Universidade de São Paulo
Escola de Engenharia de São Carlos
Departamento de Engenharia Elétrica

Trabalho de Conclusão de Curso

Uso de Rede Neural *Perceptron* Multicamadas como estimador de tempos de Flicker

Autor
Jordão Natal de Oliveira Júnior

Orientador
Prof. Dr. Rodrigo Andrade Ramos

São Carlos, 2017

Oliveira Junior, Jordao Natal
O46u Uso de Rede Neural Perceptron Multicamadas como Estimador de Tempos de Flicker / Jordao Natal Oliveira Junior; orientador Rodrigo Andrade Ramos. São Carlos, 2017.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2017.

1. Flicker. 2. Rede Neural. 3. Mineração de Dados. I. Título.

FOLHA DE APROVAÇÃO

Nome: Jordão Natal de Oliveira Júnior

Título: "Uso de rede neural Perceptron multicamadas como estimador de tempos de Flicker"

Trabalho de Conclusão de Curso defendido e aprovado em 30/06/2017

com NOTA 5,0 (cinco, zero), pela Comissão Julgadora:

Prof. Associado Rodrigo Andrade Ramos - Orientador - SEL/EESC/USP

Doutorando Edson Luis Geraldi Junior - SEL/EESC/USP

Mestre Marcel Ayres de Araujo - Doutorando - SEL/EESC/USP

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado José Carlos de Melo Vieira Júnior

Agradecimentos

Agradeço ao professor Rodrigo Ramos, por ter aceitado me orientar no projeto, mesmo possuindo a agenda completamente lotada, e à sua aluna Marley Tavares, pelas horas em que se dispôs em me ajudar de todas as maneiras possíveis

Resumo

Sistemas inteligentes vêm recebendo cada vez mais atenção em áreas da ciência aplicada e engenharia, e perdendo sua aura mística de ferramenta legada ao futuro. O presente trabalho é mais um que faz uso dessas entidades, para resolver um problema prático relacionado à distribuição de energia elétrica. Ao se abordar o problema do *flicker* em redes de distribuição de energia, e fazendo uso dos dados e experiências retirados de um trabalho teórico anterior, foram realizados testes com diversos algoritmos da área de redes neurais e mineração de dados para se prever o comportamento do sistema em questão.

Abstract

Intelligent systems are receiving more and more attention in areas of applied science and engineering, and losing their mystical aura of tool bequeathed to the future. The present work is one more that makes use of these entities, to solve a practical problem related to the distribution of electric energy. When addressing the flicker problem in energy distribution networks, and using data and experiences drawn from previous theoretical work, tests were performed with several neural network algorithms and data mining to predict the behavior of the system in question.

Capítulo 1: Introdução

A rede de geração e distribuição de energia no Brasil cresceu enormemente nos últimos 25 anos, resultado da estabilização econômica na década de 1990 e do aumento no valor dos comódites no início do século XXI. Com isso, a mesma se tornou mais complexa, e características como robustez e previsibilidade dos problemas possíveis de acontecerem se tornaram uma preocupação cada vez mais importante.

Ao mesmo tempo, a capacidade de processamento dos computadores continuava aumentando de maneira exponencial, até mesmo superando a lei de Moore, com picos entre 1995 e 2010 em que ocorreram dobra na quantidade de transístores em um chip de silício a cada ano, praticamente.

Assim, houve uma grande expectativa gerada por esse crescimento acelerado da computação em todo o mundo industrializado; surgiram novas linguagens de programação, e os sistemas inteligentes viraram assunto constante em empresas e no meio acadêmico.

Em face desse aumento paralelo de complexidade, tornou-se viável realizar análises em sistemas altamente complexos de geração e distribuição de energia elétrica usando computadores com vários núcleos, capazes de efetuar cálculos paralelos. Um dos vários problemas que assolam esse tipo de sistema é o *flicker*, uma variação na intensidade luminosa em lâmpadas causada por oscilações eletromecânicas [3]. O problema já foi anteriormente abordado em [4], pelo método dos Fatores de Participação Estendidos, e se revelou bastante preciso e minucioso, e cujos resultados intermediários são úteis para tirar conclusões importantes a respeito da ocorrência do fenômeno.

No entanto, o método demanda bastante tempo e poder de processamento, além de ter que ser recalculado todas as vezes que os dados de entrada foram diferentes. Assim sendo, deseja-se fazer uso de um método que seja preciso, enquanto bom aproximador do cálculo correto, sem, no entanto, demandar tamanho poder de processamento, nem tempo. Para isso, foram implementados vários métodos (C4.5, KNN, K-*means*, aproximação polinomial, regressão linear multivariável, todos descritos detalhadamente no apêndice A) consagrados na literatura referente, de classificação e aproximação numérica, sendo que o que obteve o melhor resultado foi a rede neural artificial do tipo Perceptron Multicamadas.

1.1 Proposta do trabalho

Os objetivos deste trabalho, portanto, são dois:

- encontrar um método que seja bom o suficiente para ser usado em termos de

eficiência (precisão dos resultados e tempo de cálculo);

- tendo em vista que o resultado será uma aproximação da realidade, aplicar um método estatístico que faça uso da distribuição normal original das variáveis de entrada para discretizar as saídas obtidas.

1.2 Estrutura da monografia

O corpo do trabalho apresentará o problema, a solução que de fato atendeu aos requisitos, e os resultados da mesma. Outras abordagens aparecem no apêndice, com cada método sendo explicado e logo em seguida com o desempenho obtido pelo mesmo.

- Capítulo 2: é apresentado o problema, de pontos de vista técnico, físico e computacional;

- Capítulo 3: a rede neural, a melhor solução encontrada para o problema, é apresentada, desde os detalhes matemáticos envolvidos na convergência, o algoritmo em si e os conceitos fundamentais que explicam porque a rede funciona.

- Capítulo 4: a rede neural é construída via MATLAB [5], e passo a passo explicada a construção da mesma em pseudocódigo; além disso, já é apresentada a abordagem de estudo do resultado da rede neural, de modo a aprimorar o mesmo com um método estatístico baseado nas médias das características da saída.

- Capítulo 5: os resultados são exibidos, e um novo algoritmo, baseado na estatística obtida anteriormente, é construído de modo a incrementar a eficiência da solução da rede neural, e comparam-se os resultados antes e depois do mesmo ser aplicado.

- Capítulo 6: uma conclusão retomando todo o trabalho, com previsões de melhorias na solução encontrada e utilização para prever o fenômeno em tempo real.

- Apêndice: aborda diversos outros métodos considerados referências para problemas da natureza do que se tem em questão, mas que não foram tão bem-sucedidos; pode ser visto também como um alerta para que não se utilize os métodos em questão em um problema análogo, uma vez que já foram testados sem bons resultados.

Capítulo 2: Apresentação do problema: cálculo dos tempos de flicker em sistemas de distribuição de energia elétrica.

O flicker é uma alteração na luminosidade de lâmpadas ocasionada por flutuações de tensão na linha de transmissão. Segundo o IEEE [3], “esse fenômeno é o resultado da aplicação de uma carga no conversor, seguindo de uma liberação dessa carga e então uma reaplicação algum tempo depois [...] Se esse processo acontece em uma frequência tal que o olho humano consegue distinguir, e se a queda de tensão resultante no sistema for grande o suficiente, uma modulação no brilho de lâmpadas incandescentes será detectada. [...] A oscilação no brilho das lâmpadas pode acontecer, mesmo sem ser detectável pelo olho humano, no entanto. ”

Entre as causas do fenômeno, segundo [3], são citadas cargas com ciclo variável, cuja frequência de operação produz uma modulação da magnitude da tensão da rede na faixa de 0 a 30Hz. Nessa faixa de frequências o olho humano é extremamente sensível às variações da emissão luminosa de lâmpadas incandescentes, sendo que a máxima sensibilidade do olho é em torno de 9Hz. Por ser um fenômeno normalmente associável com o ciclo das cargas assume-se, como regra, que a causa primária do fenômeno é a variação da corrente dessas cargas, que modula a tensão da rede, afetando assim as lâmpadas.

No entanto, segundo [4], a conexão de geradores síncronos nos sistemas de distribuição de energia altera a dinâmica da mesma, fazendo com que o problema, que inicialmente aparecia apenas na transmissão, ocorra também na distribuição. O fenômeno pode atingir níveis de irritação e até mesmo de periculosidade se a duração for longa e frequência da oscilação estiver em uma faixa crítica (figura 1), além de comportamentos não-desejados para o gerador, dado que a oscilação é eletromecânica. Tais características justificam, portanto, uma análise mais detalhada do problema, com o intuito de entende-lo e preveni-lo.

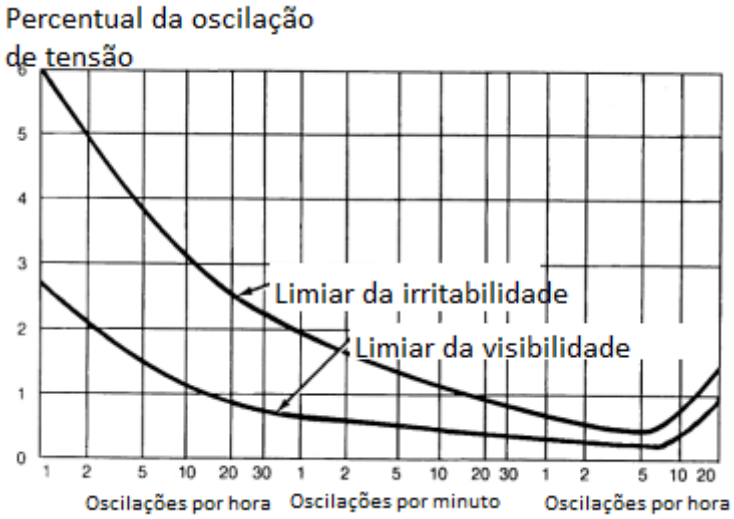


Figura 1: curva de Flicker. Fonte: *IEEE Power Engineering Society, IEEE Recommended Practice for Measurement and Limits of Voltage Fluctuations and Associated Light Flicker on AC Power Systems*, 2005, vol. 2004, março.

2.2. Apresentação do problema computacional

Este trabalho, se baseia em grande parte nos resultados obtidos em [4]. Assim sendo, é preciso explicar a filosofia do artigo citado, bem como sua metodologia, para deixar claros os objetivos e a metodologia deste.

O método que faz uso dos Fatores de Participação Estendidos foi dividido em duas partes. Em [4], foi apresentada uma metodologia para tratar o *flicker* usando Fatores de Participação Estendidos, partindo de uma geração estatística dos valores das variáveis que importam para a modelagem do sistema dinâmico, e então, calcular os tempos de flicker caso os dados em questão apontem para uma elevada probabilidade da ocorrência de oscilação eletromecânica, como é explicado abaixo.

Tal abordagem se revelou muito precisa, e permitiu tirar conclusões importantes a respeito da duração do *flicker*; no entanto, esbarra em um empecilho técnico, que não pode ser resolvido usando a metodologia vigente: o cálculo consome um enorme recurso computacional, demorando um tempo demasiado longo, variando de algumas horas, para obter os valores precisos do tempo de duração do *flicker*.

O objetivo do presente trabalho é, então, usando os dados previamente obtidos em [4] para obter os tempos de *flicker,* desenvolver algum método para diminuir o tempo de cálculo dos tempos de *flicker*.

2.2.1. Modelagem do problema

Como visto em [4], o sistema dinâmico usado como teste para o estudo do

flicker apresenta as seguintes equações:

$$\dot{\delta} = \omega_0 \omega \tag{1}$$

$$\dot{\omega} = \frac{1}{2H}\left(P_m - E'_q\, I_q\right) \tag{2}$$

$$\dot{E}'_q = \frac{1}{T'_{do}}\left(E_{fd} - E'_q + (x_d - x'_d) I_d\right) \tag{3}$$

$$\dot{V}_o = \frac{1}{T_R}\left(V_t - V_0\right) \tag{4}$$

$$\dot{E}_{FD} = \frac{1}{T_e}\left(K_e\left(V_{ref} - V_t\right) - E_{FD} + E_{FDo}\right) \tag{5}$$

$$\dot{x}_1 = \frac{K_{pss}}{T_\omega}\left(\omega + T_\omega \dot{\omega}\right) \tag{6}$$

$$\dot{x}_2 = \frac{1}{T_2}\left(x_1 - x_2 + T_1 \dot{x}_1\right) \tag{7}$$

$$\dot{V}_{esp} = \frac{1}{T_4}\left(x_2 + T_3 \dot{x}_2 - V_{pss}\right) \tag{8}$$

$$V_q = V_\infty \cos\delta - x_e I_d \tag{9}$$

$$V_d = -V_\infty \sin\delta - x_e I_q \tag{10}$$

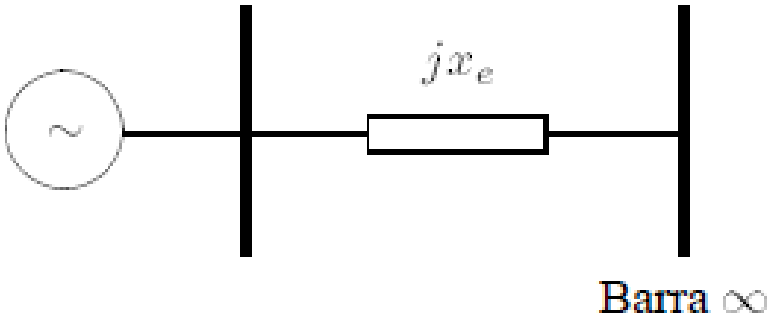
$$V_q = E'_q - x'_d\, I_d \tag{11}$$

$$V_d = -x'_d\, I_q \tag{12}$$

$$V_t = \sqrt{V_d^{\,2} + V_q^{\,2}} \tag{13}$$

Nas equações (1) a (13), como indicado em [1], δ é o ângulo da carga, ω a frequência do motor síncrono, P_m sua potência, E'_q a tensão transitória do eixo em quadratura, E_{FD} a tensão no circuito de campo e E_{FD_0}, V_t a tensão terminal e V_o a tensão na saída do transdutor; V_d e V_q são as tensões nos eixos direto e de quadratura da máquina (analogamente, I_d e I_q são as correntes nos respectivos eixos). A variável T_ω indica o torque no rotor, e K_{pss} o ganho de PSS clássico.

Figura 2: modelo de um gerador ligado a uma barra infinita.



Fonte: www.labspot.ufsc.br/~simoes/dincont/dc-cap6.pdf (acessado em 03/07/2017, às 11:21).

Trata-se de uma máquina com barra infinita com características comuns presentes no sistema de geração distribuída. A barra infinita indica que o sistema conectado à máquina é tão maior que a mesma que a frequência e a tensão no mesmo permanece aproximadamente constante [6]. A abordagem utilizada em [4] considerou, em um modelo probabilístico, as variáveis tempo de duração da perturbação, a tensão na barra infinita e a impedância equivalente da máquina, obtidas de uma gaussiana limitada, e qualquer valor aleatório pode ser obtido baseando-se em sua média e seu desvio padrão. Após isso, “de acordo com da probabilidade de ocorrência de cada uma das perturbações definida por uma função de distribuição de probabilidade correspondente, casos de parâmetros referidos a estas perturbações são escolhidos aleatoriamente para formar cada vetor de condições iniciais. Várias simulações são repetidas com o objetivo de criar o conjunto probabilístico de condições iniciais. De modo a assegurar a imprevisibilidade deste processo, este conjunto probabilístico de condições iniciais são compostos por um grande número de vetores. ” [4].

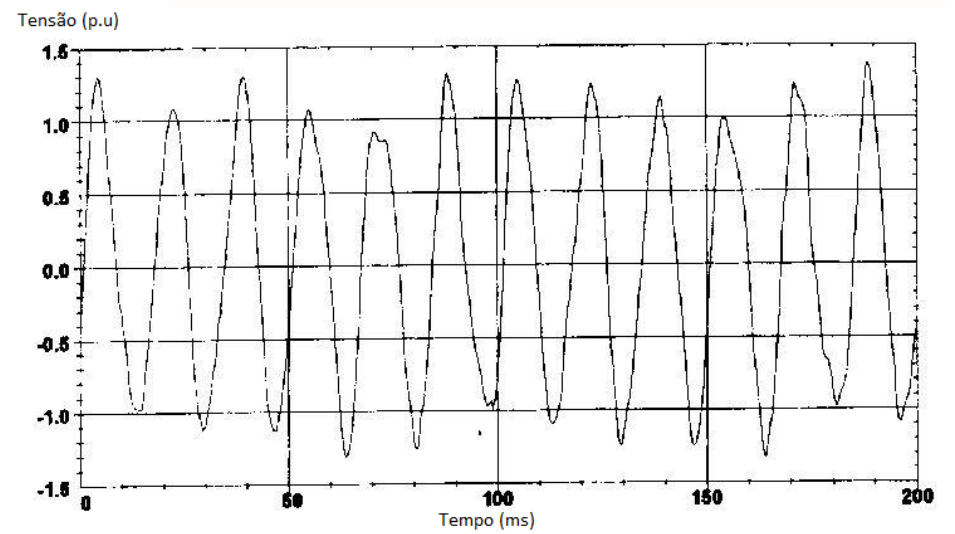
Feito isso, aplica-se o método dos Fatores de Participação Estendidos, que consistem em uma abordagem matemática das variáveis criadas a partir de uma distribuição normal via Monte Carlo, para cada conjunto de condições iniciais. A aplicação do método revela os modos eletromecânicos e, por fim, permitem saber se, dado o conjunto em análise, a probabilidade de ocorrência do Flicker é alta ou baixa.

Por fim, ao ser qualificada como elevada a probabilidade de ocorrência do fenômeno, manipulam-se os dados restantes do peneiramento anterior para obter os tempos de Flicker correspondentes. Esse tempo é dado analisando-se a diferença entre

um pico e um vale da senoide amortecida resultante do fenômeno (figura 3).

Ou seja, tem-se um conjunto de variáveis de entrada, na forma matricial, a serem trabalhadas para se obter uma saída correspondente. Essa estrutura pode ser trabalhada via mineração de dados e redes neurais, de modo a se obter uma classificação/estimação da saída.

Figura 3: *flicker* resultante de operação de forno de arco. O período é definido como a distância entre dois picos.



Fonte: MEEC, Qualidade de Energia Introdução Conceitos fundamentais, disponível em slideplayer.com.br/slide/1686434/6/images/33/Flicker+causado+pela+opera%C3%A7%C3%A3o+de+um+forno+de+arco.jpg, acessado em 02/07/2017, às 19:24.

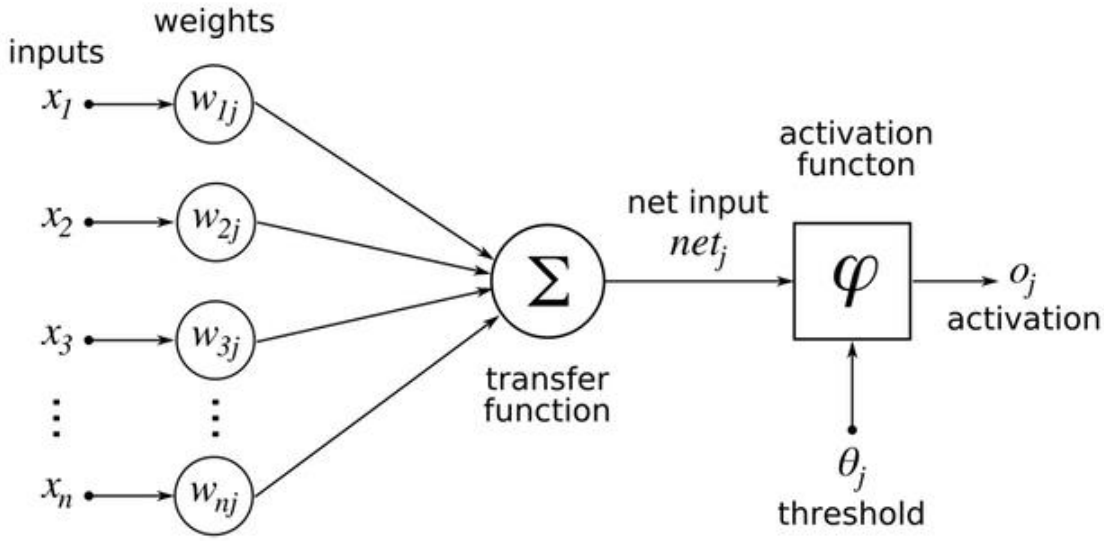
,

O método dos Fatores de Participação Estendidos, embora preciso, se revelou demorado, como citado acima. Nos capítulos seguintes, serão desenvolvidos métodos para obter aproximações do cálculo exato que sejam rápidas e facilmente implementáveis em um sistema de detecção. No apêndice encontram-se métodos não tão bem-sucedidos, de modo que, no futuro, caso venha a se deparar com um problema análogo, esteja registrado a pouca eficácia desses métodos de antemão.

Capítulo 3: Redes neurais artificiais

O assunto de redes neurais remota à década de 1960 [7]: a ideia de produzir via software um objeto que, de certa forma, imite a maneira de o cérebro humano processar informações (aprender, detectar padrões). Tal qual o cérebro biológico, as redes neurais artificiais são formadas por neurônios, chamados *perceptrons*, que se baseiam justamente no funcionamento do neurônio natural para funcionar. O mesmo possui entradas, análogas aos dendritos, que são multiplicados por um peso de modo a atender o que se espera dele; o corpo celular é composto por uma função de ativação e um potencial, semelhante ao potencial eletroquímico dos neurônios biológicos.

Figura 4: analogia entre o neurônio biológico e o neurônio artificial.



Fonte: disponível em http://www.innoarchitech.com/artificial-intelligence-deep-learning-neural- networks-explained, acessado em 02/07/2017, às 19:25.

Ao tentar imitar a maneira de processar informações do cérebro humano (que inclui várias capacidades difíceis de serem implementadas, como alto grau de paralelismo), as redes neurais artificiais se destacam por possuírem as seguintes características: adaptação por experiência, capacidade de aprendizado (resultante do treinamento), habilidade de generalização, tolerância à falhas (assim como o cérebro humano, que não se desfaz por morte de neurônios isolados, a rede neural artificial, ou RNA, é bastante robusta caso uma parte de sua estrutura esteja com problemas), armazenamento distribuído, via as sinapses, e a fácil implementação, que após um processamento complexo em um computador pessoal, pode vir a ser usado até mesmo

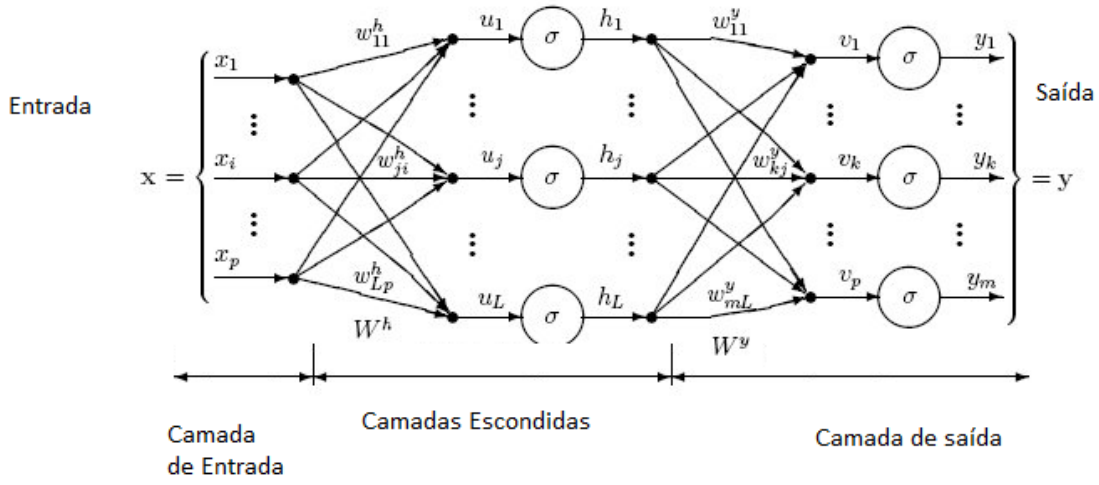
em um microcontrolador simples.

3.1. Rede tipo perceptron de múltiplas camadas (PMC)

A rede perceptrons de múltiplas camadas (*multilayer perceptron*) é uma rede neural que contém pelo menos três camadas: uma, de neurônios que recebem as entradas, uma segunda camada, chamada escondida, e uma terceira camada de saída. As camadas escondidas podem ser em maior número, embora a eficácia dessa estrutura ainda esteja sendo discutida [7].

Montada a rede adequada, é necessário treiná-la com valores conhecidos das variáveis sendo usadas como parâmetros. Ao fim do treinamento, usando o algoritmo do tipo *backpropagation*, a rede terá obtido valores de pesos ponderando as entradas da primeira camada e as saídas da camada intermediária, de modo a aproximar arbitrariamente as saídas dos valores reais fornecidos para o processo de treinamento. Esses pesos são então usados para obter os valores estimados dadas entradas diferentes das testadas [8].

Figura 5: topologia da rede PMC; cada nó representa um neurônio e as setas indicam as sinapses.



Fonte: https://www.dtreg.com/solution/view/21, acessado dia 02/07/2017, às 18:49.

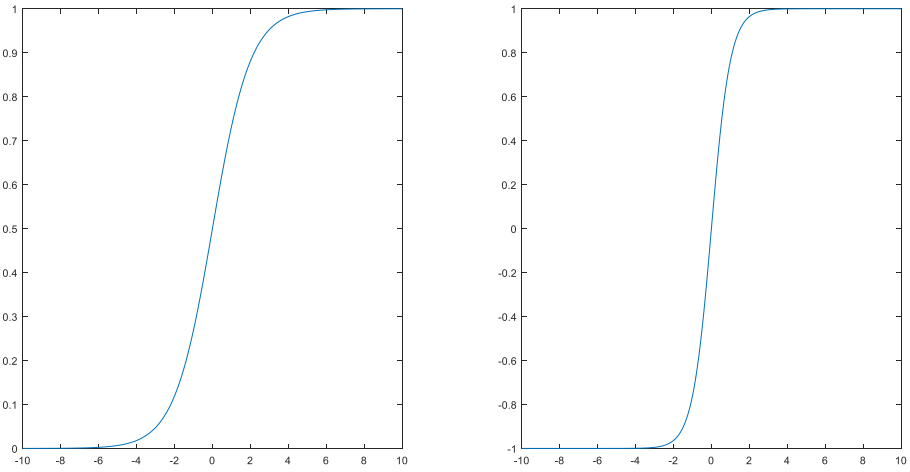
3.2. Características do treinamento do PMC

O treinamento da rede é supervisionado, ou seja, é preciso conhecer certas entradas e respectivas saídas; a partir daí o treinamento se divide em duas partes. Na primeira parte, as entradas são processadas pelas camadas com seu peso sináptico

inicial (randômico), até chegar na saída; seu valor aí é comparado com o da amostra de treinamento, e então passa-se à segunda etapa. Essa etapa, chamada *backwards*, a rede realiza cálculos para ajustar os pesos sinápticos, tendo a função de ativação o papel de ajustar a não-linearidade.

A função de ativação para a o PMC geralmente é escolhida entre a função logística (equação 14) e a função tangente hiperbólica (equação 15). Essas funções são usadas para ponderar as saídas e incluir não-linearidades. O parâmetro λ varia a inclinação da curva.

Figura 6: gráficos da função logística (à esquerda) e da função tangente hiperbólica (gráfico obtido via MATLAB).



A escolha de uma ou outra depende se a saída apresenta ou não valores negativos: como pode ser observado na figura acima, somente a função tangente hiperbólica apresenta valores negativos. De fato, as duas funções apresentam comportamento análogo (é possível chegar em uma das funções através da outra, bastando efetuar uma mudança de variáveis), sendo que a função sigmoide varia entre 0 e 1, e a função tangente hiperbólica varia entre -1 e 1. Tais variações se devem aos limites das funções, a saber:

$$f(x) = \frac{1}{1+e^{\lambda x}} \tag{14}$$

$$f(x) = \frac{e^{\lambda x} - e^{-\lambda x}}{e^{\lambda x} + e^{-\lambda x}} \tag{15}$$

O algoritmo pode ser facilmente implementado no MATLAB, sem se preocupar

com os detalhes matemáticos por trás do mesmo. No entanto, para entender a escolha desse método para o presente trabalho, é necessário entender como funciona o algoritmo.

3.3. Derivação do algoritmo *backpropagation*

É preciso definir a função de erro amostral, correspondente ao erro total entre a saída obtida via cálculo e a saída real. Seja:

$$E(k) = \frac{1}{2}\sum_{j-1}^{n_3}\left(d_j(k) - Y_k^{(3)}(k)\right)^2 \tag{16}$$

onde *d* é o valor desejado (real) da saída *Y,* os índices superiores indicam a camada em que se estão executando os cálculos, e *k* a iteração atual. O erro quadrático médio do aprendizado é dado por:

$$E_M = \frac{1}{p}\sum_{k=1}^{p} E(k) \tag{17}$$

Onde p representa o número de elementos da saída. Agora, é possível usar o erro como parâmetro para ajustar a matriz de pesos da camada de saída. Neste ponto, definimos *I* como sendo os vetores cujos elementos representam a entrada ponderada em relação ao *j*-ésimo neurônio da camada *k.* Usando a regra da cadeia com o gradiente, que indica a direção no espaço de maior crescimento de erro; em qualquer camada, temos:

$$\nabla E^{(k)} = \frac{\partial E}{\partial W_{ji}^{(k)}} = \frac{\partial E}{\partial Y_j^{(k)}}\frac{\partial Y_j^{(k)}}{\partial I_j^{(k)}}\frac{\partial I_j^{(k)}}{\partial W_{ji}^{(k)}} \tag{18}$$

Cada *Y* de índice *k-1* representa a saída de uma camada e a entrada da camada seguinte, e *W* representa a matriz de pesos, da camada *j* e neurônio *i.* Mas, por definição, tem-se que:

$$\frac{\partial I_j^{(k)}}{\partial W_{ji}^{(k)}} = Y_i^{(k-1)} \tag{19}$$

$$\frac{\partial Y_j^{(k)}}{\partial I_j^{(k)}} = g'(I_j^{(k-1)}) \tag{20}$$

$$\frac{\partial E}{\partial Y_j^{(k)}} = -\left(d_j - Y_j^{(k-1)}\right) \tag{21}$$

Substituindo em (16):

$$\nabla E^{(k)} = \frac{\partial E}{\partial W_{ji}^{(k)}} = -\left(d_j - Y_j^{(k)}\right) g'(I_j^{(k)}) Y_i^{(k-1)} \quad (22)$$

Chamando o gradiente local em relação ao *j*-ésimo neurônio da camada de saída, temos:

$$\delta_j^{(3)} = -\left(d_j - Y_j^{(k)}\right) g'(I_j^{(k)}) \quad (23)$$

O ajuste dos pesos precisa ser feito na direção oposta à do gradiente, para minimizá-lo, ou seja:

$$\nabla W_{ji}^{(k)} = \eta \delta_j^{(k)} Y_i^{(k-1)} \quad (24)$$

Onde a constante η é a taxa de aprendizagem da rede, que indica o quanto os pesos se afastam uns dos outros a cada iteração, ou seja, o quanto a rede "abandona" pesos antigos por novos, e aprende quais características são mais importantes. A taxa de aprendizagem deve ser definida empiricamente, pois valores muito baixos podem levar muito tempo para convergir, mas valores elevados indicam que a rede muda os pesos muito rapidamente, podendo alterar pesos que já foram otimizados. O processo é repetido em todas as camadas, e, para a camada de entrada *X*, temos, quando *k* vale 0:

$$Y^{(k-1)} = X \quad (25)$$

3.3.1 O Teorema da Universalidade

Conforme demonstrado por Cybenko [9] em 1989, e estendido por Hornik [10] em 1991, é possível aproximar qualquer função não linear utilizando-se uma rede neural do tipo perceptron desde que sua função de ativação seja sigmoide (logística) ou tangente hiperbólica, se o algoritmo for do tipo *backpropagation.* O teorema formal é conhecido como Teorema da Universalidade, e pode ser expresso da seguinte maneira:

Teorema (da Universalidade): Seja φ uma função não-constante, monotônica, limitada, crescente e contínua. Seja I_m o hipercubo m-dimensional, na qual se define o

espaço de todas as funções $C(I_m)$. Então, dada qualquer função $f \in C$ e $\varepsilon > 0$, existem um inteiro N e constantes reais v_i, b_i e vetores reais w N-dimensionais tais que:

$$F(x) = \sum_{i=1}^{N} v_i \varphi\left(w_i^T x - b_i\right) \tag{26}$$

$$\left|F(x) - f(x)\right| < \varepsilon \tag{27}$$

Onde f e ϕ são independentes.

Corolário: a família de funções F é densa em $C(I_m)$.

No entanto, não é possível obter precisão infinita através de nenhuma série finita de processos. Assim sendo, o objetivo passa a ser aproximar, em tempo hábil, o máximo possível o resultado obtido pela rede neural com os valores de tempo de *flicker* oriundos de [4], via Fatores de Participação Estendidos, e que se admitem como sendo corretos que se sabem serem corretos.

Capítulo 4: Implementação da rede neural PMC

A rede, como todos os procedimentos descritos no capítulo anterior, foi implementada em MATLAB, e constitui-se de duas partes: treinamento e validação.

4.1. Treinamento da rede

O treinamento foi efetuado utilizando-se as seguintes grandezas:

- Variação dos picos de tensão da oscilação (*dv_v*): constitui-se de uma matriz de 10.000 linhas e 99 colunas, sendo que, no artigo original [4], essa variável era usada para estimar a probabilidade de ocorrer flicker; a matriz aqui utilizada apresenta probabilidade que os autores de [4] julgaram considerável o suficiente para associar a cada linha da matriz um tempo de flicker, resultado do cálculo.
- Condição inicial da tensão nominal (*Ai*): vetor de 10.000 linhas.
- Tempo de *flicker* (*tflicker*): tempos resultantes do cálculo previamente realizado pelos autores em [4], obtido via força bruta.

Tomou-se uma parte dos elementos acima para treinar a rede e a outra foi deixada para validação dos resultados. Para otimizar o tempo de cálculo da rede, afinal, esse é um dos objetivos do trabalho, adotou-se a filosofia de utilizar na mesma apenas o absolutamente necessário para que a convergência. Esse critério foi estabelecido tendo em vista o erro relativo do valor encontrado pela rede e o valor correto. Tendo a referência de outros métodos tentados sem muito sucesso (Apêndice A), estabeleceu-se que o valor considerável aceitável para a aproximação deveria ter no máximo 10 % de erro em relação ao valor da simulação [4]

O tamanho da amostra de treinamento foi definido inicialmente como sendo o

valor de 66 % do total, valor este que é o padrão do WEKA (ver apêndice), para efeitos de comparação com o mesmo; no entanto depois de diversos testes (tabela 2, capítulo 5), a porcentagem da amostra usada para treinar a rede foi definida como sendo 10% do total, o que diminui drasticamente o tempo de cálculo, sem alterar a eficiência do PMC. Como indicado em [8], é necessário estabelecer para o treinamento os valores máximos e mínimos da amostra conhecida, de modo a evitar que a curva de erro assuma valores que estejam fora do *range* determinístico.

4.1.1. Algoritmo de treinamento

```
Início
        carregar as variáveis de entrada da rede (dv_v e Ai);
        carregar a variável de saída da simulação (tflicker);
        escolher 10% das variáveis;
        inicializar os neurônios de cada camada;
```

Inicialmente, carregam-se as variáveis de interesse, fornecidas de antemão, e definiu-se o tamanho da amostra de treinamento para 10% do total (ver tabela 2). Apenas para adequação dimensional no algoritmo, a saída, que é um vetor linha, precisa ser transposto para ser operado. Cria-se então uma matriz, representada por $M_{1000x101}$, onde a última coluna é a saída, e as demais funcionam como entrada da rede.

Em seguida começa o treinamento propriamente dito, com a inicialização do número de neurônios de cada camada. Os números foram escolhidos de modo que sejam os menores valores a não alterar a precisão dos cálculos; em outras palavras, tivesse a rede menos neurônios, a precisão seria menor, tivesse a rede mais neurônios, a precisão não se modificaria.

Em seguida, introduz-se uma função (no MATLAB, *newff*), que é a estruturação da rede propriamente dita. A função precisa dos seguintes parâmetros, nessa ordem:

- Na primeira linha, informam-se os valores máximos e mínimos das entradas da primeira camada da rede;
- Na segunda linha, informa-se a quantidade de neurônios de cada camada;
- Na terceira linha, informa-se a função de ativação de cada camada;
- Por fim, informa-se o tipo de treinamento para a rede.

Como os valores da matriz de entrada podem assumir valores negativos, usou-se a função tangente hiperbólica como função de ativação das duas primeiras camadas (figura 6, capítulo 3), seguida de uma ponderação linear na saída.

O método de treinamento escolhido foi de *resilient backpropagation* (“trainrp”); trata-se de uma versão otimizada do algoritmo de *backpropagation* comum, que requer menos memória. Todas as demais funções de ativação e métodos de treinamento foram testados até obter-se um treinamento ótimo: a convergência foi a máxima e o tempo de cálculo mínimo. Seguem abaixo as opções possíveis:

```
% FUNÇÕES DE ATIVAÇÃO
% purelin: Linear;
% logsig: Logística;
% tansig: Tangente hiperbólica;
% satlin: Linear saturada

% ALGORITMOS DE TREINAMEMTO
% traingd:  backpropagation com gradiente descendente;
% traingdm: backpropagation com gradiente descendente momento;
% traingda: backpropagation com gradiente descendente adaptativo;
% traingdx: backpropagation com gradiente descendente adaptativo e momento;
% trainlm:  Levenberg-Marquardt (default);
% trainrp:  backpropagation resiliente.
```

O número de épocas foi testado até se obter um valor que atenda à filosofia do projeto (ou seja, a construção de um modelo aproximador universal, rápido o suficiente para operar em tempo real, e preciso o suficiente para encontrar utilidade prática). Inicialmente colocado como sendo 10.000 (com taxa de aprendizagem inicialmente valendo 0,5), o número de iterações foi variado para baixo, tendo, no entanto, piorado significativamente a rede (tabela 3, capítulo 5). Assim, esse valor foi aumentado com passo de 5.000 até chegar em 30.000, dado que foi observado que, com 31.000 iterações, ou 35.000, ou 40.000, a precisão da rede se mantinha, e o tempo de cálculo aumentava. Em seguida, a taxa de aprendizagem foi variada entre 0 e 1, sendo que o melhor valor para a mesma foi definido empiricamente como sendo 0,5 (tabela 4, capítulo 5)

Por fim, foram testados todas os algoritmos de treinamento, com a estrutura da rede já definida pelos testes supracitados, e o que mais se adequou à filosofia do projeto foi o algoritmo inicial (“trainrp”), como pode ser observado na tabela 4 (capítulo 5)

4.1.2. A aleatoriedade dos pesos (e dos resultados)

Os pesos são inicializados de maneira aleatória pela função *initnw*, como consta a própria documentação MATLAB para o algoritmo *newff.* Essa inicialização é realizada de maneira randômica. Dessa forma, todas as vezes que o algoritmo de produção da rede for executado, os pesos iniciais serão diferentes; como as equações que regem a

convergência dos pesos finais são diferenciais, e, em dado momento (equação 24), é necessário fazer uso dos pesos iniciais, fica evidente que os pesos finais, bem como a eficiência da rede, serão diferentes todas as vezes. Então, a técnica mais lógica para os testes a serem feitos de precisão constituem-se em, ao obter um resultado que seja considerado bom, executar várias vezes o código sem alterar os parâmetros, para não obter um erro de avaliação ocasionado por um resultado viesado (tabela 5, capítulo 5).

4.2. Validação dos resultados da rede

Obtidos os vetores dos pesos da rede mediante treinamento, é a hora de validar o resultado obtido; para tanto, seleciona-se o restante das amostras de entrada, associa-as aos pesos obtidos, e verifica-se a saída calculada em relação à saída original.

4.2.1. Algoritmo da validação

```
Início da validação
        Aplicar a rede já treinada para o restante das amostras de entrada;
        Guardar as saídas da rede em um vetor;
        Comparar as saídas da rede com os valores reais do tempo de flicker;
        Criar vetor contendo o erro relativo de todas as saídas.
        Contar neste vetor quantos erros relativos são inferiores a 10% e quantos
        são inferiores a 5%
```

Obtida a rede, modifica-se a entrada com o restante da amostra, e toma-se a saída, comparando-a com os valores corretos. Em seguida, obtém-se uma matriz na qual se calcula o erro relativo entre a saída desejada e a obtida. Para termos de medir a eficiência da rede, determinou-se quantas das saídas produzidas apresentavam erro relativo em módulo menor que 10%, e quantas apresentavam erro relativo em módulo menor que 5%.

A rede neural artificial, como foi anteriormente explicado, produz resultados diferentes todas as vezes que é executada. Assim sendo, para questões de eventual prototipagem, a rede foi executada várias vezes, até que se encontrasse o melhor resultado. Tal procedimento foi realizado automaticamente no código, com a simples inserção de um comando de parada se a rede melhorasse o desempenho para valores pré-estabelecidos. Além disso, uma vez encontrados pesos que tornem o cálculo suficientemente precisos, basta calcular todas as entradas que forem fornecidas no futuro com os mesmos pesos; o teorema da universalidade garante os pesos finais produzidos pela rede são capazes de “encontrar padrões escondidos” na estrutura dos dados, para qualquer entrada daquele tipo.

4.2.2. A discretização dos dados

Há de se levar em conta o fato de os tempos de flicker possíveis assumirem apenas valores discretos; com essa informação, é possível fazer verificações com os valores obtidos pela rede neural, de modo a melhorar a precisão da mesma. A rede vai retornar valores que se aproximam dos valores entendidos como corretos (daí a importância de obter boas aproximações, uma vez que, se forem boas o suficiente, é possível trabalhar os dados e chegar no valor original).

Os tempos de flicker originais se apresentam em 177 valores distintos, sendo o menor deles 0,04 segundo e o maior deles 8,32 segundos. Com a intenção de prototipar os pesos obtidos, é necessário escrever um código para correção dos valores, de modo que retornem aos valores discretos. Para tanto, é necessário realizar uma análise estatística do erro encontrado.

Tal análise consistiu em encontrar uma distribuição para a média dos erros, de modo a tornar possível um teste de hipóteses em análises genéricas usando a prototipagem obtida da rede.

Aqui fica claro que a técnica se torna completamente estatística; portanto, é necessário analisar um grande número de amostras para se obter uma síntese não-viesada dos dados; como os dados gerados na simulação em [4] provêm de uma distribuição normal, o Teorema do Limite Central garante que, com um número suficientemente grande de saídas, é possível encontrar a média dos erros de cada saída.

Em cumprimento à essa filosofia, obtiveram-se 1000 saídas distintas da rede, e analisaram-se os resultados provenientes dos erros médios de cada tempo de flicker. Em seguida, foi possível obter as estatísticas relevantes para a solução.

Definidos os erros médios de maneira confiável (1000 iterações se configuram como “tendendo a infinito” no Teorema do Limite Central), tratou-se de estabelecer os intervalos em que a rede poderia operar para discretizar os dados, como indicado pelo seguinte fluxograma:

```
Início da discretização
        Verificar quais amostras apresentam erro médio menor que a distância
        entre dois valores discretos;
        estabelecer o intervalo de confiança das amostras encontradas como
        [a-erro*a;a+erro*a];
        Percorrer o vetor de saída aplicando o condicional acima;
```

Capítulo 5: Resultados

Os resultados serão apresentados em duas partes: a primeira envolve a capacidade da rede neural propriamente dita, enquanto a segunda trata da correção estatística a ser incluída *a posteriori*.

5.1. Eficiência da rede

A eficiência da rede levará em conta o tempo de cálculo e a precisão dos resultados.

5.1.1. Número de neurônios

O número de neurônios fio inicializado aleatoriamente como sendo 10 nas duas primeiras camadas e 1 na última (esta será sempre unitário pois possui apenas uma saída). O número de iterações foi definido inicialmente como sendo 10.000 e a taxa de aprendizagem como sendo 0,5; o algoritmo de treinamento escolhido nessa etapa foi o *default* da função, o *trainrp.*

Os valores foram então alterados até encontrar-se uma eficácia no tempo de cálculo e na precisão que não variassem se o número de neurônios fosse aumentado. Seguem abaixo os valores testados para comparação.

Tabela 1: determinação empírica do número de neurônios por camada.

Número de neurônios por camada		**Amostras com erro menor que 10 % (%)**	**Tempo (s)**
Primeira	**Segunda**		
10	10	62,4	378
10	15	65,3	414
10	20	67,7	486
15	20	67,6	510
20	20	67	528
20	25	69,1	576
20	30	70,5	666
20	35	71,2	696

20	40	80,5	738
20	45	80,6	810
25	40	81	954

5.1.2. Tamanho da amostra de treinamento

Determinados os neurônios em cada camada, o total das amostras de treinamento foi reduzido até se obter um valor para o mesmo em que a precisão da rede fosse a mesma que para conjuntos maiores. Isso reduziria o tempo de processamento e manteria a precisão da rede neural. Seguem os resultados obtidos:

Tabela 2: variação da precisão da rede neural com o tamanho da amostra de aprendizagem.

Amostras de treinamento	Amostras com erro menor que 10 % (%)
6000	80,5
5000	80,4
4000	80,5
3000	80,5
2000	80,7
1000	80,5
900	79,3
800	77,1

5.1.3. Número de épocas

Determinados os neurônios em cada camada, o número de épocas foi alterado para cima e para baixo do valor inicial de 10.000, até que se encontrasse um valor com a melhor aproximação do valor real, desde que o tempo de cálculo fosse pequeno ainda. Os resultados dessa pesquisa seguem na tabela abaixo:

Tabela 3: determinação do número de épocas empiricamente, chegando ao limiar onde variar muito o número de iterações aumenta muito pouco a eficiência da rede, e aumenta muito o tempo de processamento.

Número de épocas	Amostras com erro menor que 10 % (%)	Tempo (s)
9000	77,9	236
10000	80,5	246
12000	81,3	258
15000	82,2	290
20000	82,8	320
25000	84,5	360
30000	88	354
35000	88,1	382
100000	88,3	1502

5.1.4. Taxa de aprendizagem

Com tudo o que foi estabelecido até aqui, o próximo passo foi encontrar a taxa de aprendizagem ideal para a rede, seguindo a filosofia de a mesma ser rápida e precisa. Variou-se então o valor da taxa de aprendizagem entre 0 e 1, obtendo-se os seguintes resultados:

Tabela 4: variação da eficiência da rede com a taxa de aprendizagem.

Taxa de aprendizagem	Amostras com erro menor que 10 % (%)	Tempo (s)
0.01	75,1	119
0.02	73,2	111
0.05	77,1	115
0.08	78,3	121
0.1	85,2	122
0.15	85,3	118
0.2	85,9	112
0.3	86,3	112
0.4	86,5	113
0.45	87,2	125
0.5	88	117
0.55	86,6	126
0.6	83,2	116
0.7	84,2	125
0.8	78,9	123
0.9	81,1	118
1	80	116

5.1.5. Algoritmos de treinamento

Por fim, foram testados todos os algoritmos presentes no MATLAB, de modo a determinar o melhor deles. Os testes estão exibidos na tabela 5 abaixo (o último algoritmo não pôde ser executado):

Tabela 6: testes dos algoritmos presentes no MATLAB para finalizar a rede neural.

Algoritmo	Amostras com erro menor que 10 % (%)	Tempo (s)
trainrp	88	117
traingd	12,92	101
traingdx	22,3	108
traingdm	25,03	107
traingda	17,2	108
trainlm	?	infinito

5.1.6. Forma final da rede

Com as características estabelecidas empiricamente, como mostram as tabelas 1 a 5, a rede (tabela 6) foi capaz de obter os seguintes resultados, em termos de eficiência (contando os melhores resultados obtidos após rodar o código diversas vezes):

Tabela 6: características finais da rede neural artificial.

Iterações	Neur. 1ª camada	Neur. 2ª camada	Taxa apr.	Algoritmo	Am. Trein.
30000	20	20	0,5	trainrp	1000

- Precisão: 84,1667 % (7570, do total de 9000 usadas para a validação) das saídas da rede possuem erro relativo ao seu equivalente correto menor que 10 %, ao passo que 65,1 % (5859) possuem erro inferior a 5 %.

- O cálculo da rede demorou em média menos de um minuto no computador usado para o trabalho.

O melhor dos resultados foi salvo em uma variável com o uso da função *net.LW*, que armazena os pesos de cada camada em uma *struct* com a dimensão do número de camadas da rede.

Na tentativa de obter um resultado melhor, alterando-se o número de neurônios para 50 nas duas camadas, e o número de épocas para 30000, encontraram-se os seguintes resultados:

- Precisão: 88 % das saídas (7920) apresentara erro menor que 10 % e 69 % das saídas (6210) apresentaram erro menor que 5 %;

- O tempo de cálculo passou para cerca de 3 minutos.

Mesmo aumentando exageradamente os valores das variáveis de interesse, não se obtiveram resultados melhores.

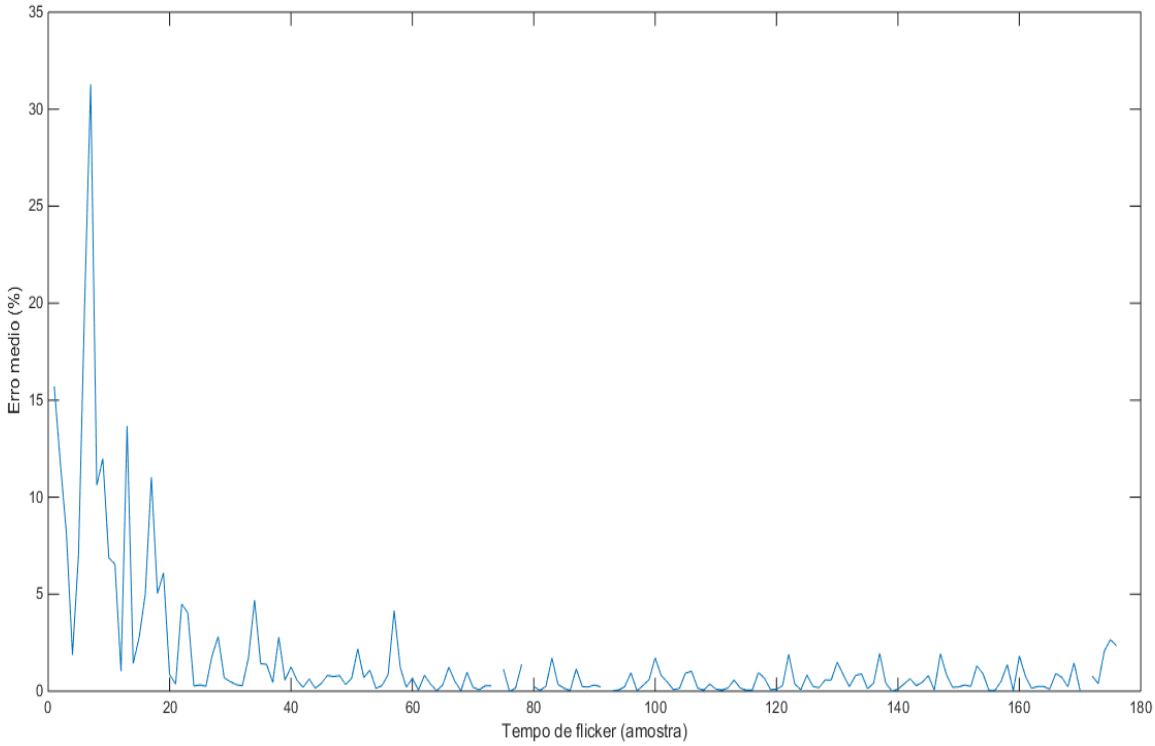
Assim sendo, essa iteração foi a última em que se gravou a variável *Wmelhor*, que será tomada como padrão para este problema, bem como qualquer desta mesma classe.

5.2. Tratamento estatístico do erro e comparação com os Fatores de Participação Estendidos

Após 1000 treinamentos distintos da rede neural, os quais foram realizados utilizando amostras de treinamento variáveis, tratou-se de avaliar as propriedades da distribuição do erro cometido pela rede enquanto aproximador universal. Observa-se, na figura abaixo, que os erros se tornam elevados nas amostras de tempo de flicker

inferiores a 15, correspondente ao tempo de flicker 0,41 segundo.

Figura 7: erro percentual médio obtido após 1000 interações da rede neural, avaliada em função da posição ordenada da amostra de tempos de flicker.



O valor discreto dos tempos de flicker, que são em número de 177, estão irregularmente distribuídos entre os valores 0,04 segundo e 8,32 segundos. Em geral dois tempos consecutivos apresentam apenas um centésimo de diferença um do outro, de modo que a correção se torna praticamente impossível para valores mais elevados, sendo mais plausível para os valores menores de tempos de flicker. Tomando uma saída de 2,30 segundos (posição 68 do vetor), por exemplo; o erro médio apresentado pela mesma é de apenas 1,4 %, valor de erro de aproximação considerado muito bom em termos de engenharia. No entanto, a próxima saída vale 2,31 segundos, e, para que não houvesse dúvida sobre qual o valor a ser aproximado, o erro da rede teria que ser

no máximo 0,44 %, o que, para a rede é algo inviável.

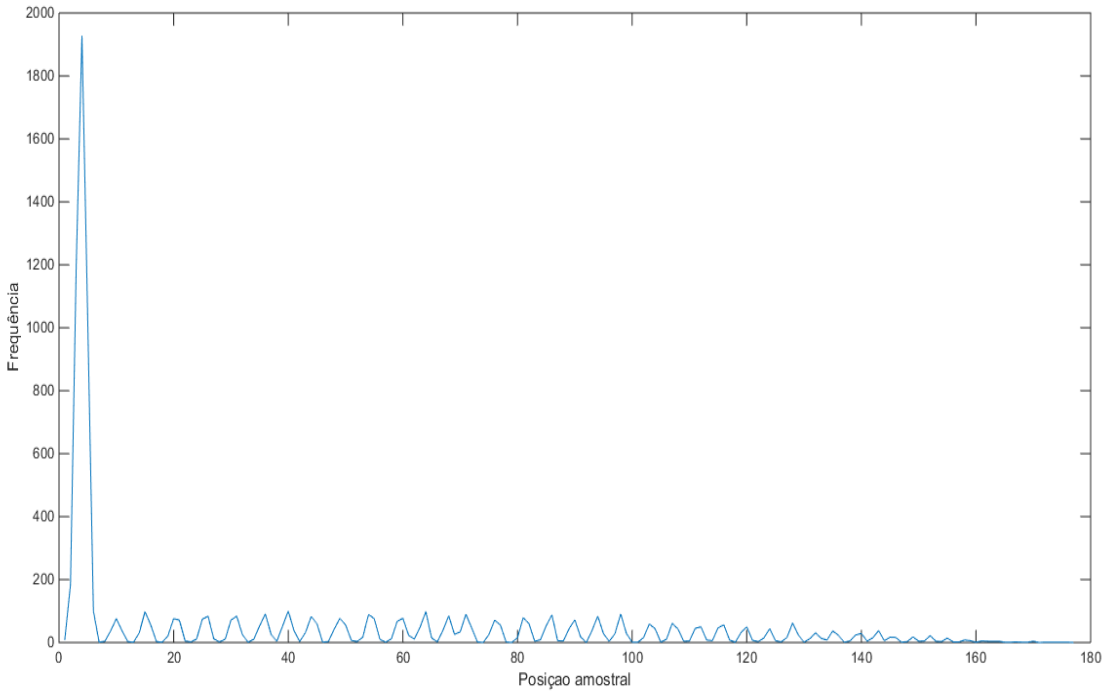
No entanto, para os valores menores, que coincidem com os mais frequentes, como mostra a tabela 7 abaixo, esse cálculo se torna mais fácil, uma vez que é possível, conhecendo-se a distribuição de erro, e a faixa de valores associada aos mesmos, é mais fácil prever e reverter os erros mais grosseiros.

Tabela 7: relação entre os três tempos de flicker que, somados, perfazem quase metade da amostra.

Tempo de flicker (s)	Posição (/177)	Frequência (/9000)	Erro médio (%)
0.06	3	1196	8.14
0.07	4	1928	1.88
0.08	5	1031	-7.04

Além disso, convém observar que alguns valores são irrelevantes, aparecendo poucas vezes, alguns somente uma vez.

Figura 8: frequência dos tempos de flicker em função da posição.



A importância da obtenção da média é de se obter um grau de confiabilidade no sinal do erro, para evitar que a correção caia no valor anterior ou posterior ao correto por engano.

A correção foi realizada por meio de um condicional para os valores obtidos com a rede usando os pesos como em *Wmelhor*. Dessa forma, é preciso inicializa-la antes de efetuar qualquer cálculo.

Para fins de comparação e averiguação da eficiência do método, foram contabilizados quantos resultados da rede batiam exatamente com o valor discreto antes de realizar a correção pelo método estatístico, e depois da aplicação do mesmo.

A rede é inicializada com os melhores pesos obtidos, para em seguida calcular a saída. As linhas seguintes obtêm os valores que coincidiam com a saída discreta antes de ser efetuada a correção estatística. Antes, nenhum valor da rede coincidia com o seu correspondente discreto. Após a correção, 3037 valores foram corrigidos. Além disso, a correção foi aplicada aos valores mais frequentes na saída, de modo que a abordagem estatística garante como bastante segurança que essa correção pode também ser estendida a demais problemas desta mesma classe.

A correção foi efetuada tendo em vista a média dos erros das variáveis mais frequentes (tabela 8), tomando aproximadamente o dobro do valor médio, para mais e para menos, já que, devido à grande distância proporcional entre os valores, não se corre o risco de intercalar as médias.

Assim, resumidamente, após realizar a correção estatística, encontram-se as seguintes informações:

Tabela 8: resultados do algoritmo após a correção estatística.

Erro (%)			
0	0<erro<5	5<erro<10	>10
33,8	42	5,4	18

Aqui percebe-se que a aplicabilidade da rede se dá em caso de prototipagem para estimação em tempo real, dentro dos limites de erro estabelecidos (que podem ser melhorados no futuro), uma vez que, apesar de ser muito rápida, a precisão do método de Fatores de Participação Estendidos é absoluta, embora o tempo de cálculo seja da ordem de horas.

Capítulo 6: Conclusão e considerações finais

Neste trabalho, foram utilizadas diversas técnicas de aproximação e classificação para obter um resultado aproximado do real, sendo que o que se destacou por sua precisão e rapidez foi a rede neural artificial do tipo perceptron multicamadas. Fazendo uso desse método, foi possível obter uma estrutura de pesos entre os neurônios que aproximava 88% dos resultados da rede com erro inferior a 10%, com a enorme vantagem computacional e com a garantia de extensão dos resultados obtidos para qualquer problema semelhante, com eficiência próxima ao que foi calculado neste trabalho.

Comparativamente, usando os Fatores de Participação Estendidos, o resultado é absolutamente preciso, porém demorado, sendo inviável implementar o mesmo para aplicações em tempo real, objetivo que tem se visado desde o início do projeto. A rede possui ainda a seguinte vantagem sobre outros métodos, incluindo o dos Fatores de Participação Estendidos: ela é capaz de generalizar os resultados, o que pode ser visto uma vez que, com apenas 10 % do total de amostras para treinar, ela consegue acertar com boa precisão os outros 90 % (ou seja, com entradas dv_v e Ai que não haviam sido usadas para treinamento).

Além disso, utilizando-se um método estatístico baseado nas médias dos erros para tempos discretos de flicker, com o qual foi possível zerar completamente o erro em um terço das saídas, havendo, no entanto, aumentado de 12 % para 18 % a quantidade de saídas cujos erros são maiores que 10 %. A vantagem de se obter esse ganho é discutível, e dependerá de estudos posteriores.

Devido à grande proximidade entre os valores discretos, a correção para valores mais elevados se torna mais difícil. Assim, como perspectiva futura para a continuidade do projeto iniciado por [4], pode-se indicar um estudo mais aprofundado, e com espaço amostral maior, do resultado evidenciado pela figura 9, de modo que, se a proporção dos erros se mantiver, será possível escrever um algoritmo que realize testes de hipóteses para chegar no limite do intervalo de confiança do processo.

Desta forma, a metodologia que aqui foi introduzida, bem como os dados da rede neural que foram otimizados, poderão ser utilizados no futuro em tempo real para prever ou evitar o *flicker* em sistemas de distribuição de energia.

Referências Bibliográficas

[1] BLACK, C., *Eventos relacionados ao superciclo de preços das Eventos relacionados ao superciclo de preços das commodities no século XXI,* Indic. Econ. FEE, Porto Alegre, v. 40, n. 2, p. 67-78, 2013.

[2] DISCO, C.; van der MEULEN, B. (1998). *Getting new technologies together* , New York: Walter de Gruyter. pp. 206–207.

[3] NASR-AZADANI, E.; CANIZARES, C.; OLIVARES, D.;K. BHATTACHARYA, K; *Stability analysis of unbalanced distribution systems with synchronous machine and dfig based distributed generators*, Smart Grid, IEEE Transactions on, vol. 5, no. 5, pp. 2326–2338, Setembro de 2014.

[4] MORACO, A. G. M. et al; *Abordagem para Avaliar a Ocorrência de Flicker Induzida por Oscilações Eletromecânicas em Sistemas com Geração Distribuída,* VI Simpósio Brasileiro de Sistemas Elétricos.

[5] https://www.mathworks.com/products/matlab.html, site oficial do produto.

[6] www.labspot.ufsc.br/~simoes/dincont/dc-cap6.pdf (acessado em 03/07/2017, às 11:21), notas de laboratório da disciplina de estabilidade em sistemas de potência da UFSC.

[7] HAYKIN, Simon. *Redes neurais: princípios e prática*. Tradução: Paulo Martins Engel. - 2.ed. - Porto Alegre: Bookman, 2001.

[8] NUNES, I.; SPATTI, D. H.; FLAUZINO, R. A. *Redes Neurais Artificiais,* ArtLiber.

[9] CYBENKO., G. (1989) *Approximations by superpositions of sigmoidal functions*, Mathematics of Control, Signals, and Systems.

[10] HORNIK, K.; (1991) *Approximation Capabilities of Multilayer Feedforward Networks*, Neural Networks.

[11] WITTEN, I.; FRANK, E.; *Data Mining: Practical Machine Learning Tools and Techniques,* Morgan Kaufmann Series in Data Management Systems.

[12] WekaMOOC, canal de um dos autores do livro citado acima, em [8], disponível em https://www.youtube.com/user/WekaMOOC.

[13] WU, X. et al.; *Top 10 algorithms in data mining,* Osaka papers, 2007.[14] SHANNON, *A Mathematical Theory of Communication, 1948.*

[15] STONE C. J. (1977). *Consistent nonparametric regression*. Annals of Statistics. 5 (4): 595–620. doi:10.1214/aos/1176343886

[16] MACQUEEN, J. B. (1967). *Some Methods for classification and Analysis of Multivariate Observations*. Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability. University of California Press. pp. 281

[17] Disponível em *https://www.mathworks.com/help/stats/mvregress*

[18] LEGENDRE, (1805), *Nouvelles méthodes pour la détermination des orbites des comètes* [*New Methods for the Determination of the Orbits of Comets*].

[19] Disponível em *https://www.mathworks.com/help/matlab/ref/polyfit.html.*

[20] HAWKING, S. W.; *Information Loss in Black Holes,* disponível em *https://arxiv.org/abs/hep-th/0507171*

Apêndice A: abordagens alternativas

Nesse apêndice, abordam-se os demais métodos desenvolvidos para a solução do problema, sem, no entanto, se obter um resultado aceitável para os padrões de engenharia. Serão discutidos o funcionamento de cada método, os resultados obtidos e as justificativas para o fato de terem falhado, uma vez que todos os algoritmos utilizados são consagrados na área de *data mining*.

A.1. Weka

O Weka é uma ferramenta usada em mineração de dados, escrita inicialmente em Java (embora atualmente seja possível usar o mesmo em Python), com interface bastante intuitiva, desde que se conheça o funcionamento dos algoritmos. A biblioteca do software é muito vasta, contendo algumas dezenas de algoritmos de classificação, indo desde os mais triviais, como o de “chutar” como padrão a classe mais numerosa no treinamento, até algumas redes neurais simples. Trata-se uma ferramenta poderosa que faz uso de aprendizagem de máquinas supervisionado para encontrar padrões de classificação ou aproximadores de funções matemáticas.

Para tratar o problema no software, foram pensadas duas abordagens diferentes para o uso dos diversos algoritmos do mesmo: inicialmente, fazendo uso de algoritmos de classificação de padrões, e em seguida, com aproximadores de função.

Devido ao grande número de algoritmos disponíveis, foram utilizados aqueles que mais foram mencionados nas vídeo-aulas do autor do software [11], [12], em conjunto com os algoritmos mencionados em [13].

O Weka possui como formato base de arquivo aqueles de extensão “.arff”; no entanto, para criar um arquivo legível pelo mesmo, foi necessário utilizar o Excel na extensão .csv. Mais abaixo, em momento mais oportuno, será explicado como proceder para fazê-lo.

A.1.1 Algoritmos de classificação de padrões

Os algoritmos de classificação de padrões fazem uso de características de entrada de um sistema e realizam operações de inter-relacionamento entre as mesmas para produzir um tipo de saída. Assim, elas relacionam diversas entradas a algumas saídas padronizadas.

A priori, o uso desses algoritmos parece uma escolha tola: uma vez que todos os dados são numéricos, a escolha mais sensata parece ser uma classificação numérica, como por exemplo uma regressão, ou uma aproximação polinomial. No entanto, dois foram os motivos que levaram à escolha de tentar métodos desse tipo: eles são constantemente citados pelo autor do livro base [11] como sendo bastante confiáveis e capazes de apresentar um alto grau de precisão em determinadas condições e, além disso, foram citados como os melhores algoritmos de mineração de dados no artigo [13]. No entanto, é evidente que isso não significa que sempre serão funcionais, nem mesmo os melhores (como foi o caso neste trabalho), mas as evidências a favor de seu uso pesam o suficiente para a tentativa.

Para fazer uso do Weka, em qualquer tipo de algoritmo, na extensão .csv, é necessário seguir os seguintes passos:

- abrir um arquivo Excel;
- inserir, na primeira linha do arquivo, os nomes das variáveis separados por vírgula;
- inserir todas as linhas a partir da primeira, com as colunas separadas por vírgula;
- salvar com a extensão “.csv”

Assim, ao usar o explorador do Weka, ele reconhecerá o arquivo. Vale ainda dizer que o próprio software reconhece se o arquivo se baseia em classificação numérica ou por padrões. Isso se deve ao fato de, no último caso, a saída a ser classificada não é um número; dessa forma, caso o arquivo .csv seja do tipo classificatório, o Weka não permite a escolha de algoritmos numéricos.

Seguem abaixo os algoritmos utilizados.

A.1.1.1 J48 (C4.5)

O algoritmo C4.5, que no Weka se chama J48, ficou em primeiro lugar na

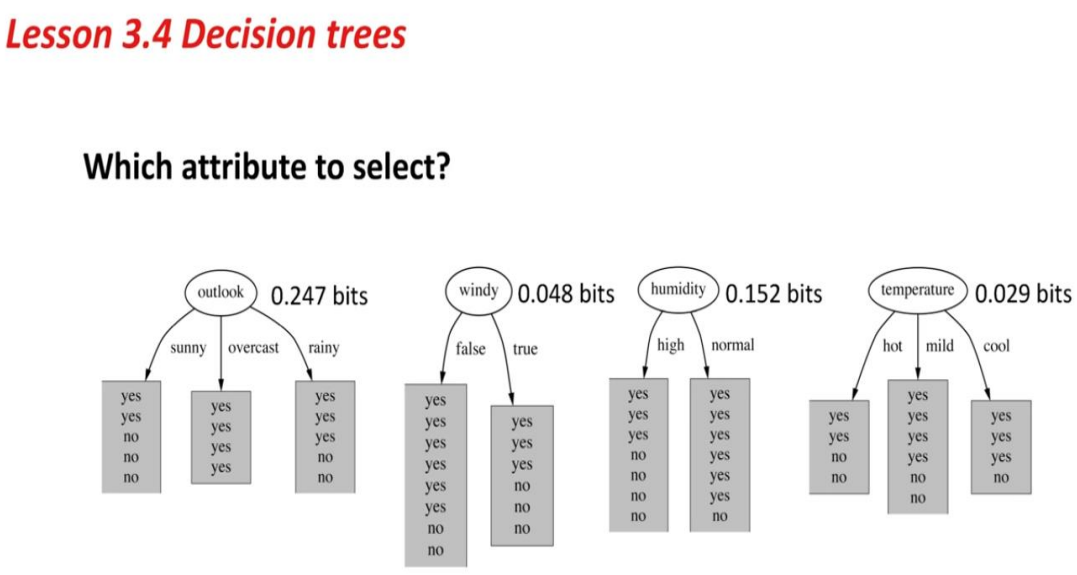
classificação em [13]. Tal algoritmo se baseia na estrutura de árvores de decisões, cujas “podas” de seus “galhos” são realizadas através de uma análise de entropia da informação de treinamento. Ou seja, as decisões são feitas de modo a minimizar a entropia e aumentar o ganho de informação. Da teoria matemática da informação de Shannon [14], tem-se que a entropia de qualquer galho de uma árvore de decisões vale:

$$S(p_1, p_2, p_3, ..., p_n) = -\sum_{i=1}^{n} \log p_i \qquad (1)$$

Essa variável recebe o nome de entropia [14] por dois motivos: o primeiro por ter uma expressão análoga à expressão da grandeza “entropia” retirada da termodinâmica, e o segundo, por se tratar da previsibilidade de um sistema (que na teoria de Shannon se chama de “informação”, e claramente é o que está sendo usado nesta abordagem), atributo que também pode ser indicado em teorias modernas de cosmologia à antiga noção de entropia [20].

Esse cálculo é feito através da escolha de qual atributo deverá ser escolhido para “podar” a árvore primeiro. O seguinte exemplo, tirado da aula “*3.4 – Decision Trees*” da seção “*Data mining with Weka*” [12],

Figura 9: exemplo de como funciona o algoritmo J48 para “podar” as árvores de decisão. Nesse caso, o ganho de informação é maior ao escolher o primeiro caminho, ou seja, o primeiro atributo a ser classificado será se o dia está ensolarado, nublado ou chuvoso. A partir daí, repete-se o cálculo de entropia até todos os atributos terem sido classificados.



Fonte: canal WekaMOOC, aula 3.4 da série “Data Mining with Weka”.

Para fazer uso desse algoritmo no Weka, bem como qualquer um dos diversos algoritmos de classificação, logo aparece um grave inconveniente: é necessário fazer com que a saída, que é um dado numérico, se torne um dado tipo char. Para tal, foi preciso manipular o mesmo de modo a obter valores tipo char em todos os tempos de flicker, o que, devido à quantidade de valores distintos (177), impossibilitou o uso de um código para associar os tempos às letras do alfabeto. Com isso, as substituições foram feitas manualmente, o que, por si só, toma um tempo dispendioso e faz com que o uso desses métodos seja repensado. No entanto, em nome da boa reputação obtida por esses algoritmos, o método foi posto em prática, apesar de penoso. Esse processo teve que ser usado em todos os algoritmos de classificação.

A.1.1.2 Vizinho mais próximo (K Nearest Neighbor - KNN)

O algoritmo do vizinho mais próximo pode ser usado para classificação ou para estimação numérica de funções contínuas. Ele se baseia na distância entre os conjuntos de saída; as distâncias de cada saída entre si são analisadas e participam do processo de aprendizagem supervisionada do algoritmo; determinadas as distâncias e as regiões no espaço de classificação correspondente, o algoritmo passa à fase de classificação.

O Weka usa como critério de distância apenas a distância euclidiana; no MATLAB, entretanto, é possível usar vários tipos de distâncias: euclidiana, euclidiana normalizada, city-block, chess-board, entre outras.

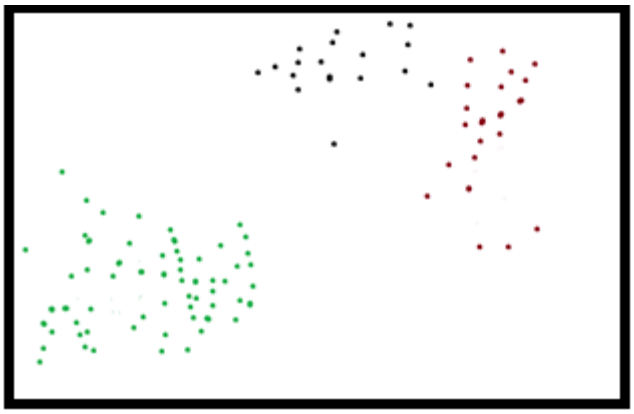
Trata-se de um algoritmo simples de ser implementado (o mesmo está presenta tanto no Weka quando no MATLAB), apresentando, no entanto, graves contrapontos. O primeiro deles é que se trata de um algoritmo guloso, ou seja, demanda não apenas um grande poder de processamento para conjuntos muito grandes (como é o caso do problema abordado), como também demanda bastante memória nesse processamento; além disso, o erro nas classificações tende a zero quando o espaço de treinamento tende a infinito (o que é uma implementação inviável devido ao tempo que demandaria para conjuntos demasiadamente grandes). Outro problema grave, que diz respeito particularmente ao problema em questão é que o algoritmo só funciona para entradas vetoriais m x 1 e saídas m x 1, e, como mencionado, a entrada em questão é 10.000 x 100 e a saída é 10.000 x 1.

Apesar da simplicidade do algoritmo, a fundamentação matemática do mesmo é relativamente complexa, e pode ser devidamente apreciada no artigo *“Consistent*

nonparametric regression" [15].

Devido aos empecilhos do algoritmo, o mesmo foi testado com algumas colunas da entrada, e avaliaram-se a precisão; em seguida, obtiveram-se os resultados de cada iteração para serem processadas da melhor maneira possível. No entanto, a baixa precisão obtida em cada coluna (entre 20% e 40%) inviabilizaram o uso do algoritmo.

Figura 10: um exemplo simples para classificação no espaço usando KNN: após a convergência, os pontos se encontram organizados em regiões distintas, obtidas através de uma métrica de distância. Observa-se que a fronteira dos pontos verdes ficou muito clara, o que não acontece com as regiões preta e vermelha (executando-se o algoritmo de maneira ligeiramente diferente, pontos vermelhos e pretos poderiam trocar de lugar).



2.1.2.3 *k*-means

O algoritmo conhecido como *k*-means, que apresenta um certo grau de "parentesco" com o KNN; no entanto, ele apresenta como novidade uma heurística que o permite convergir rapidamente em problemas np-incompletos.

Trata-se de um método de *clustering* que faz agrupamentos baseados na média [13]. Uma maneira simples de observar o funcionamento desse algoritmo é usando um artifício matemático chamado diagrama de Voronoy, que exemplifica a classificação realizada pelo algoritmo. De certa forma, a classificação por *k*-means considera as saídas como objetos e, após o treinamento, verifica o quão diferente são esses objetos.

O algoritmo pode ser resumido da seguinte maneira:

- Dado o conjunto de pontos a serem classificados, seleciona-se aleatoriamente *k* desses pontos para serem o centroide de suas respectivas regiões;

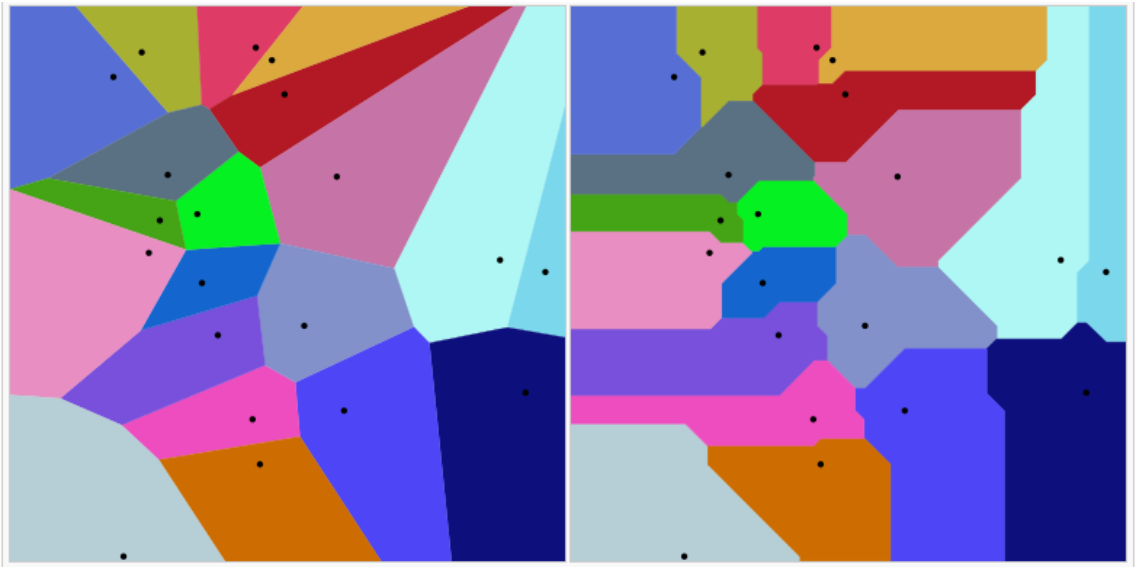
- São gerados diagramas de Voronoy para cada *cluster*;

- Os centroides (centros de massa) de cada região se tornam os novos pontos médios;

- Repete-se o algoritmo, até obter convergência.

Este método difere do anterior por ser não-supervisionado, uma vez que os pontos não possuem classificação externa. Os resultados, no entanto, foram aproximadamente iguais, o que também levou ao seu abandono.

Figura 11: diagramas de Voronoy, de 20 pontos, usando duas métricas diferentes: euclidiana (à esquerda) e Manhattan (à direita). A separação de cores no diagrama se dá, após o processamento do algoritmo, indica as regiões no espaço os quais apresentam a mesma característica que o ponto interior à ela.



Fonte: https://en.wikipedia.org/wiki/Voronoi_diagram (acessado em 14/11/2016, às 2:03).

A.2 Algoritmos de aproximação numérica

Tendo em vista o desempenho dos algoritmos de classificação, além do empecilho de ter que manipular manualmente a saída, ao atribuir a cada uma delas uma característica diferente, passou-se aos algoritmos de aproximação de funções. Essa

abordagem é mais intuitiva já que todos os dados trabalhados no problema são numéricos.

A.2.1 Regressão linear

A regressão linear é uma análise de dados que busca uma relação linear entre entrada e saída. O exemplo mais clássico dessa técnica é o método de mínimos quadrados, usado em análise bidimensional, ou seja, uma entrada vetorial m x 1 e uma saída m x 1. Dessa forma, sendo X a entrada e Y a saída, a ideia é encontrar uma equação do tipo

$$Y = AX + B \quad (3)$$

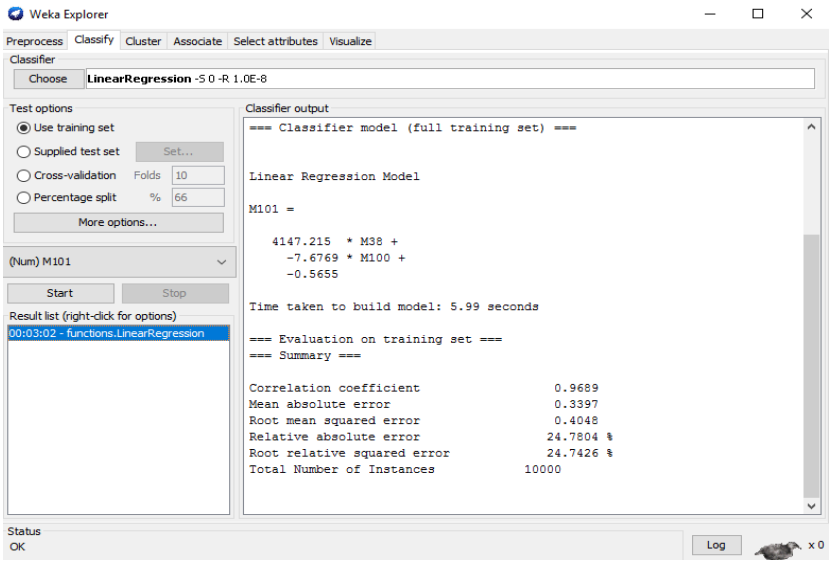
No caso do flicker, é necessário fazer uma análise multidimensional. No Weka isso não é possível; no entanto, o MATLAB permite essa análise, com o comando *mvregress*. Na própria descrição do comando, encontra-se a descrição do mesmo [17]: "*`beta = mvregress(X,Y)` returns the estimated coefficients for a multivariate normal regression of the d-dimensional responses in `Y` on the design matrices in `X`.*".

A ideia é obter uma matriz de coeficientes que, ao ser multiplicada pela entrada e somada com um termo relacionado ao erro de dispersão, resulte em uma saída que se aproxima o máximo possível [18] da saída real. No caso multidimensional, a equação resultante do produto das matrizes resulta em um hiperplano. Para atender ao coeficiente independente *B*, é necessário inserir uma coluna de 1's na matriz *A*. Dessa forma, o algoritmo consegue encontrar uma matriz *x* que representa corretamente os coeficientes, e os pontos encontrados estão à menor distância euclidiana do hiperplano [18].

A grande vantagem desse algoritmo é sua velocidade: devido à manutenção da linearidade, o algoritmo em MATLAB (que é reconhecidamente uma linguagem lenta, devido ao fato de ser de alto nível) leva cerca de 2 segundos para ser processado em um quad-core i5 4 GB/1,86 GHz.

O algoritmo também se encontra presente no WEKA, embora seja mais lento; nele, o algoritmo retorna uma expressão de dimensão 2. O resultado obtido no mesmo foi o seguinte:

Figura 12: resultado no uso da regressão linear no WEKA.



Usando a expressão retornado pelo WEKA, constata-se que apenas 1100, entre 10000 amostras de saída foram aproximadas com erro inferior a 10 %, portanto seu uso pode ser descartado. Utilizando-se a função *mvregress* do MATLAB, o resultado é ainda pior: apenas 972 saídas apresentavam erro inferior a 10 %.

Uma explicação imediata para essa falha exacerbada em um método tido como confiável refere-se ao fato de o problema não ser de classe linear; ou seja, existem não-linearidades na obtenção dos parâmetros pelos Fatores de Participação Estendidos [4], e a aproximação linear não é boa o suficiente. De fato, um dos grandes motivos da universalidade da rede neural, qualquer que seja, é o fato de sua função de ativação incluir não-linearidades em seu domínio, o que resulta em um espaço de aplicação muito mais amplo (universal).

A.2.2 Aproximação polinomial

O método de aproximação polinomial assemelha-se ao método de mínimos quadrados, com o diferencial óbvio que a aproximação não é feita por uma reta (ou hiperplano), senão por uma equação polinomial.

Essa abordagem apresenta como vantagem o fato de ser mais condizente com os dados caso os mesmos sejam não-lineares, ou apresentem muitos pontos fora da média geral; além disso, o cálculo dos coeficientes é muito rápido, e o polinômio é um objeto matemático muito bem conhecido e estudado, sendo de fácil análise. Ainda vale mencionar que os métodos numéricos para se encontrar raízes de polinômios são muito eficientes e variados, de modo que, obtida uma aproximação polinomial, é possível fazer o caminho inverso e determinar o valor de entrada que resultaria na saída. No MATLAB, o comando utilizado, bem como sua descrição, são [19]: "`p = polyfit(x,y,n)` *returns*

the coefficients for a polynomial `p(x)` *of degree* `n` *that is a best fit (in a least-squares sense) for the data in* `y`*. The coefficients in* `p` *are in descending powers, and the length of* `p` *is* `n+1`".

O algoritmo se baseia na seguinte identidade:

$$\begin{pmatrix} x_1^{n+1} & x_1^n & \dots & 1 \\ x_2^{n+1} & x_2^{n+1} & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ x_n^{n+1} & x_n^{n+1} & \dots & 1 \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

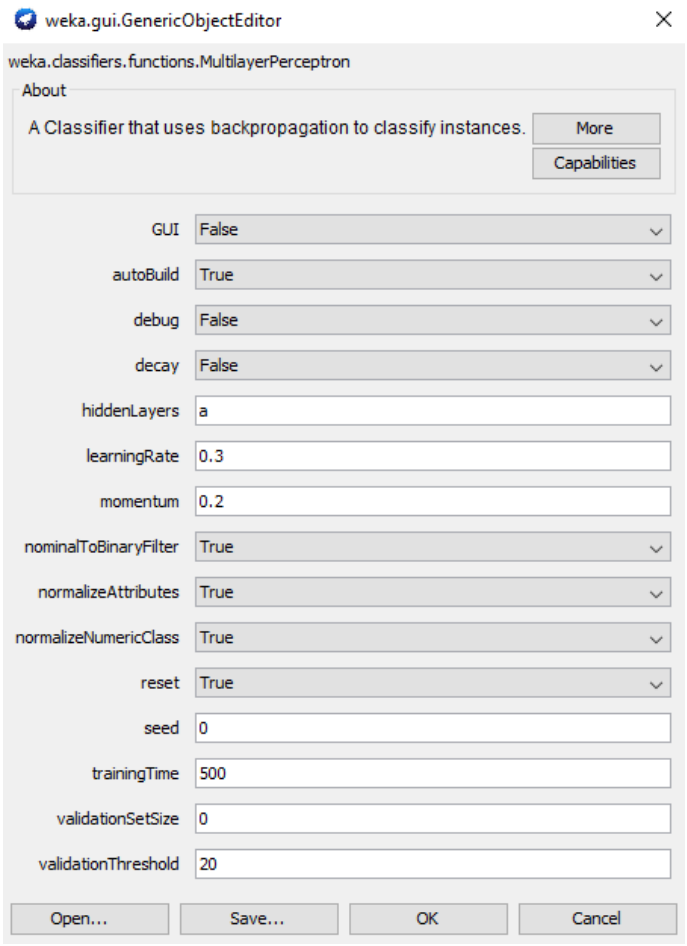
A matriz quadrada é a matriz de Vandermonde. Ao se incluir no argumento da função o número de coeficientes, os termos de grau maior que n+1 são automaticamente zerados na matriz coluna p.

A grande desvantagem deste método é a necessidade de alocar memória, que aumenta quando se aumenta o grau do polinômio utilizado. Além disso, o algoritmo funciona apenas para entradas e saídas vetoriais; no entanto, é bastante simples escrever um código que consiga selecionar a melhor coluna de dv_v para encontrar a saída calculada mais próxima da real. Essa última informação é importante pois pode ser útil para determinar quais valores iniciais de dv_v são mais relevantes na obtenção dos tempos de flicker em questão.

A.2.3. Rede Perceptron na WEKA

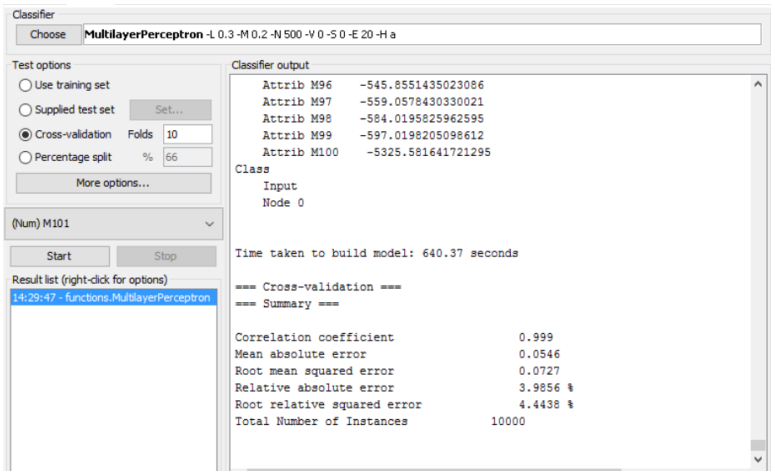
A rede neural PMC também está disponível no WEKA, apesar de bem limitada; o princípio de funcionamento é o mesmo que no MATLAB, embora ela já esteja pronta, bastando inserir os dados. No entanto, é possível alterar algumas variáveis do treinamento como momentum, taxa de aprendizagem.

Figura 13: parâmetros alteráveis da rede PMC no WEKA.



A tentativa de usar o algoritmo não foi bem-sucedida no WEKA: além de demorar mais de 10 minutos para convergir a matriz de pesos, a validação ocasionou o travamento do computador usado duas vezes, e demorou cerca de duas horas para terminar a validação, obtendo resultado análogo ao obtido no MATLAB, sendo que este último possui a vantagem de possibilitar a construção de códigos de reconstrução e correção do resultado, além de facilitar a prototipagem e a conexão em microcontroladores.

Figura 14: resultado da rede no WEKA.


Classifier
Choose
MultilayerPerceptron -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a
Test options
Use training set
Supplied test set
Set...
Cross-validation
Folds
10
Percentage split
%
66
More options...
(Num) M101
Start
Stop
Result list (right-click for options)
14:29:47 - functions.MultilayerPerceptron
Classifier output
Attrib M96 -545.8551435023086
Attrib M97 -559.0578430330021
Attrib M98 -584.0195825962595
Attrib M99 -597.0198205098612
Attrib M100 -5325.581641721295
Class
Input
Node 0
Time taken to build model: 640.37 seconds
=== Cross-validation ===
=== Summary ===
Correlation coefficient 0.999
Mean absolute error 0.0546
Root mean squared error 0.0727
Relative absolute error 3.9856 %
Root relative squared error 4.4438 %
Total Number of Instances 10000