Departamento de Engenharia Elétrica - SEL - EESC - USP

Trabalho de Conclusão de Curso

Comparação Entre Redes Neurais e Técnicas Clássicas na Previsão a Curtíssimo Prazo de Demanda de Potência Elétrica Ativa

Aluno: Luiz Henrique Barchi Bertolucci
Orientador: Prof. Dr. Ivan Nunes da Silva

10 de dezembro de 2008

Aos meus pais.

Resumo

Uma previsão adequada da demanda de potência elétrica é de fundamental importância para o planejamento e operação do sistema de distribuição de eletricidade. Perante este contexto, o objetivo deste trabalho foi estudar técnicas de previsão e suas aplicações na estimativa de demanda de potência elétrica ativa a curtíssimo prazo. Primeiramente, diferentes topologias das redes neurais denominadas *focused time delay neural network* (FTDNN) e *nonlinear autoregressive network with exogenous inputs* (NARX) foram avaliadas, e, as melhores configurações selecionadas. Em seguida, as topologias escolhidas foram comparadas com os métodos *autoregressivo de média móvel integrado* (ARIMA) e *suavização exponencial*, denominados clássicos. Utilizou-se os dados de cinco dias de observação das demadas de potência ativa e reativa da Subestação Andorinha (CPFL). Todas as técnicas foram avaliadas através do critério MAPE para previsões de 1 e 5 passos adiante. Conclui-se que as redes neurais apresentam os menores erros, no entanto, o tempo necessário para que estas forneçam o resultado pode implicar em um preterimento destas em relação aos métodos clássicos.

Palavras-chave: Previsão, redes neurais, ARIMA, suavização exponencial, demanda de potência elétrica

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Uma previsão adequada do consumo de energia elétrica por parte de uma empresa distribuidora é de fundamental importância tanto para planejamento quanto para operação do sistema de distribuição. Índices como confiabilidade de suprimento e até mesmo uma operação mais racional e econômica estão ligados a esta previsão. Atualmente, diversos métodos se propõem a realizar esta tarefa a partir dos dados históricos de demanda de potência, denominados *séries temporais*.

Uma série temporal é um conjunto de observações tomadas seqüencialmente no tempo. Para as aplicações pretendidas se deve exigir que as observações estejam dispostas em intervalos de tempo discretos e fixos. Em geral, as séries apresentam como característica fundamental a dependência entre observações adjacentes [4]. A análise de séries temporais tem por objetivo utilizar técnicas para analisar esta dependência e desenvolver modelos dinâmicos que representem a evolução dos dados.

Como o interesse do estudo é previsão, deve-se construir modelos que, a partir das observações disponíveis até o instante t, possam estimar os valores futuros da série em um dado tempo $t + h$. Desta forma, a quantidade y_t de potência demandada no instante t, e os valores y_{t-1}, y_{t-2}, y_{t-3},... , podem ser utilizados para a previsão da demanda em $h = 1, 2, 3, ...$ intervalos de tempo à frente. Denotaremos por $\hat{y}_t(h)$ a previsão feita em t da demanda y_{t+h} que será observada no tempo $t + h$. A função $\hat{y}_t(h)$, que nos oferece a previsão para todo tempo futuro, será chamada *função de previsão* na origem t. Deve-se obtê-la garantindo que os desvios $|y_{t+h} - \hat{y}_t(h)|$ sejam os menores possíveis para cada h.

Há alguns anos os modelos baseados em técnicas de estatística clássica eram considerados como ferramenta principal disponível para atingir tal meta. Porém, atualmente, as redes neurais têm apresentado excelentes resuldados surgindo como candidatas para a realização desta tarefa.

1.1 Redes Neurais

Redes Neurais são sistemas de processamento em paralelo inspirado no comportamento do sistema nervoso humano. McCulloch e Pitts (1943) foram os primeiros a criarem o neurônio computacional baseado no modelo do neurônio biológico. Este primeiro modelo caracteriza-se por ser um dispositivo binário, com limiares fixos, que se conecta a outros neurônios por sinapses [41]. As principais partes constituintes do neurônio, ou seja, o *corpo celular* com seu núcleo, os dendritos que alimentam o corpo celular com os sinais externos e o axônio que carrega os sinais para fora da célula, são apresentadas pela figura 1.1.

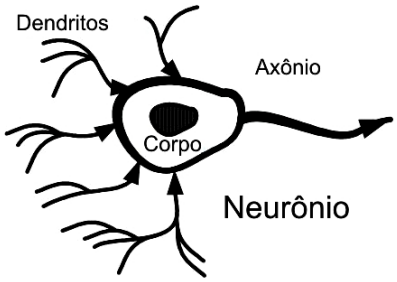

Figura 1.1: *Representação do neurônio.*

A lista dos primeiros contribuintes deste campo é vasta, com destaque para o pesquisador Rosenblatt (1958) [15], quem estendeu a idéia do neurônio computacional para o *perceptron* (figura 1.2), como um elemento auto-organizável capaz de aprender por *feedback*. Widrow e Hoff (1960) [3] criaram os dispositivos conhecidos como ADALINE (*Adaptive Linear Element*) e MADALINE (*multiple* ADALINE) que, utilizando-se do algoritmo conhecido como regra delta, são treinados a partir de vetores padrões apresentados à rede. Em 1969, Minsky e Papert [37] (1969), mostraram que os sistemas *perceptron* multicamadas (MLP, figura 1.3) apresentavam limitações de apredizado similares aos de camada única. Rumelhart e McClelland (1986) [11] provaram que a visão de Minsky e Papert estava errada, mostrando que o sistema MLP apresenta grande sucesso na tarefa de discriminação não linear, sendo também capaz de aprender padrões complexos através do método **backpropagation** (retropropagação).

Após um periodo de estagnação, o interesse da pesquisa na área passou a ser o desenvolvimento de arquiteturas de redes modificadas, tais como, **self-organizing networks** [12](Mari e Maginu, 1988), **resonating neural networks** [35] (Grossberg, 1988), **feedforward network** (Werbos, 1974) [26], **associative memory networks** (Kohonen, 1989) [38], **counterpropagation networks** (Hecht-Nielsen, 1987a), [30] **recurrent net-**

works(Elman, 1990) [22], **radial basis function networks** (Broomhead and Lowe, 1988)[7], **probabilistic networks** (Specht, 1988) [9], etc.

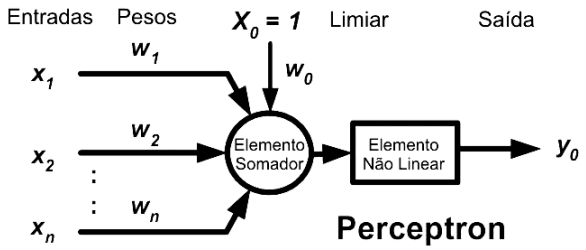

Figura 1.2: *Perceptron, o modelo computacional do neurônio.*

Desde o princípio, as redes neurais têm se mostrado como poderosas ferramentas para análise de sinais, extração de características, classificação de dados, reconhecimento de padrões, etc. Devido à sua capacidade de aprendizado e generalização, as redes têm sido largamente utilizadas por pesquisadores como ferramenta para o processamento de dados experimentais.

Suas principais aplicações incluem: *(i)* Mapeamento da relação entre os valores passados e futuros de uma série temporal (previsão); *(ii)* Captura da relação funcional essencial entre dados quando esta não é conhecida *a priori* ou é de dificil descrição matemática;*(iii)* Aproximação de funções para qualquer grau de acurácia; e *(iv)* Aprendizado e generalização a partir de exemplos.

Hu (1964) [25] foi o primeiro a demonstrar, em um exemplo de previsão de tempo, a capacidade preditiva de uma rede neural. Werbos (1974) [26] utilizou esta ferramenta para prever o comportamento de outra série temporal. Entretanto, os modelos até então pouco adequados, não motivaram avanços significativos na área. Este período de estagnação chegou ao fim quando Rumelhart e colaboradores (1986)[34] reformularam o algoritmo de treinamento *backpropagation*.

1.1.1 Arquitetura Básica de uma Rede

Inspirado nos trabalhos de McCulloch e Pitts (1943), Rosenblatt (1958) criou o *perceptron* (figura 1.2), onde a parte central do dispositivo contém um elemento somador e um elemento não linear, também conhecido como **função de ativação**, os múltiplos sinais de entrada x_i são conectados via ponderação w_i à parte central do elemento, o sinal de saída observado em y_0 é dado pela soma das entradas x_i ponderadas pelos pesos w_i aplicada na função de ativação, a entrada adicional w_0 é chamada limiar de ativação do neurônio.

Nestas condições, o sinal de saída é dado por

$$y_0 = f\Big(\sum_{i=1}^{n} w_i x_i + w_0\Big) \tag{1.1}$$

.

Originalmente, Rosenblatt propôs uma função de ativação tipo degrau unitário. Desta forma, a função é ativada, isto é, produz sinal de saída se, e somente se, a condição $w^T x + w_0 \geq 0$ é observada.

O *perceptron* é capaz de "aprender", isto é, os pesos de suas interconexões são automaticamente ajustados de acordo com o conjunto de dados apresentados. Para realizar esta tarefa, Widrow e Hoff (1960) propuseram originalmente o uso da *regra delta*, que caracteriza-se por ser um algoritmo de aprendizado recursivo baseado no gradiente do erro de saída (também chamado α**-LMC Algorithm**). Embora simples, o aprendizado através da regra delta tem, na maioria dos casos, demonstrado uma alta eficiência e excelente taxa de convergência.

Um simples perceptron não é capaz de solucionar problemas complexos, no entanto, para estes casos, uma rede mais sofisticada denominada **perceptron multicamadas** (MLP) foi construída. Esta rede apresenta, além da camada de entrada e camada de saída, as **camadas escondidas** (*hidden layers*) que são inseridas entre as primeiras para formar uma rede em *cascata*, como nos mostra a figura 1.3. O termo "camada escondida" significa que estas não podem ser acessadas diretamente, mas apenas através das camadas de entrada (*input layer*) e/ou saída (*output layer*). Na prática, apenas uma única camada escondida é capaz de extender a capacidade computacional da rede e prover boas solução para a maioria dos problemas práticos [8].

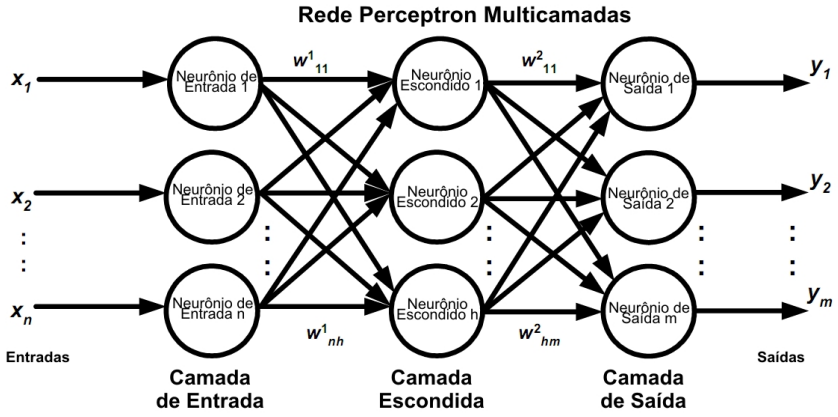

Figura 1.3: Arquitetura MLP com uma camada escondida.

1.1.2 O métodos *backpropagation*

Considerado como uma generalização da regra delta, foi inicialmente desenvolvido por Paul Werbos em 1974 mas só ficou conhecido após seu "redescobrimento" por Parker em 1982. O algoritmo, entretanto, tornou-se popular somente quando reformulado por Rumelhart e colaboradores (1986), e passa a ser intensivamente utilizado para treinar as redes perceptron multicamadas. Atualmente o backpropagation é utilizado, em uma forma modificada, para treinar também outros tipos de rede.

Neste processo de treinamento, o algoritmo procura pelo mínino da função de erro aplicada no espaço dos *pesos sinápticos* através do método *gradiente descendente*. A combinação de pesos que minimiza a função de erro é considerada a solução do processo de aprendizagem. Como o método requer o cálculo do gradiente da função de erro, deve-se garantir que esta seja contínua e diferenciável. Uma forma de obter esta garantia é utilizar funções de ativação que apresentam estas características, como por exemplo, funções do tipo *sigmóide* definida por

$$s_c(x) = \frac{1}{1 + e^{-cx}} \tag{1.2}$$

e apresentada pela figura 1.4.

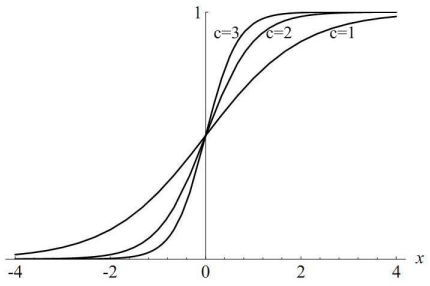

Figura 1.4: *Três funções tipo sigmóide (c=1, c=2 e c=3).*

Adotando sem perda de generalidade $c = 1$, encontra-se a derivada desta função em relação a x,

$$\frac{d}{dx} s_1(x) = \frac{e^{-x}}{(1 + e^{-x})^2} = s(x)(1 - s(x))$$

e nota-se que $s_c(x)$ é claramente, contínua e diferenciável para todo $x \in \mathbb{R}$

O maior problema na utilização destas funções é que, em determinadas circunstâncias, mínimos locais aparecem na função de erro, o que não ocorreria se fossem utilizadas funções do tipo degrau. A figura 1.5 mostra um exemplo de função de erro que apresenta um mínimo local referente a um alto valor de erro se comparado ao mínimo global. O gráfico foi obtido a

partir de uma unidade simples com dois pesos, limiar constante, e quatro conjuntos de entrada e saída apresentados durante o treinamento. Existe um vale na função de erro e, se o gradiente partir desta região, jamais o algoritmo convergirá para o mínimo global.

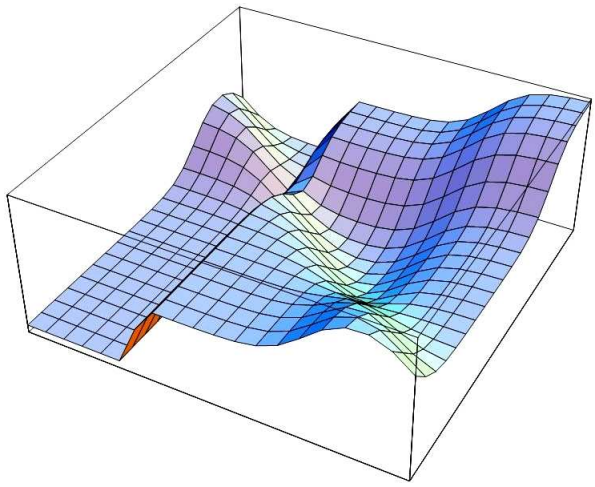

Figura 1.5: *Exemplo de função de erro.*

Em uma rede tipo *feedforward* (figura 1.6) a informação move-se em apenas uma direção, não havendo retrocesso como no caso das redes recorrentes. Cada unidade da rede é capaz de avaliar somente os valores de suas próprias entradas. A rede neural representa uma cadeia de funções compostas que transformam um conjunto de entradas em um vetor de saída. Desta forma, a rede configura-se como uma função composta, denominada *função de rede* que leva uma dada entrada a um ponto no conjunto espacial da saída. O aprendizado consiste em encontrar a combinação ótima de pesos para que a função de rede φ se aproxime o máximo possível de uma dada função f. Entretanto, nós não temos a função f explicitamente mas apenas de forma implícita através de alguns exemplos.

Considere uma rede *feedforward* com n entradas, m unidades de saídas e um número qualquer de camadas escondidas, tal qual nos mostra a figura 1.7. Considere também um conjunto de treinamento $\{(\mathbf{x}_1, \mathbf{t}_1), \ldots, (\mathbf{x}_p, \mathbf{t}_p)\}$ que consiste de p pares ordenados de vetores n e m dimensional respectivamente. Adotaremos as funções de cada nó como contínuas e diferenciáveis. Os pesos iniciais da rede são reais e dados aleatoriamente. Quando uma entrada padrão $\mathbf{x}_i$ pertencente ao conjunto de treinamento é apresentada à rede, esta produz uma saída $\mathbf{o}_i$ diferente, em geral, de $\mathbf{t}_i$. O que queremos é fazer com que $\mathbf{o}_i$ seja idêntico a $\mathbf{t}_i$ para $i = 1,\ldots,p$, utilizando um algoritmo de aprendizagem. Mais precisamente, queremos minimizar a função de erro da

rede, definida por

$$E = \frac{1}{2}\sum_{i=1}^{p} \parallel \mathbf{o}_i - \mathbf{t}_i \parallel^2$$

O algoritmo de *backpropagation* é utilizado para encontrar um mínimo local desta função. O primeiro passo deste processo é estender a rede para que sua nova saída apresente de forma direta a função de erro, como apresentado pela figura 1.7. Agora, cada uma das j unidades de saída é conectada a um nó que calcula a função $1/2(o_{ij} - t_{ij})^2$, onde o_{ij} e t_{ij} denotam o $j-$ésimo componente do vetor de saída $\mathbf{o}_i$ e do vetor alvo $\mathbf{t}_i$. O resultado dos m nós são coletados e somados para gerar a saída E_i. Somando todos os erros quadráticos $E_1 + \ldots + E_p$ encontramos a função de erro E.

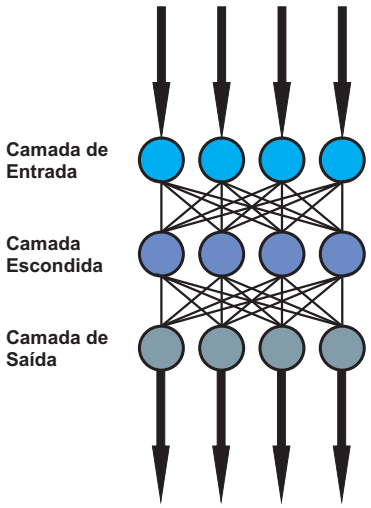

Figura 1.6: *Esquema geral de uma rede "feedforward".*

Como E é uma composição das funções de cada nó, este é por sua vez uma função contínua e diferenciável em relação aos l pesos da rede, $w_1, w_2, \ldots, w_l$. Para minimizar E deve-se calcular seu gradiente através da equação

$$\nabla E = \left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_l}\right)$$

e atualizar cada peso utilizando-se o incremento

$$\Delta w_i = -\gamma \frac{\partial E}{\partial w_i}$$

onde i=1,...,l e γ representa a taxa de aprendizado, isto é, um parâmetro que define o tamanho do passo que será dado a cada iteração na direção oposta ao crescimento do gradiente. Neste sentido, espera-se encontrar um mínimo local da função de erro, quando $\nabla E = 0$. Em geral, assume-se que

a taxa de aprendizado é fixa e uniforme para todos os pesos. Para assegurar a convergência do algoritmo, γ deve ser o menor possível. Entretanto, taxas muito pequenas acarretam baixa velocidade de convergência. Por outro lado, valores elevados podem resultar em um processo de aprendizado instável. Atualmente, alguns algoritmos calculam a taxa ótima de aprendizado em cada iteração, isto faz com que o método convirja rápidamente para a solução enquanto mantém o processo de aprendizado estável.

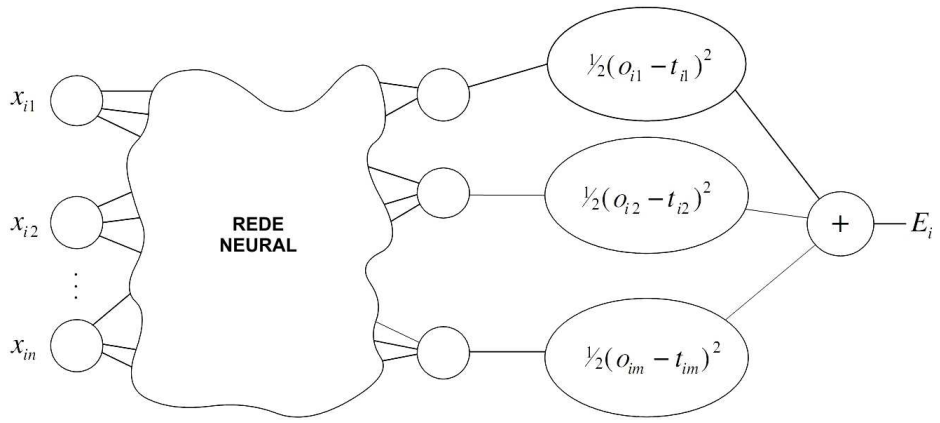

Figura 1.7: *Adaptação da rede incluindo função de erro.*

1.2 Métodos Clássicos

1.2.1 Suavização Exponencial

Embora este método tenha sido desenvolvido a partir da década de 50, algoritmos capazes de selecionar o melhor modelo para uma dada série foram desenvolvidos recentemente [33], [36] e [32]

Os métodos de suavização exponencial foram originalmente classificados por Pegels' (1969)[6] e posteriormente estendidos por [14], modificados por [36], e estendidos novamente por [21], apresentando um total de 15 métodos como nos mostra a Tabela 1.2.

Estão apresentadas, para ilustrar, as equações correspondentes ao método (A,A), também conhecido como *Holt-Winters aditivo*

Nível: $l_t = \alpha(y_t - s_{t-m}) + (1-\alpha)(l_{t-1} + b_{t-1})$
Tendência: $b_t = \beta^*(l_t - l_{t-1}) + (1-\beta^*)b_{t-1}$
Sazonalidade: $s_t = \gamma(y_t - l_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m}$
Previsão: $\hat{y}_t(h) = l_t + b_t h + s_{t-m+h_m^+}$

onde m é o tamanho da sazonalidade, l_t representa o nível da série, b_t a

tendência ou crescimento, s_t o componente sazonal, $\hat{y}_t(h)$ é a previsão para h períodos à frente, e $h_m^+ = [(h-1) \text{mod } m]+1$. Como este é um método recursivo, para calcular $\hat{y}_t(h)$ precisa-se dos estados iniciais l_0, b_0 e $s_{1-m}, \ldots, s_0$ e dos parâmetros α, β^* e γ. A Tabela 1.2 mostra as demais equações.

Tabela 1.1: *Classificação dos modelos de Suavização Exponencial.*

Componente de Tendência	**Componente Sazonal**		
	N (Nenhum)	**A** (Aditivo)	**M** (Multiplicativo)
N (Nenhum)	**N,N**	**N,A**	**N,M**
A (Aditivo)	**A,N**	**A,A**	**A,M**
$\mathbf{A_d}$ (Aditivo amortecido)	$\mathbf{A_d}$**,N**	$\mathbf{A_d}$**,M**	$\mathbf{A_d}$**,M**
M (Multiplicativo)	**M,N**	**M,A**	**M,M**
$\mathbf{M_d}$ (Multiplicativo amortecido)	$\mathbf{M_d}$**,N**	$\mathbf{M_d}$**,A**	$\mathbf{M_d}$**,M**

Para cada método apresentado pela Tabela 1.1, são possíveis duas modelagens no espaço de estados, uma correspondendo ao erro classificado como aditivo e a outra ao erro multiplicativo. Desta forma, o método compreende ao todo 30 modelos candidatos à descrever uma série temporal. A determinação do modelo final inclui o cálculo dos parâmetros e, em seguida, a determinação da quantidade dos mesmos que otimizam a previsão.

Nesta primeira etapa, precisa-se conhecer os valores dos parâmetros θ=(α,β,γ,ϕ) e de $\mathbf{x}_0 = (l_0, b_0, s_0, s_{-1}, \ldots, s_{-m+1})'$. Esta tarefa é realizada utilizando-se o estimador de *máximo verossimilhança* definido por

$$L^*(\theta, \mathbf{x}_0) = n \; log\left(\sum_{t=1}^{n} \varepsilon_t^2 \right) + 2 \sum_{t=1}^{n} log|r(\mathbf{x}_{t-1})| \tag{1.3}$$

condicionado aos parâmetros θ e $\mathbf{x}_0$, onde n é o número de observações. Os parâmetros são então estimados para todos os modelos enquanto se procura minimizar L^*.

Para definir o número de parâmetros, o algoritmo utiliza o critério de informação Akaike (AIC) [17] , definido por

$$AIC = L^*(\hat{\theta}, \hat{\mathbf{x}}_0) + 2q \tag{1.4}$$

onde q é o número de parâmetros em θ adicionado à quantidade de estados em $\mathbf{x}_0$, e $\hat{\theta}$ e $\hat{\mathbf{x}}_0$ denotam os valores estimados na etapa anterior. O modelo que apresentar o menor AIC dentre todas as combinações citadas é selecionado. Detalhes do funcionamento do algoritmo são encontrados em [13].

Tabela 1.2: *Equações para o cálculo dos parâmetros recursivos e previsão*

Tendência	Sazonalidade		
	N	A	M
N	$\ell_t = \alpha y_t + (1-\alpha)\ell_{t-1}$	$\ell_t = \alpha(y_t - s_{t-m}) + (1-\alpha)\ell_{t-1}$	$\ell_t = \alpha(y_t/s_{t-m}) + (1-\alpha)\ell_{t-1}$
		$s_t = \gamma(y_t - \ell_{t-1}) + (1-\gamma)s_{t-m}$	$s_t = \gamma(y_t/\ell_{t-1}) + (1-\gamma)s_{t-m}$
	$\hat{y}_{t+h\|t} = \ell_t$	$\hat{y}_{t+h\|t} = \ell_t + s_{t-m+h_m^+}$	$\hat{y}_{t+h\|t} = \ell_t s_{t-m+h_m^+}$
A	$\ell_t = \alpha y_t + (1-\alpha)(\ell_{t-1} + b_{t-1})$	$\ell_t = \alpha(y_t - s_{t-m}) + (1-\alpha)(\ell_{t-1} + b_{t-1})$	$\ell_t = \alpha(y_t/s_{t-m}) + (1-\alpha)(\ell_{t-1} + b_{t-1})$
	$b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1}$	$b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1}$	$b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1}$
		$s_t = \gamma(y_t - \ell_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m}$	$s_t = \gamma(y_t/(\ell_{t-1} - b_{t-1})) + (1-\gamma)s_{t-m}$
	$\hat{y}_{t+h\|t} = \ell_t + hb_t$	$\hat{y}_{t+h\|t} = \ell_t + hb_t + s_{t-m+h_m^+}$	$\hat{y}_{t+h\|t} = (\ell_t + hb_t)s_{t-m+h_m^+}$
$\mathrm{A_d}$	$\ell_t = \alpha y_t + (1-\alpha)(\ell_{t-1} + \phi b_{t-1})$	$\ell_t = \alpha(y_t - s_{t-m}) + (1-\alpha)(\ell_{t-1} + \phi b_{t-1})$	$\ell_t = \alpha(y_t/s_{t-m}) + (1-\alpha)(\ell_{t-1} + \phi b_{t-1})$
	$b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)\phi b_{t-1}$	$b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)\phi b_{t-1}$	$b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)\phi b_{t-1}$
		$s_t = \gamma(y_t - \ell_{t-1} - \phi b_{t-1}) + (1-\gamma)s_{t-m}$	$s_t = \gamma(y_t/(\ell_{t-1} - \phi b_{t-1})) + (1-\gamma)s_{t-m}$
	$\hat{y}_{t+h\|t} = \ell_t + \phi_h b_t$	$\hat{y}_{t+h\|t} = \ell_t + \phi_h b_t + s_{t-m+h_m^+}$	$\hat{y}_{t+h\|t} = (\ell_t + \phi_h b_t)s_{t-m+h_m^+}$
M	$\ell_t = \alpha y_t + (1-\alpha)\ell_{t-1} b_{t-1}$	$\ell_t = \alpha(y_t - s_{t-m}) + (1-\alpha)\ell_{t-1} b_{t-1}$	$\ell_t = \alpha(y_t/s_{t-m}) + (1-\alpha)\ell_{t-1} b_{t-1}$
	$b_t = \beta^*(\ell_t/\ell_{t-1}) + (1-\beta^*)b_{t-1}$	$b_t = \beta^*(\ell_t/\ell_{t-1}) + (1-\beta^*)b_{t-1}$	$b_t = \beta^*(\ell_t/\ell_{t-1}) + (1-\beta^*)b_{t-1}$
		$s_t = \gamma(y_t - \ell_{t-1} b_{t-1}) + (1-\gamma)s_{t-m}$	$s_t = \gamma(y_t/(\ell_{t-1} b_{t-1})) + (1-\gamma)s_{t-m}$
	$\hat{y}_{t+h\|t} = \ell_t b_t^h$	$\hat{y}_{t+h\|t} = \ell_t b_t^h + s_{t-m+h_m^+}$	$\hat{y}_{t+h\|t} = \ell_t b_t^h s_{t-m+h_m^+}$
$\mathrm{M_d}$	$\ell_t = \alpha y_t + (1-\alpha)\ell_{t-1} b_{t-1}^{\phi}$	$\ell_t = \alpha(y_t - s_{t-m}) + (1-\alpha)\ell_{t-1} b_{t-1}^{\phi}$	$\ell_t = \alpha(y_t/s_{t-m}) + (1-\alpha)\ell_{t-1} b_{t-1}^{\phi}$
	$b_t = \beta^*(\ell_t/\ell_{t-1}) + (1-\beta^*)b_{t-1}^{\phi}$	$b_t = \beta^*(\ell_t/\ell_{t-1}) + (1-\beta^*)b_{t-1}^{\phi}$	$b_t = \beta^*(\ell_t/\ell_{t-1}) + (1-\beta^*)b_{t-1}^{\phi}$
		$s_t = \gamma(y_t - \ell_{t-1} b_{t-1}^{\phi}) + (1-\gamma)s_{t-m}$	$s_t = \gamma(y_t/(\ell_{t-1} b_{t-1}^{\phi})) + (1-\gamma)s_{t-m}$
	$\hat{y}_{t+h\|t} = \ell_t b_t^{\phi_h}$	$\hat{y}_{t+h\|t} = \ell_t b_t^{\phi_h} + s_{t-m+h_m^+}$	$\hat{y}_{t+h\|t} = \ell_t b_t^{\phi_h} s_{t-m+h_m^+}$

1.2.2 ARIMA

Uma maneira comumente empregada para compreender o comportamento de uma série temporal, é assumir que estes são a realização de um *processos estocástico*. Assim posto, considera-se a série como os valores gerados por uma variável aleatória indexada no tempo, que se comporta de acordo com uma lei probabilística. Então, supõe-se que a seqüência de dados em estudo é uma realização particular de um processo estocástico.

Um processo estocástico é dito *estritamente estacionário* se sua distribuição de probabilidade conjunta não é afetada pela alteração da origem temporal, isto é, o processo apresenta um certo equilíbrio estatístico. Um processo apresenta *estacionariedade fraca* quando sua média e matriz de covariância independem do tempo.

O desenvolvimento e aplicação dos modelos lineares que serão apresentados são justificados quando a presença da estacionariedade fraca é observada. A partir de então se utliza a palavra "estacionária" para aludir à propriedade recém definida.

Uma série temporal gerada por processo estacionário tipo $ARMA(p, q)$ pode ser modelada pela equação

$$y_t - \phi_1 y_{t-1} - \ldots - \phi_p y_{t-p} = z_t + \theta_1 z_{t-1} + \ldots + \theta_q z_{t-q} \tag{1.5}$$

onde $\{Z_t\} \sim WN(0, \sigma^2)$, isto é, a série $\{Z_t\}$ é um ruído branco com média zero e variância σ^2.

De forma equivalente reescreve-se as equação como

$$\phi(B) y_t = \theta(B) z_t \tag{1.6}$$

onde B é o operador de atraso, isto é, $B^d y_t = y_{t-d}$, e, $\phi(.)$ e $\theta(.)$ são os polinômios

$$\phi(z) = 1 - \phi_1 z - \ldots - \phi_p z^p$$

e

$$\theta(z) = 1 + \theta_1 z + \ldots + \theta_q z^q$$

Os modelos ARIMA não sazonal e ARIMA sazonal são generalizações do modelo ARMA. No entanto, está inclusa nestes modelos uma pré-diferenciação da série com o objetivo de fazer com que os comportamentos de tendência e sazonalidades sejam removidos, ou seja, produzir uma série aproximadamente estacionária.

Um processo ARIMA(p, d, q) não sazonal é dado por:

$$\phi(B)(1 - B)^d y_t = \theta(B) z_t \tag{1.7}$$

onde $\{Z_t\}$ é um ruído branco com média zero e variância σ^2, B é o operador de atraso e $\phi(z)$ e $\theta(z)$ são polinômios de ordem p e q respectivamente.

O processo ARIMA$(p, d, q)(P, D, Q)_m$ sazonal é definido como:

$$\Phi(B^m)\phi(B)(1-B^m)^D(1-B)^d y_t = \Theta(B^m)\theta(B)z_t \tag{1.8}$$

onde $\Phi(z)$ e $\Theta(z)$ são polinômios de ordem P e Q respectivamente, e m é o tamanho da sazonalidade.

A escolha do melhor modelo *ARIMA* para representar uma série é considerada uma tarefa subjetiva e de grande dificuldade. No entanto, o pacote desenvolvido por Hyndman implementa uma série de algoritmos desenvolvidos nos últimos 25 anos capazes de automatizar este processo. Alguns destes algoritmos são encontrados em [19], [1], [23] e [20] dentre outros.

Para se construir um modelo ARIMA, a principal tarefa resume-se em definir os valores de p, q, P, Q, D e d, como também os coeficientes de cada polinômio $\Phi(.)$, $\phi(.)$, $\Theta(.)$ e $\theta(.)$. A seleção automática destes valores também baseia-se no critério de informação Akaike (AIC), definido de maneira sutilmente diferente da anterior. Para d e D conhecidos define-se

$$AIC = -2log(L) + 2(p + q + P + Q + 1) \tag{1.9}$$

que estima os valores dos parâmetros que melhor representam a série, e concomitantemente, penaliza os parâmetros de ordem elevada, na tentativa de construir modelos precisos e ao mesmo tempo simples.

Os parâmetros d e D não são escolhidos diretamente por AIC pois, em geral, este processo leva a uma sobrediferenciação da série, provocando queda na acurácia da previsão. Para resolver este impasse, outros testes, tais como *Dick-Fuller* [42], HEGY [2] e KPSS [45], são aplicados previamente. Maiores detalhes do funcionamento do algoritmo são encontrados em [13].

1.2.3 Previsão Inocente

Consiste em adotar os valores futuros da série como iguais ao valor atual, isto é, $\hat{y}_t(h) = y_t$, para todo $h > 0$. Esta técnica é utilizada para avaliar o ganho preditivo ao se adotar os métodos mais elaborados e pode ser trivialmente implementada através de uma translação da série. Como exemplo, a figura 1.8 apresenta uma previsão para 5 passos adiante.

Cabe observar que previsão inocente é um caso especial do método de suavização exponencial (N,N) quando $\alpha = 1$.

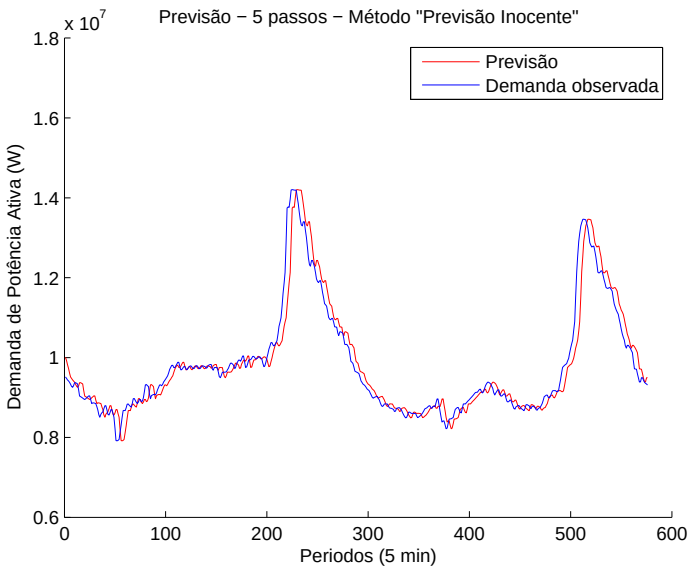

Figura 1.8: *Exemplo de previsão inocente - 5 passos.*

Capítulo 2

Objetivos

Podem ser divididos em

- **Encontrar as arquiteturas das redes FTDNN e NARX que apresentam o menor erro médio na previsão a curtíssimo prazo da máxima demanda de potência ativa**;

- **Levando em consideração o erro médio e o tempo gasto para previsão, comparar as arquiteturas previamente encontradas com as técnicas clássicas na previsão a curtíssmo prazo da máxima demanda de potência ativa de uma subestação, onde a alta freqüência de observações exige uma constante reconstrução do modelo empregado.**

Esta divisão é necessária pois, como veremos a seguir, antes da utilização dos modelos neurais é necessário um minucioso estudo para determinar a melhor arquitetura capaz de solucionar o problema proposto. Por outro lado, os modelos clássicos dispensam um estudo prévio, pois são gerados de forma automática a partir dos critérios definidos na seção 1.2.

Capítulo 3

Material e Métodos

Todos os processos referentes às redes neurais foram implementados através do software MATLAB (MATrix LABoratory) [24], um programa de alta performance amplamente utilizado na comunidade acadêmica que integra análise numérica, cálculo com matrizes e construção de gráficos [43]. Para a aplicação dos métodos clássicos, foi utilizada a linguagem de programação R [39], que caracteriza-se por ser uma linguagem de programação e um *software* voltado para a criação de gráficos e análises estatísticas [44].

As séries de demanda de potência ativa e reativa utilizadas apresentam um total de 5 dias de observações coletados em períodos fixos de 10 segundos, referente à Subestação Andorinha, SE40 da CPFL (Companhia Paulista de Força e Luz). As demandas são caracterizada por parcelas de consumo residencial, industrial e comercial.

Após a divisão dos dados em intervalos de 5 minutos (ou 30 observações), foi construída uma série modificada considerando somente as demandas máximas observadas em cada intervalo. Então foi prevista a potência máxima demandada nos 5 minutos vindouros (1 passo à frente) e no intervalo de 20 a 25 minutos (5 passos à frente), a partir das demandas máximas ocorridas em intervalos passados e presente. As séries obtidas estão representadas nas figuras 3.1 e 3.2.

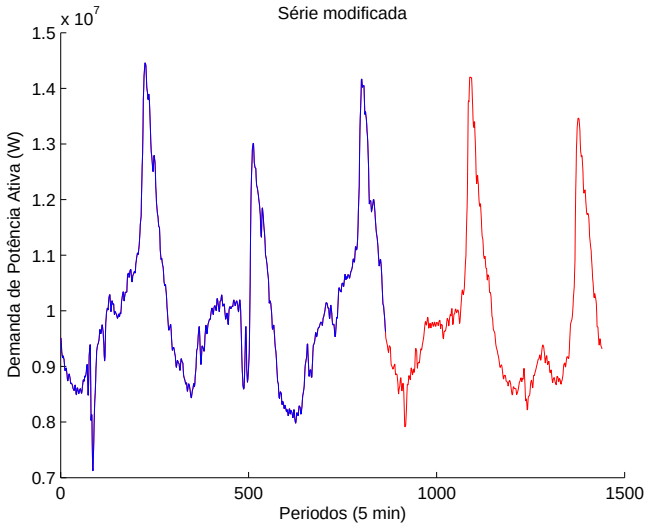

Figura 3.1: *Série modificada da demanda de potência ativa.*

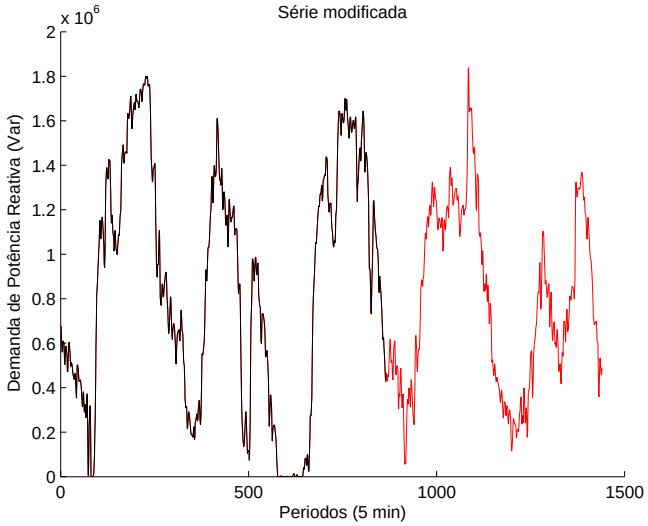

Figura 3.2: *Série modificada da demanda de potência reativa.*

3.1 Definição das arquiteturas de rede

Pode-se sintetizar a seqüência de ações desta etapa em (i) preparação dos dados, (ii) variação da arquitetura da rede, (iii) treinamento e (iv) avaliação e (v) seleção da melhor arquitetura.

Utilizou-se as redes classificadas como *feedforwards* conhecidas como *Fo-*

cused Time-Delay Neural Network (FTDNN) e *Nonlinear Autoregressive Network with Exogenous Inputs* (NARX). A vantagem de utilizarmos redes pertecentes a esta classe reside na elevada rapidez com que estas são treinadas através do método *backpropagation*. Esta característica é altamente desejável em aplicações que exige um retreinamento constante.

A rede **FTDNN** (figura 3.3) apresenta a arquitetura mais simples dentre as integrantes do conjunto denominado *focused networks*.

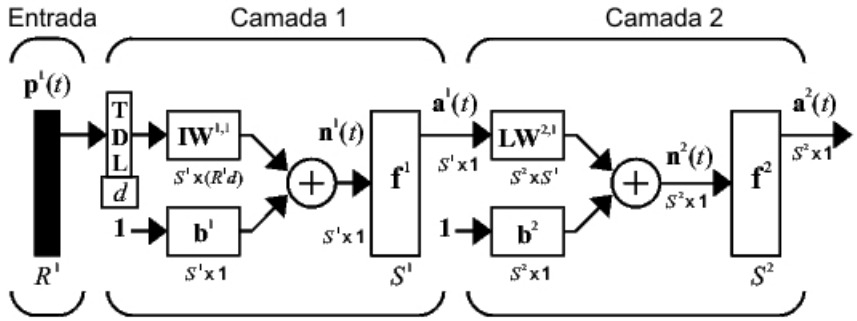

Figura 3.3: *Representação da rede FTDNN.*

A rede **NARX** (figura 3.4) apresenta uma topologia similar à rede anterior, porém, além dos valores da série a ser prevista, é possível adicionar também como entrada valores exógenos. Neste trabalho foi utilizada como entrada exógena os valores da *demanda de potência reativa* com o objetivo de investigar se estes refletem melhorias na previsão. A rede ilustrada pela figura 3.4 apresenta a versão denominada *paralela*, em que os valores de entradas previstos, $\{\hat{y}_t\}$, são realimentados na entrada.

No presente trabalho utilizou-se a configuração *série-paralela* (figura 3.5, lado direito), onde os valores reais da série, $\{y_t\}$, são utilizados como entrada e, portanto, não há realimentação de estados. Esta arquitetura apresenta duas vantagens em relação à paralela. A primeira, é que produz resultados mais precisos. A segunda, é que se trata de uma arquitetura *feedforward*, o que lhe confere as vantagens supracitadas.

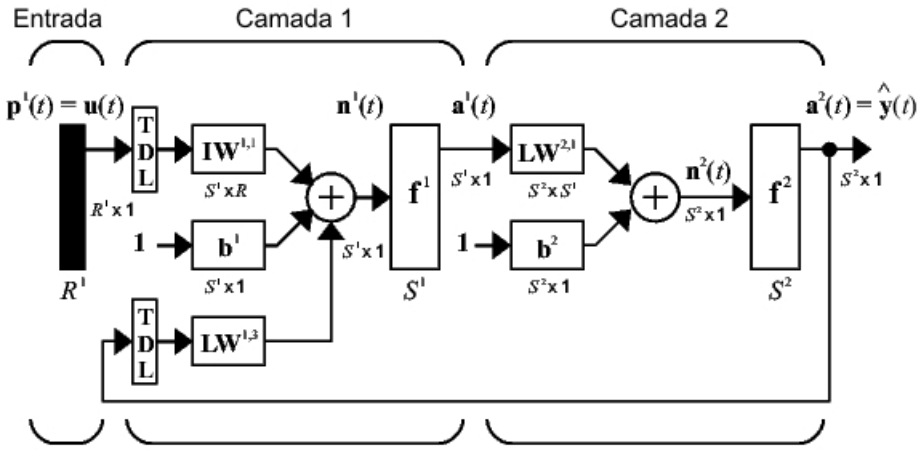

Figura 3.4: *Representação da rede NARX.*

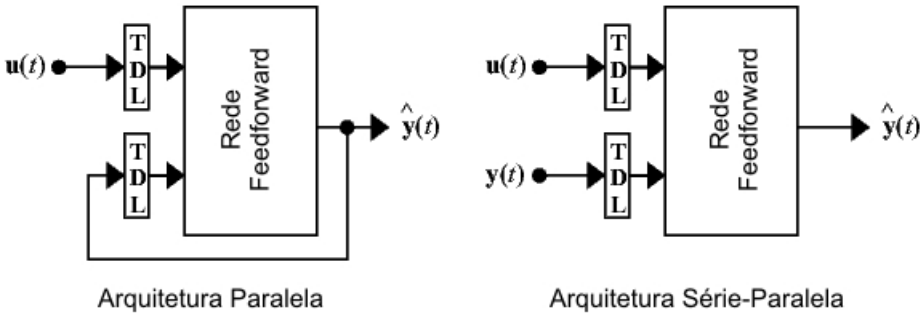

Figura 3.5: Configurações paralela e série-paralela, rede NARX.

Preparação dos dados

Objetivando melhorar a eficiência do treinamento das redes, adotou-se uma **normalização linear** dos dados para que estes ficassem compreendidos dentro do conjunto imagem das funções de ativação. Esta transformação obedeceu à seguinte equação

$$y_{ni} = \frac{y_i - y_{min}}{y_{max} - y_{min}} \tag{3.1}$$

onde y_{ni} é o valor normalizado da observação y_i, e y_{min} e y_{max} são o menor e o maior valor da série, respectivamente.

Variação do modelo neural

A determinação do modelo neural é uma tarefa fundamental em qualquer aplicação de redes neurais. Deve-se encontrar a configuração de rede

ótima capaz de resolver o problema em questão, ou seja, definir o número de nós de entrada, o número de camada escondida, o número de neurônios na camada escondida, o número de neurônios na camada de saída, as funções de ativação, etc., para os quais a rede apresenta os melhores resultados.

Para os objetivos do trabalho, o número de entrada reflete a quantidade de observações passadas, adjacentes à observação presente, que podem trazer informações significativas para a previsão. Para definir o número de entradas, iniciou-se um processo investigativo que implicou em variar este número enquanto se monitorava a eficiência da previsão. No presente trabalho testou-se os seguintes valores: 1, 2, 3, 5, 6, 7, 8, 9 e 11.

O número de saídas é, em geral, dedutível diretamente das necessidades do problema. Neste caso pretendeu-se fazer a previsão de um único valor que se encontrava 1 passo à frente, o que implicou na adoção de apenas um nó de saída.

Como já mencionado, para a maioria das aplicações de redes neurais, a adoção de uma única camada escondida é suficiente para solucionar o problema (teorema de superposição de *Kolmogorov*), enquanto garante uma arquitetura simples e de rápido treinamento. Assim, as arquiteturas com nenhuma ou uma camada escondida foram avaliadas.

Decidir pelo *número de neurônios na camada escondida*, mais uma vez, implica em um procedimento sem uma regra geral. No entanto, algumas sugestões nos guiaram nesta tarefa. Sugere-se utilizar de 0.5 a 3 vezes o número de neurônios da camada de entrada. A **regra da pirâmide geométrica**, por outro lado, aconselha

$$N_h = \alpha \sqrt[2]{N_i N_o}$$

neurônios na camada escondida quando esta é unica, onde N_i é o número de entradas da rede, N_0 é o número de saída, e α é o fator multiplicativo que depende da complexidade do problema a ser resolvido ($0.5 < \alpha < 2$). Como exemplo, para uma rede tipo *FTDNN* adotando o máximo valor de α, 11 neurônios na camada de entrada e 1 na saída, obtemos $N_h = 6.6332$. Considerando as duas sugestões, foram testados para cada valor de neurônios na camada de entrada, os seguintes números de neurônio na camada escondida: 1, 2, 3, 4, 6, 8, 10, 12, 14 e 16, e 0 (ausência da camada).

Para todos os neurônios da camada de saída utilizou-se a função de ativação *purelin* (Matlab), que tem como saída a regressão linear dos valores de entrada. Como a adoção de uma função de ativação simétrica implica vantagens no processo de aprendizagem dos perceptrons e melhoras na eficiência da rede, para todos os outros neurônio foi implementada a função denominada *tansig* (Matlab), dada pela equação

$$S(x) = 2s(x) - 1 = \frac{1 - e^{-x}}{1 + e^{-x}}$$

e apresentada pela figura 3.6.

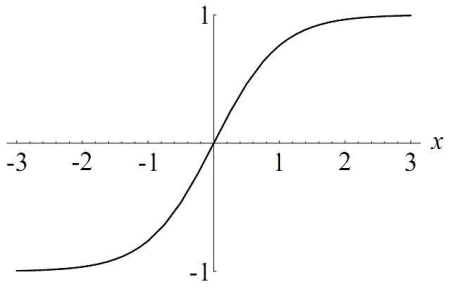

Figura 3.6: *Função tangente hiperbólica.*

Como as funções de ativação foram fixadas, o estudo do impacto da alteração destas na eficiência da rede está fora do escopo deste trabalho.

Finalizando, observa-se que foram avaliadas um total de 121 diferentes arquiteturas para cada tipo de rede.

Treinamento

As funções do tipo sigmóide apresentam baixas derivadas para grandes valores de entrada. Durante o treinamento, isto pode causar pequenas variações nos pesos, mesmo que estes estejam longe do valor ótimo. Assim, foi utilizado em todos os casos o algoritmo de treinamento *Levenberg-Marquardt*, denominado *trainlm* no Matlab, que se utiliza de aproximações matriciais para acelerar a convergência dos pesos.

O número de épocas de treinamento foi definido como 100, isto é, os dados foram apresentados no máximo 100 vezes para cada rede antes da obtenção dos valores finais dos pesos. Cada configuração foi treinada 8 vezes para se aumentar as chances de encontrar os mínimos globais.

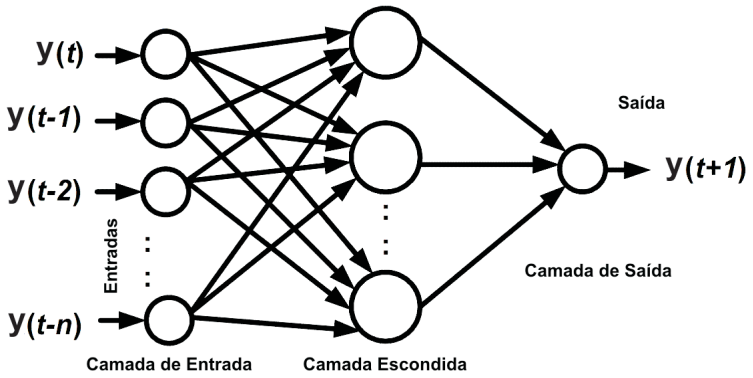

Figura 3.7: *Rede neural para previsão de 1 passo.*

Durante o treinamento da rede FTDNN, foram apresentados em sua entrada os valores do vetor de demanda de potência ativa de acordo com o número n de entradas configurado, enquanto na saída foi apresentado o valor do vetor referente a um passo à frente, como nos mostra a figura 3.7. Neste caso, procurou-se construir a seguinte função

$$\hat{y}_{t+1} = f(y_t, y_{t-1}, y_{t-2}, \ldots, y_{t-n})$$

onde y_t é a série de demanda de potência ativa.

Já para o treinamento da rede NARX, forneceu-se também como entrada a série de potência reativa. Desta forma, encontrou-se a seguinte função

$$\hat{y}_{t+1} = f(y_t, y_{t-1}, y_{t-2}, \ldots, y_{t-n}, u_t, u_{t-1}, u_{t-2}, \ldots, u_{t-n})$$

onde y_t é a série de demanda de potência ativa e u_t a série de demanda de potência reativa.

Ambas as arquiteturas foram implementadas em ambiente Matlab através das funções *newfftd* e *newnarx*, respectivamente.

Avaliação

Para avaliar cada arquitetura de rede os dados foram divididos em dois conjuntos. O primeiro evidenciado nas figuras 3.1 e 3.2 pelas cores azul e preta, denomina-se conjunto de treinamento, ou seja, são os dados que foram utilizados para treinar a rede. Compreende as observações efetuadas em t igual 1 até 864. O segundo, em cor vermelha, é dito conjunto de teste, utilizado para avaliar a rede previamente treinada. Corresponde ao conjunto das observações efetuadas em t igual 865 até 1440.

Seleção da melhor arquitetura

O erro médio percentual absoluto, também denominado como MAPE (*mean absolute percentage error*), foi definido como índice de avaliação das arquiteturas e é dado por

$$MAPE = \frac{1}{n} \sum_{i=1}^{n} \mid \frac{y_i(h) - \hat{y}_i(h)}{y_i(h)} \mid \tag{3.2}$$

onde n é o número de pontos da série.

O índice da arquitetura foi tomado como o menor MAPE dos 8 treinamentos executados. Já a configuração ótima das redes FTDNN e NARX foram tomadas como as configurações que apresentaram o menor erro dentre todos as variações do modelo. A figura 3.8 mostra o algoritmo utilizado para encontrar estas arquiteturas.

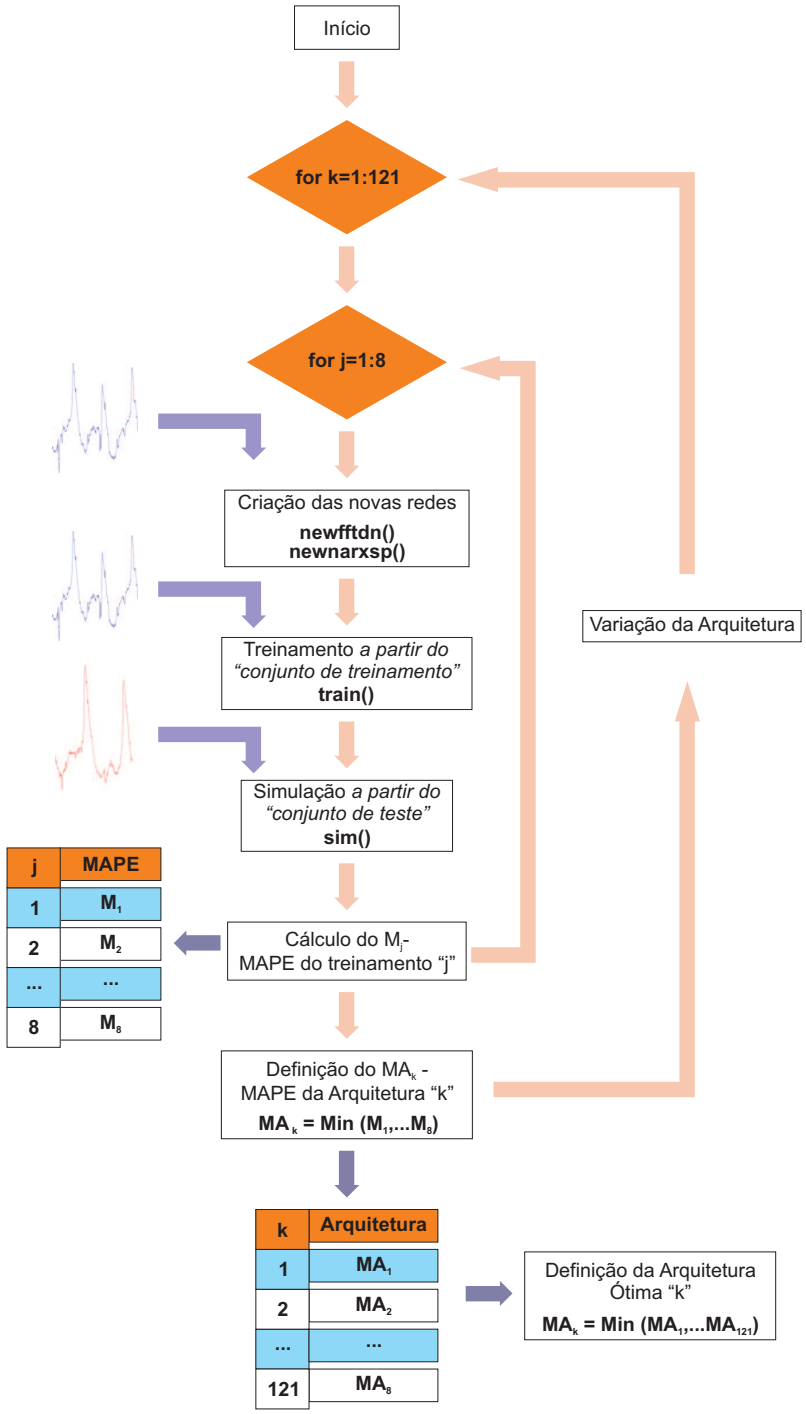

Figura 3.8: *Algoritmo utilizado para a definição da arquitetura ótima das redes NARX e FTDNN.*

3.2 Comparação dos modelos

Para a comparação entre os modelos foi avaliado o erro de cada método quando submetidos a um conjunto de treinamento com número crescente de dados. Primeiramente, as redes foram treinadas e os modelos clássicos definidos para o mesmo conjunto de treinamento adotado anteriormente (figuras 3.1 e 3.2, cores azul e preto). Depois disso, cada modelo imprimiu suas previsões para 1 e 5 passos adiante. A próxima observação da série, já pertencente ao conjunto de teste (cor vermelha), foi adicionada ao conjunto de treinamento e o processo se repetiu até que o conjunto de treinamento contivesse todos os dados das séries (com exceção do último). O resultado desta seqüência de ações é a previsão da demanda para os dois últimos dias dos dados.

Redes Neurais

Os dados foram normalizados como definidos na seção 3.1. As arquiteturas FTDNN e NARX selecionadas na etapa anterior foram treinadas 8 vezes para cada novo valor inserido no conjunto de treinamento. Após cada treinamento, foi calculado o MAPE quando a mesma foi simulada com o próprio conjunto de treinamento. A rede que apresentou o menor erro dentre os treinamentos foi selecionada para realizar as previsões.

Para obter a previsão de 1 passo adiante, o processo foi direto. Inseriu-se na entrada da rede os valores $y_t, y_{t-1}, y_{t-2}, \ldots, y_{t-n}$ para FTDNN e, adicionalmente, os valores $u_t, u_{t-1}, u_{t-2}, \ldots, u_{t-n}$ para NARX, e obteve-se na saída $\hat{y}_{t+1}$. Como a rede foi treinada para prever apenas o valor seguinte da série, a previsão de 5 passos adiante nos exigiu a estimativa dos passos intermediários. Desta forma, para a rede FTDNN, o cálculo de $\hat{y}_{t+5}$ foi alcançado a partir da obtenção dos valores $\hat{y}_{t+1}, \ldots, \hat{y}_{t+4}$. Assim apresentou-se na entrada da rede o conjunto $\hat{y}_{t+4}, \ldots, \hat{y}_{t+1}, y_t, y_{t-1}, \ldots, y_{t-n+4}$, para obter a saída desejada. No caso de NARX deve-se dispor também de $\hat{u}_{t+1}, \ldots, \hat{u}_{t+4}$, no entanto, devido à propria construção da rede, obtem-se em sua saída somente a previsão para a série de potência ativa. Para solucionar este problema sem alterar a topologia da rede, adotou-se a previsão inocente. Assim, inseriu-se como entradas exógenas as variáveis $\hat{u}_{t+4}, \ldots, \hat{u}_{t+1}, u_t, u_{t-1}, \ldots, u_{t-n+4}$, onde $\hat{u}_{t+4}, \ldots, \hat{u}_{t+1}$ são iguais a u_t.

As figuras 3.9 e 3.10 apresentam esta sequência de ações, bem como as principais funções utilizadas em ambiente MATLAB. As previsões para 1 passo foram armazenadas no vetor *for1_FTDNN* e *for1_NARX*. Jás para 5 passos adiante, as previsões foram gravadas nos vetores *for5_FTDNN* e *for5_NARX*.

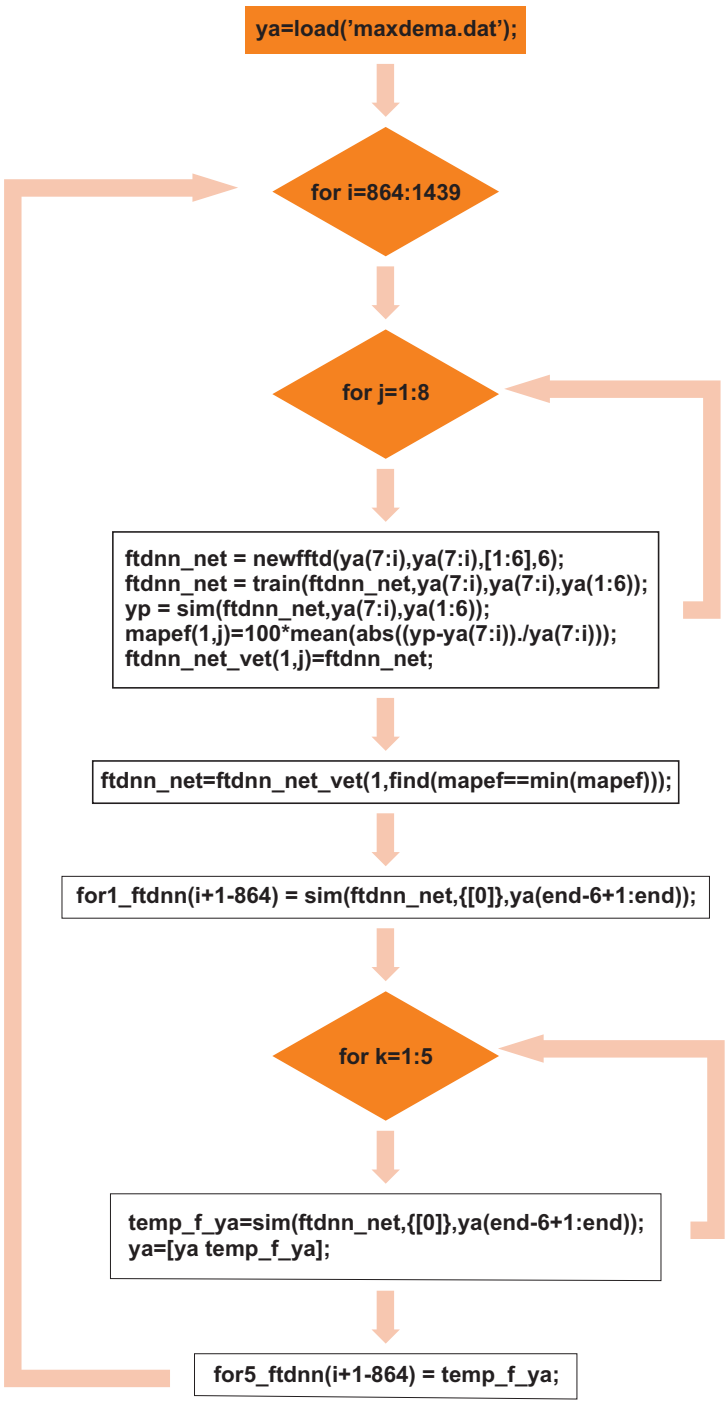

Figura 3.9: *Algoritmo utilizado para a previsão - rede FTDNN.*

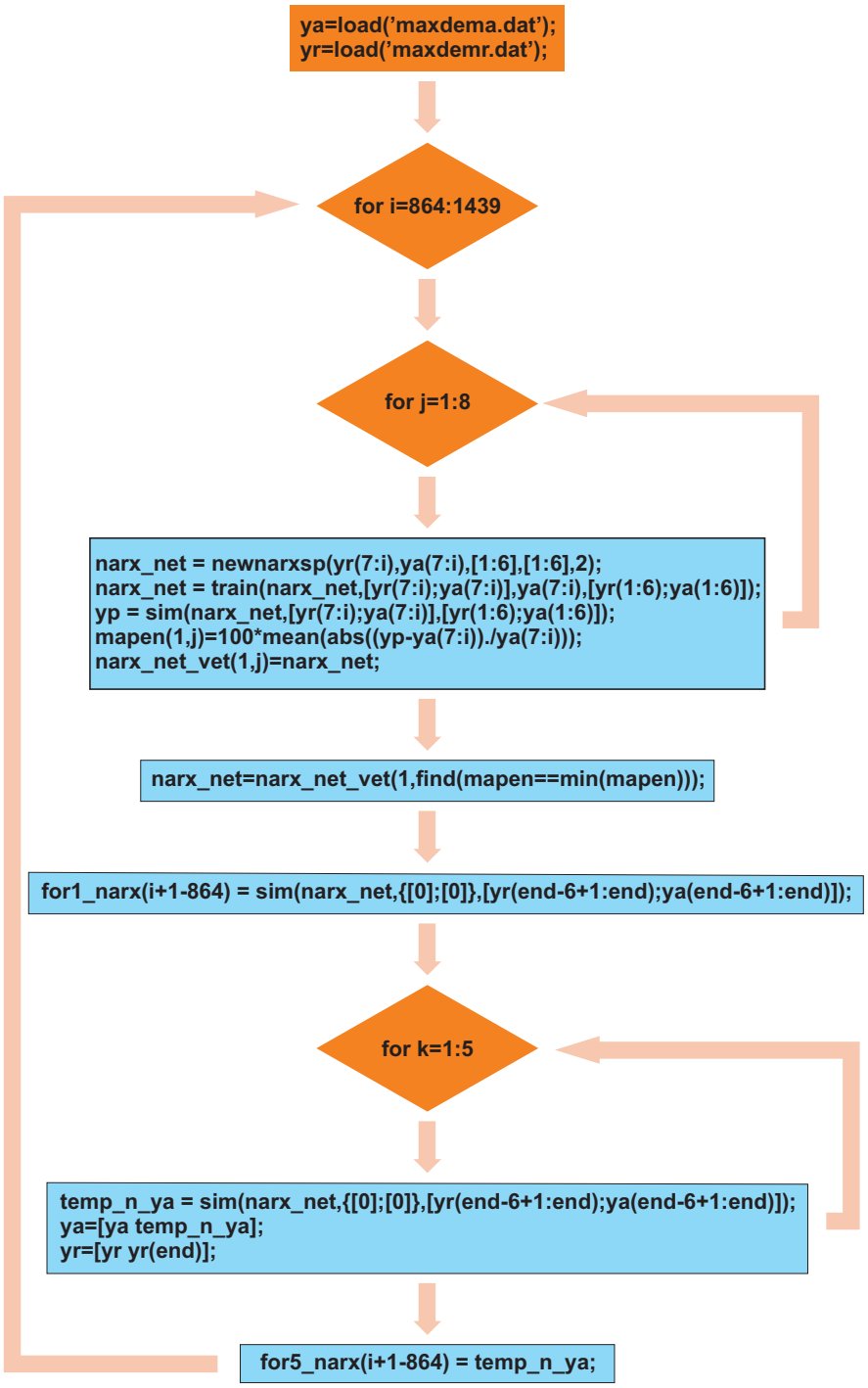

Figura 3.10: *Algoritmo utilizado para a previsão - rede NARX.*

Modelos Clássicos

Tendo em vista que valores elevados da série podem causar problemas numéricos nos algoritmos de definição dos modelos clássicos, os dados foram previamente normalizados da seguinte maneira

$$y_{ni} = \frac{y_i}{y_{max}}$$

onde y_{ni} é o valor normalizado da observação y_i e y_{max} é o maior valor da série.

Os modelos de *Suavização Exponencial* e *ARIMA* foram determinados automaticamente com a ajuda das funções *ets()* e *auto.arima()* inclusas no pacote *forecast* [13] do programa R. Estas funções recebem como entrada uma séries temporal e retornam o melhor modelo a partir dos critérios apresentados nas seções 1.2.1 e 1.2.2.

As figuras 3.11 e 3.12 apresentam a sequência de ações, bem como as principais funções utilizadas em linguagem R. As previsões para 1 passo foram armazenadas no vetor *for1_ets* e *for1_arima*. Jás para 5 passos adiante, as previsões foram gravadas nos vetores *for5_ets* e *for5_arima*.

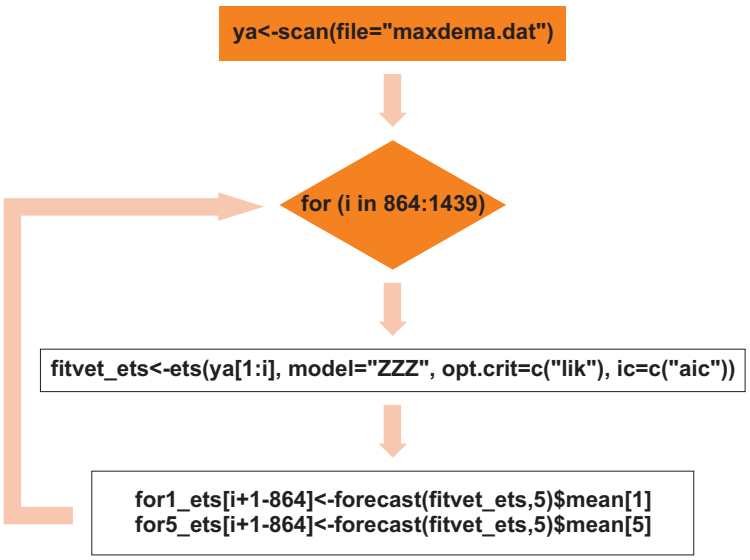

Figura 3.11: *Algoritmo utilizado para a previsão - Suavização Exponencial.*

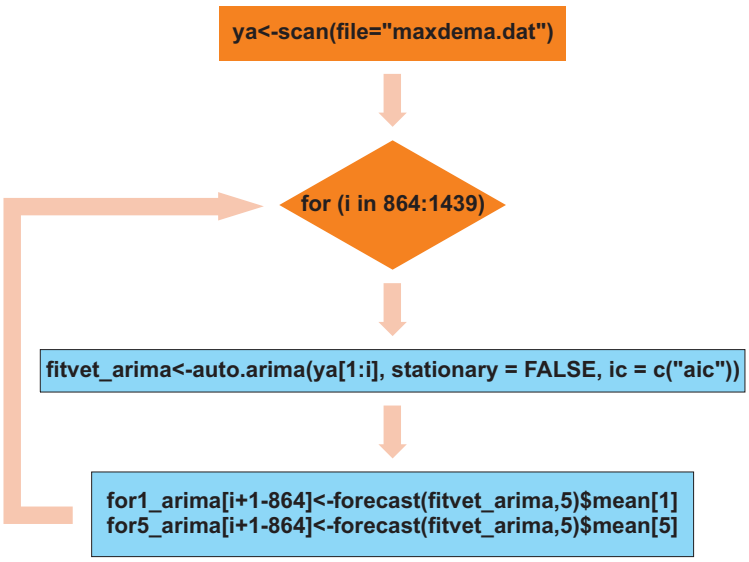

Figura 3.12: *Algoritmo utilizado para a previsão - método ARIMA.*

Capítulo 4

Resultados e Discussão

4.1 Definição das arquiteturas de rede

Os gráficos 4.1 e 4.2 nos revelam o MAPE médio, mínimo e seu desvio padrão, das redes FTDNN e NARX que apresentaram os melhores desempenho para cada número de neurônios na camada de entrada. As tabelas 4.1 e 4.2 apresentam os detalhes de cada ponto mostrado nos gráficos recem citados.

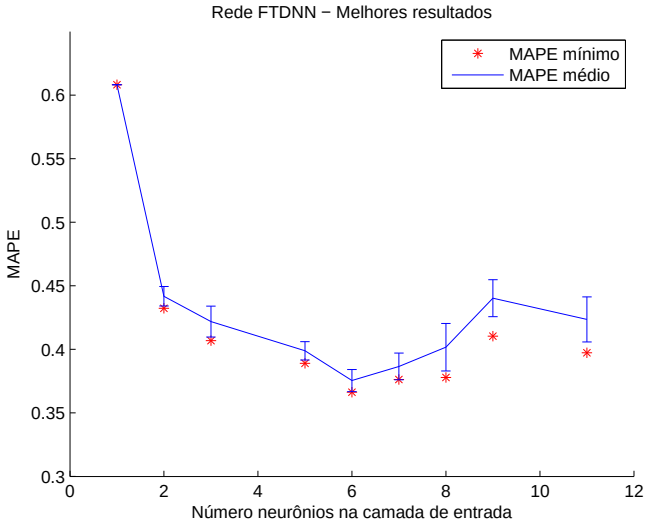

Figura 4.1: *MAPE das melhores arquiteturas FTDNN.*

Tabela 4.1: *Sumário dos resultados FTDNN.*

Rede	NNIL	NNHL	MAPE médio	MAPE max	MAPE min	MAPE dp
FTDNN	1	0	0,608177245	0,608177245	0,608177245	3,33733E-13
FTDNN	2	4	0,441748463	0,453502795	0,432327463	0,007663887
FTDNN	3	10	0,421835742	0,437862455	0,406823435	0,012099027
FTDNN	5	8	0,398833866	0,409767211	0,388969813	0,007197783
FTDNN	6	6	0,375435433	0,391955345	0,366284783	0,008671219
FTDNN	7	6	0,386521076	0,40745615	0,376256644	0,01052197
FTDNN	8	6	0,401685642	0,425390005	0,377919341	0,018686587
FTDNN	9	8	0,440218914	0,460780308	0,410405146	0,014546498
FTDNN	11	4	0,423592608	0,452003139	0,397325321	0,017723766

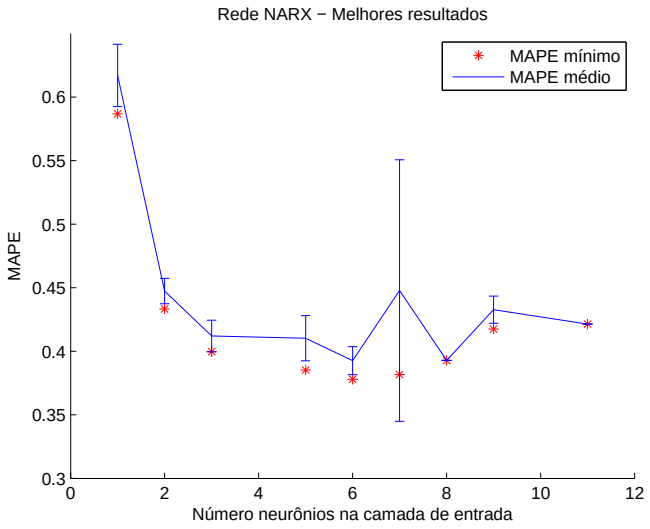

Figura 4.2: *MAPE das melhores arquiteturas NARX.*

Tabela 4.2: *Sumário dos resultados NARX.*

Rede	NNIL	NNHL	MAPE médio	MAPE max	MAPE min	MAPE dp
NARX	1	3	0,61701041	0,666802032	0,586771349	0,024573245
NARX	2	2	0,447409324	0,461470231	0,433350013	0,010001525
NARX	3	2	0,412016769	0,433420234	0,399739118	0,012392889
NARX	5	4	0,410274466	0,428986372	0,385299911	0,01777174
NARX	6	2	0,392627436	0,414689768	0,378014995	0,011019812
NARX	7	4	0,447823979	0,698901869	0,38169342	0,102861648
NARX	8	1	0,392814663	0,392814663	0,392814663	1,5029E-12
NARX	9	2	0,432723572	0,445943473	0,417525962	0,010703694
NARX	11	1	0,42135978	0,42135978	0,42135978	5,9905E-11

A partir dos resultados encontrados foram definidas como arquiteturas ótimas:

- **FTDNN** - 6 neurônios na camada de entrada e 6 na camada de escondida;
- **NARX** - 6 neurônios na camada de entrada e 2 na camada de escondida.

As redes escolhidas para representar o grupo apresentaram um desvio padrão do erro relativamente pequeno. Isto sugere que a cada treinamento da rede, o MAPE calculado será próximo ao valor mínimo, o que implica em confiabilidade da eficiência de ambas.

4.2 Comparação dos modelos

Previsão de 1 passo

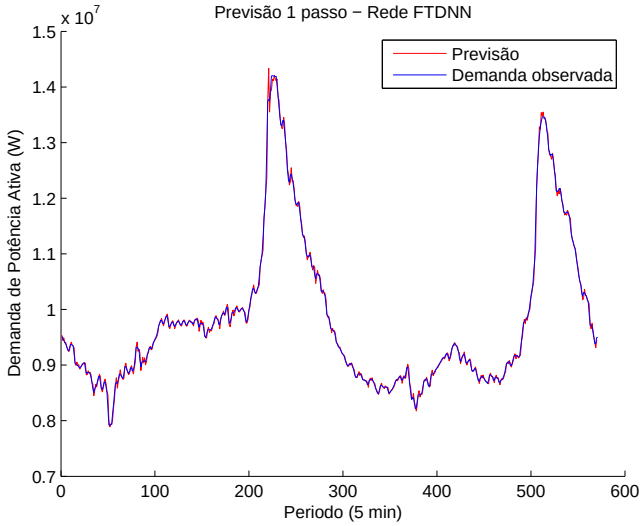

Figura 4.3: *Previsão de 1 passo - Rede FTDNN.*

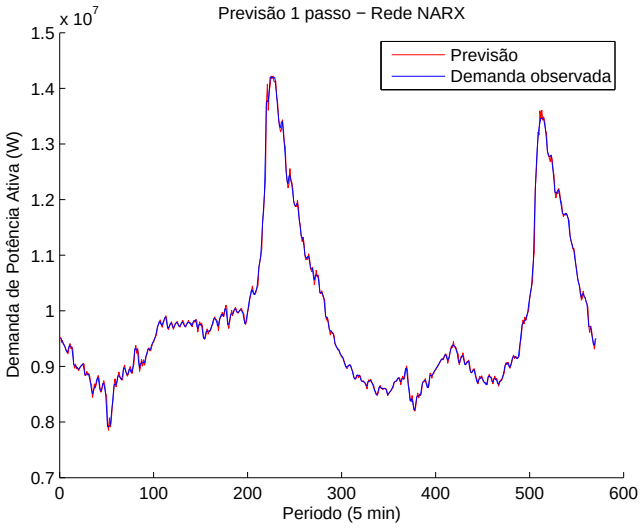

Figura 4.4: *Previsão de 1 passo - Rede NARX.*

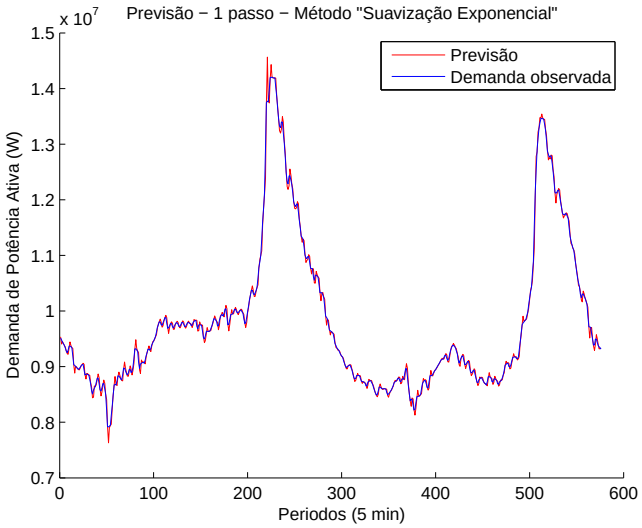

Figura 4.5: *Previsão de 1 passo - Suavização Exponencial.*

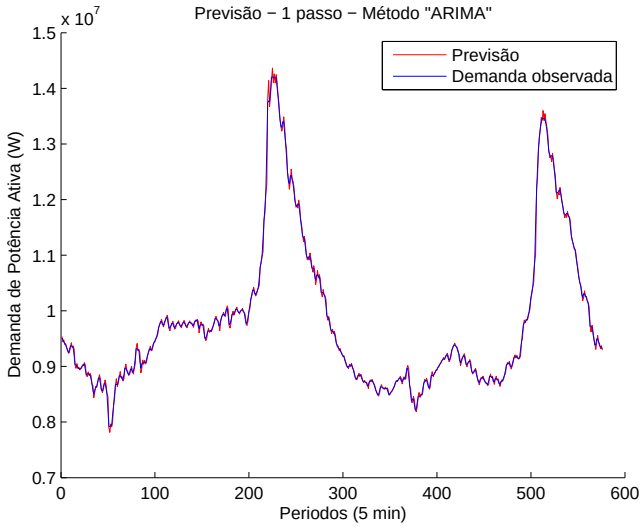

Figura 4.6: *Previsão de 1 passo - ARIMA.*

Previsão de 5 passos

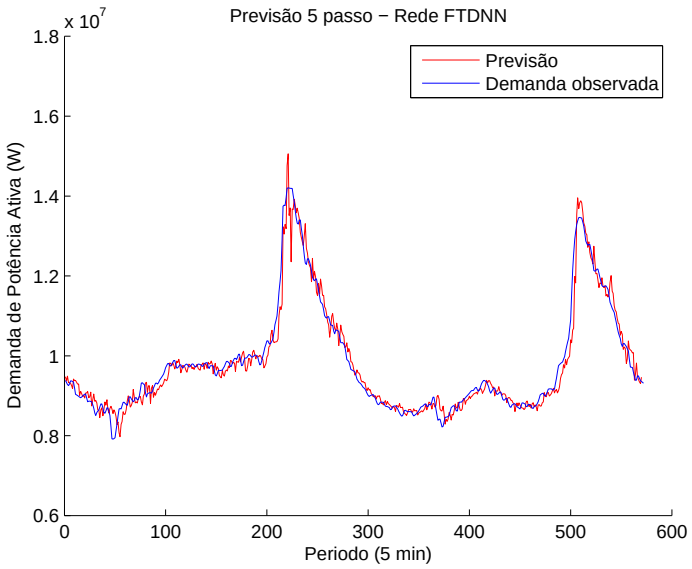

Figura 4.7: *Previsão de 5 passos - Rede FTDNN.*

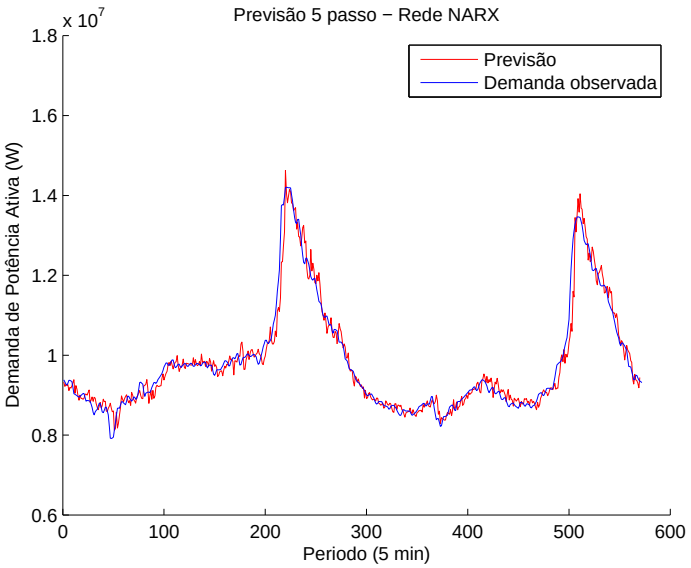

Figura 4.8: *Previsão de 5 passos - Rede NARX.*

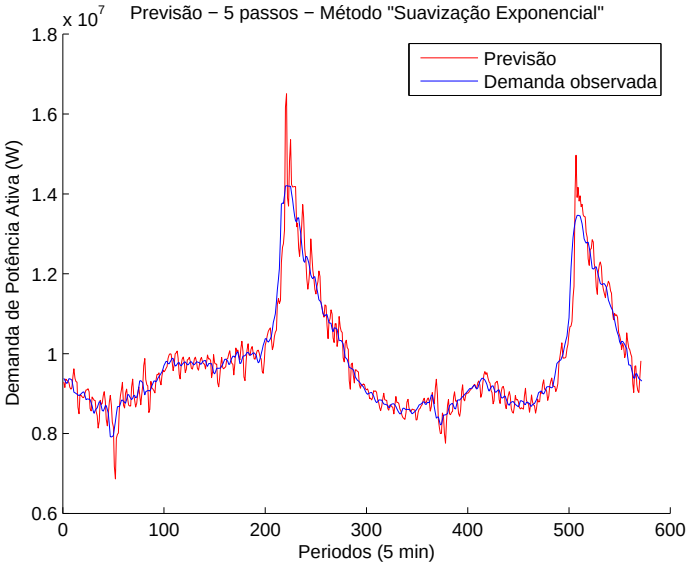

Figura 4.9: *Previsão de 5 passos - Suavização Exponencial.*

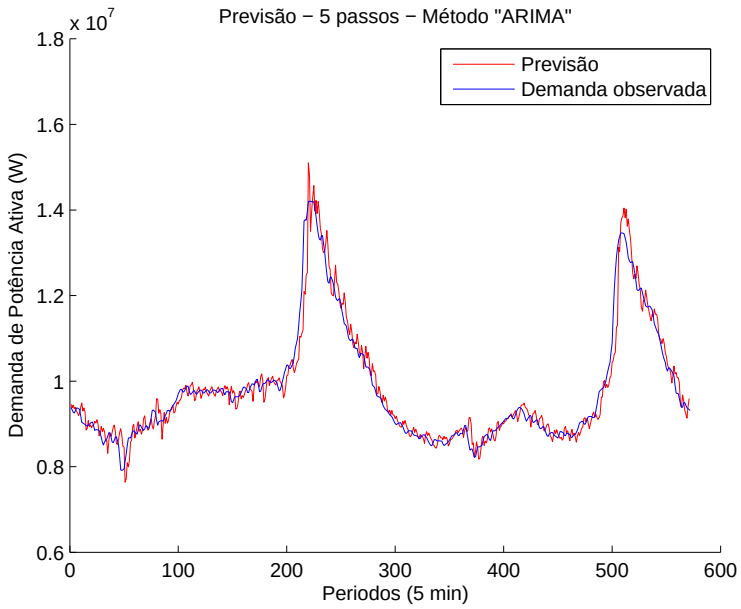

Figura 4.10: *Previsão de 5 passos - ARIMA.*

A Tabela 4.3 apresenta os índices obtidos por cada técnica para a previsão de 1 e 5 passos. A coluna denominada **Ganho (%)** revela o ganho médio obtido pela utilização do método em relação à previsão inocente para ambos os dias. Por fim, a última coluna mostra o tempo total gasto por cada algoritmo em todo o processo.

Tabela 4.3: *Sumário dos resultados finais.*

Método	h	MAPE (%)	Ganho (%)	Tempo		
				h	min	s
Prev. Inocente	1	0,611278	0,000	0	0	0
Suav. Exponencial	1	0,439393	28,119	1	56	38
ARIMA	1	0,380200	37,802	2	6	36
Rede FTDNN	1	0,372249	39,103	13	22	51
Rede NARX	1	0,370660	39,363	16	25	58
Prev. Inocente	5	2,335423	0,000	0	0	0
Suav. Exponencial	5	2,634627	-12,812	1	56	38
ARIMA	5	2,126752	8,935	2	6	36
Rede FTDNN	5	2,191997	6,141	13	22	51
Rede NARX	5	2,087889	10,599	16	25	58

O tempo total (T_t) gasto na execução dos algoritmos apresentados pelas figuras 3.9, 3.10, 3.12 e 3.11, foi encontrado a partir de

$$T_t = \sum_{i=864}^{1439} t_{i+1} \tag{4.1}$$

onde t_{i+1} foi o tempo necessário para que a técnica realizasse a previsão da observação $i+1$. O tempo dos algoritmos considerados apresentam, aproximadamente, uma relação linear com o número de observações utilizadas na construção de cada modelo, da seguinte forma

$$t_{i+1} = kn_i = ki \tag{4.2}$$

onde k é uma constante e n_i é o tamanho do conjunto de dados utilizado para a previsão da observação $i+1$. A última igualdade da equação 4.2 decorre diretamente de $n_i = i$. A partir destas considerações construímos a Tabela 4.4.

Tabela 4.4: *Tempo gasto para previsão de uma observação.*

Método	k (s/obs)	t_{865} (s)	t_{1440} (s)
Suav. Exponencial	0.0106	9,116	15,183
ARIMA	0.0115	9,895	16,480
Rede FTDNN	0.0726	62,750	104,511
Rede NARX	0.0901	77,844	129,649

O gráfico apresentado pela figura 4.11 mostra a média do ganho de cada modelo em relação à previsão inocente para cada dia de previsão. Claramente se observa um aumento da eficiência dos algoritmos para um maior número de observações utilizados na construção dos modelos. No entanto, da Tabela 4.4, nota-se que este aumento de eficiência é compensado negativamente por um maior tempo necessário à previsão.

Werbos (1989,1990)[27], Ansuj e colaboradores (1996) [10], Kohzadi e colaboradores [18], Chin e Arthur [29], Hill [40] e colaboradores e Caire [5] e colaboradores [28] concluíram que as redes neurais podem apresentar resultados superiores aos obtidos pela estatística clássica. Da mesma forma, Lapedes e Farber (1988) [31] obtiveram sucesso na utilização de redes neurais para modelar e prever séries temporais não lineares. No entanto, Maier e Dandy [16] sugerem que os modelos ARIMA apresentam melhores resultados para previsões a curto prazo e que as redes neurais se ajustam melhor para as previsões a longo prazo. Em suma, não há um modelo que prevaleça, e a escolha deste está diretamente relacionada com a aplicação desejada.

Como o objetivo do presente trabalho foi realizar uma previsão a curtíssimo prazo a cada nova observação, obviamente o tempo utilizado pela técnica

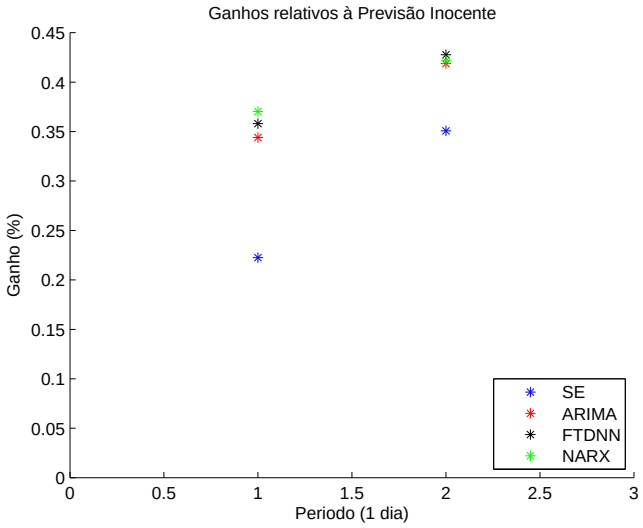

Figura 4.11: *Ganho em relação à Previsão Inocente - 1 passo.*

também é importante para a escolha do melhor modelo. A rede NARX apresentou o menor erro e o maior tempo gasto para a previsão. No entanto, para o apontamento do modelo ideal, faz-se necessário um detalhado estudo que pondere a importância de cada parâmetro para o caso considerado.

Capítulo 5

Conclusões

A arquitetura NARX que apresentou o menor erro médio na previsão a curtíssimo prazo da máxima demanda de potência ativa apresentou 6 entradas e 2 neurônios na camada escondida. Já a rede FTDNN registrou o valor 6 para ambas as características.

Na comparação das técnicas, a rede NARX apresentou o menor erro de previsão. No entanto, para a definição do melhor modelo para a previsão a curtíssimo prazo da máxima demanda ativa, deve-se considerar também o tempo gasto na previsão. Sendo assim, o modelo adequado deve apresentar um ajuste ótimo entre acurácia e tempo. Como a rede NARX gastou mais tempo durante o processo de previsão, a escolha de um modelo torna-se dificil quando esses dois parâmetros são considerados concomitantemente.

Capítulo 6

Trabalhos Futuros

A partir de um conjunto de dados maior, investigar de forma mais detalhada a relação entre acurácia dos modelos e tempo gasto para a previsão. Estudar o impacto gerado por uma previsão menos errônea e pelo tempo necessário à previsão. A partir destes definir o melhor modelo para resolver o problema em questão.

Referências Bibliográficas

[1] Gómez V. Maravall A. Programs tramo and seats, instructions for the users. *Working paper 97001, Dirección General de Anilisis y Programación Presupuestaria, Ministerio de Economia y Hacienda*, 1998.

[2] Hylleberg S. Engle R. Granger C. Yoo B. Seasonal integration and cointegration. *Journal of Econometrics*, 44:215 238, 1990.

[3] Reinsel G. C. Box, G. E. P. Jenkins G. M. *Adaptive Switching Circuits. In: Anderson J and Rosenfeld E. (eds.) Neurocomputing.* MIT Press, 1960.

[4] G. M. Reinsel G. C. Box G. E. P. Jenkins. *Time Series Analysis, Forecasting and Control.* Prentice Hall, 1994.

[5] Caire D. Hatabian G. Muller C. Progress in forecasting by neural networks. *Int. Conf. Neural Networks II*, page 540 545, 1992.

[6] Pegels C.C. Exponential forecasting: some new variations. *Management Science*, 15(5):311 15, 1969.

[7] Broomhead D.S. Lowe D. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2:321 355, 1988.

[8] Palit A. K. Popovic D. *Computational Intelligence in Time Series Forecasting Theory and Engineering Applications.* Springer, 2005.

[9] Specht D.F. Probabilistic neural networks for classification, or associative memory. *Proc. of IEEE Intern. Conf. on Neural Networks*, 1:525 532, 1988.

[10] Ansuj A.P. Camargo M.E. Radharamanan R. Petry D.G. Sales forecasting using time series and neural networks. *Comput. Ind. Eng*, 31(1):421 424, 1996.

[11] McClelland J. L. Rumelhart D. E. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* MIT Press, 1986.

[12] Amari S e Maginu K. Statistical neurodynamics of associative memory. *Neural Networks*, 1:63 73, 1988.

[13] Rob J. Hyndman e Yeasmin Khandakar. Automatic time series forecasting: the forecast package for r. *Journal of Statistical Software*, 26(3), 2008.

[14] Gardner Jr. E.S. Exponential smoothing: The state of the art. *Journal of Forecasting*, 4:1 28, 1985.

[15] Rosenblatt F. The perceptron: A probabilistic model for information storage and organisation of the brain. *Psych. Review*, 65:386 408, 1958.

[16] Maier H.R. Dandy G.C. Neural network models for forecasting univariate time series. *Neural Networks World*, 6(5):747 772, 1996.

[17] Akaike H. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716 723, 1974.

[18] Kohzadi N. Boyd M.S. Kermanshahi B. Kaastra I. A comparison of artificial neural network and time series model for forecasting commodity prices. *Neurocomputing*, 10(2):169 181, 1996.

[19] Hannan E.J. Rissanen J. Recursive estimation of mixed autoregressive-moving average order. *Biometrika*, 69(1):81 94, 1982.

[20] Mélard G. Pasteels J.M. Automatic arima modeling including intervention, using time series expert software. *International Journal of Forecasting*, 16:497 508, 2000.

[21] Taylor J.W. Exponential smoothing with a damped multiplicative trend. *International Journal of Forecasting*, 19:715 725, 2003.

[22] Elman J. L. Finding structure in time. *Cognitive Science*, 14:179 211, 1990.

[23] Liu L.M. Identification of seasonal arima models using a altering method. *Communications in Statistics Part A Theory and Methods*, 18:2279 2288, 1989.

[24] The MathWorks. *MATLAB technical documentation*. The MathWorks, Inc, Natick, Massachusetts, 2008.

[25] Hu M.J.C. Application of the adaline system to weather forecasting. master thesis 6775 1, stanford el. lab., stanford, ca. *Technical Report*, 1964.

[26] Werbos P.J. Beyond regression: New tool for prediction and analysis in the behavioural sciences. *Ph.D. Thesis, Harvard University, Cambridge, MA.*, 1974.

[27] Werbos P.J. Backpropagation and neural control: A review and prospectus. internat. *Joint Conf of Neural Networks, Washington*, 1:209 216, 1989.

[28] Werbos P.J. Backpropagation through time what it does and how to do it. *Proc. of IEEE*, 78(10):1550 1560, 1990.

[29] Chin K. Arthur R. Neural network vs. conventional methods of forecasting. *J. Bus. Forecast.*, 14(4):17 22, 1996.

[30] Hecht-Nielsen R. Counterpropagation networks. *Applied Optics*, 26(23):4979 4984, 1987a.

[31] Lapedes A. Farber R. Nonlinear signal processing using neural network: prediction and system modelling. *Joint Conf of Neural Networks, Washington*, 1987.

[32] Hyndman R.J. Koehler A.B. Ord J.K. Snyder R.D. Prediction intervals for exponential smoothing using two new classes of state space models. *Journal of Forecasting*, 24:17 37, 2005.

[33] Ord J.K. Koehler A.B. Snyder R.D. Estimation and prediction for a class of dynamic nonlinear statistical models. *Journal of the American Statistical Association*, 92:1621 1629, 1997.

[34] Rumelhart D.E. Hinton G.E. Williams R.J. *Learning internal representation by back-propagation errors. In: Rumelhart DE, McClelland JL, the PDP Research Group(Eds.), Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* MIT Press, 1986.

[35] Grossberg S. *Competitive Learning: From interactive activation to adaptive resonance, Neural Networks and Neural Intelligence.* MIT Press, 1988.

[36] Hyndman R.J. Koehler A.B. Snyder R.D. Grose S. A state space framework for automatic forecasting using exponential smoothing methods. *International Journal of Forecasting*, 18(3):439 454, 2002.

[37] Minsky M. L. Papert S. *Perceptrons.* MIT Press, 1960.

[38] Kohonen T. *Self-Organisation and Associative Memory. 3rd Edition.* Springer, 1989.

[39] R Development Core Team. *R: A language and environment for statistical computing.* R Foundation for Statistical Computing, Vienna, Austria, 2005. ISBN 3-900051-07-0.

[40] Hill T. O Connor M. Remus W. Neural network models for time series forecasts. *Manage. Sci.*, 42(7):1082 1092, 1996.

[41] McCulloch W.S. Pitts W. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115 133, 1943.

[42] Dickey D.A. Fuller W.A. Likelihood ratio statistics for autoregressive time series with a unit root. *Econometrica*, 49:1057 1071, 1981.

[43] WIKIPEDIA. *http://en.wikipedia.org/wiki/MATLAB.*

[44] WIKIPEDIA. *http://en.wikipedia.org/wiki/R.*

[45] Kwiatkowski D. Phillips P.C. Schmidt P. Shin Y. Testing the null hypothesis of statioanrity against the alternative of a unit root. *Journal of Econometrics*, 54:159 178, 1992.