begin{aligned} E[w_{n+1}] &= E[\exp(\varepsilon_{n+1} + \theta_1 \varepsilon_n + \theta_2 \varepsilon_{n-1})] \\ &= \exp\left(\frac{\sigma^2}{2}\right) E[\exp(\theta_1 \varepsilon_n + \theta_2 \varepsilon_{n-1})] \\ &= \exp\left(\frac{\sigma^2}{2}\right) \hat{w}_{n+1} \end{aligned} \tag{2.115}$$

Dessa forma, uma previsão não-viesada de w_{n+1} no caso de modelos utilizando logaritmos é dada por $\exp\left(\frac{\sigma^2}{2}\right)\hat{w}_{n+1}$, onde $\hat{w}_{n+1}$ é chamada de *previsão ingênua*.

Para dois passos à frente tem-se que:

$$\begin{aligned} E[w_{n+2}] &= E[\exp(\varepsilon_{n+2} + \theta_1 \varepsilon_{n+1} + \theta_2 \varepsilon_n)] \\ &= \exp\left(\frac{(1+\theta_1^2)\sigma^2}{2}\right) E[\exp(\theta_2 \varepsilon_n)] \\ &= \exp\left(\frac{(1+\theta_1^2)\sigma^2}{2}\right) \hat{w}_{n+2} \end{aligned} \tag{2.116}$$

Quando modelos ARMA são escritos na forma de MA, expressões apropriadas podem ser derivadas para o fator de correção das *previsões ingênuas* para w_{n+h} (Granger e Xewbold/1976).

A seguir vamos discutir sobre como realizar comparação entre as previsões.
Um procedimento prático comum é manter m observações para avaliar previsões de modelos que foram ajustados para as n primeiras observações. Uma possibilidade é checar se 95% das previsões de fato encontram-se no intervalo se 95%. Se isso acontece, os modelos criados passam a ter confiança. Se não acontecer, é provável que a variância dos dados será sub-estimada.

Estudos empíricos mostram que os modelos que tendem a ser melhores para dados amostrais não necessariamente realizam previsões melhores para dados que estejam fora dessa amostra. Não há uma regra estrita para isso, mas experiências empíricas sugerem que pode ser melhor selecionar poucos modelos baseados no AIC e SIC, e avaliar estes sobre m dados desejados fora da amostra (Diebold e Mariano/1995). Esta avaliação pode ser baseada na raiz do erro de predição quadrático médio *(root mean squared prediction error* - RMSPE) também chamado de RMSE:

$$RMSPE = \sqrt{\left(\frac{1}{m}\right)\left[\sum_{h=1}^{m}(\hat{y}_{n+h} - y_{n+h})^2\right]} \qquad (2.117)$$

Ou pode ser baseada no erro absoluto médio percentual *(mean absolute percentage error* -MAPE) dado por:

$$MAPE = \left(\frac{1}{m}\right)\sum_{h=1}^{m}\frac{|\hat{y}_{n+h} - y_{n+h}|}{y_{n+h}} \qquad (2.118)$$

*(extraído de Oliveira, 2003)

3 REDES NEURAIS ARTIFICIAIS (ANS).

3.1 INTRODUÇÃO.

Em muitas aplicações do mundo real, queremos que os nossos computadores executem problemas complexos de reconhecimento de padrões. Já como os computadores convencionais não são obviamente feitos para solucionar este tipo de problema, foram necessárias características fisiológicas do cérebro como a base para os nossos novos modelos. Assim, a tecnologia começou a ser conhecida como Sistemas Neurais Artificiais (ANS), ou simplesmente redes neurais. Talvez os modelos que iremos discutir aqui permitirão eventualmente produzir máquinas que possam interpretar padrões complexos.

Começamos definindo uma estrutura NN (Neural Network) como uma coleção de processadores paralelos conectados juntos na forma de um gráfico dirigido, organizado tal que a estrutura da rede se encaixa ao problema que está sendo considerado. Referindo-se a figura 3.1 como um típico diagrama de rede, podemos esquematicamente representar cada elemento processado (ou unidade) na rede como um nó, com as conexões entre as unidades indicadas pelos arcos. Nós indicaremos o sentido do fluxo de informação na rede com o uso de setas nas conexões.

Primeiramente, para simplificar o estudo das Redes Neurais, iremos começar utilizando um exemplo que, de certa forma, não se assemelha ao que iremos utilizar no projeto, mas que irá facilitar o melhor entendimento. No projeto, iremos utilizar uma Rede Neural para prever variações financeiras (Forecast Neural Network) e o exemplo a seguir trata-se uma Rede Neural que fará reconhecimento de padrões (figuras em bmp convertidas para o computador). O exemplo a seguir irá receber números em formato de figura (escritos a mão) e tentará reconhecer, transformando-o em código ASCII.

Para simplificar nosso exemplo, restringiremos o número dos caracteres que a rede neural deve reconhecer aos 10 dígitos decimais, 0, 1. . . , 9, melhor que usar todos os caracteres ASCII. Adotamos este agrupamento para somente facilitar o exemplo; entretanto, não há nenhuma razão porque uma ANS não poderia ser usada para reconhecer todos os caracteres, indiferente do uso ou estilo.

Desde que nosso objetivo é fazer com que a rede neural determine quais dos 10 dígitos um caractere particular escrito à mão é, podemos criar uma estrutura da rede que tenha 10 unidades de saída discretas (ou processadores), um para cada caractere que será identificado. Esta estratégia simplifica a função de discriminação de caractere da rede, porque permite que usemos uma rede que contenha unidades binárias na camada de saída (por exemplo, para todo o teste padrão dado da entrada, nossa rede deve ativar uma e somente uma das 10 unidades de saída, representando os 10 dígitos que estamos tentando reconhecer e que a entrada mais se assemelha). Além disso, se insistirmos que as unidades de saída se comportam de acordo com uma estratégia on-off simples, o processo de converter um sinal de entrada em um sinal de saída transforma-se em uma função simples.

Baseado nestas considerações, sabemos agora que nossa rede deve conter 10 unidades binárias como sua estrutura de saída. Similarmente, devemos determinar como modelaremos o caractere de entrada da rede. Mantendo-se na mente que temos indicado já uma preferência para unidades de saída binária, podemos outra vez simplificar nossa tarefa se modelarmos os dados de entrada como um vetor que contém os elementos binários, que permitirão que usemos uma rede com somente um tipo de unidade processada. Para criar este tipo de entrada, utilizaremos a idéia de vídeo e transformaremos o caractere em pixels. Faremos sob medida arbitrariamente a imagem do pixel como uma matriz 10 x 8, fazendo com que um 1 represente um pixel que seja "on" e um 0 um pixel que seja "off."

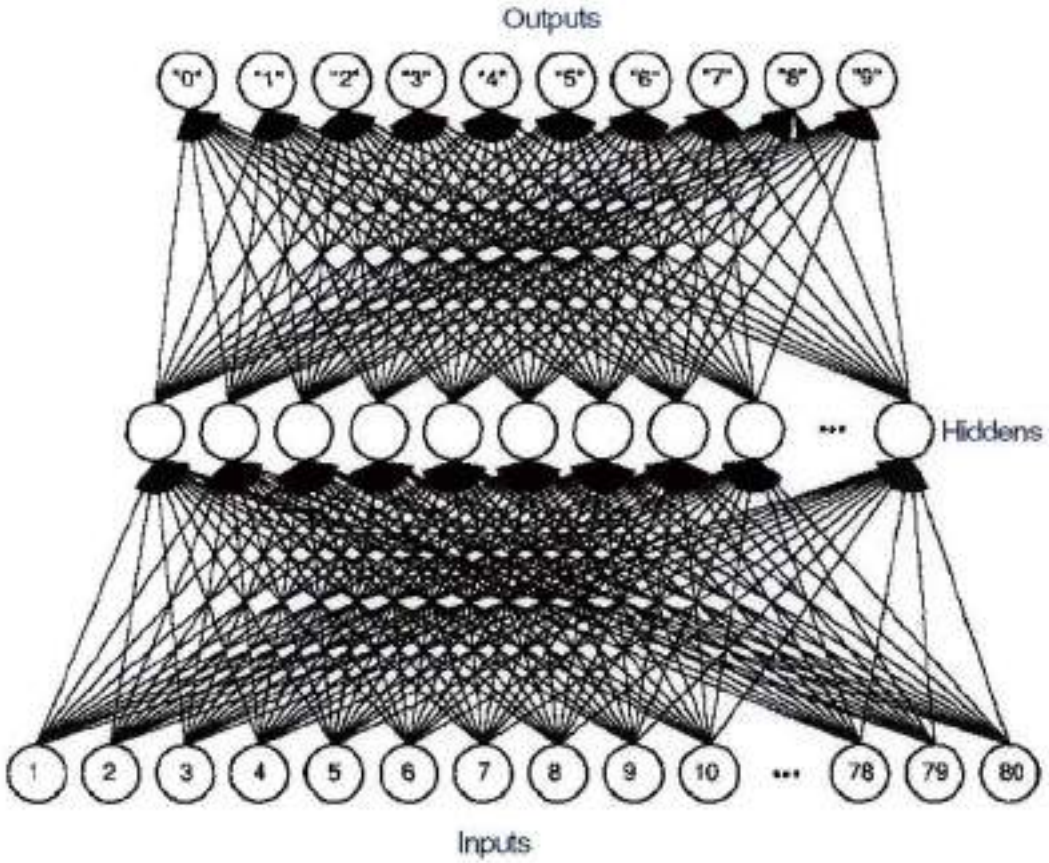


Figura 3.1 Este diagrama esquemático representa o problema do reconhecimento de caractere descrito no texto. Neste exemplo, a aplicação de um teste padrão da entrada na camada inferior dos processadores pode fazer com que muitas das unidades da segunda camada, ou hidden-layer, sejam ativadas. A atividade na hidden-layer deve então ativar uma unidade da camada de saída que está associada com o padrão a ser identificado. Deve também notas o grande número de conexões necessárias para esta rede relativamente pequena.

Além disso, podemos dissecar esta matriz em uma fileira de vetores, que podem então ser concatenados em um único vetor em fila de dimensão 80. Assim, definimos agora a dimensão e as características do padrão de entrada da nossa. Neste momento, tudo que remanesce deve ser feito para o número das unidades processadas (chamadas unidades escondidas – hidden-units) que deve ser usado internamente, conectá-las às unidades de entrada e de saída definidas utilizando conexões com

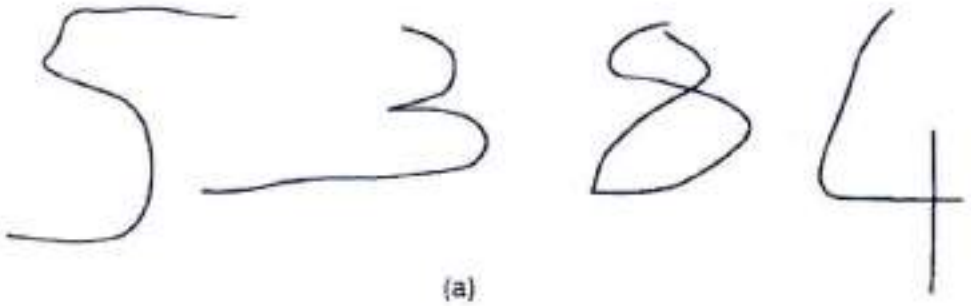
pesos distintos. Este conceito de aprendizagem pelo exemplo é extremamente importante. Como veremos, uma vantagem significativa de uma abordagem por ANS para solucionar um problema é que não necessitamos ter um processo bem definido algoritmicamente para converter uma entrada em saída, tudo que necessitamos para a maioria das redes é uma coleção de exemplos representativos da tradução desejada. A ANS adapta-se então para reproduzir as saídas desejadas quando apresentada com as entradas do exemplo.

Além disso, como nossa rede do exemplo ilustra, uma ANS é robusta no sentido que responderá com uma saída mesmo quando apresentada com entradas de que nunca viu antes, como os testes do padrão que contêm o ruído. Se o ruído da entrada não apagar a imagem do caractere, a rede produzirá uma suposição boa usando aquelas parcelas da imagem que não foram obscurecidas e da informação que armazenou sobre como os caracteres são dispostos. A habilidade inerente de tratar os padrões ruidosos ou obscurecidos é uma vantagem significativa de uma aproximação da ANS sobre uma solução algoritmica tradicional. Ilustra também uma máxima da rede neural: O poder de uma aproximação da ANS não necessariamente está na elegância de uma solução particular, mas sim em uma generalização da rede para encontrar sua própria solução aos problemas particulares, dado somente exemplos do comportamento desejado.

Uma vez que nossa rede é treinada adequadamente, podemos mostrar-lhe imagens dos numerais escritos por pessoas cuja escrita não foi usada para treinar a rede. Se o treinamento for adequado, a informação que propaga através da rede resultará em um único elemento na saída que tem um 1 como valor binário, e essa unidade será correspondente ao numeral que foi escrito. A Figura 3.2 ilustra os caracteres que a rede treinada pode reconhecer, assim como diversos que não pode. Na discussão precedente, fazemos alusão a dois tipos diferentes de operação da rede: modalidade de treinamento e modalidade de produção. A natureza distinta destas duas modalidades de operação é uma outra característica útil da tecnologia da ANS. Se notarmos que o processo de treinar a rede é simplesmente um modo de codificar a informação sobre o problema a ser resolvido e que a rede gasta a maioria de seu

tempo produtivo se exercitando após o término do treinamento, teremos descoberto meios de permitir que os sistemas automatizados evoluam sem uma reprogramação explicita.

Como um exemplo de como podemos nos beneficiar desta separação, considere um sistema que utilize uma simulação do software de uma rede neural como parte de sua programação. Neste caso, a rede seria modelada no sistema computadorizado do host como um conjunto de estruturas de dados que representasse o estado atual da rede. O processo de treinar a rede é simplesmente um problema de alterar os pesos da conexão sistematicamente para codificar os relacionamentos desejados de entrada-saida Se codificarmos o simulador da rede tal que as estruturas de dados usadas pela rede estão alocadas dinamicamente, e inicializadas pela leitura de dados da conexão-peso de um arquivo do disco, podemos também criar um simulador da rede com uma estrutura similar em outro sistema computadorizado off-line. Quando o sistema em linha deve mudar para satisfazer as exigências operacionais novas, podemos desenvolver os pesos novos das conexões off-line treinando o simulador da rede no sistema remoto. Mais tarde, podemos atualizar o sistema operacional simplesmente mudando o arquivo de inicialização da conexão-peso da versão precedente à versão nova produzida pelo sistema off-line.



(a)

(b)

Figura 3.2: Os caracteres escritos à mão variam extremamente, (a) estes caracteres foram reconhecidos pela rede da figura 3.2; (b) estes caracteres não foram reconhecidos.

Estes exemplos mostram a habilidade das redes neurais de tratar complexos problemas de reconhecimento de padrões, mas não são indicadores dos limites da tecnologia. Muitos são os usos das redes neurais, desde diagnosticar problemas já existentes e sintomas, como traçar modelos topográficos complexos, ou mesmo na utilização de padrões temporais (o qual iremos tratar mais à frente).

Finalmente, a distinção feita entre os sistemas artificiais e naturais é intencional. Nós não podemos enfatizar demasiadamente o fato que os modelos da ANS que iremos examinar suporte somente semelhanças esperadas de suas contrapartes biológicas. O que é importante sobre estes modelos é que mostram todos os comportamentos úteis

do aprendizado, do reconhecimento, e da aplicação de relacionamentos entre objetos e padrões dos objetos do mundo real. Assim, fornecem-nos um novo conjunto de ferramentas capazes de auxiliar nas soluções de problemas complexos.

3.2 Dos neurônios à ANS.

Nesta parte, faremos uma transição de algumas das idéias recolhidas da neurobiologia às estruturas idealizadas que dão forma à base da maioria dos modelos de ANS. Nós descrevemos primeiramente um neurônio artificial geral que incorpore a maioria das caracteristicas que necessitaremos para as discussões futuras de modelos específicos.

3.2.1 Os elementos gerais de processamento.

Os elementos computacionais individuais que fazem à maioria dos modelos artificiais dos sistemas neurais são raramente chamados de neurônios artificiais; estes são frequentemente referenciados como nós, unidades, ou elementos de processamento (PEs). Um outro ponto a ter em mente é que não é sempre apropriado pensar nos elementos de processamento em uma rede neural como estando em um relacionamento one-to-one com os neurônios biológicos reais. É às vezes melhor imaginar um único elemento de processamento como o representante da atividade coletiva de um grupo de neurônios. Não somente esta interpretação nos ajudará a evitar a armadilha do discurso como se nossos sistemas eram modelos reais do cérebro, mas também tornará o problema mais fácil de ser solucionado quando estamos tentando modelar o comportamento de alguma estrutura biológica.

A Figura 3.3 mostra o modelo geral do PE. Cada PE é numerado, esse na figura que é o i_{th}. Advertindo que não se deve fazer demasiadas analogias biológicas, agora ignoraremos os conselhos e faremos alguns por nossa conta. Para o exemplo, como um neurônio real, o PE tem muitas entradas, mas tem somente uma única saída, que pode alimentar todos os outros PEs da rede. A entrada que o i_{th} recebe do PE do j_{th} é indicada como Xj (note que este valor é também a saída do nó do j_{th}, apenas como a saída gerada pelo nó do i_{th} é chamada x_j). Cada conexão ao PE do i_{th} tem associada

com ela uma quantidade chamada peso ou força da conexão. O peso na conexão do nó do j_{th} ao nó do i_{th} é denotado w_{ij}. Todas estas quantidades têm analogias no modelo padrão do neurônio: A saída do PE corresponde à freqüência do acendimento do neurônio, e o peso corresponde à força de conexão sinapse entre os neurônios. Em nossos modelos, estas quantidades serão representadas como números reais.

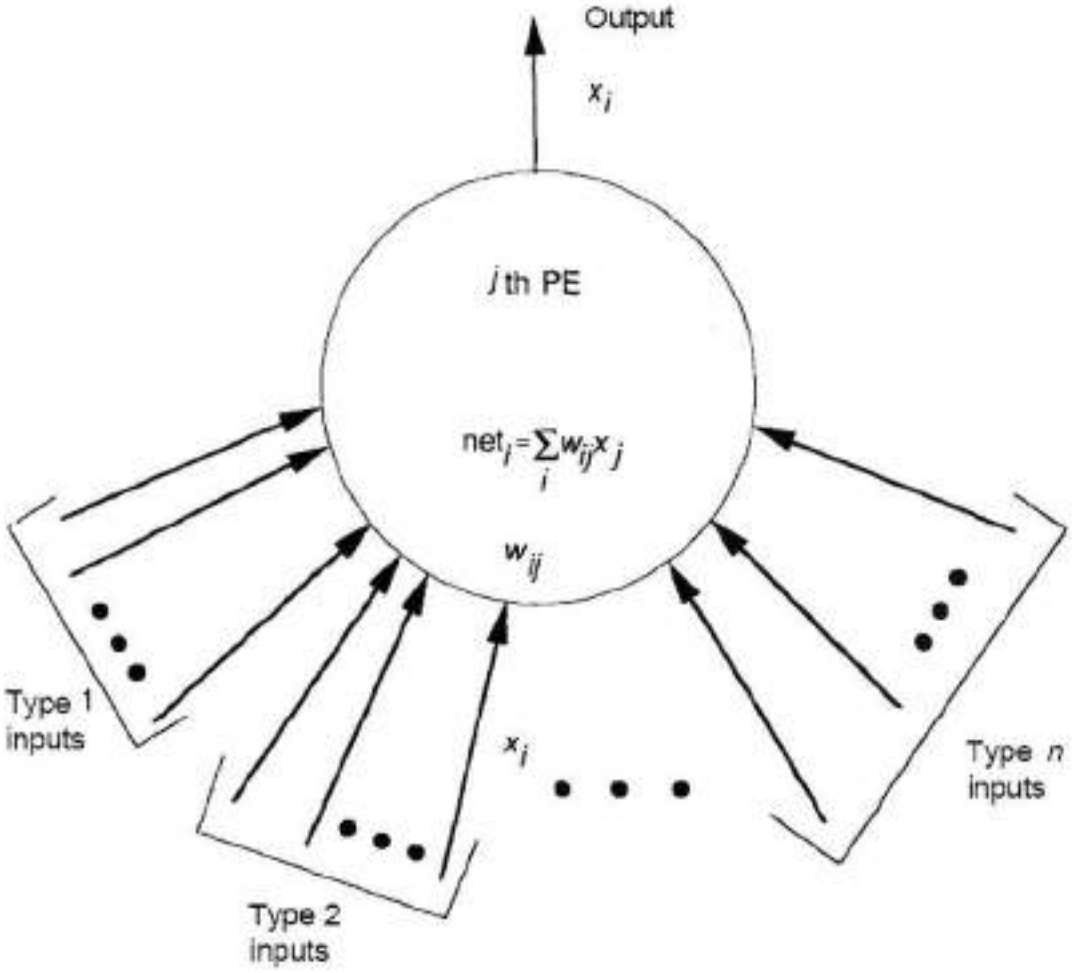


Figura 3.3: Esta estrutura representa um único PE em uma rede. As conexões da entrada são modeladas como setas de outros elementos de processamento. Cada conexão de entrada associou uma quantidade, w_{ij}, chamada peso. Há um único valor de saída, que pode alimentar outras unidades.

Observe que as entradas do PE estão segregadas em vários tipos. Esta segregação reconhece que uma conexão particular da entrada pode ter diversos efeitos. Por exemplo, uma conexão da entrada pode ser excitada ou inibida. Em nossos modelos, as conexões excitadas têm pesos positivos, e as conexões inibidas têm pesos negativos. Outros tipos são possíveis. As conexões excitadas e inibidas são consideradas geralmente juntas, e constituem as formas mais comuns de entrada de um PE.

Cada PE determina um valor de entrada da rede baseado em todas suas conexões da entrada. Na ausência de conexões especiais, calculamos tipicamente a entrada líquida somando os valores da entrada, multiplicado por seus pesos correspondentes. Ou seja, a entrada líquida à unidade do i_{th} pode ser escrita como:

$$\mathrm{net}_i = \sum_j x_j w_{ij} \tag{3.1}$$

onde o índice j funciona sobre todas as conexões ao PE. Note que a excitação e a inibição estão automaticamente explicadas pelo sinal dos pesos. Este cálculo da somatória dos produtos tem um papel importante nas simulações da rede que nós estaremos descrevendo mais tarde. Porque há frequentemente um número muito grande de interconexões em uma rede, a velocidade em que este cálculo pode ser executado determina geralmente o desempenho de toda a simulação dada da rede.

Uma vez que a entrada líquida é calculada, esta está convertida a um **valor de ativação**, ou simplesmente à **ativação**, para o PE. Podemos escrever este valor de ativação como

$$a_i(t) = F_i(a_i(t - 1), \mathrm{net}_i(t)) \tag{3.2}$$

para denotar que a ativação é uma função explícita da entrada líquida. Observe que a ativação atual pode depender do valor precedente da ativação, $a(t - 1)^2$. Incluímos esta dependência na definição para generalizar. Na maioria dos casos, a entrada de ativação e da rede é idêntica, e os termos são usados permutavelmente. Às vezes, a entrada da ativação e da rede não é a mesma, e devemos prestar atenção nesta diferença. Em sua maioria, entretanto, poderemos usar a ativação no sentido da entrada líquida, ou vice versa.

Uma vez a ativação do PE seja calculada, podemos determinar o valor da saída aplicando uma função de saída:

$$x_i = f_i(a_i) \tag{3.3}$$

Desde que, geralmente, a, = net_i, esta função é escrita normalmente como:

$$x_i = f_i(\text{net}_i) \tag{3.4}$$

Um razão para enfatizar o assunto de ativação versus a entrada líquida é que o termo **função de ativação** é, às vezes, usado para referenciar a função f_i, que converte o valor líquido de entrada, net_i, no valor do nó de saída x_i. Neste texto, consistentemente usaremos o termo função de saída para $f_i(\ )$ das Eqs. (3.3) e (3.4). Esteja ciente, entretanto, que a literatura não é sempre consistente neste respeito.

Quando nós estivermos descrevendo a base matemática para modelos da rede, será frequentemente útil pensar na rede como um **sistema dinâmico**, como um sistema que evolua com o tempo. Para descrever tal rede, escreveremos as equações diferenciais que descrevem a taxa do tempo de mudança das saídas dos vários PEs. Por exemplo, $\dot{x}_i = g_i(x_i, net_i)$ representa uma equação diferencial geral para a saída do PE do *ith*, onde o ponto acima do x é a diferencial relacionada ao tempo. Como net_i depende das saídas de muitas outras unidades, realmente temos um sistema de equações diferenciais. Como exemplo, apenas olhe a equação:

$$\dot{x}_i = -x_i + f_i(\text{net}_i) \tag{3.5}$$

para a saída do PE de *ith*. Aplicamos alguns valores de entrada ao PE de modo que $net_i > 0$. Se as entradas remanescerem por um tempo suficientemente longo, o valor da saída alcançará um valor de equilíbrio, quando $x_i = 0$, dado por:

$$x_i = f_i(\text{net}_i) \tag{3.6}$$

que é idêntico à Eq. (3.4). Podemos frequentemente supor que os valores de entrada remanescem até que o equilíbrio seja alcançado. Uma vez que a unidade tem um valor não zero de saída, a remoção das entradas fará com que a saída retorne a zero.

Se $net_i = 0$, então:

$$\dot{x}_i = -x \tag{3.7}$$

que significa que $x \longrightarrow 0$.

É também útil ver a coleção de valores do peso como um sistema dinâmico. Recorde a discussão anterior, onde foi afirmado que aprender é um resultado da modificação da força de junções das sinapses entre os neurônios. Em uma ANS, o aprendizado geralmente é realizado pela modificação dos valores do peso. Podemos escrever um sistema de equações diferenciais para os valores do peso, $\dot{w}_{ij} = G_i(w_{ij}, x_i, x_j, \ldots)$, onde G representa a **lei de aprendizagem**. O processo de aprendizagem consiste encontrar os pesos que codificam o conhecimento que queremos que o sistema aprenda. Para a maioria dos sistemas realísticos, não é fácil determinar uma solução fechada para este sistema de equações. As técnicas existem, entretanto, esse resultado é uma aproximação aceitável a uma solução. Provar a existência de soluções estáveis a tais sistemas de equações é uma área de

pesquisa ativa em redes neurais hoje em dia, e provavelmente continuará a existir por algum tempo.

3.2.2 Forma Vetorial.

Em muitos dos modelos de rede que discutiremos, é útil descrever determinadas quantidades em relação aos vetores. Pense em uma rede neural composta de diversas camadas (**layers**) de elementos de processamento idênticos. Se uma camada particular possui n unidades, as saídas dessa camada podem ser pensadas como um vetor de n dimensões, $X = (x_1, X2, \bullet\bullet, x_n)^t$, onde t significa transposta. Em nossa notação, os vetores serão escritos em negrito, como x, serão supostos para ser colunas do vetor. Quando são descritos em forma de linha, o símbolo *transposto* estará adicionado para indicar que o vetor deve realmente ser pensado como uma coluna. Inversamente, a notação $\mathbf{x}^t$ indica uma fileira do vetor.

Suponha que o vetor de n dimensões da saída descrito anteriormente fornece os valores de entrada de cada unidade em uma camada m-dimensional (uma camada com m unidades). Cada unidade na camada m-dimensional terá os pesos de n associados com as conexões da camada precedente. Assim, há m n-dimensional vetores peso associados com esta camada; há um n-dimensional vetor peso para cada uma das camadas de m. O vetor peso da unidade *ith* pode ser escrito como $\mathbf{w}_i = (w_{i1}, w_{i2}, \bullet\bullet, w_{in})^t$. Um sobrescrito pode ser adicionado à notação do peso para distinguir entre pesos em camadas diferentes.

A entrada líquida à unidade *ith* pode ser escrita nos termos do produto interno, ou no produto pontual, do vetor de entrada e do vetor peso. Para vetores de dimensões iguais, o produto interno é a soma dos produtos dos componentes correspondentes dos dois vetores. Utilizando a notação anterior,

$$\text{net}_i = \sum_{j=1}^{n} x_j w_{ij} \tag{3.8}$$

onde n é o número de conexões à unidade *ith*. Esta equação pode ser escrita sucintamente na notação do vetor como

$$\text{net}_i = \mathbf{x} \cdot \mathbf{w}_i \tag{3.9}$$

ou

$$\text{net}_i = \mathbf{x}^t \mathbf{w}_i \tag{3.10}$$

Note também, por causa das regras de multiplicação de vetores:

$$\mathbf{x}^t \mathbf{w}_i = \mathbf{w}_i^t \mathbf{x} \tag{3.11}$$

Falaremos frequentemente de vetores de entrada, vetores de saída e de vetores peso. mas tenderemos a reservar a notação do vetor para os casos onde é particularmente apropriado. Os conceitos adicionais de vetor serão introduzidos mais tarde quando necessitados.

3.3 Backpropagation.

Há diversas potenciais aplicações de computador que são difíceis de executar porque há muitos problemas que não se adequam à solução por um processo seqüencial. As aplicações que devem executar alguma tradução de dados complexos, contudo não há nenhuma função predefinida para o processo de tradução, ou aquelas que devem fornecer a melhor aproximação como saída quando apresentados dados ruidosos na entrada são dois exemplos dos problemas deste tipo.

Uma ANS encontrada que é útil em se dirigir aos problemas que requerem o reconhecimento de padrões complexos e que executam funções não triviais é **backpropagation network (BPN)**, formalizada primeiramente por Werbos, e mais tarde por Parker e por Rummelhart e por McClelland. Esta rede, ilustrada na figura 3.4, é projetada para operar como uma rede multi camada (**multilayer**), feedfoward network, usando a modalidade supervisionada de aprendizagem.

Iremos discutir um problema em traçar uma imagem de caractere ASCII, que parece simples primeiramente, mas pode rapidamente oprimir aproximações tradicionais. Então, olharemos como a rede backpropagation se comporta para resolver tal problema.

Para ilustrar alguns problemas que frequentemente observamos quando estamos tentando automatizar aplicações complexas de reconhecimento de padrões, consideraremos o projeto de um programa de computador que deva traduzir uma matriz 5 x 7 dos números binários que representam a imagem .bmp do pixel de um caractere alfanumérico e seu código equivalente 8-bits ASCII. Este problema básico, retratado na figura 3.5, parece ser relativamente trivial à primeira vista. Já que não há nenhuma função matemática óbvia,

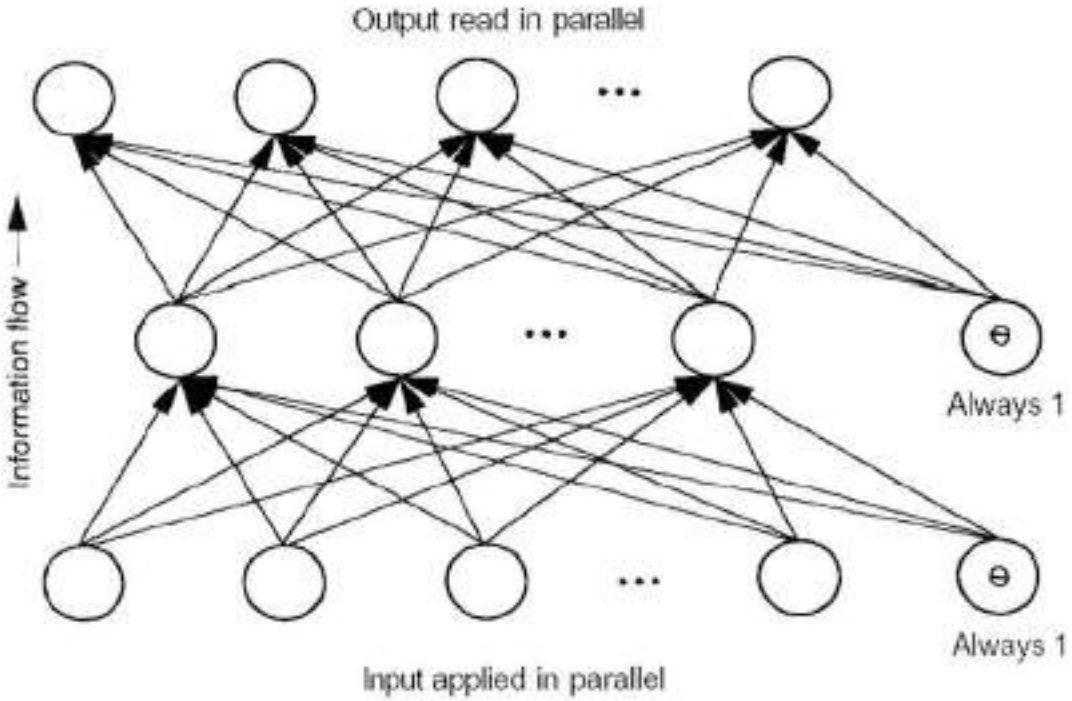
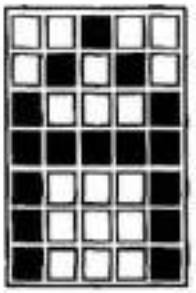


Figure 3.4: A arquitetura geral de uma rede backpropagation.

que faça a tradução desejada, e porque tomaria demasiado tempo (tanto humano quanto tempo de processamento) para executar uma correlação pixel-por-pixel, a melhor solução algoritmica deveria usar uma tabela de consulta. A tabela consulta necessitada para resolver este problema seria um vetor linear uni-dimensional ordenado em pares, cada um na forma:

```
record AELEMENT =
        pattern : long integer;
        ascii : byte;
end record;
```



$0010010101000111111100011000110001_2$

$0951FC631_{16}$

$= 65_{10}$ ASCII

Figure 3.5: Cada caractere é mapeado no seu correspondente código ASCII.

O primeiro é o equivalente numérico do código do padrão em bit, que foi gerado movendo as sete fileiras da matriz para uma única fileira e considerando o resultado para ser um número 35-bit binário. O terceiro é o código ASCII associado ao caractere. A disposição conteria exatamente o mesmo número de pares ordenados como os caracteres a serem convertidos.

Embora a aproximação da tabela de consulta seja razoavelmente rápida e fácil de manter, há muitas situações que ocorrem nos sistemas reais que não podem ser manipulados por este método. Por exemplo, considere o mesmo processo de conversão do pixel da imagem para ASCII em um ambiente mais realístico. Vamos supor que o nosso varredor de imagem de caractere altere um pixel aleatório na matriz da imagem de entrada devido ao ruído quando a imagem foi lida. Este único erro do pixel faria com que o algoritmo de consulta retornasse o código nulo ou errado do ASCII, desde que a combinação entre o padrão de entrada e o padrão de saída do alvo seja exata.

Considere agora a quantidade de software adicional (e, portanto, tempo de processamento da CPU) que deve ser adicionada ao algoritmo da tabela de consulta (**lookup table**) para melhorar a habilidade do computador em adivinhar o caractere da imagem ruidosa. Os erros de bit único são razoavelmente fáceis de encontrar e corrigir. Os erros multi bits tornam-se cada vez mais difíceis à medida que o número de bits de erro aumenta. Para complicar ainda mais o problema, como nosso software poderia compensar o ruído na imagem se esse ruído acontecesse fazendo com que

um "O" seja visto como um "Q", ou um "E" seja visto como um "F"? Se nosso sistema de conversão de caractere tivesse que produzir uma saída exata toda a hora, uma imensa quantidade de tempo de processamento seria gasta para eliminar o ruído do padrão de entrada previamente a tentar traduzi-lo para ASCII.

Uma solução a este dilema é tomar vantagem da natureza paralela das redes neurais para reduzir o tempo requerido por um processador seqüencial para executar um mapeamento. Além, o tempo de desenvolvimento do sistema pode ser reduzido porque a rede pode aprender o algoritmo apropriado sem ter alguém deduzindo adiantadamente.

3.3.1 A abordagem Backpropagation.

Os problemas tais como o exemplo do ruído na conversão da imagem para ASCII são difíceis de serem solucionados pelo computador devido à incompatibilidade entre a máquina e o problema. A maioria dos sistemas computadorizados de hoje foi projetado para executar funções matemáticas e de lógica em velocidades que são incompreensíveis aos seres humanos. Mesmo os menos sofisticados desktops de hoje em dia podem executar centenas de milhares de comparações ou de combinações numéricas a cada segundo.

Entretanto, como o nosso exemplo ilustra uma abordagem matemática não é o mais necessário para o reconhecimento destes padrões complexos em ambientes ruidosos. De fato, uma busca algorítmica uniforme em um espaço relativamente pequeno de entrada pode provar ser um consumidor de tempo. O problema é a própria natureza seqüencial do computador; o ciclo da arquitetura de von Neumann permite que a máquina execute somente uma operação de cada vez. Na maioria de casos, o tempo requerido pelo computador para executar cada instrução é curto. Entretanto, para as aplicações que devem procurar através de um espaço grande de entrada, ou tente correlacionar todas as permutações possíveis de um teste de padrão complexo, o tempo requerido por uma máquina muito rápida pode rapidamente tornar-se intolerável.

O que necessitamos é um novo sistema de processamento que possa examinar todos os pixels na imagem paralelamente. Idealmente, tal sistema não teria que ser programado explicitamente, iria se adaptar ou aprender o relacionamento do conjunto de padrões do exemplo, e poderia aplicar o mesmo relacionamento aos novos padrões de entrada. Este sistema poderia focar nas características de uma entrada arbitrária que se assemelham a outros padrões vistos previamente, como aqueles pixels na imagem ruidosa. Felizmente, tal sistema existe e é chamado **backpropagation network (BPN)**.

3.3.2 Dados de treinamento.

Infelizmente, não há uma definição correta sobre os termos suficientes e apropriados relacionados ao número de dados para o treinamento da BPN. Como muitos aspectos de sistemas de redes neurais, a experiência é frequentemente o melhor professor. Como acabamos ganhando mais habilidades à medida que utilizamos a rede neural, ganharemos também com o tempo a habilidade de escolher melhor o conjunto de aprendizagem.

Geralmente, você pode utilizar quantos dados forem disponíveis para treinar a rede neural, embora não seja necessária a utilização de todos. Dos dados disponíveis para o treinamento, um pequeno subconjunto é frequentemente, tudo que irá necessitar para treinar com sucesso uma rede neural. Os dados restantes podem ser usados para testar a rede ou para verificar se a rede executa o mapeamento desejado utilizando vetores de entrada não usados no treinamento.

O BPN é bom para generalizações. Nesse caso, generalizações significam que, dado diversas entradas em diferentes vetores, pertencendo à mesma classe, uma BPN aprenderá as similaridades do vetor de entrada. Os dados irrelevantes serão ignorados. Como exemplo, suponha que queremos treinar uma rede para determinar se um número bipolar de comprimento 5 é uniforme ou impar. Com somente um conjunto pequeno de exemplos foi utilizado para o treinamento, o BPN ajustará seus

pesos de modo que uma classificação seja feita unicamente na base do valor do bit menos significativo no número. A rede aprende a ignorar os dados irrelevantes nos outros bits.

Em contraste à generalização, o BPN não extrapola bem. Se um BPN for treinado inadequadamente ou insuficientemente em uma classe particular de vetores de entrada, a identificação subseqüente dos membros dessa classe pode ser não confiável. Certifique-se de que os dados de treinamento cobrem o espaço inteiro das entradas esperadas. Durante o processo de treinamento, selecione pares de vetores de treinamento aleatórios do conjunto. Em nenhum caso, não treine a rede completamente com os vetores da entrada de uma classe, e então a comute a uma outra classe: A rede irá esquecer do treinamento original.

Se a função de saída for sigmóide, então terá que escalar os valores de saída. Por causa da forma da função sigmóide, as saídas da rede nunca podem alcançar 0 ou 1. Conseqüentemente, use valores tais como 0.1 e 0.9 para representar os menores e maiores valores de saída. Como exemplo, também pode descolar o sigmóide de modo que os valores limitantes sejam em torno de ±0.4. Além disso, você pode mudar a inclinação da porção linear da curva sigmóide incluindo uma constante multiplicativa no exponencial. Há muitas possibilidades que largamente dependem do problema a ser solucionado.

3.3.3 Arquitetura da Rede Neural.

Geralmente, três camadas são suficientes. Entretanto, às vezes, um problema parece ser mais fácil de resolver com mais de uma camada intermediária. O tamanho da camada da entrada é ditado geralmente pela natureza da aplicação. Você pode frequentemente determinar o número de nós de saída decidindo-se se você quer valores analógicos ou valores binários nas unidades de saída.

Determinar o número de unidades que será usado na camada intermediária geralmente não é tão direto como é para as camadas de entrada e de saída. A idéia

principal é a menor quantidade de unidades na camada intermediária possível, porque cada unidade irá adicionar uma carga no processador durante as simulações.

Pesos devem ser inicializados por pequenos e randômicos valores, algo em torno de ±0.5, até o valor bias, θ_i, que aparece na equação. É prática comum tratar este valor bias como um outro peso, que seja conectado a uma unidade fictícia que tenha sempre uma saída 1. Para ver como este esquema trabalha, Eq. (3.12):

$$\text{net}_{pk}^{o} = \sum_{j=1}^{L} w_{kj}^{o} i_{pj} + \theta_k^{o} \tag{3.12}$$

Pela definição, $\theta_k^{o} = w_{k(L+1)}^{o}$ e $i_{p(L+1)} = 1$ podemos escrever:

$$\text{net}_{pk}^{o} = \sum_{j=1}^{L+1} w_{kj}^{o} i_{pj} \tag{3.13}$$

Então θ_k^{o} tratado como um peso, e participando do processo de aprendizagem como um peso. Uma outra possibilidade é simplesmente remover completamente os termos bias; seu uso é opcional.

Uma maneira de aumentar a velocidade de convergência é utilizar a técnica chamada de **momentum**. Quando calculando o valor do peso alterado, $\Delta_p w^{o}$, adicionamos uma fração da mudança anterior. Este termo adicional tende a manter as mudanças de peso no mesmo sentido, por isso o termo momentum. A equação de mudança de peso na camada de saída torna-se então

$$w_{kj}^{o}(t+1) = w_{kj}^{o}(t) + \eta \delta_{pk}^{o} i_{pj} + \alpha \Delta_p w_{kj}^{o}(t-1) \tag{3.14}$$

com uma equação similar na camada intermediária. Em Eq. (3.14), α é o parâmetro do momentum, e é ajustado geralmente a um valor positivo menor que 1. O uso do termo do momentum é também opcional.

Um tópico final diz respeito à possibilidade de convergência a um mínimo local no espaço do peso. A Figura 3.6 ilustra a idéia. Uma vez que uma rede se estabelece em um mínimo, se local ou global, o aprendizado pára. Se um mínimo local for alcançado, o erro nas saídas da rede pode ainda ser inaceitavelmente elevando. Felizmente, este problema não parece causar na prática muita dificuldade. Se uma rede parar de aprender antes de alcançar uma solução aceitável, uma mudança no número de nós intermediário ou nos parâmetros de aprendizagem reparará frequentemente o problema; ou podemos simplesmente começar por um conjunto diferente de pesos iniciais. Quando uma rede alcança uma solução aceitável, não há nenhuma garantia que alcançou o mínimo global melhor que local. Se a solução fosse aceitável de um ponto de vista do erro, não importa se o mínimo é global ou local, ou mesmo se o treinamento esteve parado em algum ponto antes que um mínimo verdadeiro tenha sido alcançado.

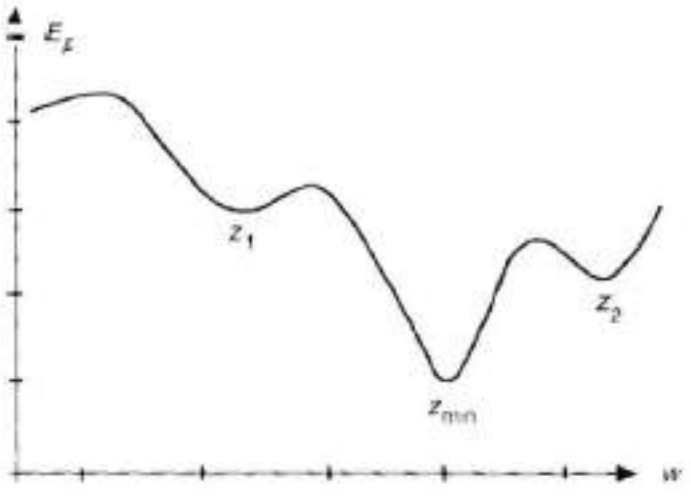


Figura 3.6: Este gráfico mostra uma hipotética superfície de erro em um espaço de pesos.

3.4 Criação de uma Rede Neural de previsão.

3.4.1 Seleção das variáveis.

O sucesso de uma rede neural depende de um perfeito entendimento do problema. Conhecer cada variável de entrada é muito importante em previsão de um mercado crítico. Isto é mais fácil ao falar do que realmente fazer porque a razão relacionada às redes neurais é a grande habilidade de detectar relacionamentos não lineares entre diferentes quantidades de variáveis. Entretanto, teorias econômicas podem auxiliar na escolha das variáveis, as quais são importantes elementos para uma boa previsão. Neste momento do processo de desenvolvimento, a grande preocupação está no conjunto principal de dados que será desenvolvido mais à frente.

Os pesquisadores financeiros interessados em previsões dos preços de mercado devem decidir se usam ambas as técnicas e fundamentos econômicos como entrada em um ou mais mercados. As entradas técnicas são definidas como valores retardados da variável dependente ou dos indicadores calculados dos valores retardados. As entradas fundamentais são as variáveis econômicas que acreditamos influenciar a variável dependente. Modelos simples de redes neurais usam valores retardados das variáveis dependentes ou a diferença como entrada.

Uma aproximação mais popular deve calcular os vários indicadores técnicos que são baseados somente sobre preços passados (e ocasionalmente volume e/ou o interesse) do mercado que está sendo previsto. Como uma melhoria adicional, os dados do mercado interno podem ser usados desde que a ligação seja próxima entre todos os tipos de mercado, doméstico e internacional, e sugere-se que usar entradas técnicas de um número de mercados inter-relacionados deve melhorar a performance da previsão. Por exemplo, os dados do mercado interno tais como o mercado alemão, taxas transversais e diferenciais da taxa de interesse podiam ser usados como entradas das redes neurais ao prever o mercado. A informação fundamental tal como

o balanço, o suprimento de moeda, ou o índice de preço por atacado pode também ser útil.

A freqüência dos dados depende dos objetivos do pesquisador. Um típico off-floor trader na bolsa ou no mercado futuro de comodities usaria muito provavelmente dados diários ao projetar uma rede neural como um componente de um sistema de negociação maior. Um investidor com um horizonte mais de longo prazo pode usar dados semanais ou mensais como entradas à rede neural para formular a melhor mistura de recursos ao invés de usar uma estratégia de compra passiva ou de espera. Um economista que prevê o PIB, o desemprego, ou outros indicadores macroeconômicos usaria provavelmente dados mensais ou trimestrais.

3.4.2 Coleção de Dados.

O pesquisador deve considerar custo e disponibilidade ao coletar dados para as variáveis escolhidas na etapa precedente. Os dados técnicos estão prontamente disponíveis por muitos fabricantes por um custo razoável ao passo que a informação fundamental é mais difícil de obter. O tempo gasto coletando dados não pode ser usado para o pré-processamento, o treinamento, e o desempenho de avaliação da rede. O fabricante deve ter uma reputação de fornecer dados de qualidade elevada; entretanto, todos os dados devem ainda ser verificados para ver se há erros examinando as mudanças do dia a dia, consistência lógica (por exemplo, alta maior ou igual ao fechamento, abertura maior ou igual à baixa) e faltando observações.

O problema da falta de observações que ocorre frequentemente pode ser lidado de algumas maneiras. Todas as observações faltantes podem ser descartadas ou uma segunda opção é supor que as observações faltantes surgem da interpolação ou do uso da média dos valores próximos.

Quando usado dados fundamentais como uma entrada de uma rede neural quatro pontos devem ser mantidos em mente. Primeiramente, o método de calcular o indicador fundamental deve ser consistente sobre a série de tempo. Em segundo, os

dados não devem retroativar depois que sua publicação inicial como é feito geralmente nas bases de dados desde que os números revisados não estejam disponíveis na previsão real. Em terceiro lugar, os dados devem ser apropriadamente retardados como uma entrada na rede neural desde que a informação fundamental não esteja disponível tão rapidamente quanto citações de mercado. Quarto, o pesquisador deve confiar que a fonte continuará a publicar a informação fundamental particular ou outras fontes idênticas.

3.4.3 Pré-processamento.

O pré-processamento dos dados tem a função de analisar e transformar as variáveis de entrada e saída para minimizar o ruído, destacar relações importantes, detecta tendências e nivelar a distribuição das variáveis para acompanhar o aprendizado da rede neural nos padrões relevantes. Desde que as redes neurais são confrontadores de padrões, a representação dos dados é crítica em projetar uma rede bem sucedida. As variáveis de entrada e de saída para os dados coletados são raramente alimentadas na rede no formato não tratado. Muito embora, os dados não tratados devem ser escalados entre os limites superiores e mais baixos de funções de transferência (geralmente entre 0 e 1 ou –1 e 1).

Duas das transformações mais comuns dos dados em redes neurais de previsão e mesmo as tradicionais são a primeira diferencial e pegar logaritmo natural da variável. Primeira diferença, ou utilizar mudanças em uma variável, pode ser usado para remover uma forma linear da tendência os dados. A transformação logarítmica é útil para quando os dados variam entre valores bem pequenos e é caracterizada por uma distribuição direita prolongada em forma de cauda. A transformação logarítmica converte também relações multiplicativas ou razões em adições que simplifica e melhora a fase de treinamento da rede neural.

Uma outra transformação popular dos dados é usar razões das variáveis de entrada. A proporção destaca as importantes relações (por exemplo suíno/milho, proporções entre indicadores financeiros) e, ao mesmo tempo, conserva os graus de liberdade já

que poucos neurônios da entrada são requeridos para codificar as variáveis independentes.

Além das diferenças, registros, e as proporções, a análise técnica pode fornecer uma rede neural com uma riqueza dos indicadores incluindo uma variedade de médias móveis, osciladores, movimentos direcionais e filtros de volatilidade. É uma boa idéia utilizar a variação de diferentes indicadores para reduzir a redundância da variável e fornecer à rede a habilidade de adaptar-se à condição de mercados em mudança com pós-treinamentos periódicos.

Nivelar dados de entrada e de saída usando médias móveis simples ou exponenciais é empregado frequentemente. O trabalho empírico em testar a hipótese eficiente do mercado encontrou que os preços exibem a dependência do tempo ou uma auto correlação positiva quando as mudanças de preço em torno de uma tendência forem um tanto aleatórias. Conseqüentemente, tentar predizer mudanças de preço em torno da tendência usando preços não filtrados ou preços que mudam constantemente como entrada pode ser bem difícil. Usando médias móveis para nivelar as variáveis independentes e prever tendências podem ser uma aproximação mais promissora.

Amostrar ou filtrar dados refere-se a retirar observações da fase de treinamento e teste para criar uma distribuição mais uniforme. O tipo de filtragem empregada deve ser consistente com os objetivos do pesquisador. Por exemplo, um histograma de mudanças do preço de uma commodity pode revelar muitas pequenas mudanças as quais um operador que tem um perfil especulativo não terá lucro após ter deduzido o custo real da execução. Entretanto, esta densa região da distribuição impactará extremamente o treinamento da rede neural desde que as pequenas mudanças de preço esclareçam a maioria dos fatos do treinamento.

A rede neural minimiza a soma dos erros quadráticos (ou qualquer outra função de erro) acima de todos os fatos do treinamento. Removendo estas pequenas mudanças de preço, o desempenho negociado pode ser melhorado desde que a rede se especialize no preço maior, potencialmente a lucratividade muda. É possível para

sistemas de trade não ser lucrativo mesmo se a rede neural predizer 85% dos pontos críticos, como o ponto crítico pode ser somente mudanças sem importância pequenas de preço.

Por outro lado, um operador que mantém suas posições à noite, interessa-se provavelmente por estar pequenas mudanças de preço. O pesquisador deve ter claro onde exatamente a rede neural irá aprender. Uma outra vantagem de filtrar é um decréscimo no número de observações de treinamento que permite testar mais variáveis de entrada, pesos iniciais aleatórios, ou neurônios intermediários ao invés de grandes séries de dados para o treinamento.

Na prática, o pré-processamento dos dados envolve muitas experimentações e erros. Um método para selecionar variáveis de entrada deve testar várias combinações. Por exemplo, uma lista das "top 20" variáveis que consistem em uma variedade de indicadores técnicos podiam testar primeiramente 10 de cada vez com cada combinação sendo diferente de duas ou três variáveis. Embora computacionalmente intensa, este procedimento reconhece a probabilidade de algumas variáveis serem melhores para a previsão somente quando em combinação com outras variáveis. A teoria do caos e o teste estatístico não podem fazer tal determinação. Também, a lista "top 20" acima pode ser modificada sobre o tempo enquanto o pesquisador ganha a experiência no tipo de pré-processamento que trabalha para a aplicação. Esta aproximação é especialmente útil se o conjunto de treinamento for relativamente pequeno ao número dos parâmetros (pesos) que é provavelmente o caso se todas as 20 variáveis de entradas forem apresentadas à rede neural em uma única vez.

3.4.4 Treinamento, teste e validação.

A prática comum diz que se deve dividir a série de tempo em três conjuntos distintos chamados treinamento, teste, e validação. O conjunto de treinamento é o maior conjunto e é usado pela rede neural para aprender o padrão atual dos dados. O conjunto de teste, varia entre 10% a 30% do conjunto de treinamento, é usado para avaliar a habilidade da generalização da rede supostamente treinada. O pesquisador

deve selecionar a rede que melhor se adapta ao conjunto de teste. Uma verificação final na performance da rede treinada é feita usando o conjunto de validação. O tamanho do conjunto escolhido deve ser um contrapeso entre obter um tamanho de amostra suficiente para avaliar uma rede treinada que tem bastante observações restantes para o treinamento e testar. O conjunto de validação deve consistir nas observações contínuas mais recentes. Deve tomar cuidado para não utilizar o conjunto de validação como um conjunto de teste executando repetidamente uma série de etapas de treinamento-teste-validação e ajustando as variáveis de entrada baseadas no desempenho da rede no conjunto de validação.

O conjunto de teste pode ser aleatoriamente selecionado do treinamento ou consistir em um conjunto de observações imediatamente após o conjunto de treinamento. A vantagem de selecionar fatos aleatórios testando é que o perigo de usar um conjunto de teste caracterizado por um tipo de mercado é evitado pela maioria. Por exemplo, um conjunto de teste pequeno pode somente consistir em preços em uma época de forte alta. O conjunto de teste favorecerá as redes que se especializem em fortes altas à custa das redes que generalizam, e que possuem uma boa performance em fortes altas e fortes baixas. A vantagem de usar as observações que seguem o treinamento como observações de teste é que estas são observações mais recentes (excluindo a fase de validação) que podem ser mais importantes do que dados mais antigos.

Os fatos para serem testados que foram selecionados aleatoriamente não devem ser substituídos no conjunto de treinamento porque isto inclinaria a habilidade de avaliar a generalização especial se o conjunto testado é relativamente grande ao conjunto de treinamento (por exemplo 30%). Um método determinístico, tal como selecionar cada observação n_{th} como uma observação testada, não é recomendado também pois pode resultar em ciclos nos dados provados devido unicamente à técnica de amostragem empregada.

Uma abordagem mais rigorosa para avaliar a rede neural deve usar uma rotina de teste do tipo "walk-forward" conhecida como "sliding" ou deslocamento de janela. Bem popular em sistema de avaliação de commodities, o teste "walk-forward"

consiste em dividir os dados em uma série de conjuntos sobrepondo o conjunto de treinamento-teste-validação. Cada conjunto é movido para a frente com a série de tempo. O teste "walk-forward" tenta simular a negociação na vida real e testar a robustez do modelo com seu treinamento freqüente em uma série de dados de um grande conjunto fora da amostra. No teste "walk-forward", o tamanho do conjunto de validação guia a freqüência do treinamento da rede neural.O treinamento freqüente consome mais tempo, mas permite que a rede adapte-se mais rapidamente às condições de mercado em mudança. A consistência ou a variação do resultado nos conjuntos de dados fora da amostra são uma medida de desempenho importante. Por exemplo, no caso de sistemas de negociação de commodities, uma rede neural com um baixo desempenho de previsão não será implementado para evitar qualquer tipo de risco.

Recomenda-se que o treinamento e os conjuntos testados sejam escalados conjuntamente desde que a finalidade do conjunto testado é determinar a habilidade da rede em generalizar. Entretanto, de nenhuma maneira, o conjunto de validação deve ser usado como um uma verificação final e independente da rede neural. No uso real, o pesquisador não tem nenhuma maneira de saber a escala exata dos valores futuros, mas tem somente uma estimativa razoável na escala de treinamento e/ou conjunto de testes.

3.4.5 Paradigma das Redes Neurais.

Há um número infinito das maneiras para se construir uma rede neural. Neurodinâmica e arquitetura são dois termos usados para descrever a maneira em que uma rede neural é organizada. A combinação de neurodinâmica e arquitetura definem o paradigma da rede neural. Neurodinâmica descreve as propriedades de um neurônio individual tais como sua função de transferência e como as entradas são combinadas. Uma arquitetura de rede neural define sua estrutura incluindo o número de neurônios em cada camada e o número e o tipo de interconexões.

O número dos neurônios da entrada é um dos parâmetros mais fáceis de selecionar, uma vez que as variáveis independentes foram pré-processadas pois cada variável interdependente está representada por seu próprio neurônio de entrada.

3.4.5.1 Número de camadas intermediárias.

A (s) camada (s) intermediária (s) (hidden layers) fornece(m) a rede sua habilidade em generalizar. Na teoria, uma rede neural com uma camada intermediária com um número suficiente de neurônio é capaz de aproximar toda função contínua. Na prática, uma rede neural com uma ou ocasionalmente duas camadas intermediárias são amplamente utilizadas com bons resultados. Aumentando o número de camadas intermediárias, aumenta também o tempo de computação e o perigo de uma pouca generalização da rede (overfitting), o que conduz ao baixo desempenho na previsão de dados fora da amostra. Overfitting ocorre quando um modelo de previsão tem poucos graus de liberdade. Em outras palavras, tem relativamente poucas observações com relação a seus parâmetros e conseqüentemente pode memorizar pontos individuais de uma forma melhor ao invés de aprender padrões gerais. N caos de redes neurais, o número dos pesos, que é fortemente ligado ao número de camadas intermediárias e de neurônios, e o tamanho do conjunto de treinamento (número de observações) determinam a probabilidade de "overfitting". Quanto maior o número dos pesos relativo ao tamanho do conjunto do treinamento, maior a habilidade da rede de memorizar as observações individuais. Em conseqüência, a generalização para o conjunto de validação é perdida e os modelos são de pouco uso na previsão real.

Conseqüentemente, é recomendado que todas as redes neurais devam começar preferivelmente uma ou no máximo duas camadas intermediárias. Se uma rede neural de quatro camadas mostre resultados insatisfatórios após testas os neurônios intermediários múltiplos e usando um número razoável de pesos iniciais selecionados aleatoriamente, então o pesquisador deve modificar as variáveis de entrada algumas vezes antes de adicionar uma nova camada intermediária. Tanto teórica como

virtualmente, todo o trabalho empírico sugerem que as redes com mais de quatro camadas não melhorarão os resultados.

3.4.5.2 Número de Neurônios Intermediários.

Apesar de sua importância, não há nenhuma fórmula mágica para selecionar o número ótimo de neurônios na camada intermediária. Conseqüentemente, os pesquisadores utilizam mais da experimentação, entretanto, algumas receitas foram aprimoradas.

Para uma rede de três camadas com n neurônios de entrada e m neurônios de saida, a camada intermediária teria um número de neurônios da ordem do quadrado de (n X m). O número real de neurônios intermediários pode ainda variar entre um e meio e dois do valor geométrico da pirâmide dependendo da complexidade do problema.

Sugere-se que o número de neurônios intermediários em uma rede neural de três camadas deve ser 75% do número dos neurônios de entrada. É importante notar que as regras que calculam o número dos neurônios intermediários como um múltiplo do número de neurônios de entrada implicitamente supõe que o conjunto de treinamento é ao menos duas vezes maior que o número dos pesos e, preferivelmente, quatro ou mais vezes maior. Se este não for o caso, então as regras podem rapidamente conduzir ao "overfitted" desde que o número de neurônios intermediários é diretamente dependente do número de neurônios de entrada (que determinam por sua vez o número dos pesos). A solução é o aumento do tamanho do treinamento ajustado ou, se esta não for possível, ajustar um limite superior do número de neurônios de entrada de modo que o número dos pesos seja ao menos metade das observações do treinamento. A seleção de variáveis de entrada torna-se mais crítica em redes pequenas desde que o luxo de apresentar a rede com um grande número entradas e de ignorar os das irrelevantes.

Selecionar o melhor número de neurônios intermediários envolve experimentação. Três métodos usados frequentemente são o fixo, construtivo e destrutivo. Na

aproximação fixa, um grupo de redes neurais com números diferentes de neurônios intermediários é treinado e cada um é avaliado no conjunto testado usando um número razoável de pesos iniciais selecionados aleatoriamente. O incremento no número de neurônios intermediários pode ser um, dois, ou mais dependendo dos recursos computacionais disponíveis. Traçar o critério de avaliação (por exemplo soma de erros quadrados) no conjunto testado em função do número de neurônios intermediários para cada rede neural produz geralmente um gráfico com uma forma redonda do erro A rede neural com o menor erro é encontrada no ponto mais baixo do gráfico pois é a que possui a maior capacidade de generalização. Esta abordagem consome muito tempo de processamento, mas geralmente funciona muito bem.

A aproximação construtiva e destrutiva envolve mudar o número de neurônios intermediária, como na aproximação fixa. Muitos pacotes de software comerciais de rede neural não suportam a adição ou a remoção de neurônios intermediários durante o treinamento. A aproximação construtiva envolve adicionar os neurônios intermediários até que o desempenho da rede comece se deteriorar, a aproximação destrutiva é similar embora retire neurônios intermediários durante o treinamento.

Além do método utilizado para selecionar a escala de neurônios intermediários para serem testados, a regra é selecionar sempre a rede que executa melhor no conjunto testado com menos número de neurônios intermediários. Ao testar uma escala dos neurônios intermediários é importante manter todos os parâmetros restantes constantes. Mudar todo o parâmetro cria de fato uma rede neural nova com uma superfície potencial diferente do erro que pode complicar a seleção do melhor número de neurônios intermediários.

3.4.5.3 Número de neurônios de saída.

Decidir-se o número de neurônios de saída é um tanto quanto direto desde que haja razões para a não utilização apenas um neurônio de saída. As redes neurais com saídas múltiplas, especialmente se estas saídas são extensamente espaçadas, produzirão resultados inferiores em comparação a uma rede com uma única saída.

Uma rede neural treina escolhendo pesos tais que o erro médio sobre todos os neurônios de saída seja minimizado.

Por exemplo, uma rede neural que tenta prever um mês adiante e seis meses adiante do preço futuro do gado concentrará a maioria de seus esforços em reduzir a previsão com o maior erro que é provavelmente previsão de seis meses. Em conseqüência, uma melhoria relativamente grande em uma previsão mensal não será feita se aumentar o erro absoluto das seis previsões se aumentado o montante da previsão mensal absoluta. A solução deve mandar as redes neurais especializar-se usando as redes separadas para cada tipo de previsão. A especialização faz também a experimentação e o procedimento um tanto mais simples desde que cada rede neural seja menor e poucos parâmetros necessitem ser mudados no ajuste fino no modelo final.

3.4.5.4 Funções de Transferência.

As funções de transferência são as fórmulas matemáticas que determinam a saída de um neurônio de processamento. São também conhecidos como transformação, "squashing", ativação, ou função "threshold". A maioria dos modelos atuais de rede neural usa a função sigmóide (S-shaped), mas outras como tangente hiperbólica, por etapas, rampa, arco tangente e linear são também utilizadas. A finalidade da função de transferência é impedir que as saídas alcancem valores muito elevados que podem paralisar a rede neural e, desse modo, inibir o treinamento.

A função de transferência tal como o sigmóide é usada geralmente para dados da série de tempo porque são não-lineares e continuamente diferenciáveis que são propriedades desejáveis para a aprendizagem da rede.

Os dados não tratados são escalados geralmente entre 0 e 1 ou -1 e +1, assim que são consistentes com o tipo de função de transferência que está sendo usada. As escalas de desvio linear e da média padrão são dois dos métodos mais comuns usados em

redes neurais. Na escala linear, todas as observações são escaladas linearmente entre os valores mínimos e máximos de acordo com a seguinte fórmula:

$$SV = TF_{min} + (TF_{max} - TF_{min}) \times \frac{(D - D_{min})}{(D_{max} - D_{min})} \qquad (3.15)$$

onde *SV* é o valor escalado, *TFmin* e *TFmax* são o mínimo respectivo e os valores máximos de transferência funcionam, *D* é o valor da observação, e *Dmin* e *Dmax* são os valores respectivos de mínimo e máximo de todas as observações.

3.4.6 Critério de Avaliação.

A função para minimizar o erro mais comum em redes neurais é a soma dos erros quadráticos. A outra função de erro oferecida por fabricantes de software inclui o menor desvio absoluto, diferenças de porcentagem, entre outros. Estas funções de erro não podem ser os critérios finais de avaliação já que outros métodos comuns de avaliação de previsões tais como o erro percentual absoluto médio (MAPE) não são tipicamente minimizados em redes neurais.

No caso de sistemas de negociação de commodities, as redes neurais de previsões seriam convertidas em sinais de compra/venda de acordo com um critério predeterminado. Por exemplo, toda previsão maior que 0.8 ou 0.9 podem ser considerados sinais de compra e as previsões menores de 0.2 ou 0.1 como sinais de venda. Os sinais de compra/venda são então alimentados em um programa para calcular algum tipo de retorno ajustado do risco e as redes com melhor risco do retorno ajustado (não o menor erro de teste) seriam selecionadas. Os baixos erros de previsão e os lucros negociados não são necessariamente sinônimos desde que um única grande negociação incorreta por uma rede neural poderia ter explicado a maioria dos lucros do sistema negociado.

Filtrar a série de tempo para remover muitas das mudanças menores do preço pode, na maioria das vezes, impedir a situação onde uma rede neural com exatidão elevada

de previsão do ponto crítico remanesce não lucrativa. Também, o valor de todo único sistema negociado pode somente ser estabelecido dentro do contexto do portfolio do usuário de sistemas de commodity. Nesta consideração, as redes neurais podem ser especialmente úteis caso se comportem mais como sistemas contrários da tendência ao contrário dos seguintes sistemas de uma tendência mais comum usados por fundos de commodities.

3.4.7 Treinamento da Rede Neural.

Treinar uma rede neural para aprender o padrão nos dados envolve iterativamente apresentar os dados junto com respostas conhecidas. O objetivo do treinamento é encontrar o conjunto dos pesos entre os neurônios que determinam o mínimo global da função de erro. A menos que o modelo for "overfitted", este conjunto dos pesos pode fornecer uma boa generalização. O BPN usa um algoritmo de treinamento de descida do gradiente que ajusta os pesos de acordo com a inclinação mais íngreme da superfície do erro. Encontrar o mínimo global não é garantido já que a superfície de erro pode incluir muitos mínimos locais em que o algoritmo pode se tornar inútil. Um termo momentum e cinco a dez conjuntos randômicos de pesos iniciais podem melhorar as possibilidades de alcançar um mínimo global.

3.4.7.1 Número de Iterações na fase de Treinamento.

Há duas escolas de pensamento a respeito do ponto em que o treinamento deve ser parado. O primeiro força o perigo de começar preso em um mínimo local e a dificuldade de alcançar um mínimo global. O pesquisador deve somente parar o treinamento quando não há melhoria na função de erro tendo em vista um número razoável de pesos iniciais selecionados aleatoriamente. O ponto em que a rede não melhora é chamado convergência. A segunda escola advoga para uma série de interrupções na fase de treinamento. O treinamento pára depois que um número predeterminado de iterações e quando a habilidade da rede em generalizar o conjunto testado é avaliada e o treinamento recomeça. A generalização é a idéia que um

modelo baseado em uma amostra de dados é apropriado para prever a população geral.

Ambas as escolas do pensamento concordam que a generalização no conjunto de validação é o objetivo final e ambos os conjuntos testados são usados para avaliar um grande número redes. A aproximação da convergência indica que não há nada como um excessivo treinamento, apenas "overfitting". Overfitting é simplesmente um sintoma de uma rede que tenha muitos pesos. A solução deve reduzir o número dos neurônios intermediários (ou de camadas intermediárias se houver mais de uma) e/ou aumentar o tamanho do conjunto de treinamento. A abordagem da fase de treinamento tenta diminuir o overfitting parando o treinamento baseado na habilidade da rede de generalizar.

A vantagem da abordagem de convergência é que pode ser mais confiável quando o mínimo global for alcançado. Replicar é mais difícil para a abordagem da fase de treinamento já que os pesos iniciais são randômicos e a correlação média pode flutuar muito com o processo de treinamento. Uma outra vantagem é que o pesquisador tem dois parâmetros a menos para se preocupar; saber o ponto em que se pare o treinamento e o método para avaliar a rede é o esperado. Uma vantagem da aproximação da fase de treinamento pode ser que redes com poucos graus de liberdade podem ser executadas com uma maior generalização do que o treinamento de convergência que resultaria em overfitting. A aproximação da série de testes requer também menos tempo de treinamento.

O objetivo do treinamento de convergência é alcançar um mínimo global. Isto requer o treinamento para um número suficiente de iterações usando um número razoável de pesos iniciais aleatórios. Mesmo assim, não há nenhuma garantia com um BPN que o mínimo global foi alcançado já que se pode prender em um mínimo local. Na prática, os recursos computacionais são limitados e os tradeoffs são constantes. O pesquisador deve escolher o número de combinações variáveis da entrada a ser treinadas, a quantidade de neurônios intermediários que cada rede deve ter, o número dos pesos iniciais aleatórios e o número máximo de iterações. Por exemplo, 50

entradas são testadas sobre três neurônios intermediários diferentes com os cinco conjuntos de pesos iniciais aleatórios e um número máximo de funcionamentos de 4.000 resultados em 3.000.000 iterações. O mesmo tempo de processamento é requerido para 10 entradas testadas em seis neurônios intermediários com dez pesos iniciais e 5000 iterações.

Um método para determinar um valor razoável para o número máximo de funcionamentos deve traçar a correlação média, a soma de erros quadráticos, ou a outra medida apropriada do erro para cada iteração ou em intervalos predeterminados até o ponto onde a melhoria é insignificante (geralmente até um máximo de 10.000 iterações). Cada iteração pode facilmente ser traçada se o software de rede neural criar estatísticas e arquivá-las ou, se este não for o caso, se a correlação média puder ser gravada em intervalos de 100 ou de 200. Após ter traçado a correlação média para o um número de pesos iniciais, os pesquisadores podem escolher o número máximo de funcionamentos baseados no ponto onde a correlação média para de aumentar rapidamente.

É recomendado que os pesquisadores determinem o número das iterações requeridas para conseguir a melhoria insignificante para seu problema particular.

3.4.7.2 Momentum e Taxa de Aprendizagem.

Um BPN é treinado usando um algoritmo de descida do gradiente que siga o contorno da superfície do erro sempre abaixando a inclinação o mais íngreme possível. O objetivo do treinamento é minimizar os erros quadráticos totais, definidos como segue:

$$E = \frac{1}{2}\sum_{h}^{M} E_h = \frac{1}{2}\sum_{h}^{M}\sum_{i}^{N}(t_{hi} - O_{hi})^2 \qquad (3.16)$$

onde E é o erro total de todos os testes padrões. E_h representa o erro no teste padrão h, o índice h varia sobre o conjunto de testes padrões da entrada, e i refere-se ao neurônio i_{th} de saída. A variável t_{hi} é a saída desejada para o neurônio de saída i_{th} quando o padrão de h_{th} é apresentado, e, O_{hi} é a saída atual do neurônio de saída i_{th} quando o teste padrão h está apresentado. A regra de aprendizagem para ajustar o peso entre os neurônios i e j é definida como:

$$\delta_{hi} = (t_{hi} - O_{hi})O_{hi}(1 - O_{hi}) \tag{3.17}$$

$$\delta_{hi} = O_{hi}(1 - O_{hi})\sum_{k}\delta_{hk}w_{jk} \tag{3.18}$$

$$\Delta w_{ij}(n+1) = \varepsilon(\delta_{hi}O_{hj}) \tag{3.19}$$

onde n é o número da apresentação, δ_{hi} é o sinal do erro do neurônio i para o teste padrão h, e ε é a taxa que de aprendizagem. A taxa de aprendizagem é uma constante proporcional que determina o tamanho das mudanças dos pesos. A mudança dos pesos de um neurônio é proporcional ao impacto do peso desse neurônio no erro. O sinal do erro para um neurônio da saída e um neurônio intermediário é calculado por Eq.(3.17) e por (3.18), respectivamente.

Um método para aumentar a taxa de aprendizagem e, desse modo, a velocidade do tempo de treinamento e que discutimos anteriormente deve incluir um termo de momentum no na regra de aprendizagem. O termo momentum determina como as mudanças dos pesos antigos afetam mudanças atuais do peso. A regra modificada do treinamento de BPN é definida como segue:

$$\Delta w_{ij}(n+1) = \varepsilon(\delta_{hi}O_{hj}) + \alpha\Delta w_{ij}(n) \tag{3.20}$$

onde α é o momento, e os outros termos já foram previamente discutidos.

3.4.8 Implementação.

A etapa da execução é listada como sendo última, mas na verdade, requer alguns cuidados antes mesmo da coleta dos dados. A disponibilidade de dados, os critérios da avaliação, e os tempos de treinamento são dados que formam o ambiente da rede neural. A maioria dos fabricantes de software de rede neural fornece os meios que cada rede neural pode ser implementada no próprio programa ou mesmo em uma arquivo executável. Caso contrário, uma rede treinada pode facilmente ser criada em sabendo sua arquitetura, funções de transferência e pesos. Deve tomar cuidado pois transformações de dados e outros parâmetros podem variar entre o treinamento e o uso real.

Uma vantagem da redes neural é sua habilidade de adaptar-se às condições de um ambiente em mudança. Uma vez implementada, o desempenho da rede neural poderá cair ao longo do tempo ao menos que seja utilizado um re-treinamento. Entretanto, mesmo com treinamentos periódicos, não há nenhuma garantia que o desempenho da rede pode ser mantido como as variáveis independentes selecionadas podem se tornar mais ou menos importante.

Recomenda-se que a freqüência de re-treinamento para a rede implementada deve ser a mesma que usada durante teste no modelo final. Entretanto, quando se utiliza um grande número para testar uma rede em seu modelo final, menos re-treinamentos são suficientes a fim de manter um tempo de processamento razoável. Um modelo bom deve ser robusto com respeito à freqüência de re-treinamento e geralmente irá melhorar estes re-treinamentos com uma frequência maior.

4 Parte Experimental.

4.1 Modelo Econométrico.

Os dados utilizados foram as cotações de fechamento da ação preferencial da Petrobrás, código na BOVESPA PETR4, do período de 4 de janeiro de 1999 até 15 de junho de 2004 (fonte Economática). Como base para estimativa dos modelos utilizou-se os dados até 30 de abril de 2004, e como base para verificação do modelo os dados restantes (30 observações).

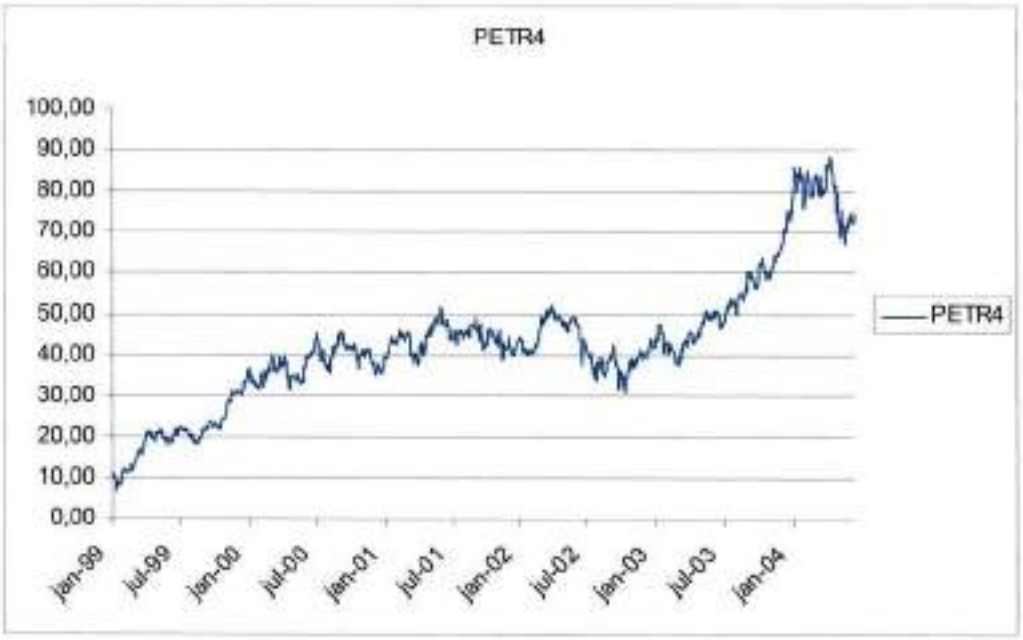


Figura 4.1 Série de preço de fechamento PETR4

A primeira etapa consiste em analisar a série temporal. As estatísticas obtidas são as seguintes:

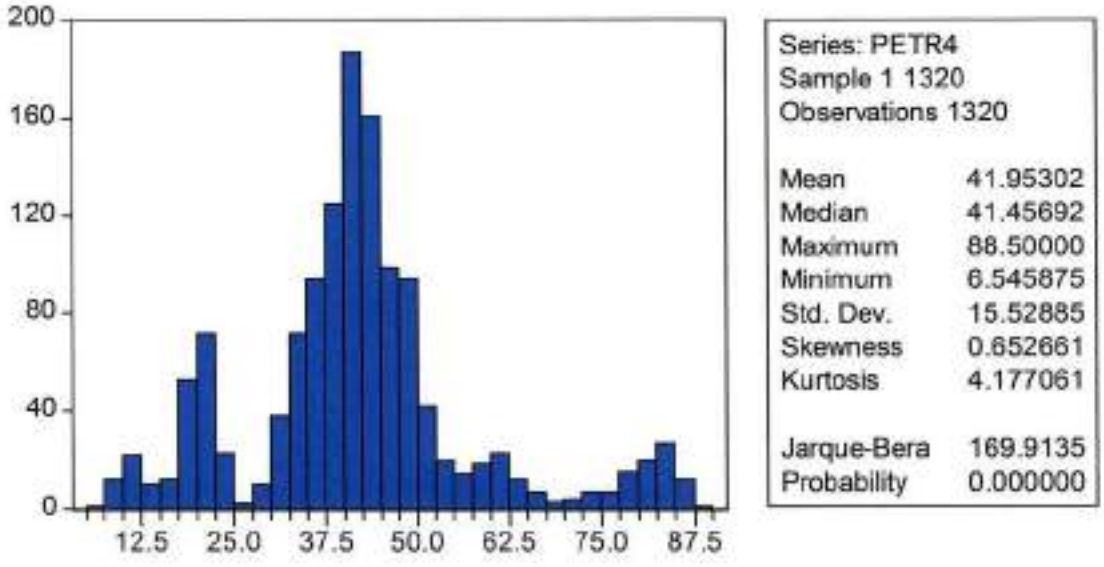


Figura 4.2 Estatísticas do preço de fechamento de PETR4

Observando o resultado do teste de Jarque-Bera a hipótese de normalidade é rejeitada. O teste de Jarque-Bera mede a diferença entre a curtose (achatamento) e a assimetria (skewness) da série com a de uma distribuição normal. A estatística é calculada da seguinte forma:

$$Jarque-Bera = \frac{N-k}{6}\left(S^2 + \frac{(K-3)^2}{4}\right)$$

onde S é a assimetria, K é a curtose, e k representa o número de coeficientes estimados utilizados para se criar a série. Um valor pequeno para a probabilidade implica na rejeição da hipótese nula de distribuição normal.

Como início da especificação do modelo, analisou-se a correlação entre a série PETR4 e a série PTAX, que é a série das taxas médias de câmbio BRL/USD, partindo-se do pressuposto que o resultado da Petrobrás estava diretamente ligado ao preço do dólar. Obtendo-se um valor de 0,502123.

Como a principal *commoditie* da Petrobrás é o petróleo, é natural que o retorno da ação esperado pelo mercado acompanhe os preços internacionais do petróleo, o que reflete no preço da ação. Portanto se analisou a correlação com o preço do primeiro futuro de petróleo bruto NYMEX (New York Mercantile Exchange), obtendo-se 0,730874.

A correlação é uma medida entre 0 e 1 que nos diz quão correlacionadas duas variáveis são. Quanto maior este valor, mais uma variável se comporta como a outra.

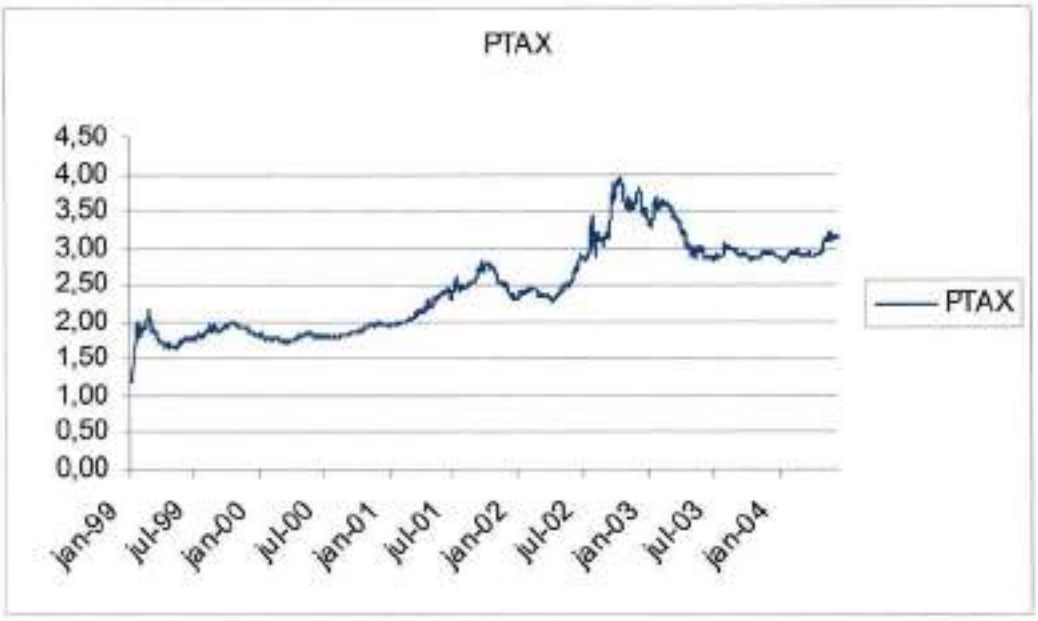


Figura 4.3 Taxa de câmbio PTAX (BACEN)

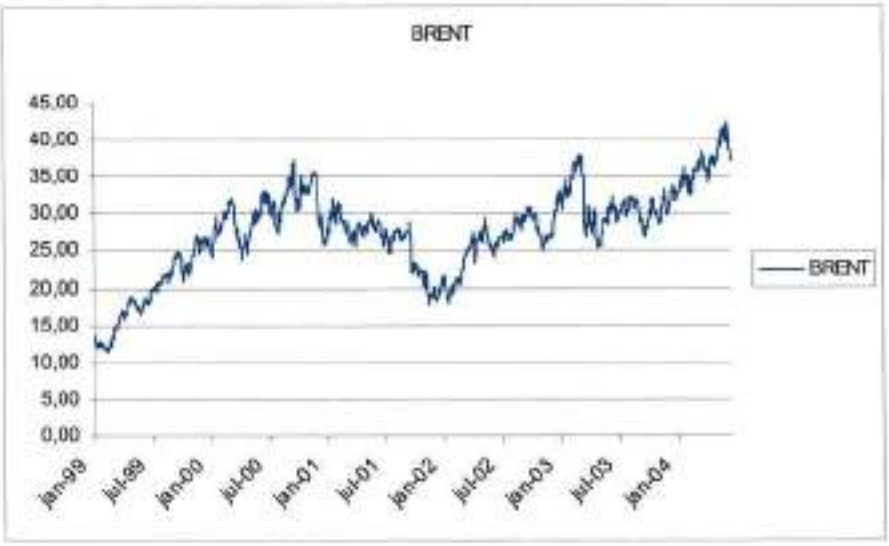


Figura 4.4 Primeiro futuro do Crude Oil NYMEX

O modelo inicial é uma regressão simples com a seguinte forma:

$$\log(PETR4) = \alpha_1 + \alpha_2 \log(PTAX) + \alpha_3 (BRENT)$$

Os resultados obtidos, utilizando o método dos mínimos quadrados (Ordinary Least Squares – OLS), foram:

Variável Dependente: LOG(PETR4)

Método: Mínimos Quadrados

Amostra: 1 1321

Observações incluídas: 1321

Variável	Coeficiente	Dv. Padrão	t-Statistic	Prob.
C	-0.601885	0.089146	-6.751684	0.0000
LOG(PTAX)	0.494983	0.028754	17.21415	0.0000
LOG(BRENT)	1.171361	0.029569	39.61414	0.0000
R-squared	0.697096	Mean dependent var		3.659928
Adjusted R-squared	0.696637	S.D. dependent var		0.418510
S.E. of regression	0.230509	Akaike info criterion		-0.094789
Sum squared resid	70.03089	Schwarz criterion		-0.083011
Log likelihood	65.60804	F-statistic		1516.610
Durbin-Watson stat	0.026106	Prob(F-statistic)		0.000000

Tabela 4.1 Resultado da regressão simples

Ou seja:

$$\log(PETR4) = -0.601885 + 0.494983.\log(PTAX) + 1.171361.\log(BRENT)$$

A coluna "Dv. Padrão" reporta os desvios padrões estimados para os coeficientes obtidos na regressão. O desvio padrão mede a significância estatística do coeficiente estimado – quanto maior o desvio padrão, maior é o ruído estatístico nas estimações. Se os desvios forem normalmente distribuídos, existem aproximadamente duas chances em três de que os verdadeiros coeficientes da regressão estejam dentro de um desvio padrão do coeficiente reportado, e 95% de chance que estejam dentro de dois desvios padrões.

A estatística-t (t-Statistic), que é computada como a razão entre o coeficiente estimado e seu desvio padrão, é utilizada para testar a hipótese de que um coeficiente é igual a zero (quantos desvios padrões de "distância" entre o coeficiente e zero).

A última coluna da tabela mostra a probabilidade do coeficiente a se estimar ser igual a zero, assumindo-se que os erros são normalmente distribuídos, ou de que os coeficientes estimados são assintoticamente normalmente distribuídos. Esta probabilidade também é conhecida como p-value ou nível marginal de significância. Significa que o resultado tem significado estatístico quando o p-value é menor do que o nível de significância adotado.

A estatística R^2 (R-squared) mede o sucesso da regressão em prever os valores da variável dependente baseando-se na amostra. Esta estatística é igual a 1 se a regressão se encaixa perfeitamente, e 0 caso a regressão não seja melhor método do que se utilizar a média simples da variável dependente como valor esperado.

Um problema em se utilizar R^2 como uma medida de qualidade da regressão, é que o R^2 nunca diminuirá ao se adicionar mais regressores. Como caso extremo, é possível obter um R^2 igual a 1, caso inclua-se tantos regressores quanto existam observações na amostra. O R^2 ajustado (Adjusted R-squared), normalmente

representado por $\overline{R}^2$, penaliza o R^2 pela adição de regressores que não contribuem ao poder de explicação do modelo. O $\overline{R}^2$ nunca é maior do que o R^2, pode diminuir ao se incluir novos regressores, e, para regressões que se encaixam fracamente , podem ser negativos.

A estatística DW (Durbin-Watson stat) mede a correlação serial dos resíduos, e é calculada da seguinte forma:

$$DW = \frac{\sum_{t=2}^{T}(\hat{\varepsilon}_t - \hat{\varepsilon}_{t-1})^2}{\sum_{t=1}^{T}\hat{\varepsilon}_t^2}$$

Como uma regra geral, caso DW seja menor do que 2, existe evidência de correlação serial.

A estatística-F (F-stat) é o resultado de um teste de que todos os coeficientes da regressão (excluindo-se a constante) são nulos. Novamente, deve-se analisar seu p-value.

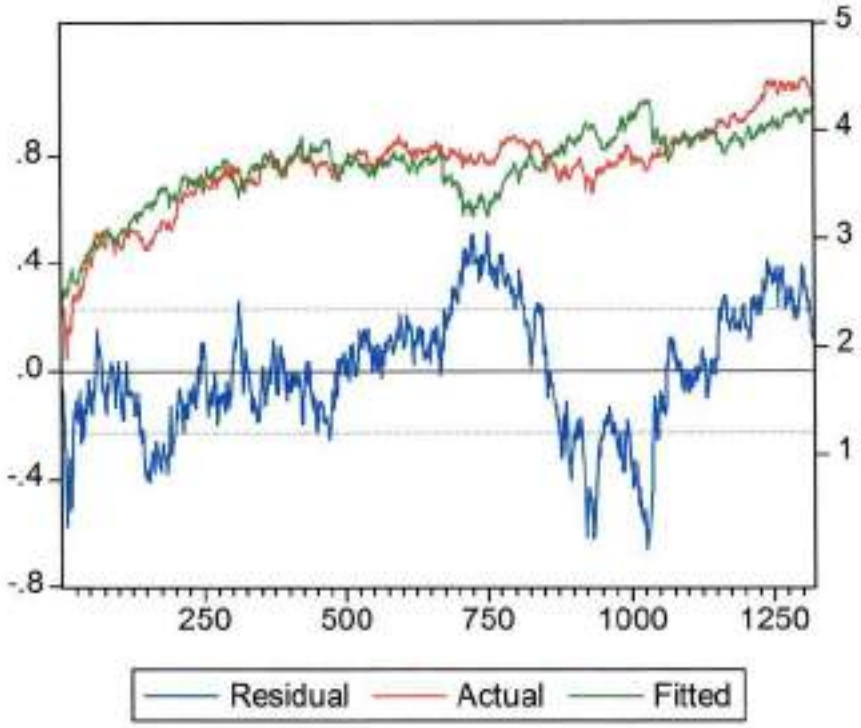


Figura 4.5 Resultado da regressão simples

Os resultados da regressão simples não são satisfatórios, tanto em termos do R^2 ou observando-se os resíduos.

Analisando-se os resíduos em busca de correlação serial pois DW é menor do que dois, tem-se:

Breusch-Godfrey Serial Correlation LM Test:

F-statistic	49077.14	Probability	0.000000
Obs*R-squared	1286.477	Probability	0.000000

Test Equation:
Variável Dependente: RESID
Método: Mínimos Quadrados

Presample missing value lagged residuals set to zero.

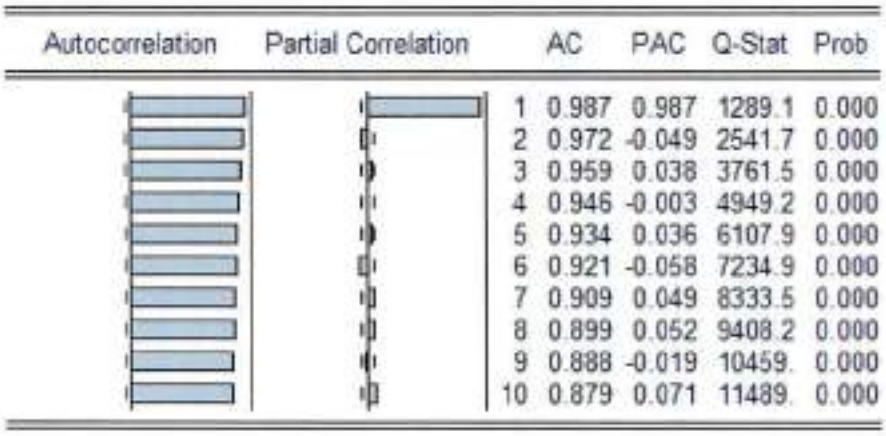
Variável	Coeficiente	Dv. Padrão	t-Statistic	Prob.
C	0.016956	0.014417	1.176143	0.2398

LOG(PTAX)	0.005839	0.004650	1.255679	0.2095
LOG(BRENT)	-0.006682	0.004782	-1.397337	0.1625
RESID(-1)	0.986974	0.004455	221.5336	0.0000

R-squared	0.973866	Mean dependent var	1.03E-15
Adjusted R-squared	0.973806	S.D. dependent var	0.230334
S.E. of regression	0.037278	Akaike info criterion	-3.737793
Sum squared resid	1.830186	Schwarz criterion	-3.722090
Log likelihood	2472.813	F-statistic	16359.05
Durbin-Watson stat	1.893565	Prob(F-statistic)	0.000000

Tabela 4.2 Análise residual.

O que rejeita a hipótese de não haver correlação serial para um atraso de 1 (Observar p-value para RESID(-1), que é o coeficiente dos resíduos de t-1). Construindo-se o correlograma (FAC e PAC):



Autocorrelation	Partial Correlation		AC	PAC	Q-Stat	Prob
		1	0.987	0.987	1289.1	0.000
		2	0.972	-0.049	2541.7	0.000
		3	0.959	0.038	3761.5	0.000
		4	0.946	-0.003	4949.2	0.000
		5	0.934	0.036	6107.9	0.000
		6	0.921	-0.058	7234.9	0.000
		7	0.909	0.049	8333.5	0.000
		8	0.899	0.052	9408.2	0.000
		9	0.888	-0.019	10459.	0.000
		10	0.879	0.071	11489.	0.000

Figura 4.6 PAC/FAC

Observa-se o decaimento da PAC após um atraso e o valor muito próximo a 1 para o primeiro atraso, indício de um modelo auto-regressivo de primeira ordem. O termo AR(1) é incluído na regressão, obtendo-se:

Variável Dependente: LOG(PETR4)

Método: Mínimos Quadrados

Amostra(ajustada): 2 1321

Observações incluídas: 1320 após ajuste

Convergence achieved after 7 iterations

Variável	Coeficiente	Dv. Padrão	t-Statistic	Prob.
C	3.993630	0.212353	18.80653	0.0000
LOG(PTAX)	-0.311052	0.057739	-5.387211	0.0000
LOG(BRENT)	0.085557	0.026205	3.033369	0.0025
AR(1)	0.995195	0.001517	656.2246	0.0000
R-squared	0.996466	Mean dependent var		3.660956
Adjusted R-squared	0.996458	S.D. dependent var		0.416997
S.E. of regression	0.024818	Akaike info criterion		-4.551477
Sum squared resid	0.810561	Schwarz criterion		-4.535764
Log likelihood	3007.975	F-statistic		123686.6
Durbin-Watson stat	1.825074	Prob(F-statistic)		0.000000
Inverted AR Roots	1.00			

Tabela 4.3 Resultado da regressão, modelo AR

Ou seja:

$$\log(PETR4) = 3.99363 - 0.311052.\log(PTAX) + 0.08557.\log(BRENT) + 0.995195.\log(PETR4_{t-1})$$

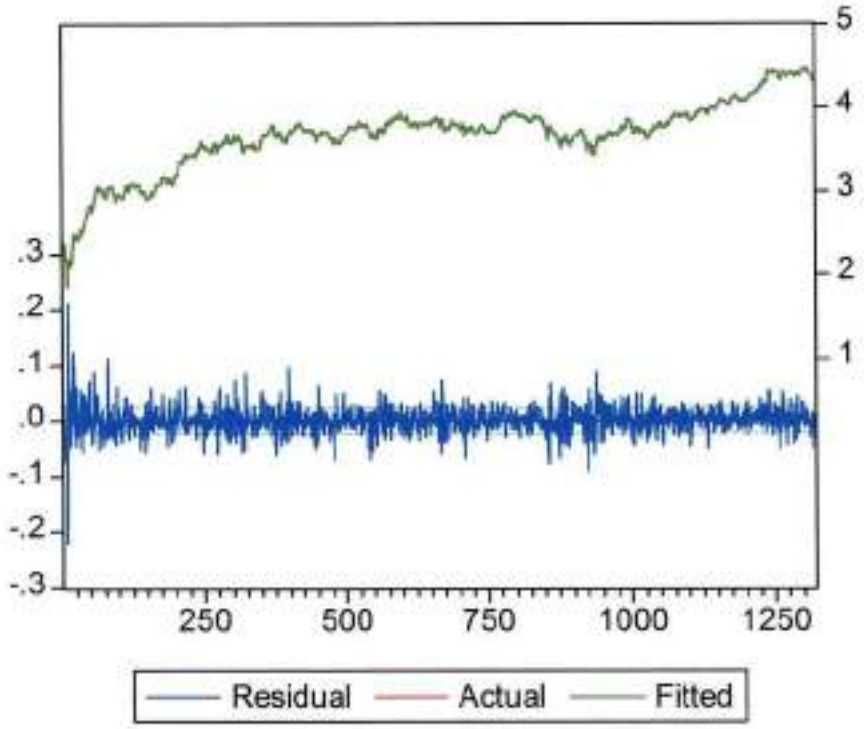


Figura 4.7 Regressão, modelo AR

Os resultados obtidos foram muito melhores do que os da regressão simples. Para validar o modelo, uma previsão foi feita com os dados de teste (30 observações à frente).

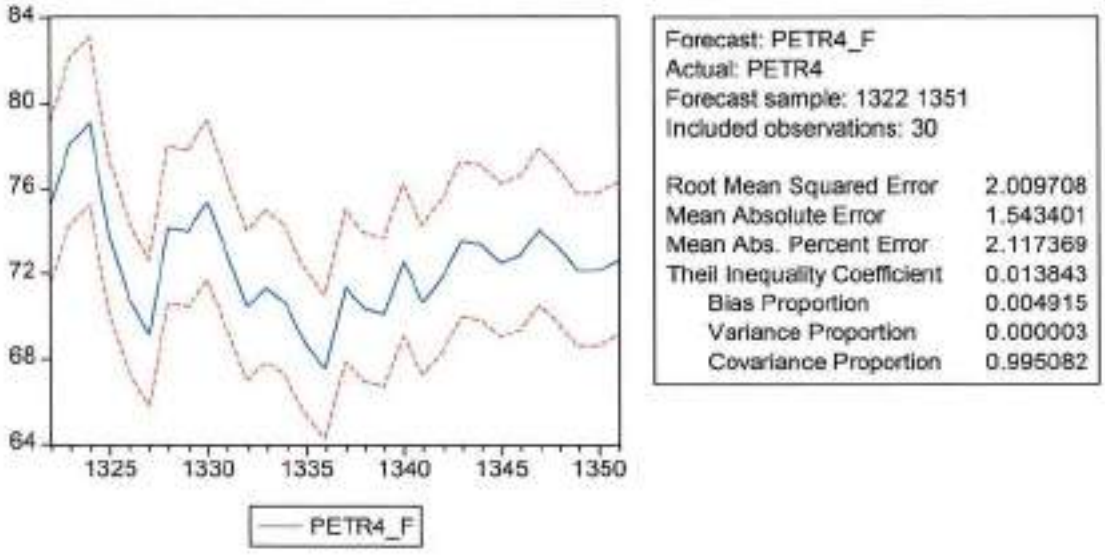


Figura 4.8 Resultado da previsão, modelo AR

As medidas de qualidade do modelo são quatro. O RMSE (Root Mean Squared Error) e o MAE (Mean Absolute Error) dependem da escala da variável dependente. E são utilizados para comparar-se previsões entre modelos diferentes aplicados à mesma série; quanto menor o erro, melhor a habilidade de previsão do modelo de acordo com este critério. Os outros dois, MAPE (Mean Absolute Percentage Error) e TIC (Teil Inequality Coeficient), são independentes à escala utilizada. O TIC sempre está entre 0 e 1, sendo 0 uma previsão perfeita. O RMSE pode ser dividido em três partes:

$$\sum(\hat{y}_t - y_t)^2 / h = \left(\left(\sum \hat{y}_t / h\right) - \bar{y}\right)^2 + \left(s_{\hat{y}} - s_y\right)^2 + 2(1-r)s_{\hat{y}}s_y$$

onde $s_{\hat{y}}, s_y, r$ são os desvios padrão e a correlação de $\hat{y}$ e y.

- Bias Proportion, que mede quão distante a média da previsão está da média da série original. $\dfrac{\left(\left(\sum \hat{y}/h\right) - \bar{y}\right)^2}{\sum(\hat{y}_t - y_t)^2 / h}$
- Variance proportion, que mede quão distante a variação da previsão está da variação da série original. $\dfrac{\left(s_{\hat{y}} - s_y\right)^2}{\sum(\hat{y}_t - y_t)^2 / h}$
- Covariance proportion, mede os erros não sistemáticos remanescentes. $\dfrac{2(1-r)s_{\hat{y}}s_y}{\sum(\hat{y}_t - y_t)^2 / h}$

Notar que todos podem chegar a 1.

Se a previsão for "boa", o bias e variance proportions devem ser bem pequenos, sendo que a maior parte do erro está no covariance proportion.

No caso do modelo AR construído, todos os erros obtidos são satisfatórios.

O gráfico da série dos dados de teste mais o intervalo de confiança de 95% (2σ) do modelo obtido é o seguinte:

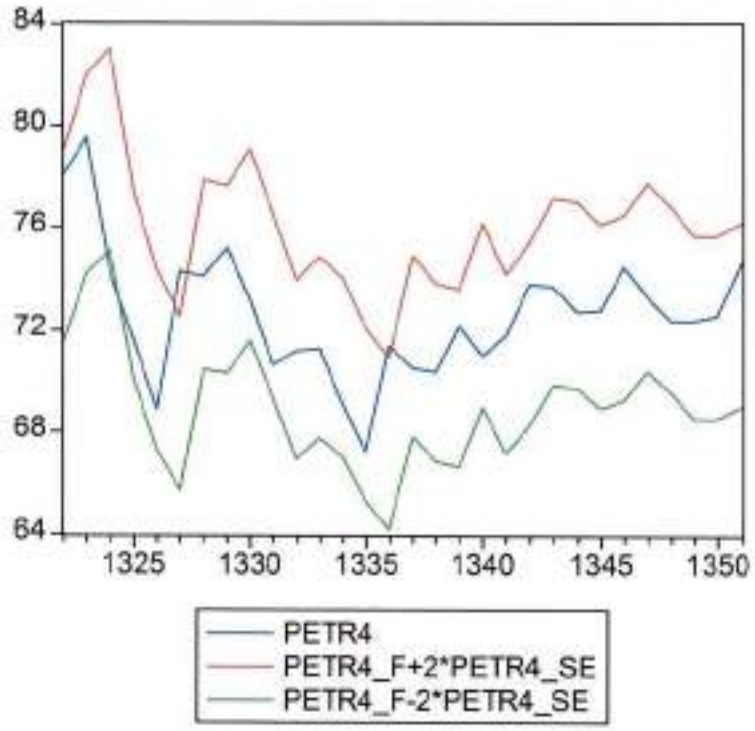


Figura 4.9 Intervalo de confiança, modelo AR.

Como se vê, a série fica fora do intervalo de confiança em alguns pontos, mas isso considerando-se que a série segue uma distribuição normal. De qualquer forma, como a série apresenta pontos de volatilidade concentrada (clusters), um modelo AR(1) GARCH(1,1) é adequado.

Utilizando um componente GARCH(1,1) no modelo temos:

Variável Dependente: LOG(PETR4)
Método: ML - ARCH (Marquardt)

Amostra(ajustada): 2 1321
Observações incluídas: 1320 após ajustes
Convergence achieved after 28 iterations
Variance backcast: ON

	Coeficiente	Dv. Padrão	z-Statistic	Prob.
C	4.314705	0.276688	15.59409	0.0000
LOG(PTAX)	-0.420780	0.056292	-7.474902	0.0000
LOG(BRENT)	0.078866	0.024294	3.246336	0.0012
AR(1)	0.996202	0.001468	678.4004	0.0000
Variance Equation				
C	1.69E-05	3.94E-06	4.292477	0.0000
ARCH(1)	0.049174	0.009189	5.351211	0.0000
GARCH(1)	0.915778	0.012865	71.18512	0.0000
R-squared	0.996454	Mean dependent var		3.660956
Adjusted R-squared	0.996438	S.D. dependent var		0.416997
S.E. of regression	0.024887	Akaike info criterion		-4.739091
Sum squared resid	0.813234	Schwarz criterion		-4.711592
Log likelihood	3134.800	F-statistic		61498.79
Durbin-Watson stat	1.845022	Prob(F-statistic)		0.000000
Inverted AR Roots	1.00			

Tabela 4.4 Resultado da regressão, modelo GARCH

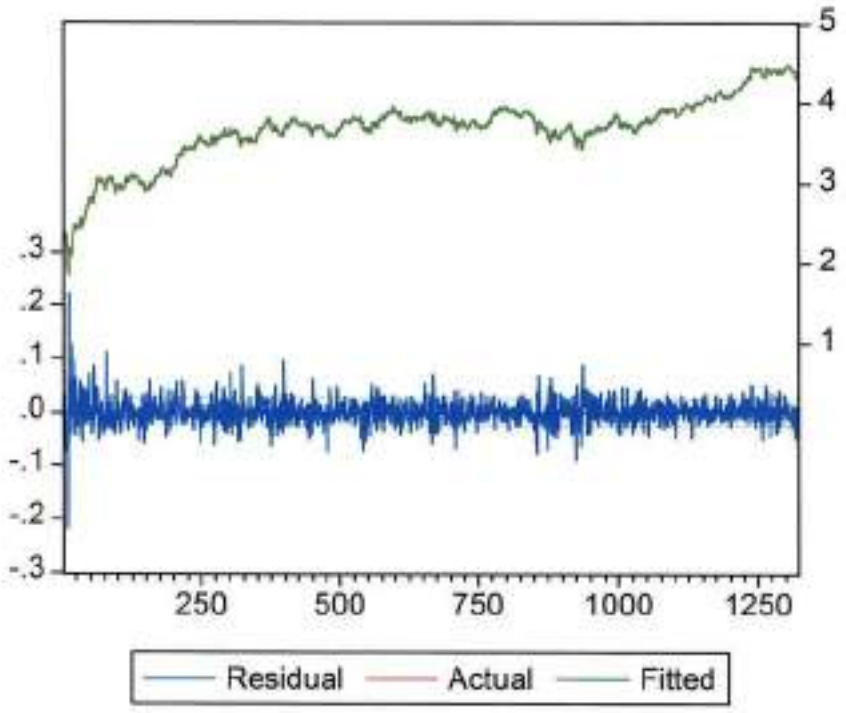


Figura 4.10 Resultado , modelo GARCH

E a previsão:

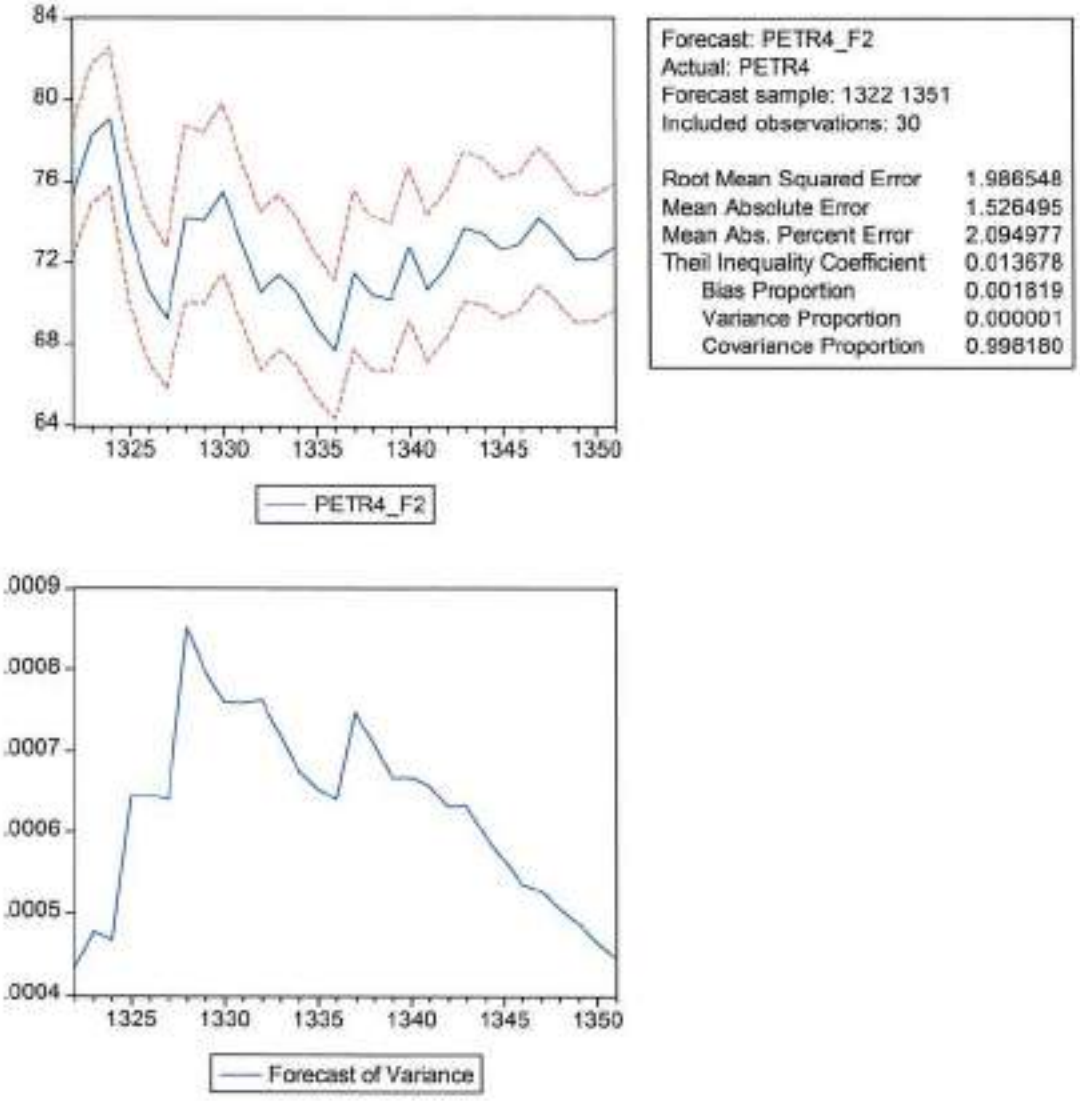


Figura 4.11 Resultados da previsão, modelo GARCH

Os resultados do modelo GARCH são melhores do que o modelo AR simples, sendo todas as medidas de erro menores e a única componente representativa do RMSE a covariance proportion.

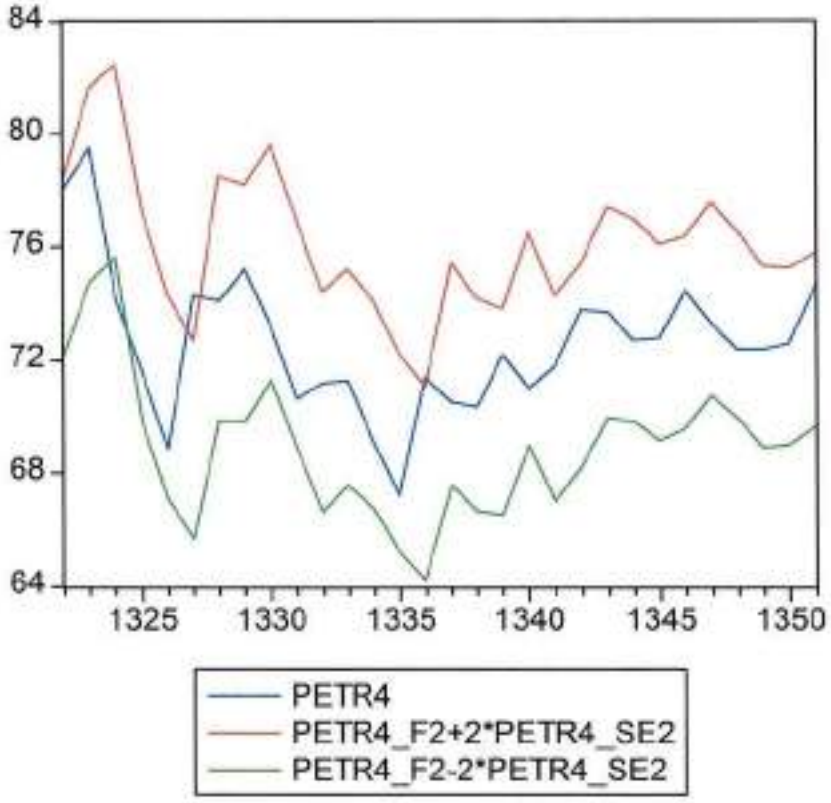


Figura 4.12 Intervalo de confiança, modelo GARCH

Alguns pontos ainda ficam de fora do intervalo de confiança, porém é visível a melhor resposta do modelo a choques de volatilidade.

5 PREVISÃO COM REDES NEURAIS.

Outro grande objetivo deste trabalho é efetuar testes no Matlab, abordando as características das redes neurais estudadas tanto na teoria quanto na prática. Para isso fizemos um estudo detalhado do seguinte modo:

1. Estudo da teoria de Redes Neurais utilizando a bibliografia referenciada.
2. Estudo da ferramenta Matlab e de todas as funcionalidades encontradas na biblioteca de Redes Neurais (neural network toolbox).
3. Definição de cenários de testes com os modelos estudados.

5.1 Matlab.

Utilizado em larga escala em muitos estudos qualitativos sobre os mais diversos assuntos e pesquisas, o software Matlab é uma poderosa ferramenta de projeto e simulação de sistemas, circuitos e modelos matemáticos. Como o próprio nome diz, é considerado um verdadeiro laboratório matemático. Proporciona ainda um ambiente amigável de programação de rotinas executáveis, interfaces gráficas e inúmeras bibliotecas das mais diversas aplicações.

Devido às suas inúmeras qualidades e também ao prévio contato com a ferramenta, o Matlab foi escolhido para atuar como suporte ao estudo prático das redes neurais.

A seguir apresentamos um roteiro detalhado de como as redes neurais se comportam no software: como os algoritmos funcionam; como são implementados; e como os dados são tratados. Em seguida apresentamos os cenários elaborados de teste e também os resultados do treinamento e simulação desses cenários.

5.2 Redes Neurais no Matlab

Há geralmente quatro etapas no processo do treinamento de redes neurais feedforward no Matlab:

1. Arranjo dos dados
2. Desenhar a rede neural
3. Treinar a rede
4. Simular a resposta da rede às entradas novas

5.2.1 Arranjo de Dados.

As redes multicamadas freqüentemente utilizam a função de transferência logsig (log-sigmoid). A função logsig gera saídas entre zero e um enquanto a entrada do neurônio vai de negativo ao infinito positivo. Alternativamente, as redes multicamadas podem utilizar a função de transferência tansig (tan-sigmoid). Ocasionalmente, a função de transferência purelin (linear) é usada em redes backpropagation. Se a última camada de uma rede multicamadas tiver os neurônios sigmoid, as saídas da rede estão então limitadas a uma escala pequena. Se os neurônios de saída forem lineares então as saídas da rede podem assumir qualquer valor. Ao utilizar backpropagation é importante saber calcular as derivadas de todas as funções de transferência usadas. Cada função de transferência acima (tansig, logsig, e o purelin) tem uma função derivada correspondente: dtansig, dlogsig, e dpurelin.

As três funções de transferência descritas aqui são geralmente as funções de transferência mais usadas para o backpropagation, mas outras funções diferenciais de transferência podem ser criadas e usadas com backpropagation.

As redes feedforward têm freqüentemente uma ou mais camadas escondidas de neurônios sigmoid seguido por uma camada da saída de neurônios lineares. As

camadas múltiplas de neurônios com funções não-lineares de transferência permitem que a rede aprenda relacionamentos não-lineares e lineares entre a entrada e os vetores da saída. A camada linear de saída possibilita que a rede produza saídas fora da escala -1 a +1. Na outra mão, se desejado confinar as saídas de uma rede (como entre zero e 1), então a camada da saída deve usar uma função sigmoid de transferência (tal como o logsig). Pode-se aproximar toda função com um número finito de descontinuidades arbitrariamente bem, desde de que haja neurônios suficientes nas camadas intermediárias.

5.2.2 Criando uma rede.

A primeira etapa em treinar uma rede feedforward é criar o objeto rede. A rotina newff cria uma rede feedforward a partir de quatro entradas, retornando o objeto rede. A primeira entrada é uma matriz Rx2 de valores mínimos e máximos para cada um dos elementos de R do vetor da entrada. A segunda entrada é uma disposição que contem os tamanhos de cada camada. A terceira entrada é uma lista que contem os nomes das funções de transferência a serem usadas em cada camada. A entrada final contém o nome da função de treinamento a ser utilizada.

Para o exemplo abaixo, o seguinte comando cria uma rede de três camadas. Há um vetor input com três elementos: o valor para o primeiro elemento (PETR4), o valor para o segundo elemento (PTAX) e o valor para o terceiro elemento (BRENT). Há três neurônios na primeira camada, quarenta neurônios na camada intermediária e um neurônio na terceira camada (de saída). A função de transferência na primeira e segunda camada é tan-sigmoid, e a função de transferência da camada de saída é linear. A função de treinamento é trainlm (que será descrita adiante).

```
net = newff(minmax(ptr),[3 40 1],{'tansig'
'tansig' 'purelin'},'trainlm');
```

Este comando cria o objeto rede e inicializa também os pesos e polarizações da rede; conseqüentemente a rede está pronta para o treinamento.

5.2.3 Inicializando pesos (init).

Antes de treinar uma rede feedforward, os pesos e as polarizações devem ser inicializados. O comando newff inicializa automaticamente os pesos. Para reinicializá-los pode ser utilizado o comando init, que tem como a entrada um objeto de rede e retorna outro objeto de rede com todos os pesos e as polarizações inicializadas.

```
net = init(net);
```

5.2.4 Simulação (sim).

A função sim simula a rede. Toma como parâmetros de entrada um conjunto de dados p e o objeto de rede net, retornando as saídas a.

Abaixo, a função sim é chamada para calcular as saídas para um conjunto simultâneo de três vetores input. Este é o modo de simulação em batch, em que todos os vetores da entrada estão em uma matriz. Isto é muito mais eficiente do que apresentar os vetores um de cada vez.

```
p = [1 3 2;2 4 1];
a=sim(net,p)
a =
   -0.1011    -0.2308     0.4955
```

5.2.5 Treinamento.

O processo de treinamento requer um conjunto de exemplos de comportamento apropriado à rede - entradas p e saídas-alvo t. Durante o treinamento os pesos e polarizações da rede são ajustadas iterativamente para minimizar a função

net.performFcn, que representa o desempenho da rede. A função de desempenho padrão para redes feedforward é o erro quadrático médio mse (Mean Square Error) entre as saídas da rede a e as saídas-alvo t.

Todos estes algoritmos usam o gradiente da função de desempenho para determinar o ajuste nos pesos, a fim de minimizar o desempenho. O gradiente é determinado usando uma técnica chamada backpropagation, que envolve computações para trás da rede. O algoritmo básico de treinamento do backpropagation, em que os pesos são movidos no sentido do gradiente negativo, é descrito adiante, além da descrição de algoritmos mais complexos que aumentem a velocidade da convergência. Há duas maneiras diferentes em que este algoritmo da descida do gradiente pode ser executado: modalidade incremental e modalidade batch. Na modalidade incremental, o gradiente é computado e os pesos são atualizados depois que cada entrada é aplicada à rede. Na modalidade batch todas as entradas são aplicadas na rede antes que os pesos sejam atualizados.
Treinamento Batch (train). Na modalidade batch os pesos e as polarizações da rede são atualizados somente depois que todos os dados de treinamento foram aplicados à rede. Os gradientes calculados em cada exemplo de treinamento são adicionados juntos para determinar a mudança nos pesos e nas polarizações. Os dois algoritmos de treinamento de backpropagation, descida do gradiente e descida do gradiente com momentum, são muito lentos para problemas práticos.

Assim apresentamos diversos algoritmos de desempenho elevado que podem convergir dez a cem vezes mais rapidamente do que os algoritmos discutidos previamente. Todos os algoritmos nesta seção se operam na modalidade batch e são invocados usando train. Estes algoritmos mais rápidos caem em duas categorias principais. A primeira categoria usa as técnicas heurísticas, que foram desenvolvidas a partir de uma análise do desempenho do algoritmo standard steepest descent (descida mais íngreme). A segunda categoria utiliza técnicas numéricas de otimização. Apresentamos um tipo de técnica numérica de otimização para o treinamento da rede neural, utilizados nos testes: o Levenberg-Marquardt.

5.2.6 Levenberg-Marquardt (trainlm).

Do mesmo modo que os métodos quasi-Newton, o algoritmo de Levenberg-Marquardt foi projetado pára se aproximar à velocidade de segunda-ordem de treinamento sem ter que computar a matriz de Hessian. Quando a função de desempenho tem a forma de uma soma dos quadrados (como é típico no treinamento em redes feedforward), então a matriz de Hessian pode ser aproximada como:

$$\mathbf{H} = \mathbf{J}^T\mathbf{J}$$

e o gradiente pode ser computado como:

$$\mathbf{g} = \mathbf{J}^T\mathbf{e}$$

onde J é a matriz Jacobiana que contém as primeiras derivadas dos erros da rede com respeito aos pesos e às polarizações, e **e** é um vetor de erros da rede. A matriz Jacobiana pode ser computada através da técnica padrão de backpropagation que é muito menos complexa do que computar a matriz de Hessian. O algoritmo de Levenberg-Marquardt usa uma aproximação à matriz de Hessian no seguinte modelo de Newton:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - [\mathbf{J}^T\mathbf{J} + \mu\mathbf{I}]^{-1}\mathbf{J}^T\mathbf{e}$$

Quando o mu escalar é zero, obtém-se apenas o método de Newton, usando a matriz aproximada de Hessian. Quando o mu é grande este se transformar em "descida do gradiente" com um pequeno passo. O método do Newton é mais rápido e mais exato próximo a um valor mínimo de erro. Logo o melhor é deslocar o mais rápido possível para o método do Newton. Assim, o mu diminui após cada etapa bem

sucedida (redução na função do desempenho) e aumenta somente quando uma etapa de tentativa aumentaria a função de desempenho. Nesta maneira, a função de desempenho será reduzida sempre em cada iteração do algoritmo.

Os parâmetros de treinamento para o trainlm são epochs, show, goal, time, min_grad, max_fail, mu, mu_dec, mu_inc, mu_max, mem_reduc.

O parâmetro mu é o valor inicial para um. Ele é multiplicado pelo mu_dec sempre que a função do desempenho é reduzida por uma etapa. É multiplicado pelo mu_inc sempre que uma etapa aumentaria a função do desempenho. Se o mu se tornar maior do que o mu_max, o algoritmo é interrompido. O parâmetro mem_reduc é usado para controlar a quantidade de memória utilizada pelo algoritmo.

Este algoritmo parece ser o método mais rápido para treinar redes neurais feedforward de tamanho médio (até algumas centenas de pesos). Apresenta também uma execução muito eficiente no MATLAB, desde que a solução da equação da matriz seja uma função interna, assim seus atributos tornam-se melhor apresentáveis no MATLAB.

5.2.7 Melhorando a generalização.

Um dos problemas que ocorre durante o treinamento de uma rede neural é o chamado overfitting. O erro no conjunto de dados de treinamento é levado a um valor muito pequeno, mas quando os dados novos são apresentados à rede o erro é grande. A rede memorizou os exemplos de treinamento, mas não aprendeu a generalizar às situações novas.

Um método para melhorar a generalização deve utilizar uma rede grande o suficiente para fornecer um ajuste adequado. Quanto maior uma rede que você usa mais complexas são as funções que a rede pode criar. Se nós usarmos uma rede grande o suficiente, não terá bastante poder para provocar overfitting dos dados. Infelizmente, é difícil saber de antemão como deve ser uma rede para uma aplicação específica.

Há outros dois métodos para melhorar a generalização, que são executados no toolbox da rede neural: regularização e early stopping. A seguir descrevemos estas duas técnicas e as rotinas para executá-las. Note que se o número dos parâmetros na rede for muito menor do que o número total dos pontos no conjunto de treinamento, então há quase nenhuma possibilidade de overfitting. Ao coletar mais dados e aumentar o tamanho do conjunto de dados de treinamento diminui-se a necessidade de preocupar-se com o overfitting.

5.2.7.1 Regularização.

O primeiro método para melhorar a generalização é chamado regularização. Isto envolve modificar a função do desempenho, que normalmente é escolhida para ser a soma dos quadrados dos erros da rede no treinamento. A seguir é explicado como a função de desempenho pode ser modificada, e em seguida é descrita uma rotina que ajusta automaticamente a função ótima de desempenho para conseguir a melhor generalização.

5.2.7.1.1 Função modificada de desempenho.

A função típica de desempenho que é usada para treinamento de redes neurais feedforward é a soma média dos quadrados dos erros da rede.

$$F = mse = \frac{1}{N}\sum_{i=1}^{N}(e_i)^2 = \frac{1}{N}\sum_{i=1}^{N}(t_i - a_i)^2$$

É possível melhorar a generalização se nós modificarmos a função do desempenho adicionando um termo que consista na média da soma dos quadrados dos pesos e das polarizações da rede

$$mscreg = \gamma msc + (1 - \gamma) msw$$

Onde gama é a performance ratio e

$$msw = \frac{1}{n} \sum_{j=1}^{n} w_j^2$$

Utilizar esta função do desempenho fará com que a rede tenha pesos e polarizações menores, e este forçará a resposta da rede para ser mais leve e menos suscetível ao overfitting.

Um problema na regularização consiste na dificuldade em determinar o melhor valor para o parâmetro performance ratio. Se nós denotarmos este parâmetro demasiado grande, nós podemos desencadear um overfitting. Se a relação for demasiado pequena, a rede não absorverá adequadamente os dados de treinamento.

5.2.7.1.2 Regularização automática (trainbr).

É desejável determinar os parâmetros ótimos da regularização de forma automatizada. Uma aproximação a este processo é a estrutura Bayesiana de David MacKay [MacK92]. Nesta estrutura, os pesos e as polarizações da rede são supostamente variáveis aleatórias com distribuições especificas. Os parâmetros de regularização são relacionados às variações desconhecidas associadas com estas distribuições. Nós podemos então estimar estes parâmetros usando técnicas estatísticas. A regularização Bayesiana foi implementada na função trainbr.

Uma característica deste algoritmo é que fornece uma medida de quantos parâmetros de rede (pesos e polarizações) estão sendo utilizados eficazmente pela rede. Este

número eficaz dos parâmetros deve remanescer aproximadamente o mesmo, não importando quão grande se torne o número total dos parâmetros na rede (supondo que a rede esteve treinada para um número suficiente de iterações para assegurar a convergência.).

A seguinte figura mostra a resposta de uma rede de exemplo treinada. Aqui se vê que a resposta da rede é muito perto da função subjacente do seno (linha pontilhada), e, conseqüentemente, a rede generalizará bem às entradas novas. Ao usar o trainbr, deve-se deixar o algoritmo funcionar até que o número eficaz dos parâmetros convirja. Analisando se o erro quadrático da soma (SSE) e os pesos quadrados da soma (SSW) são relativamente constantes sobre diversas iterações pode dizer também que o algoritmo convergiu ou não. No caso positivo pode-se interromper o treinamento.

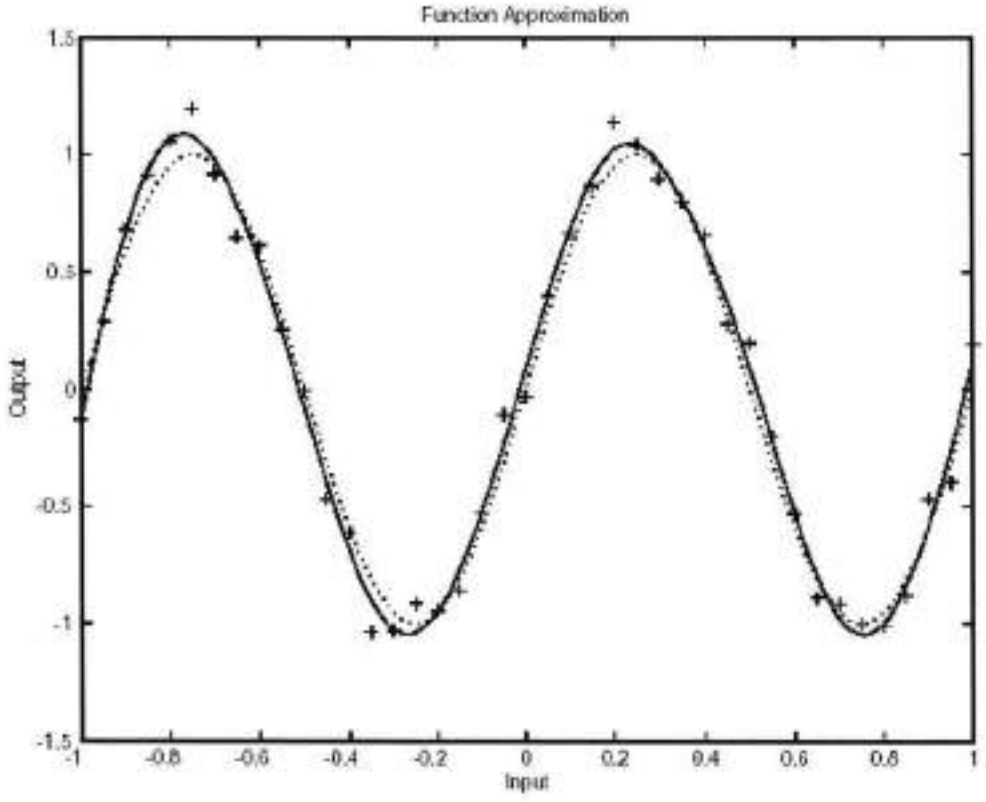


Figura 5.1 Resposta da rede à senóide.

5.2.7.2 Early Stopping.

Um outro método para melhorar a generalização é chamado early stopping. Nesta técnica os dados disponíveis são divididos em três subconjuntos. O primeiro subconjunto é o composto pelos dados de treinamento, que são usados para computar o gradiente e atualizar os pesos e as polarizações da rede. O segundo subconjunto são os dados de validação. O erro nesse conjunto de dados é monitorado durante o processo de treinamento. O erro de validação diminuirá normalmente durante a fase inicial do treinamento, assim como o erro de treinamento. Entretanto, quando a rede começa a provocar overfitting dos dados, o erro de validação começará tipicamente a crescer. Quando o erro de validação aumenta durante um número especificado de iterações, o treinamento é interrompido, e os pesos e as polarizações correspondentes ao erro mínimo de validação são retornados.

O erro de teste não é usado durante o treinamento, mas é útil para comparar modelos diferentes. É também útil traçar o erro de teste durante o treinamento. Se o erro no conjunto de teste alcançar um mínimo em um número significativamente diferente de iterações de validação, isto pode indicar uma má divisão dos grupos de dados. O Early Stopping pode ser usado com qualquer uma das funções de treinamento disponibilizadas no MATLAB.Basta simplesmente passar os dados de validação à função de treinamento.

Tanto a regularização como o early stopping podem assegurar a generalização da rede quando aplicados corretamente. Ao usar a regularização Bayesiana, é importante treinar a rede até que se alcance a convergência. O erro quadrático da soma, os pesos quadráticos da soma e o número eficaz de parâmetros deve sempre alcançar valores constantes quando a rede convergir. Para o caso do early stopping, deve-se ter cuidado para não usar um algoritmo que convirja muito rápido. No caso da utilização de um algoritmo rápido (como o trainlm), devem-se ajustar os parâmetros de treinamento de modo que a convergência seja relativamente lenta (por exemplo, ajustando o mu a um valor relativamente grande, tal como um, e o mu_dec e o mu_inc do jogo aos valores perto de um, tal como 0,8 e 1,5 , respectivamente).

Com early stopping, a escolha do conjunto de validação é também importante. Os dados de validação devem ser representativos frente a todos os pontos de treinamento.

A regularização Bayesiana geralmente fornece desempenho melhor de generalização do que o early stopping ao treinar redes de aproximação de função. Isto porque a regularização Bayesiana não requer que uma série de dados de validação esteja separada da série de dados de treinamento. Esta vantagem é especialmente visível quando o tamanho da série de dados é pequeno.

5.2.8 Análise Pós-Treinamento (postreg).

O desempenho de uma rede treinada pode ser medida em determinada extensão pelos erros nos dados de treinamento, validação e teste, mas é freqüentemente útil investigar mais detalhadamente a resposta da rede. Uma opção consiste na análise da regressão entre a resposta da rede e os alvos correspondentes. A rotina postreg é utilizada para tal finalidade. Inicialmente são passados a saída da rede e os alvos correspondentes ao postreg. Três parâmetros são retornados. Os primeiros dois, m e b, correspondem à inclinação e o deslocamento no eixo y da melhor regressão linear que relaciona os alvos às saídas da rede. No caso de um ajuste perfeito (saídas exatamente iguais aos alvos), a inclinação seria um, e o deslocamento seria zero. A terceira variável retornada pelo postreg é o coeficiente de correlação (R-value) entre as saídas e os alvos. É uma medida de quão bem a variação na saída é explicada pelos alvos. Se este número for igual a um, então há uma correlação perfeita entre alvos e saídas.

As redes multicamadas são capazes de executar qualquer computação linear ou não-linear, e podem aproximar qualquer função arbitrariamente bem. Entretanto, ao mesmo tempo em que a rede que está sendo treinada possa ser teoricamente capaz de funcionar corretamente, o backpropagation e suas variações podem nem sempre encontrar uma boa solução.

5.2.9 Limitações.

Escolher a taxa da aprendizagem para uma rede não-linear é um desafio. Assim como em redes lineares, uma taxa da aprendizagem que seja demasiada grande conduz a uma aprendizagem instável. Inversamente, uma taxa da aprendizagem que seja demasiada pequena pode acarretar em tempos de treinamento incrivelmente longos. Ao contrário das redes lineares, não há uma maneira fácil de escolher uma taxa boa de aprendizagem para redes multicamadas não-lineares. Com os algoritmos mais rápidos de treinamento, os valores de parâmetro padrão normalmente se enquadram adequadamente. As redes também são sensíveis ao número dos neurônios em suas camadas intermediárias. Poucos neurônios podem conduzir a underfitting. Muitos neurônios podem contribuir a overfitting, em que todos os pontos do treinamento têm um bom ajuste, mas a curva entre esses pontos apresenta grandes oscilações.

5.3 Testes.

5.3.1 Cenários.

A melhor maneira de se estudar uma rede neural é analisar a performance de convergência e as saídas de teste, variando sua arquitetura (numero de camadas, numero de neurônios) e seus parâmetros de treinamento (número de épocas, learning ratio, dentre outros). Para isso foram definidos diversos cenários de treinamento, buscando relatar os resultados seguindo um padrão, para facilitar a comparação entre eles. Cada cenário é constituído por uma combinação de:

- Arquitetura da rede
- Arranjo de dados
- Parâmetros de treinamento e teste

Esses itens são variados em cada cenário para tentar identificar as características de convergência e aprendizado para cada rede, além de principalmente tentar identificar o melhor modelo possível para o nosso problema proposto.

Pode-se verificar na tabela abaixo as características de cada cenário.

	Arquitetura	**Dados de Entrada**	**Algoritmo de Treinamento**
Cenário um	3-40-1	PETR4 + PTAX + OIL	Levenberg-Marquardt
Cenário 2	3-40-1	PETR4 + PTAX + OIL	Bayesian Regularization
Cenário 3	7-40-1	5 x PETR4 + PTAX + OIL	Levenberg-Marquardt
Cenário 4	7-40-1	5 x PETR4 + PTAX + OIL	Bayesian Regularization
Cenário 5	3-8-1	PETR4 + PTAX + OIL	Levenberg-Marquardt
Cenário 6	3-20-10-1	PETR4 + PTAX + OIL	Levenberg-Marquardt

Tabela 5.1 Características dos cenários

5.3.2 Resultados.

Todos os 6 cenários foram implementados e testados a partir do grupo de dados do preço da ação PETR4 da Petrobrás, utilizado também para definição dos modelos econométricos discutidos anteriormente.

Os cenários 3 e 4 dispõem de um arranjo de dados de entrada diferente. Enquanto que nos demais cenários as entradas para a rede são preço da ação, cotação do dólar e preço do barril de petróleo, nestes dois cenários citados foram utilizadas séries das últimas cinco cotações correntes da PETR4. A idéia consiste na análise da resposta

da rede no caso de disponibilizar entradas mais completas, com uma série de cotações no lugar de apenas uma única cotação da ação.

Outra variação proposta foi o algoritmo de treinamento utilizado. Foram experimentados inicialmente inúmeros algoritmos para treinar a rede:

- Levenberg-Marquardt (trainlm)
- Bayesian Regularization (trainbr)
- Quasi-Newton (trainbfg)
- Resilient Backpropagation (trainrp)
- Scaled Conjugate Gradient (trainscg)
- Conjugate Gradient with Powell/Beale Restarts (traincgb)
- Fletcher-Powell Conjugate Gradient (traincgf)
- Polak-Ribiére Conjugate Gradient (traincgp)
- One-Step Secant (trainoss)
- Variable Learning Rate Backpropagation (traingdx)

Apenas os algoritmos Levenberg-Marquardt e Bayesian Regularization convergiram de maneira mais eficiente. Alguns levaram muitos passos para convergir, outros não convergiram e outros definiram padrões de reconhecimento extremamente ruins. Logo se concentrou todo o esforço adicional para analisar o comportamento das redes utilizando esses dois algoritmos.

A última, mas não menos importante ocorreu com a arquitetura. No cenário 5 foi reduzido o número de neurônios da camada interna, que era de 40, para 8. Já no cenário 6 experimentou-se aumentar o numero de camadas internas para 2, com 20 neurônios na primeira camada interna e 10 na segunda.

A seguir são apresentados em duas tabelas os resultados dos testes.

A primeira mostra os resultados da análise do treinamento da rede.

	Epochs	Performance	SSE	m	b	R
Cenário 1	69	0.821918	-	0,995	0,145	0,998
Cenário 2	77	-	546.147	0,995	0,166	0,998
Cenário 3	18	0,648189	-	0,991	0,3	0,998
Cenário 4	40	-	362,991	0,995	0,152	0,998
Cenário 5	62	0,824323	-	0,995	0,134	0,998
Cenário 6	13	124,375	-	0,478	22	0,689

Tabela 5.2 Resultados da análise do treinamento das redes.

Para analisar as saídas obtidas na simulação podem-se calcular os seguintes erros estatísticos:

Root Mean Squared Error	$\sqrt{\sum_{t=T+1}^{T+h} (\hat{y}_t - y_t)^2 / h}$
Mean Absolute Error	$\sum_{t=T+1}^{T+h} \left\|\hat{y}_t - y_t\right\| / h$
Mean Absolute Percentage Error	$100 \sum_{t=T+1}^{T+h} \left\|\frac{\hat{y}_t - y_t}{y_t}\right\| / h$
Theil Inequality Coefficient	$\frac{\sqrt{\sum_{t=T+1}^{T+h} (\hat{y}_t - y_t)^2 / h}}{\sqrt{\sum_{t=T+1}^{T+h} \hat{y}_t^{\,2} / h} + \sqrt{\sum_{t=T+1}^{T+h} y_t^{\,2} / h}}$

Tabela 5.3 Cálculo dos erros estatísticos.

Os dois primeiros (rms e mae) dependem da escala da variável analisada, por isso devem ser utilizados apenas para comparação quando os modelos em questão utilizarem os mesmos dados de teste. Os outros dois (mape e tic) são independentes da escala. O Theil Inequality Coefficient vai de zero a um, com zero indicando um encaixe perfeito entre a saída esperada e a prevista.

A segunda tabela mostra os resultados da análise das saídas simuladas.

	RMS	**MAE**	**TIC**
Cenário 1	2.0962	1.6178	0.0144
Cenário 2	2.0861	1.6163	0.0143
Cenário 3	4.5436	3.2646	0.0314
Cenário 4	2.3834	1.8681	0.0163
Cenário 5	2.2273	1.7636	0.0153
Cenário 6	22.2194	22.0828	0.1804

Tabela 5.4 Resultados da análise das saídas simuladas.

Analisando essa tabela, pode-se verificar que as redes dos cenários um e dois apresentaram os melhores resultados, além de estarem muito próximos entre si. Isso mostra que a arquitetura multicamadas com os neurônios dispostos na configuração 3-40-1 representa um bom modelo para o problema em questão. Observa-se, através da tabela 1, que a regressão entre a resposta da rede e as correspondentes saídas-alvo é ligeiramente melhor no cenário um, pois seu deslocamento é de apenas 0,144 contra 0,166 do cenário dois. Além disso, a convergência do algoritmo Levenberg-Marquardt no primeiro cenário é mais rápida, realizada em 69 épocas contra 77 do cenário dois.

No teste da rede com duas camadas intermediárias os resultados foram péssimos. Após apenas 13 épocas o algoritmo já é interrompido, apresentando um valor de performance de 124,375.

Os cenários três e quatro, que utilizaram uma serie dos últimos 5 preços da ação para cada previsão, não apresentaram resultados satisfatórios, demonstrando que a correlação entre o preço corrente do barril de petróleo e a taxa de dólar com o preço corrente da ação é maior do que utilizar a série das últimas cinco cotações.

O cenário cinco, onde a camada intermediária da rede possui 8 neurônios, apresenta uma boa regressão entre a resposta da rede e as saídas esperadas, entretanto a previsão a partir de novos dados apresenta grandes erros estatísticos. Pode-se deduzir que o pequeno número de neurônios na camada interna não possibilita, de modo qualitativo, que a rede neural se adapte a novos dados e realize boas previsões.

Considerando a rede neural do primeiro cenário como a melhor aproximação para o problema proposto, pode-se comparar as suas previsões com as realizadas com os modelos econométricos.

O gráfico abaixa apresenta as previsões realizadas pela rede e também pelos modelos econométricos AR e GARCH.

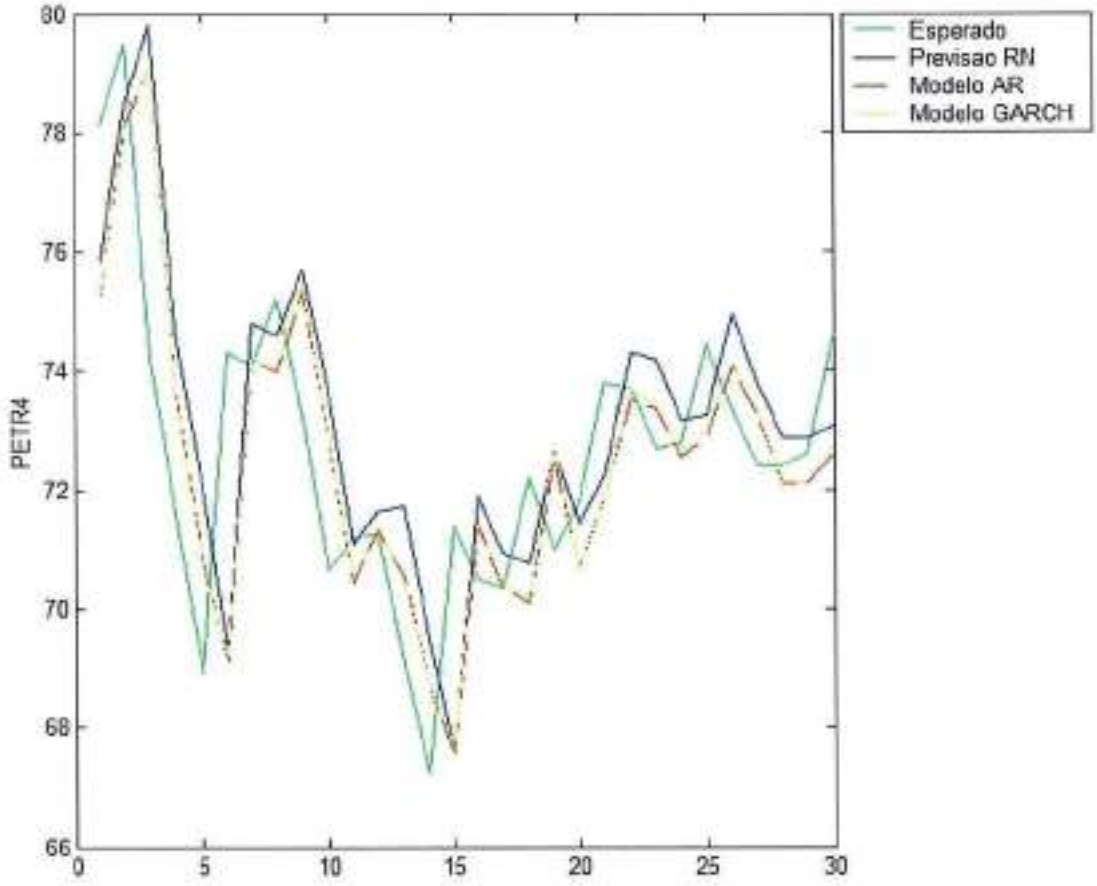


Figura 5.2 Previsões realizadas pela rede e pelos modelos econométricos AR e GARCH

Pode-se notar pelo gráfico que as previsões nitidamente acompanham os valores esperados, mostrando o bom aprendizado da rede e a boa definição dos modelos econométricos.

Pode-se notar, através da seguinte tabela, que os erros estatísticos confirmam o alinhamento dos modelos acima citados.

	RMS	MAE	TIC
Rede Neural	2.0962	1.6178	0.0144
Modelo AR	2,0097	1,5434	0,0138
Modelo GARCH	1,9865	1,5265	0,0137

Tabela 5.5 Erros estatísticos do modelo de RN e dos modelos econométricos.

A partir da série real e da série prevista, pode-se simular processos de compra e venda das ações PETR4, a fim de verificar-se o quanto aplicável seriam os modelos no ambiente real. Deve-se ressaltar que o escopo do problema consiste apenas na previsão das séries econômico-financeiras a partir dos modelos e a comparação entre as previsões obtidas. A utilização dos resultados na determinação de ações de compra e venda é exclusivamente discutível.

Para os casos em que a previsão do próximo valor da PETR4 é maior que o valor corrente considera-se naturalmente uma decisão de compra, de forma a garantir ganho financeiro. Para os casos em que o valor previsto é menor, define-se uma decisão de venda.

A seguir a tabela com as taxas de acerto para cada modelo estudado.

	GARCH	AR	RN
Taxa Acerto	70%	67%	43%

5.3.3 Gráficos de Suporte para Análise

A seguir são apresentados os gráficos para cada cenário de treinamento e teste. Esses gráficos serviram como base para a análise da performance, convergência e regressão no treinamento de cada rede, além de gráficos comparando a resposta da rede aos dados de teste com as saídas esperadas.

- Gráfico A – Função de desempenho na convergência do algoritmo.
- Gráfico B – Erros de treinamento, validação e teste durante o treinamento.
- Gráfico C – Regressão entre a resposta da rede e as saídas esperadas.
- Gráfico D – Comparação entre a previsão e as saídas esperadas dos dados de teste.
- Gráfico E – Variação do erro entre a previsão e o esperado.

Cenário 1

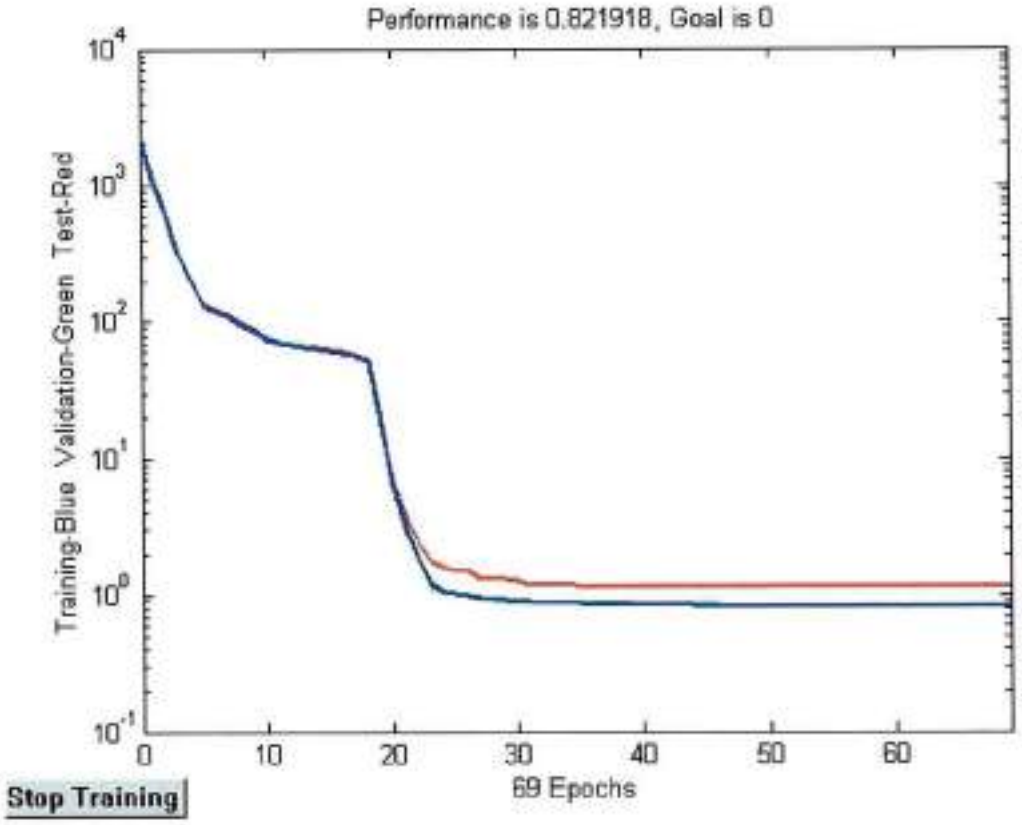


Figura 5.3 Gráfico A

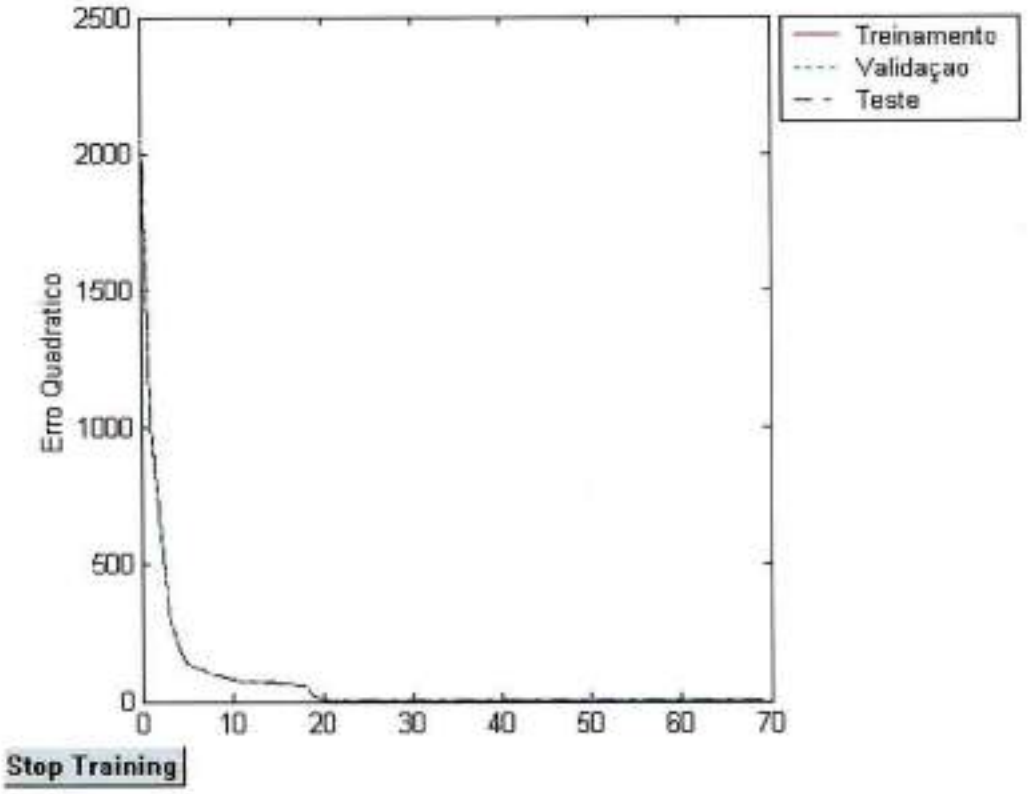


Figura 5.4 Gráfico B

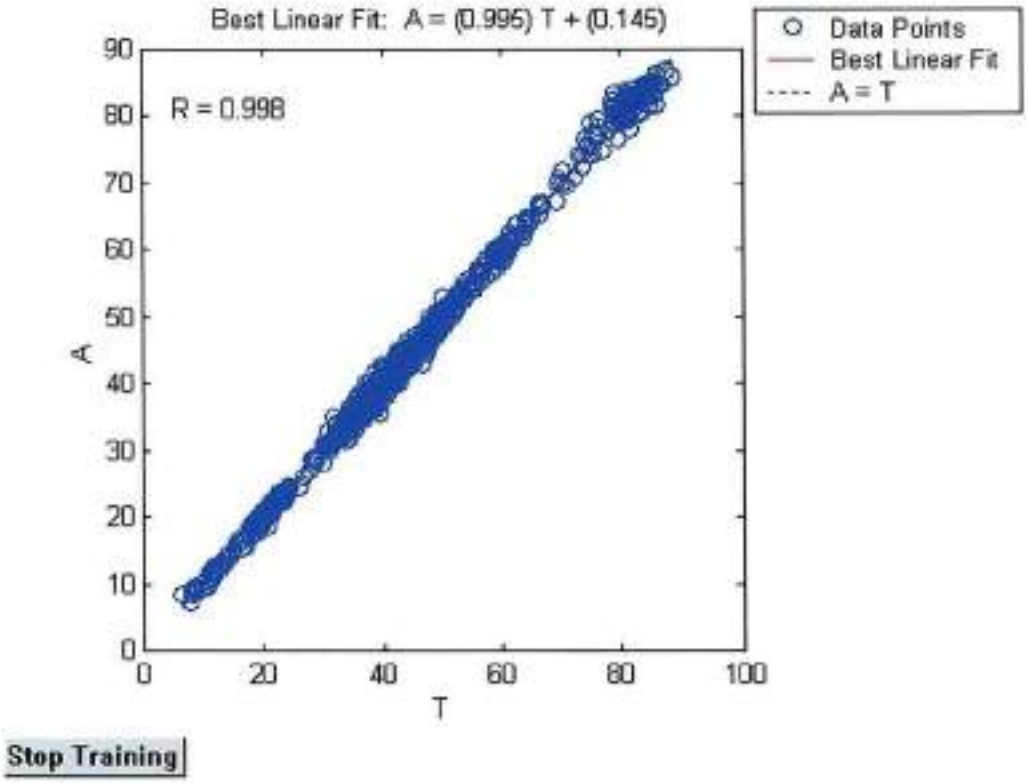


Figura 5.5 Gráfico C

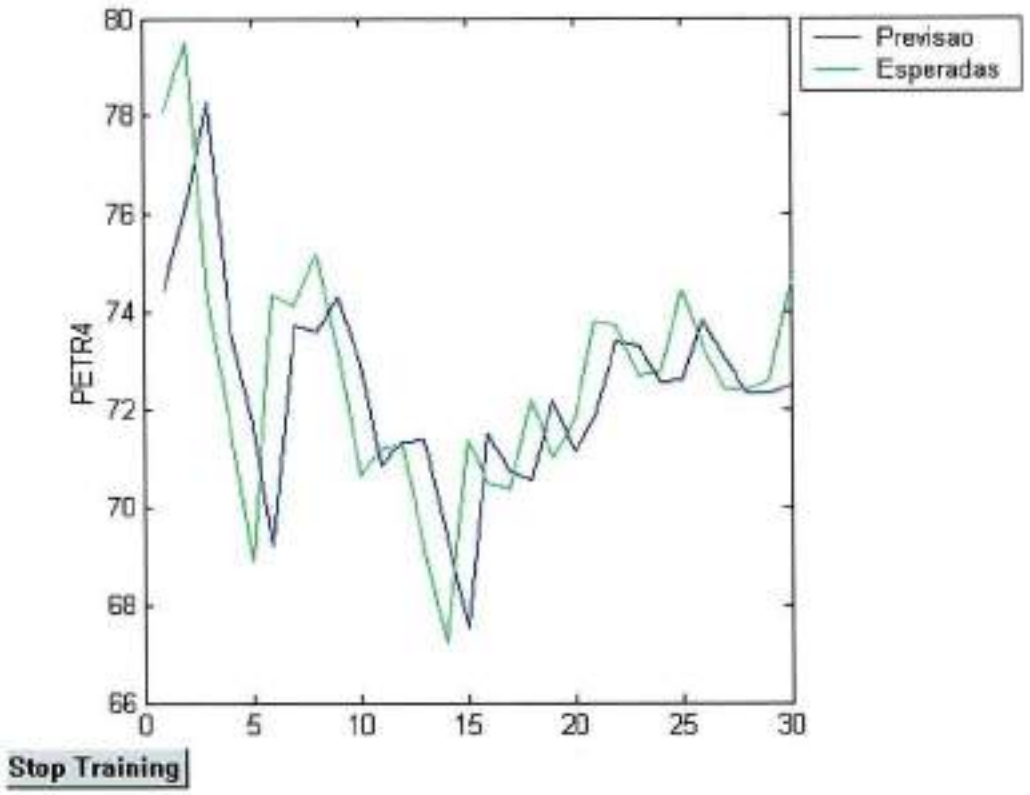


Figura 5.6 Gráfico D

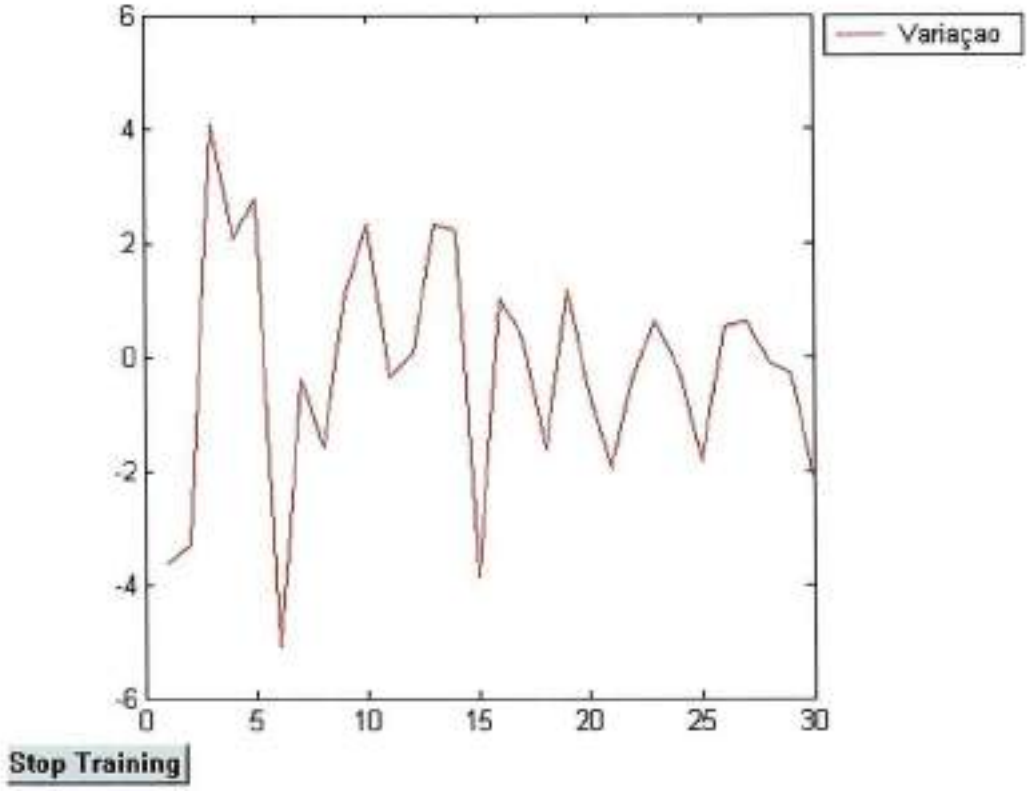


Figura 5.7 Gráfico E

Cenário 2

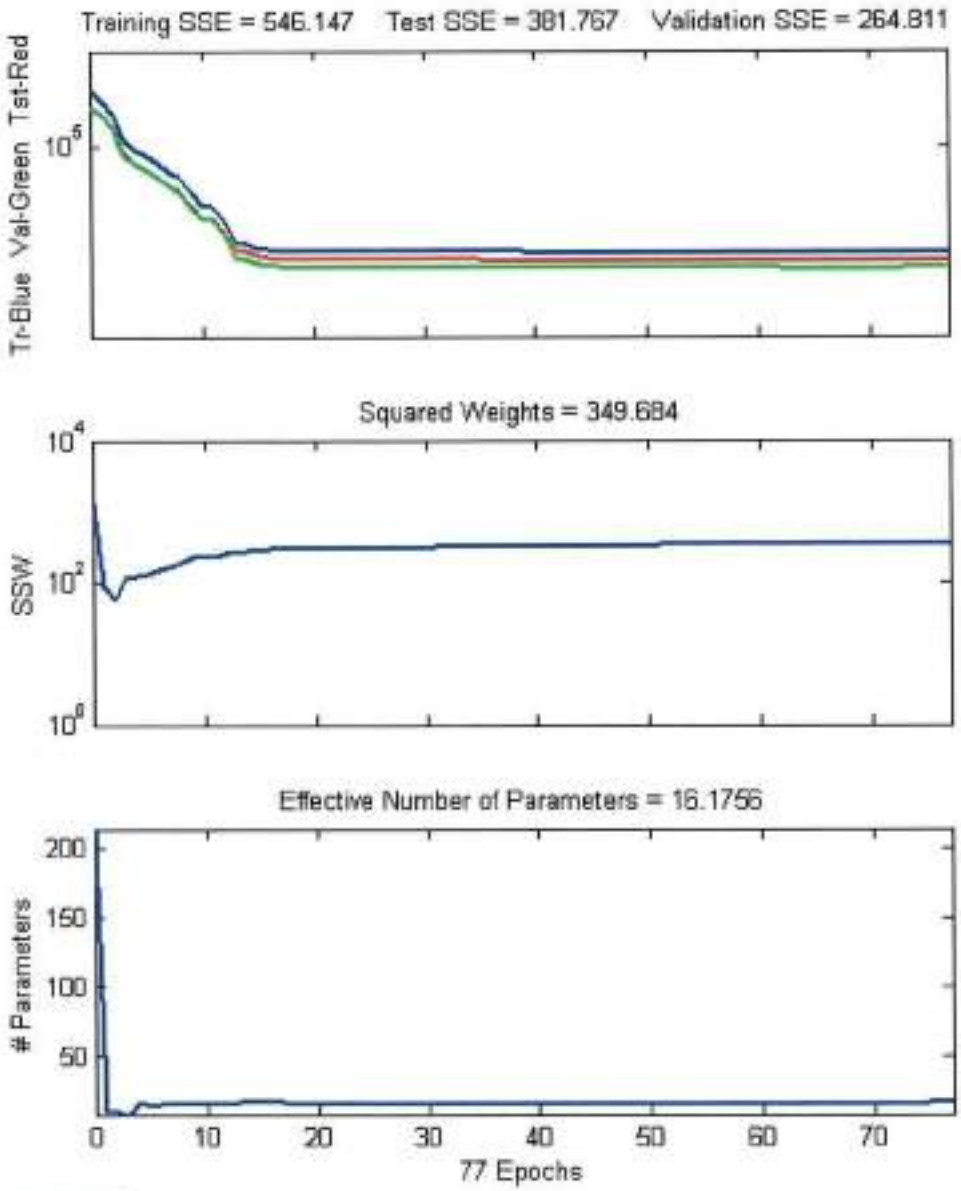


Figura 5.8 Gráfico A

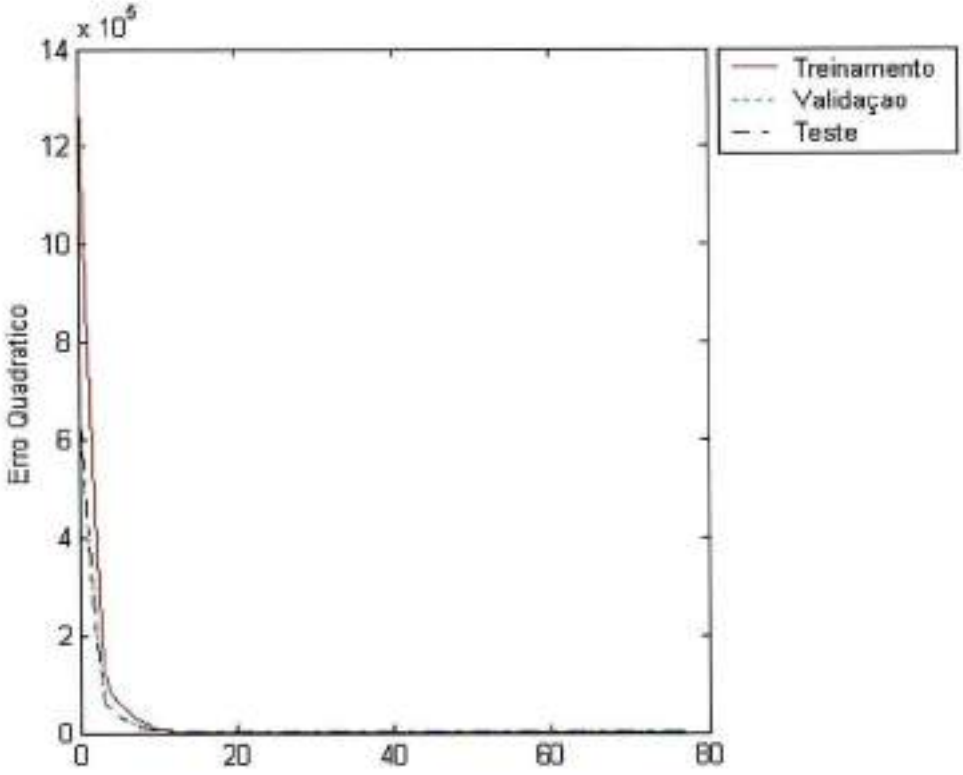


Figura 5.9 Gráfico B

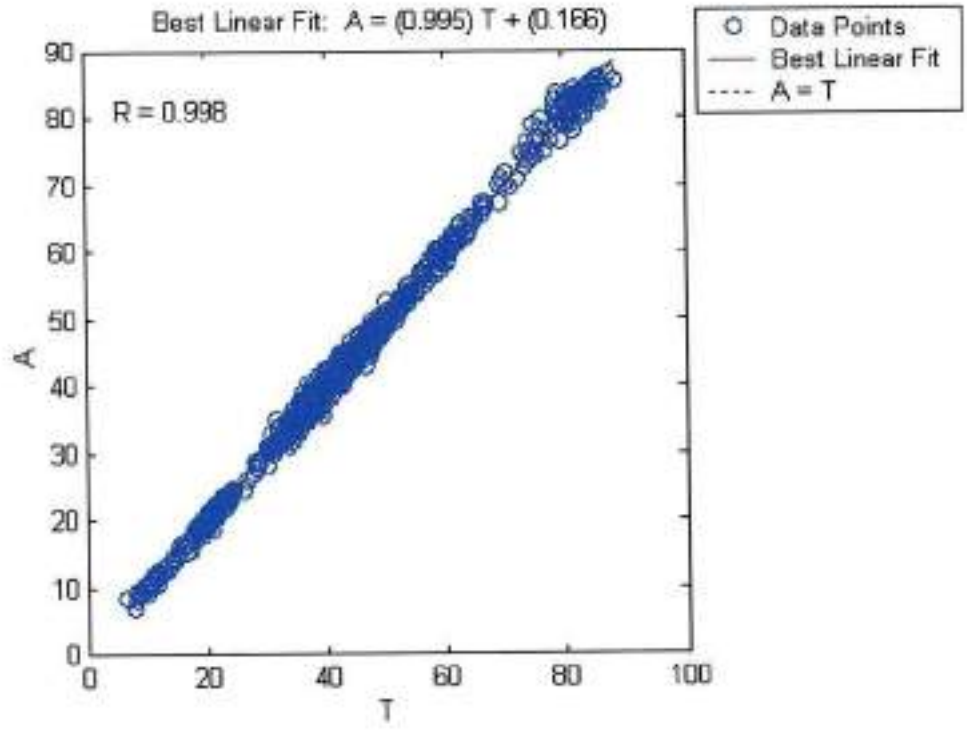


Figura 5.10 Gráfico C

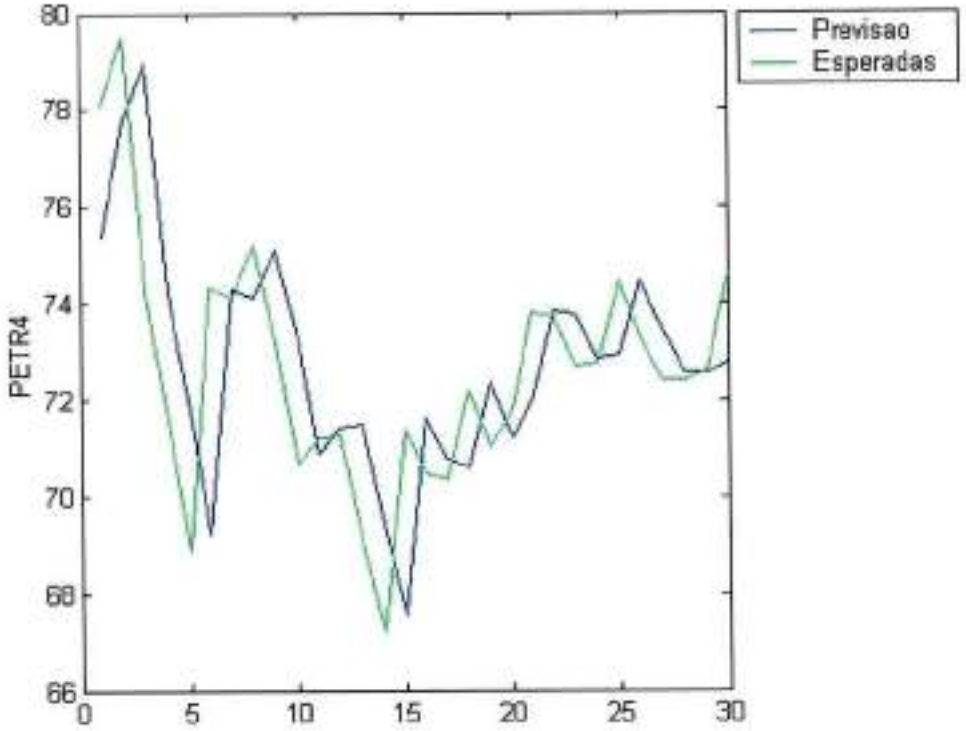


Figura 5.11 Gráfico D

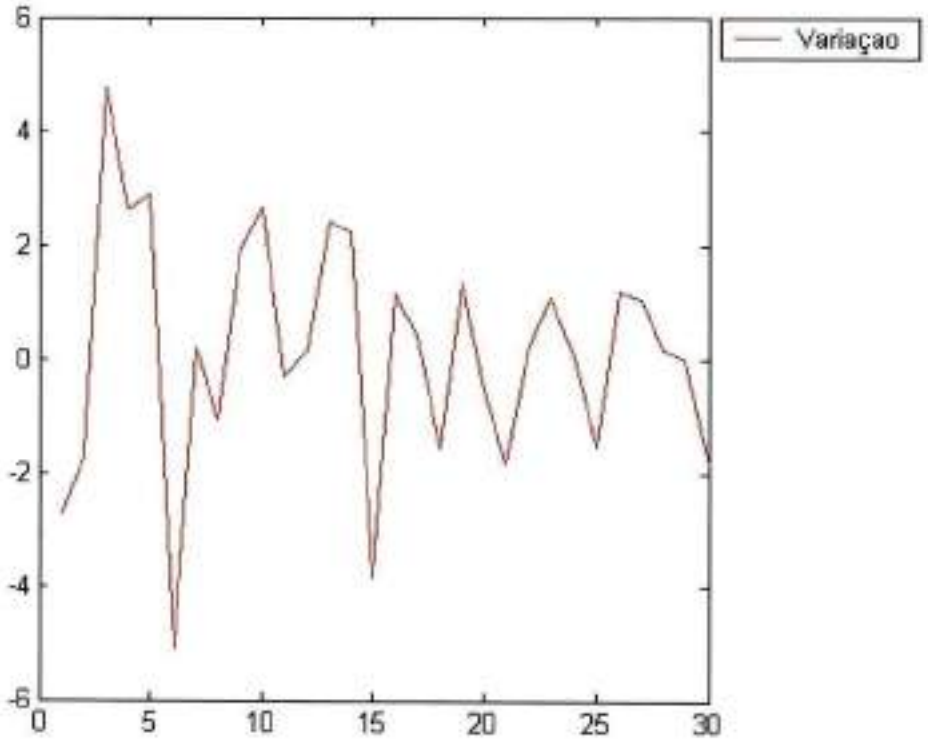


Figura 5.12 Gráfico E

Cenário 3

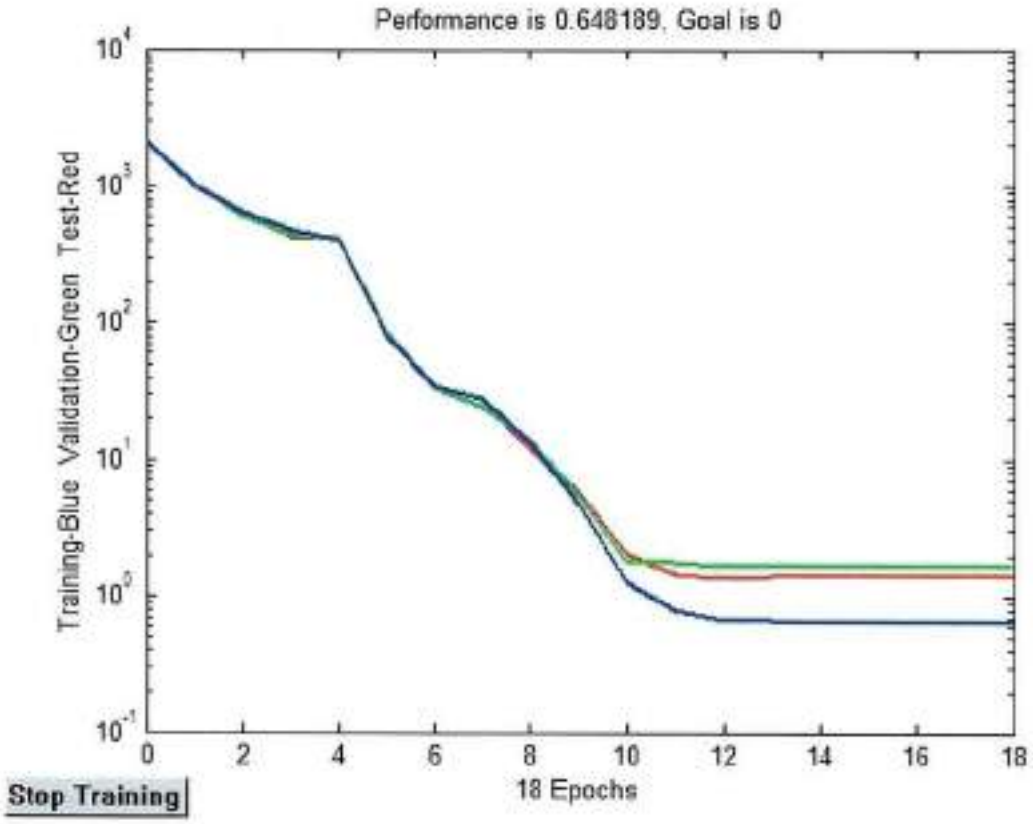


Figura 5.13 Gráfico A

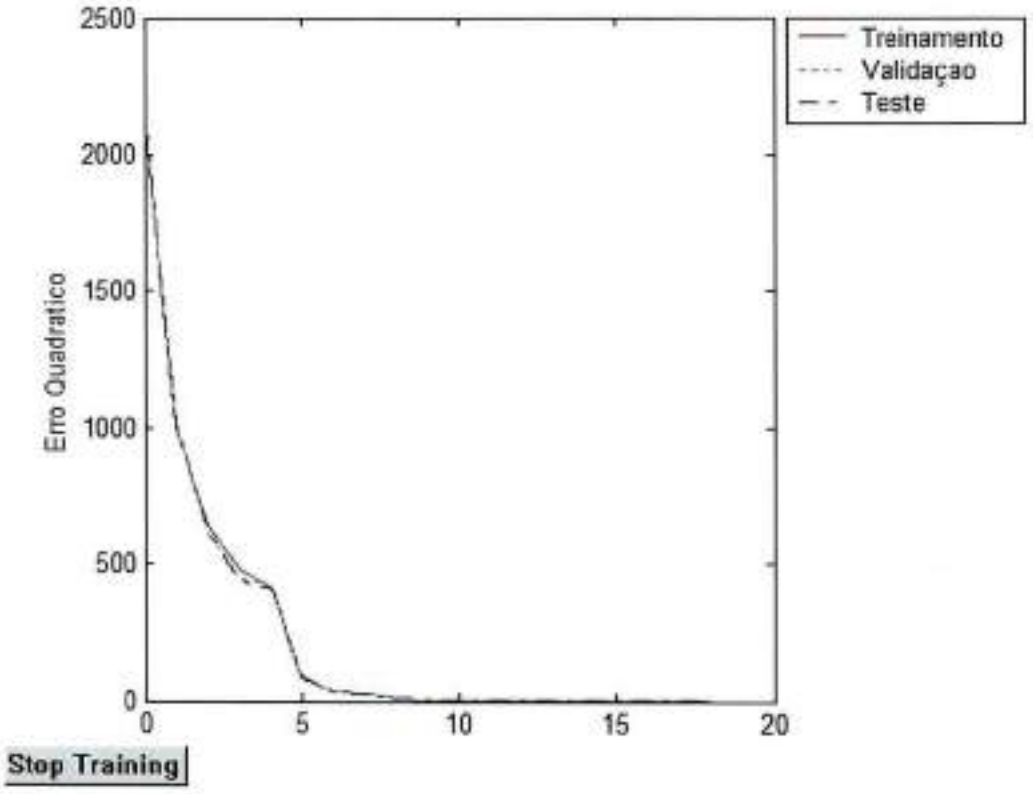


Figura 5.14 Gráfico B

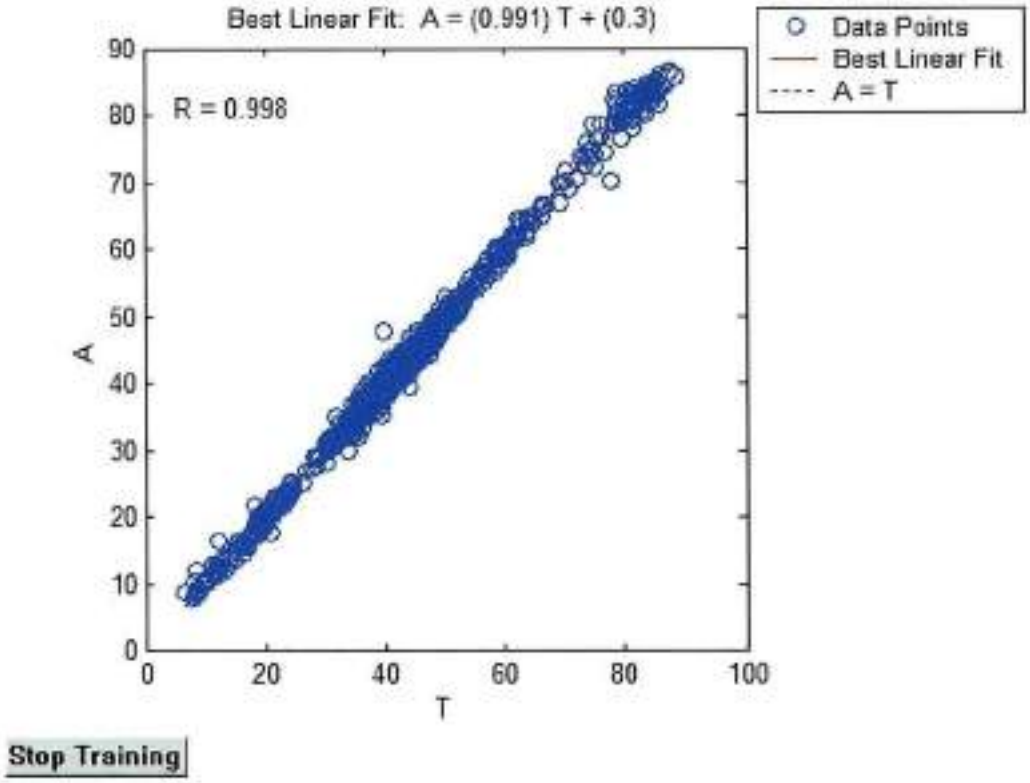


Figura 5.15 Gráfico C

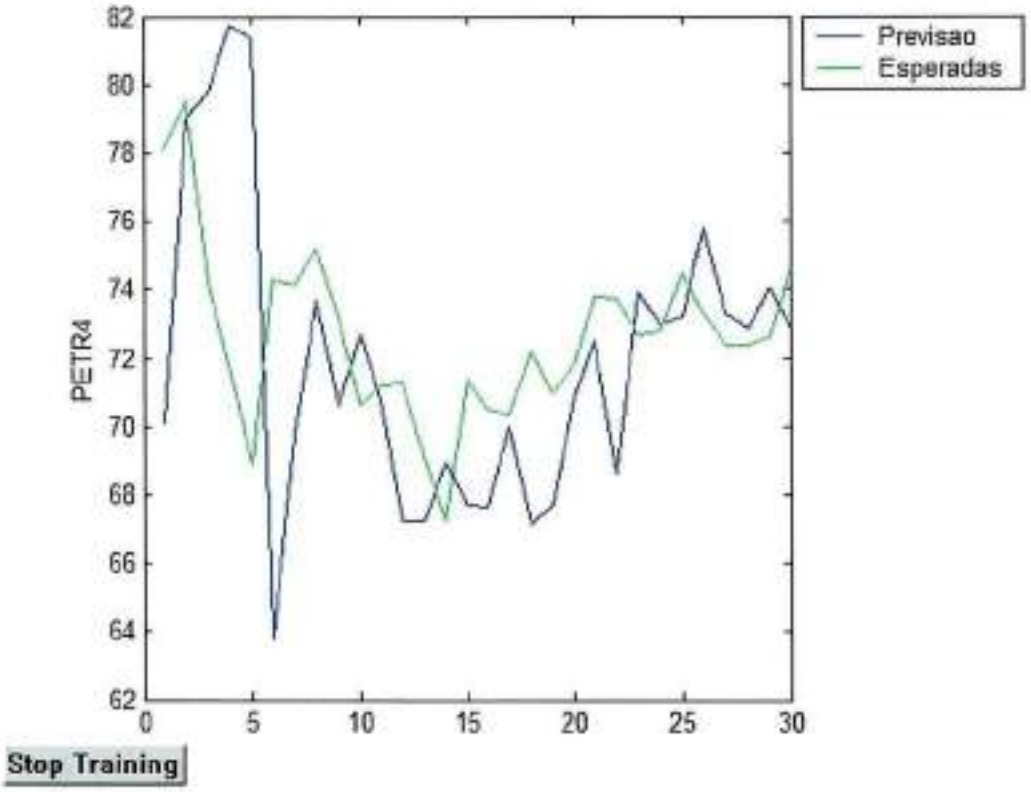


Figura 5.16 Gráfico D

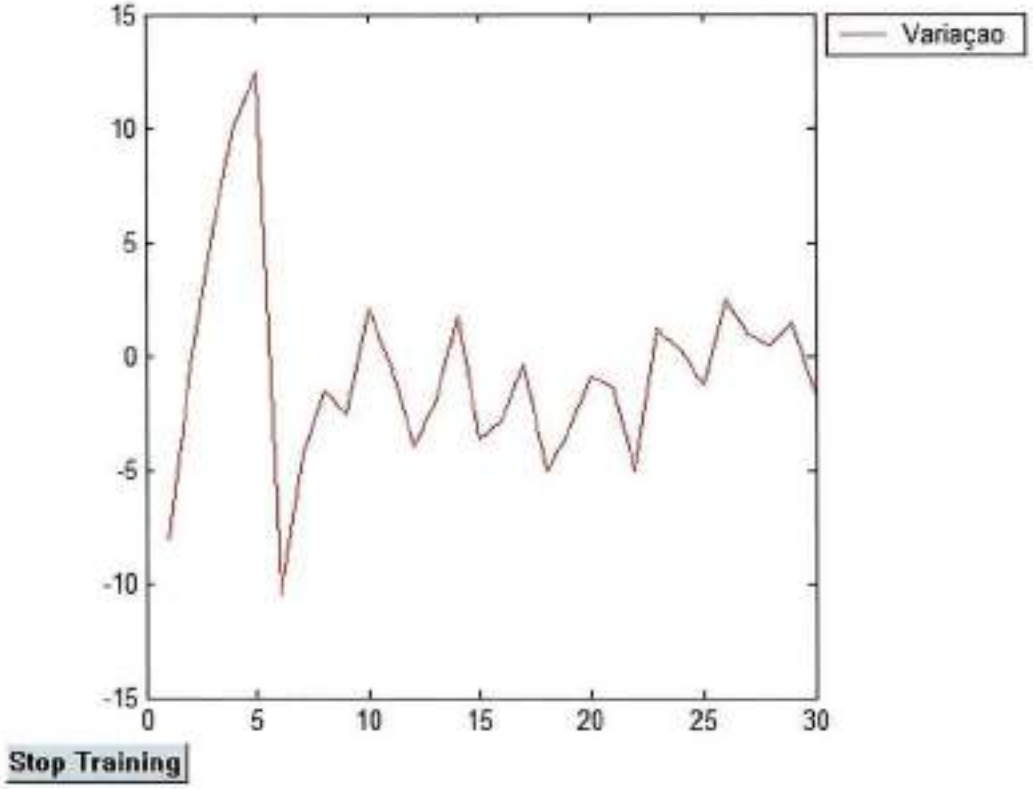


Figura 5.17 Gráfico E

Cenário 4

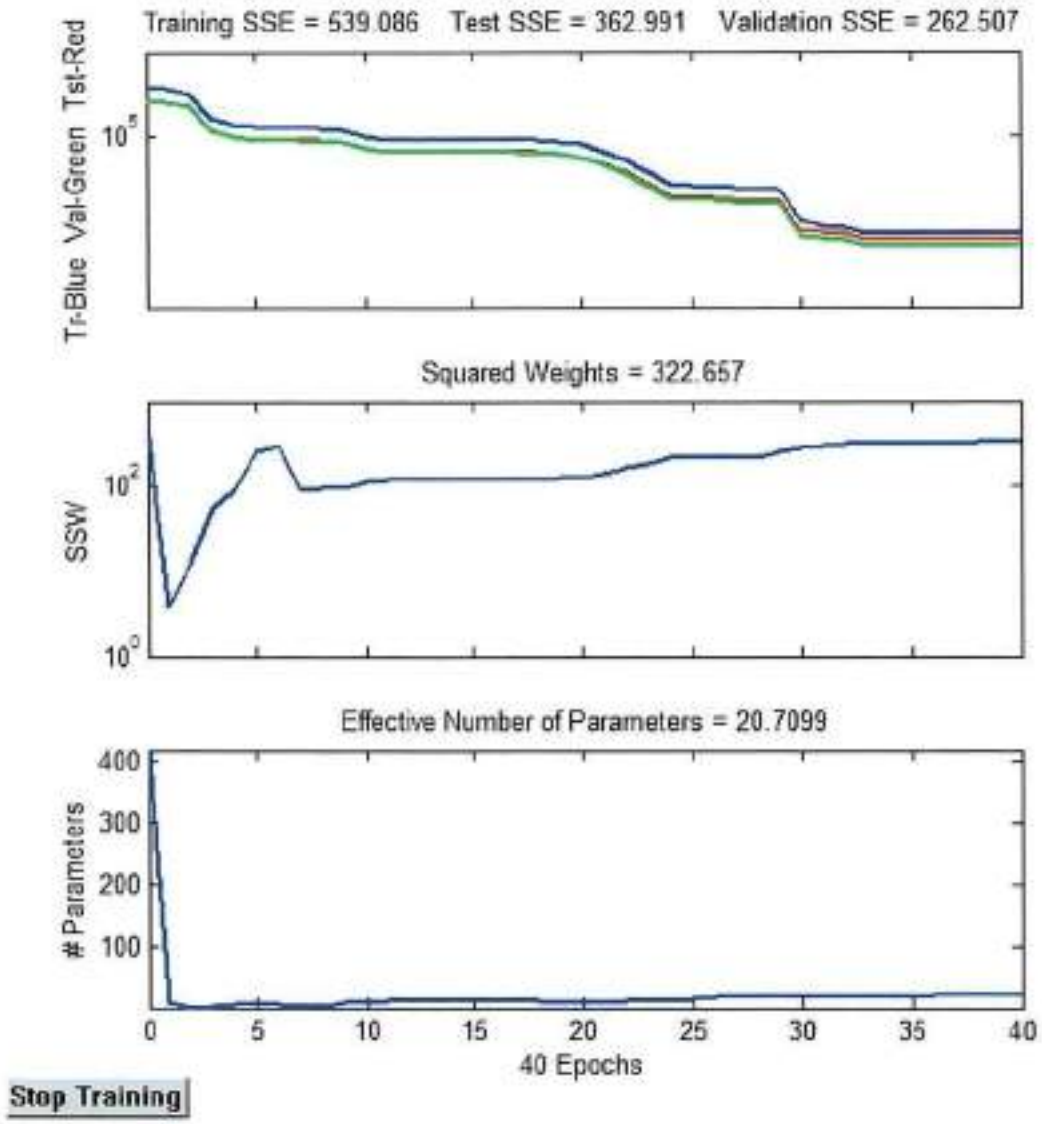


Figura 5.18 Gráfico A

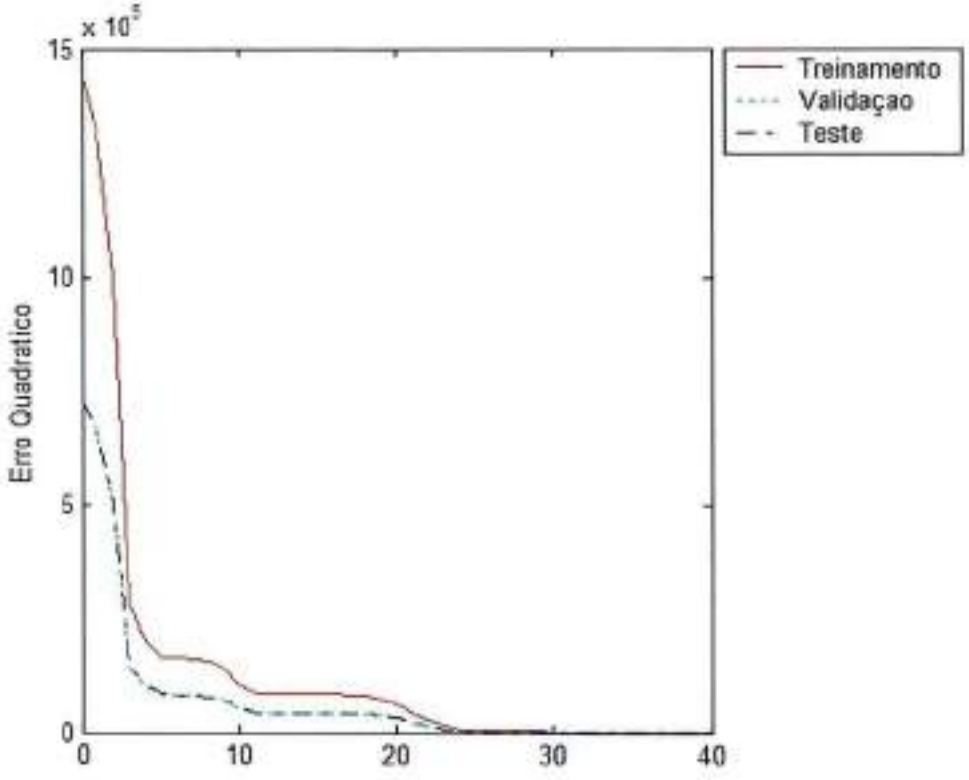


Figura 5.19 Gráfico B

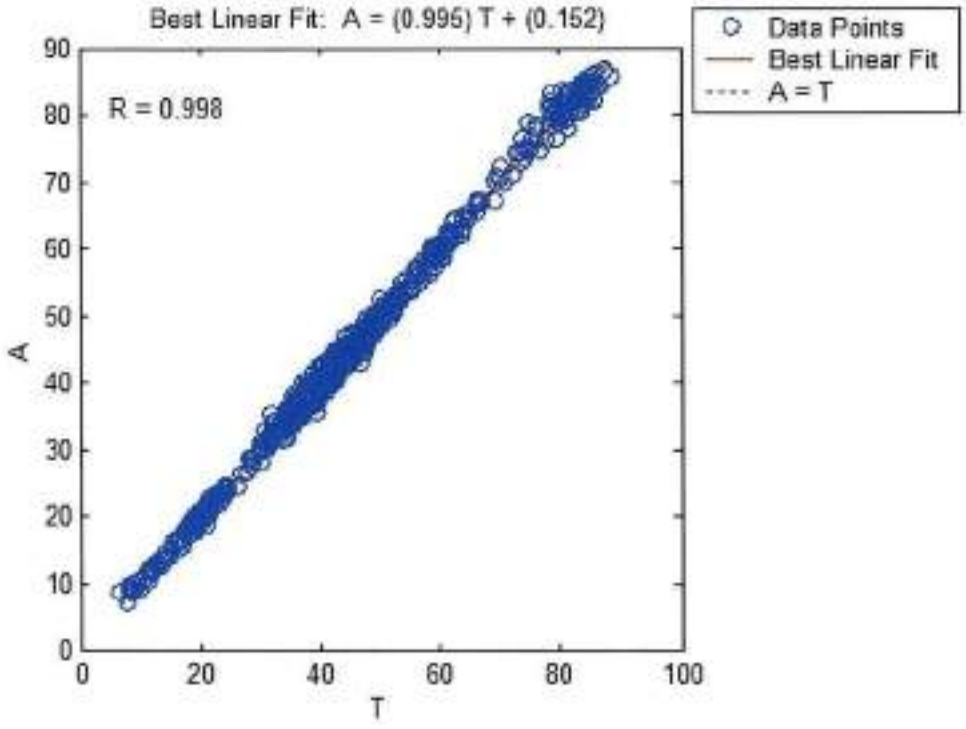


Figura 5.20 Gráfico C

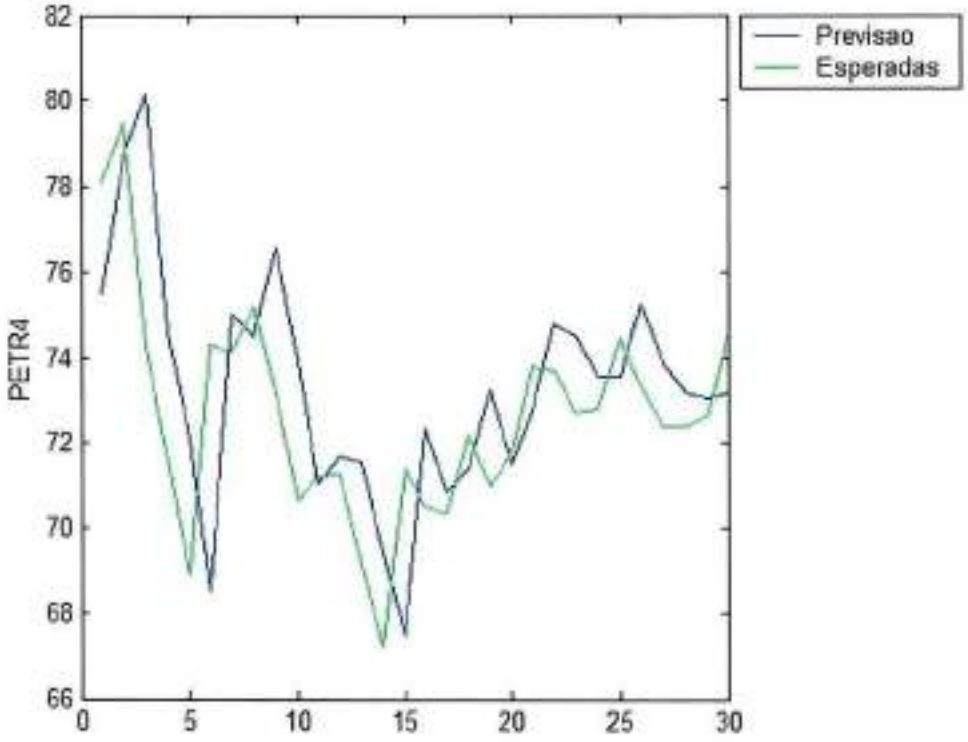


Figura 5.21 Gráfico D

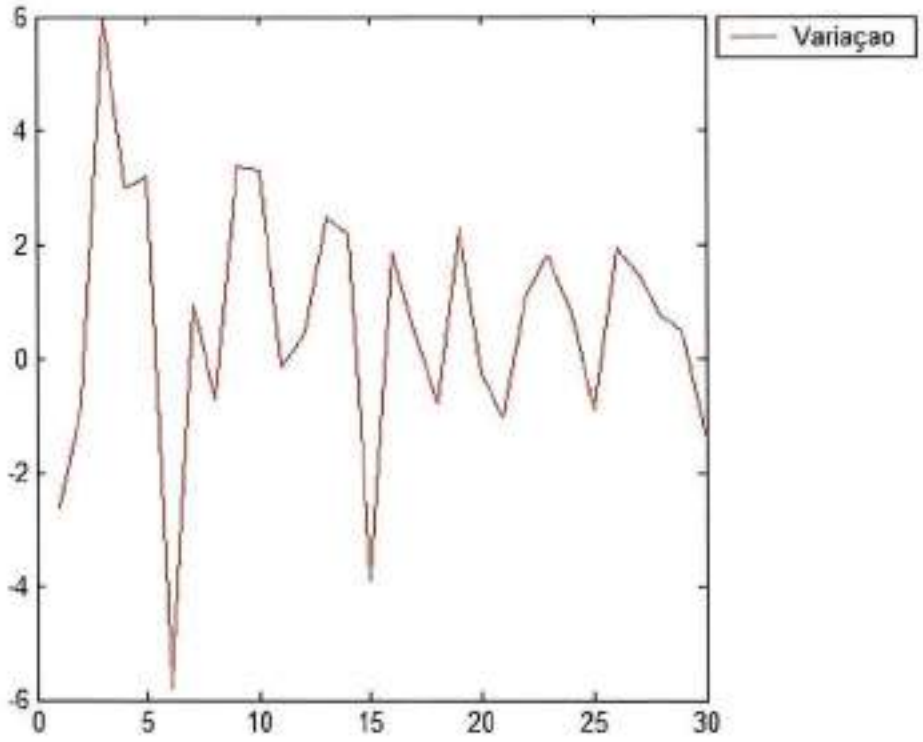


Figura 5.22 Gráfico E

Cenário 5

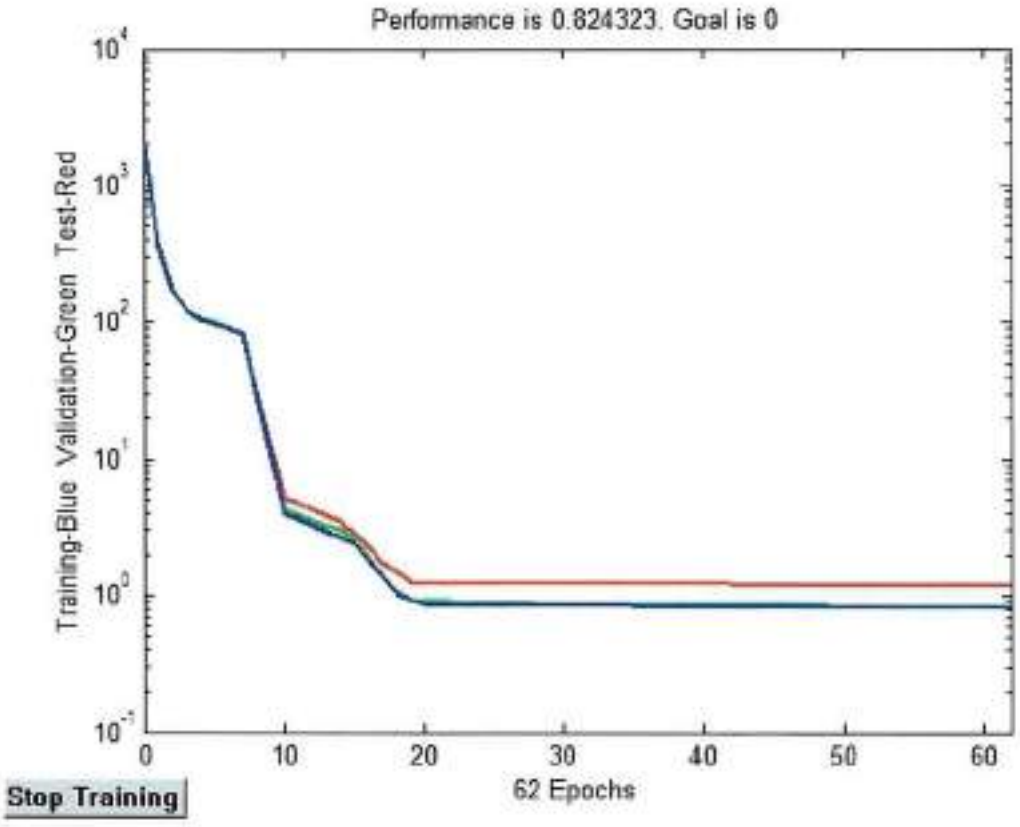


Figura 5.23 Gráfico A

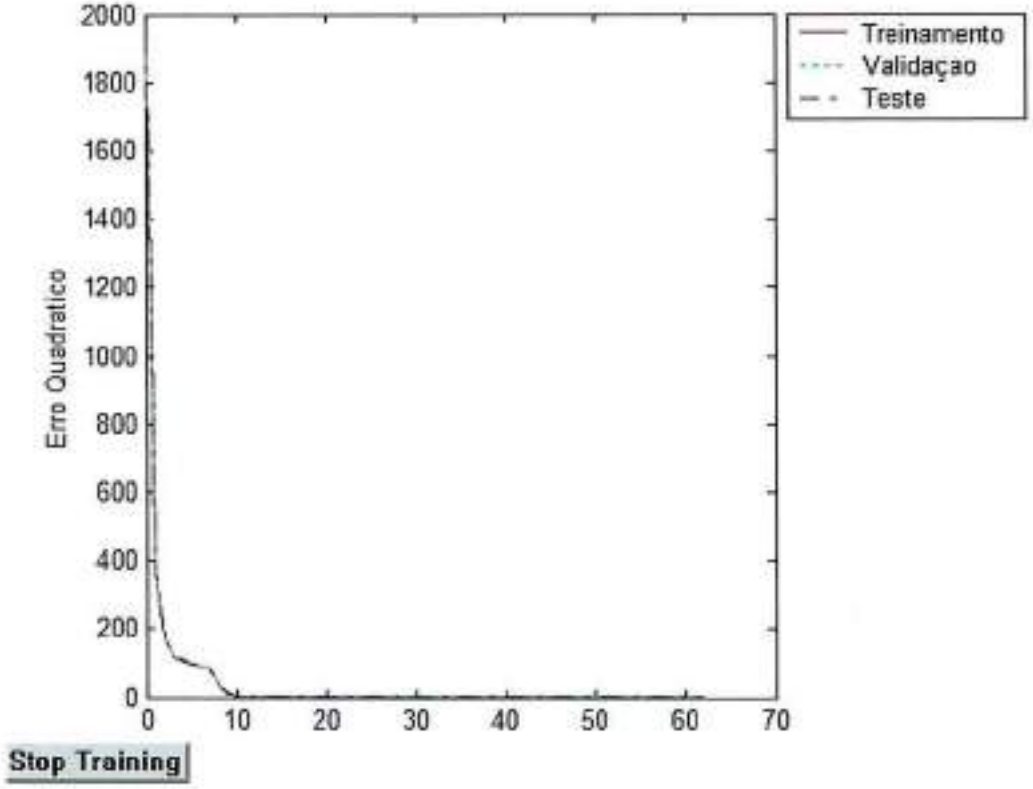


Figura 5.24 Gráfico B

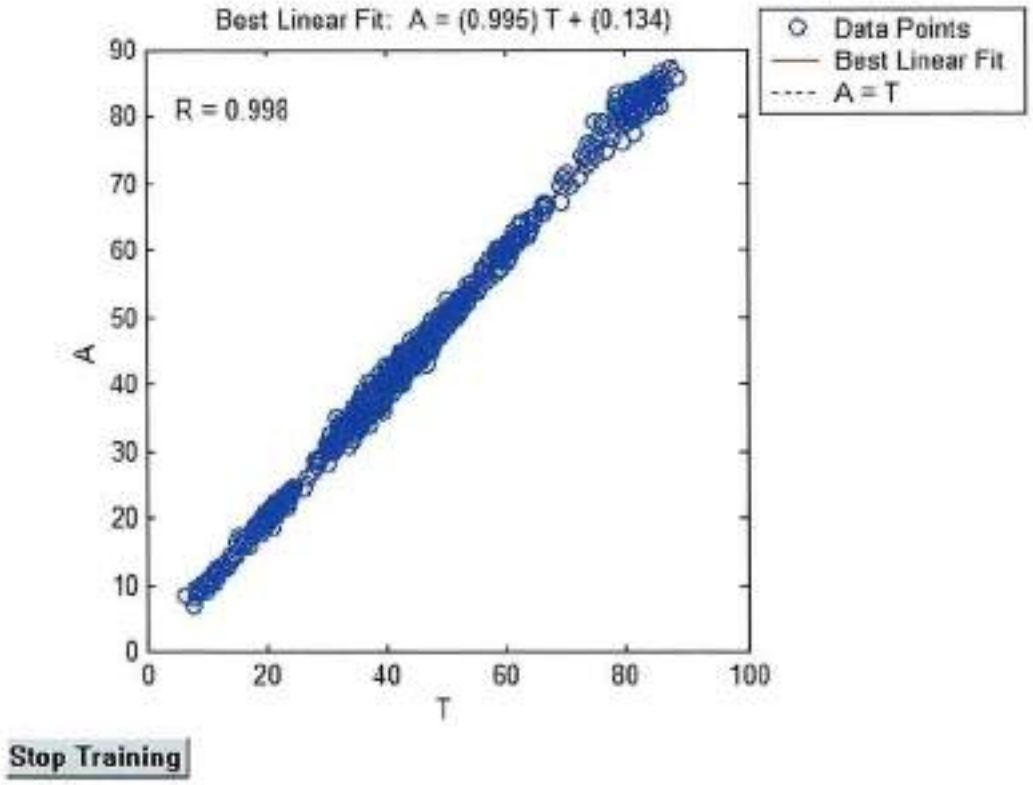


Figura 5.25 Gráfico C

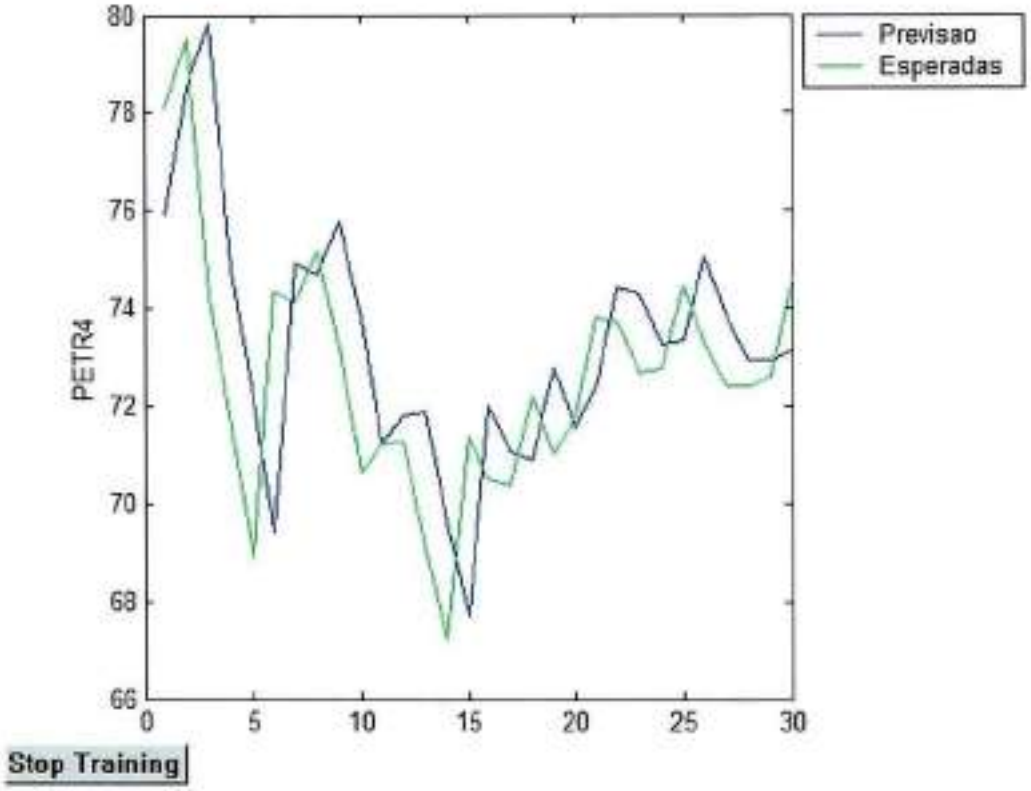


Figura 5.26 Gráfico D

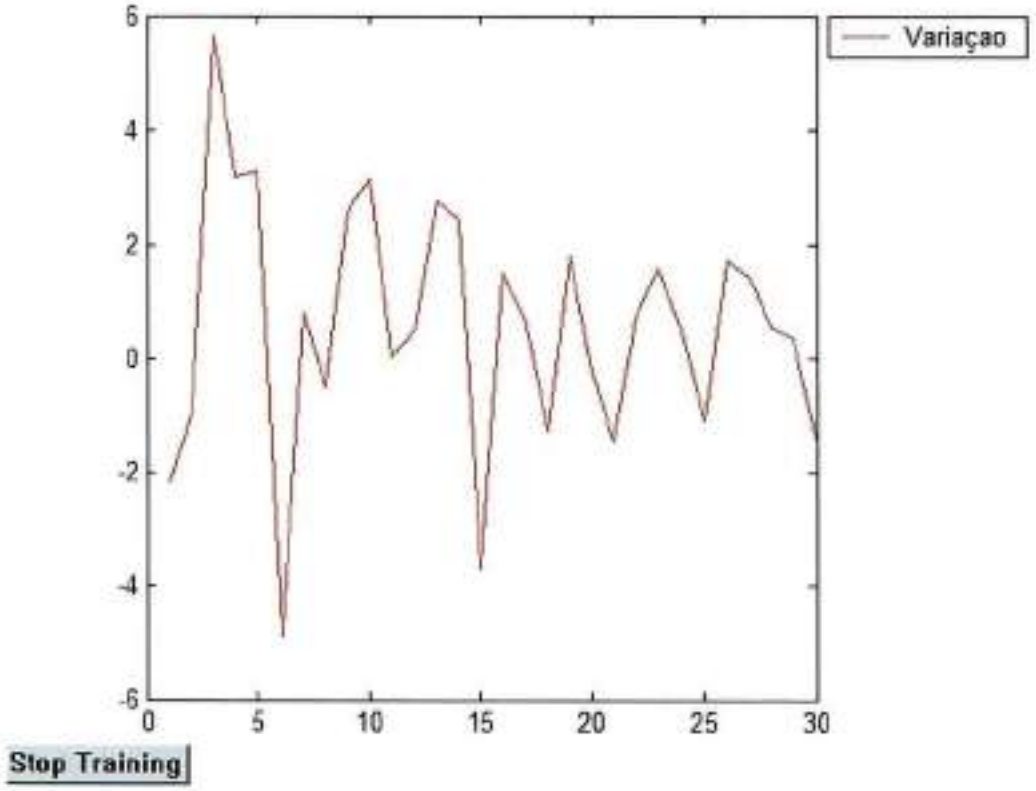


Figura 5.27 Gráfico E

Cenário 6

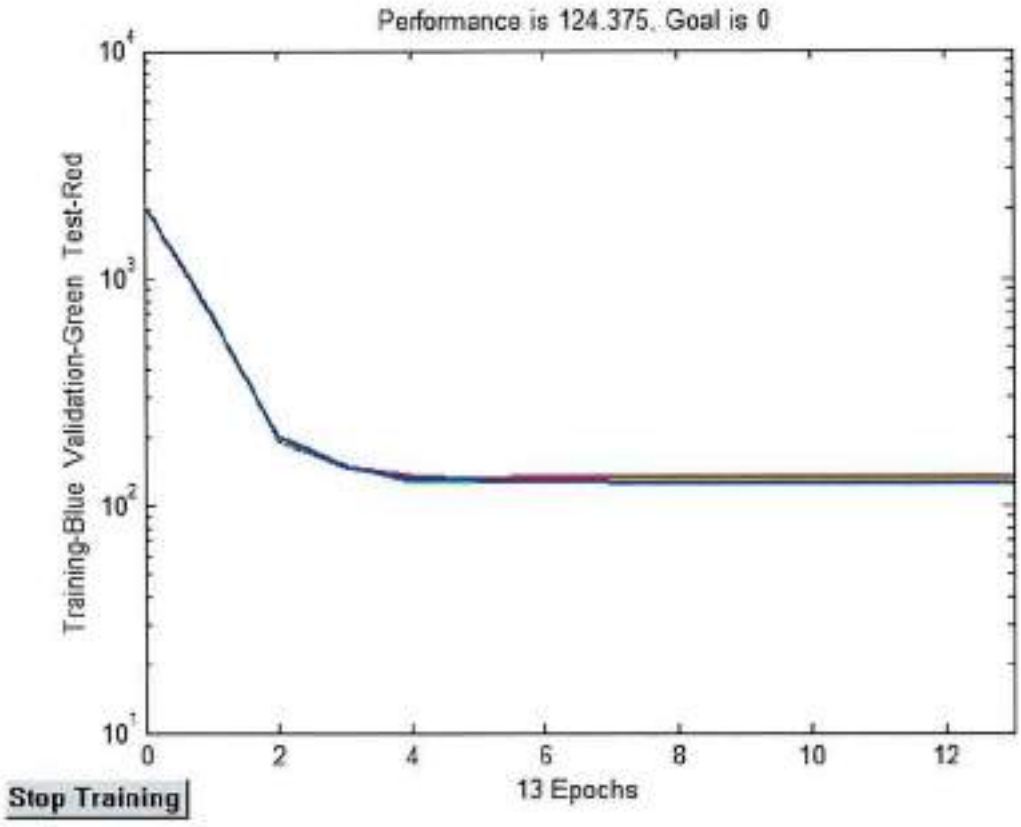


Figura 5.28 Gráfico A

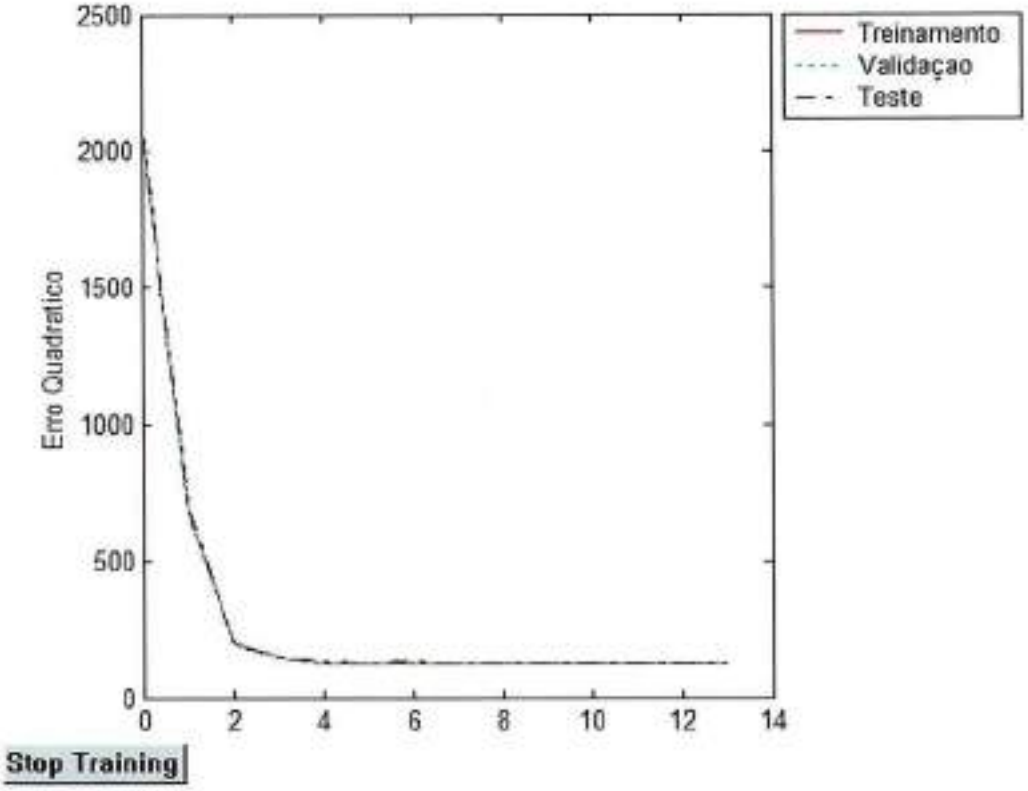


Figura 5.29 Gráfico B

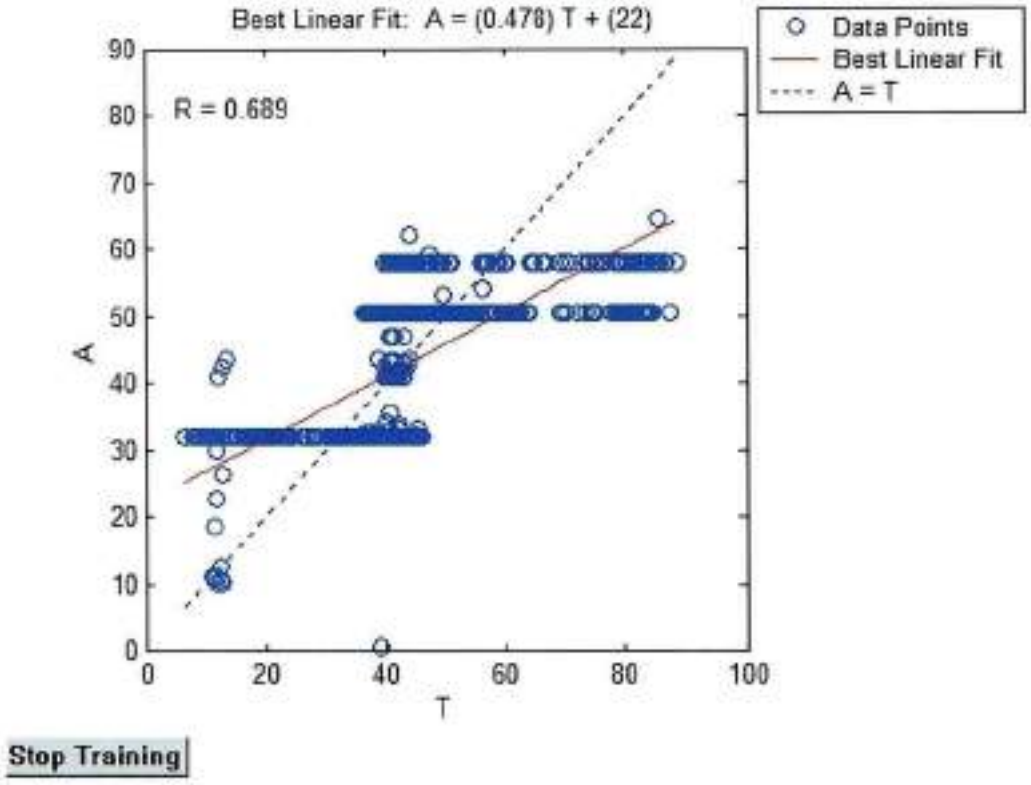


Figura 5.30 Gráfico C

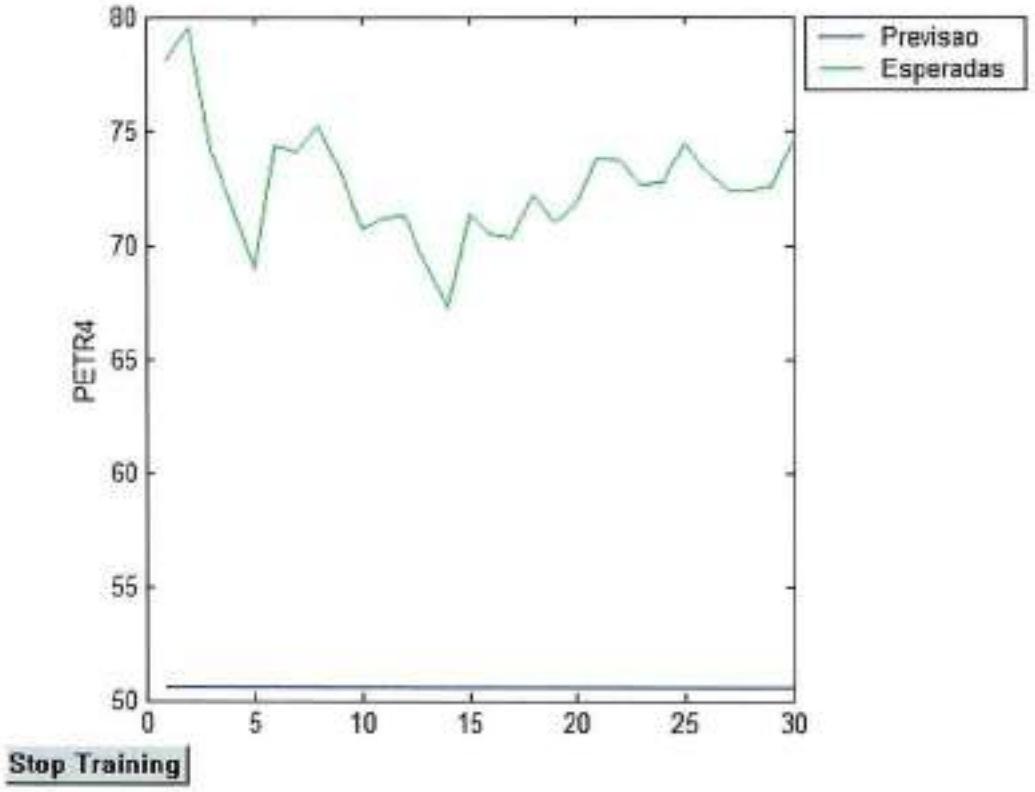


Figura 5.31 Gráfico D

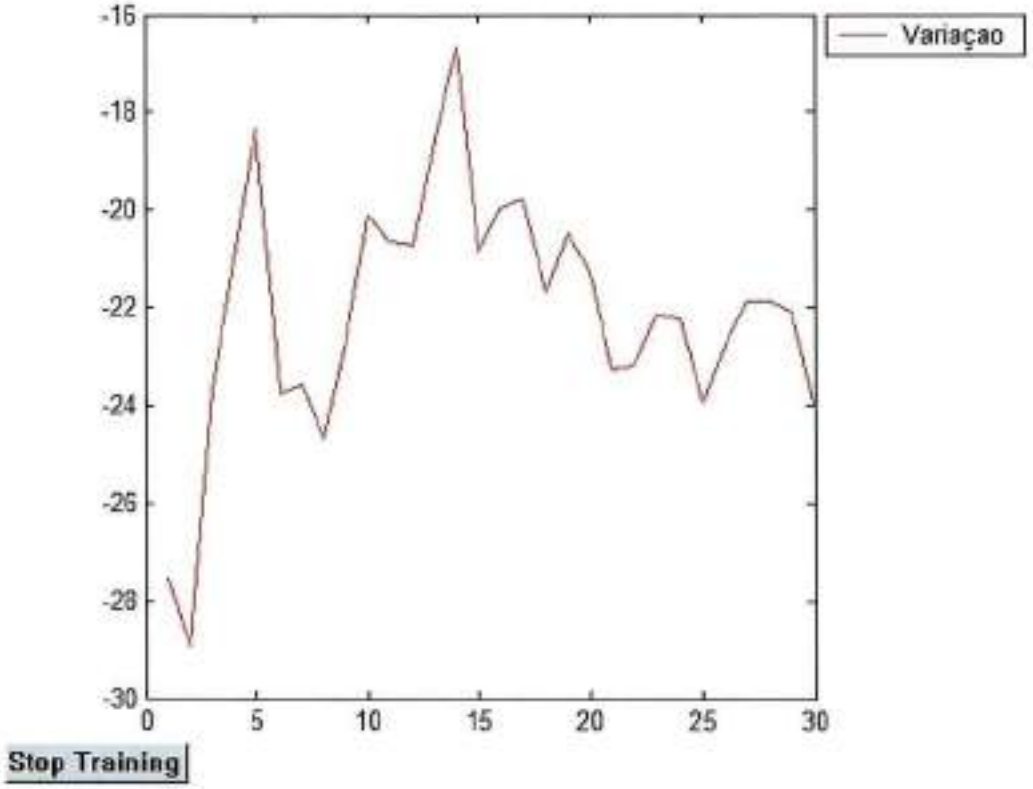


Figura 5.32 Gráfico E

5.3.4 Rotina de Treinamento e Simulação do Modelo de Rede Neural.

Essa é a rotina utilizada para treinar o modelo do cenário 1, o qual foi escolhido como a melhor aproximação de previsão.

```
% Rotina de Treinamento e Teste - PETROBRAS

% figure(gcf)
% clf;
echo on
clc

%   O arquivo petro_all contem as matrizes dos dados de entrada e target, baseados
nas series de entrada
clear;
load petro_all

% Divide os dados em dados de treinamento(metade), validaçao(um quarto) e
teste(um quarto).
% A ideia eh pegar os dados de forma bem distribuida. Para isso montamos as
series de cada tipo pegando
% dados de quatro em quatro dias.

[R,Q] = size(p);
iitst = 2:4:Q;
iival = 4:4:Q;
iitr = [1:4:Q 3:4:Q];
validation.P = p(:,iival);
validation.T = t(:,iival);
testing.P = p(:,iitst);
testing.T = t(:,iitst);
ptr = p(:,iitr);
```

```
ttr = t(:,iitr);

%   DEFININDO A REDE
%   ================

% Cria a rede feedforward com uma camada de entrada, uma cama interna e uma
de saida.
% Nas camadas de entrada e interna utilizamos a funçao de transferencia TANSIG,
e na de saida
% a funçao PURELIN.
% Utilizamos a funçao de treinamento Levenberg-Marquardt - TRAINLM
% O comando NEWFF tambem inicializa os pesos e bias da rede.

net = newff(minmax(ptr),[3 40 1],{'tansig' 'tansig' 'purelin'},'trainlm');

%   TREINANDO A REDE
%   ================

% Alguns parametros devem ser setados: Show configura como devem ser mostrados
os resultados.
% O comando abaixo configura para mostrar os resultados a cada 5 iteraçoes.

net.trainParam.show = 5;

%   A rede sera treinada. Por favor aguarde...

% No treinamento utilizamos early stopping, por isso passamos os dados de
validaçao.
% Tambem computamos os erros entao passamos os dados de teste.

[net,tr]=train(net,ptr,ttr,[],[],validation,testing);
```

```
pause % Pressione uma tecla para continuar...

% TESTANDO A REDE
% ====================

% Plota os erros de treinamento, validaçao e teste.
plot(tr.epoch,tr.perf,'r',tr.epoch,tr.vperf,':g',tr.epoch,tr.tperf,'-.b')
legend('Treinamento','Validaçao','Teste',-1);
ylabel('Erro Quadratico')

pause % Pressione uma tecla para continuar...

% Simula a rede treinada com os todos os dados.

a = sim(net,p);

% RESULTADOS
% ================

% Agora mostramos uma analise de regressao entre os as saidas obtidas na rede
% e as saidas esperadas.

[m,b,r] = postreg(a,t);

pause % Pressione uma tecla para continuar...

% TESTANDO COM A BASE DE TESTES
% ==============================

% Testa a rede com os ultimos 30 dados
```

```
forecast = sim(net,p_test);

% Plota um grafico com as saidas esperadas e as obtidas na rede.


plot(p_index, forecast,'b', p_index, t_test,'g');
legend('Previsao','Esperadas',-1);
ylabel('PETR4');

pause % Pressione uma tecla para continuar...

% Plota um grafico com a variacao das saidas

plot(p_index, forecast - t_test,'r');
legend('Variaçao',-1);


% Analisa a qualidade dos resultados atraves de Root Mean Square, Mean Absolute
error,
% Mean Absolute Percent Error e Theil Inequality Coefficient.

rms = rms(forecast,t_test);
mae = mae(forecast,t_test);
mape = mape(forecast,t_test);
tic = tic(forecast,t_test);
rms
mae
mape
tic

echo off
```

5.3.5 Saída do Treinamento e Simulação da Rede Neural.

Essa é a saída obtida ao executar a rotina de treinamento e simulação descrita no item anterior.

```
%  O arquivo petro_all contem as matrizes dos dados de entrada e target, baseados
nas series de entrada
clear;
load petro_all

% Divide os dados em dados de treinamento(metade), validaçao(um quarto) e
teste(um quarto).
% A ideia eh pegar os dados de forma bem distribuida. Para isso montamos as
series de cada tipo pegando
% dados de quatro em quatro dias.
[R,Q] = size(p);
iitst = 2:4:Q;
iival = 4:4:Q;
iitr = [1:4:Q 3:4:Q];
validation.P = p(:,iival);
validation.T = t(:,iival);
testing.P = p(:,iitst);
testing.T = t(:,iitst);
ptr = p(:,iitr);
ttr = t(:,iitr);

%  DEFININDO A REDE
%  ====================
% Cria a rede feedforward com uma camada de entrada, uma cama interna e uma
de saida.
% Nas camadas de entrada e interna utilizamos a funçao de transferencia TANSIG,
e na de saida
```

% a funçao PURELIN.

% Utilizamos a funçao de treinamento Levenberg-Marquardt - TRAINLM

% O comando NEWFF tambem inicializa os pesos e bias da rede.

net = newff(minmax(ptr),[3 40 1],{'tansig' 'tansig' 'purelin'},'trainlm');

% TREINANDO A REDE

% ==========

% Alguns parametros devem ser setados: Show configura como devem ser mostrados os resultados.

% O comando abaixo configura para mostrar os resultados a cada 5 iteraçoes.

net.trainParam.show = 5;

% A rede sera treinada. Por favor aguarde...

% No treinamento utilizamos early stopping, por isso passamos os dados de validaçao.

% Tambem computamos os erros entao passamos os dados de teste.

[net,tr]=train(net,ptr,ttr,[],[],validation,testing);

TRAINLM, Epoch 0/100, MSE 2108.79/0, Gradient 2.09068e+006/1e-010

TRAINLM, Epoch 5/100, MSE 133.808/0, Gradient 99525.4/1e-010

TRAINLM, Epoch 10/100, MSE 76.0216/0, Gradient 138605/1e-010

TRAINLM, Epoch 15/100, MSE 61.1145/0, Gradient 31783.1/1e-010

TRAINLM, Epoch 20/100, MSE 6.44172/0, Gradient 68143.8/1e-010

TRAINLM, Epoch 25/100, MSE 1.05267/0, Gradient 61326.6/1e-010

TRAINLM, Epoch 30/100, MSE 0.916914/0, Gradient 5549.03/1e-010

TRAINLM, Epoch 35/100, MSE 0.864604/0, Gradient 2757.83/1e-010

TRAINLM, Epoch 40/100, MSE 0.849748/0, Gradient 4848.78/1e-010

TRAINLM, Epoch 45/100, MSE 0.833406/0, Gradient 667.997/1e-010

TRAINLM, Epoch 50/100, MSE 0.824441/0, Gradient 357.665/1e-010

TRAINLM, Epoch 55/100, MSE 0.824146/0, Gradient 22.4364/1e-010

TRAINLM, Epoch 60/100, MSE 0.824088/0, Gradient 12.2686/1e-010

TRAINLM, Epoch 65/100, MSE 0.821927/0, Gradient 6.83114/1e-010

TRAINLM, Epoch 69/100, MSE 0.821918/0, Gradient 32.6254/1e-010

```
TRAINLM, Validation stop.

pause % Pressione uma tecla para continuar...

%  TESTANDO A REDE
%  ====================
% Plota os erros de treinamento, validaçao e teste.
plot(tr.epoch,tr.perf,'r',tr.epoch,tr.vperf,':g',tr.epoch,tr.tperf,'-.b')
legend('Treinamento','Validaçao','Teste',-1);
ylabel('Erro Quadratico')

pause % Pressione uma tecla para continuar...

% Simula a rede treinada com os todos os dados.
a = sim(net,p);

%  RESULTADOS
%  ================
%  Agora mostramos uma analise de regressao entre os as saidas obtidas na rede
%  e as saidas esperadas.
[m,b,r] = postreg(a,t);

pause % Pressione uma tecla para continuar...

%  TESTANDO COM A BASE DE TESTES
%  ============================
% Testa a rede com os ultimos 30 dados
forecast = sim(net,p_test);

% Plota um grafico com as saidas esperadas e as obtidas na rede.
plot(p_index, forecast,'b', p_index, t_test,'g');
legend('Previsao','Esperadas',-1);
```

```
ylabel('PETR4');

pause % Pressione uma tecla para continuar...

% Plota um grafico com a variacao das saidas
plot(p_index, forecast - t_test,'r');
legend('Variaçao',-1);

% Analisa a qualidade dos resultados atraves de Root Mean Square, Mean Absolute
error,
% Mean Absolute Percent Error e Theil Inequality Coefficient.
rms = rms(forecast,t_test);
mae = mae(forecast,t_test);
mape = mape(forecast,t_test);
tic = tic(forecast,t_test);
rms

rms =   2.0962

mae

mae = 1.6178

mape

mape =  -0.2357

tic

tic =   0.0144

echo off
```

6 CONCLUSÃO

Prever séries econômico-financeiras requer modelos muitos bem delineados, de forma que os resultados representem, ao menos, um esboço do futuro. Sem dúvida, a tarefa mais difícil consistiu na determinação do melhor modelo capaz de realizar tais previsões. Primeiramente, os modelos econométricos AR e GARCH se mostraram bastante razoáveis na previsão da série temporal de retornos da ação preferencial da Petrobras. A partir da utilização de dados de entrada bem selecionados conseguiu-se testar e analisar a resposta dos modelos Esses resultados serviram como suporte ao estudo das redes neurais, validando a resposta obtida por estas últimas.

O estudo das redes neurais envolveu fases bem distintas ao longo do projeto.

A primeira delas foi o estudo detalhado das redes neurais e as características de modelagem e treinamento, através de conteúdo teórico. Essa etapa foi muito importante para que as etapas seguintes pudessem fluir de forma sinérgica e que os resultados fossem mais bem analisados.

A próxima etapa foi o estudo das redes neurais no Matlab. Uma vez desenhada a rede no Matlab, ela funciona como uma complexa "caixa preta", ao contrário de modelos econométricos, que necessitam de conhecimentos de um especialista, e onde se parte de hipóteses sobre o comportamento da série temporal.

A seguir esse conhecimento foi utilizado no treinamento das redes e na discussão dos resultados obtidos. Essa etapa foi no começo preocupante, devido à dificuldade de determinar uma arquitetura inicial de teste. Quantas camadas seriam usadas? Quantos neurônios seriam utilizados nas camadas intermediárias?

A partir do momento em que os resultados começaram a parecer razoáveis, os modelos foram mais facilmente implementados e testados, pois tinha-se agora uma base de comparação.

Quanto ao treinamento da rede tentou-se utilizar o máximo de funcionalidades disponíveis no Matlab. Esse fato acabou alavancando o estudo das redes no software.

O conceito de generalização foi utilizado através da técnica do early stopping e através da regularização com o algoritmo de treinamento Bayesian Regularization. Por último foi utilizado o cálculo de erros estatísticos e da rotina postreg para análise da resposta da rede.

Os gráficos de convergência e erros durante o processo de treinamento possibilitaram a verificação das características de cada algoritmo de treinamento e da influência da arquitetura da rede. Os gráficos de comparação entre as saídas simuladas e as esperadas possibilitaram uma análise qualitativa do modelo. A partir do teste em vários cenários pré-estabelecidos, chegou-se ao modelo de melhor desempenho para o escopo da Petrobrás proposto no projeto. Esse modelo é composto de uma camada de entrada, uma camada intermediária, com 40 neurônios e uma camada de saída. Foi treinado pelo algoritmo de Levenberg-Marquardt e apresentou a melhor regressão entre a resposta da rede e as saídas esperadas. Juntamente com um modelo dotado de mesma arquitetura e treinado pelo algoritmo Bayesian Regularization, essas duas redes neurais apresentaram os menores erros estatísticos entre as previsões e os valores reais.

A etapa final foi a comparação do modelo de rede neural com os modelos econométricos, que serviram como base de validação para o desempenho da rede. Essa comparação foi feita baseando-se no gráfico contendo as previsões dos modelos e a serie de valores esperados. Observou-se que os dois tipos de modelos proporcionaram resultados muito próximos, e os erros estatísticos foram pequenos nos dois casos, como pode ser verificado na Tabela 5.5 - Erro estatístico do modelo de RN e dos modelos econométricos. Isso comprova a suposição de que as Redes Neurais são capazes de prever séries econômico-financeiras, desde que treinadas corretamente a partir de um conjunto completo de dados. Sua característica não linear e capacidade de generalização são ideais na modelagem de séries financeiras, no entanto a falta de controle do que ocorre na rede prejudica a estimação de um

modelo, ficando a sensação de que existem inúmeros modelos que funcionariam da mesma forma. De qualquer maneira, a grande vantagem de uma rede neural é de ela aprender por si só, enquanto que para desenvolver o modelo econométrico teve-se de aplicar diversos conceitos avançados de econometria, na modelagem da RN a própria rede "compreende" o comportamento da série.

De acordo com os resultados da simulação de decisões de compra e venda verificou-se a eficiência dos modelos quando aplicados em um ambiente real de mercado de ações. Verificou-se que os modelos econométricos apresentaram uma melhor taxa de acerto do que as redes neurais. Entretanto deve-se atentar ao fato de que a série estudada apresenta muito componentes aleatórios, e que a aplicação dos modelos estudados no cenário real é muito discutível.

O escopo abordado por este projeto fica apenas na superfície do real potencial do que as RNs podem oferecer, no momento em que uma maior sensibilidade das mudanças na arquitetura de uma RN ficar clara ao modelador, maior será a capacidade de se criar RNs específicas ao problema a se solucionar, podendo-se cruzar conhecimentos de outras disciplinas na modelagem da rede.

Lista de Referências

BERNSTEIN, P. L. **Desafio aos Deuses – A Fascinante História do Risco**. Rio de Janeiro: Elsevier, 1997

BELTRATTI , A., MARGARITA, S., TERNA, P. **Neural Networks for Economic and Financial Modelling**. Thomson , 1996.

BOX, G. E. P., JENKINS, G.M. **Time series analysis: forecasting and control.** San Francisco: Holden Day, 1976.

BOLLERSLEV, T., ENGLE R. F., NELSON, D. B. **ARCH models, Handbook of Econometrics,** 1994.

ENDERS, W. **Applied Econometric Time Series,** Wiley, 2003.

MCNELIS, P. D. **Financial Forecasting with Neural Networks,** Georgetown University, 2002.

OLIVEIRA, M. A. **Previsão de sucessões cronológicas econômico-financeiras por meio de redes neurais artificiais recorrentes de tempo real e de processos ARMA-GARCH: um estudo comparativo quanto à eficiência de previsão,** FEA-USP, 2003

RUUD, P. **An introduction to classical econometric theory,** Oxford University Press, Oxford 2000.

Eviews User Guide, Quantitative Software , 2002.

Freeman, James A., Skapura, David M. **Algorithms, Applications and Programming Techniques,** Addison-Wesley Publishing Company, 1991

Kaastra, Iebeling, Boyd, Milton. **Designing a Neural Network for Forecasting Financial and Economic Time Series,** Neurcomputing, 1996

Demuth, H., Beale, M., **Neural Network Toolbox User's Guide**, Ver. 4

MacKay, D. J. C., **"Bayesian interpolation,"** ***Neural Computation***, vol. 4, no. 3, 1992.