UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

Luiz Victor Linhares Rocha

Utilização de redes neurais para determinação de tensão nodal em barras não observáveis

São Carlos

2018

Luiz Victor Linhares Rocha

Utilização de redes neurais para determinação de tensão nodal em barras não observáveis

Monografia apresentada ao Curso de Curso de Engenharia Elétrica com Ênfase em Sistemas de Energia e Automação, da Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Engenheiro Eletricista.

Orientador: Prof. Dr. Eduardo Nobuhiro Asada

São Carlos

2018

Ficha catalográfica elaborada pela Biblioteca Prof. Dr. Sérgio Rodrigues Fontes da EESC/USP com os dados inseridos pelo(a) autor(a).

Rocha, Luiz Victor Linhares
RL672u Utilização de redes neurais para determinação de tensão nodal em barras não observáveis / Luiz Victor Linhares Rocha; orientador Eduardo Nobuhiro Asada. São Carlos, 2018.

Monografia (Graduação em Engenharia Elétrica com ênfase em Sistemas de Energia e Automação) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2018.

1. Rede neural artificial. 2. Aprendizado de máquina. 3. Estimação. 4. Gauss-Seidel. 5. Rede de potência. 6. Unidade de medida fasorial. 7. Fluxo de potência. 8. Árvore de possibilidade. I. Título.

Eduardo Graziosi Silva - CRB - 8/8907

FOLHA DE APROVAÇÃO

Nome: Luiz Victor Linhares Rocha

Título: "Utilização de redes neurais para determinação de tensão nodal em barras não observáveis"

Trabalho de Conclusão de Curso defendido e aprovado em 27/11/2018,

com NOTA 8,0 (oito, zero), pela Comissão Julgadora:

Prof. Associado Eduardo Nobuhiro Asada - Orientador - SEL/EESC/USP

Prof. Dr. Benvindo Rodrigues Pereira Junior - SEL/EESC/USP

Mestre Antônio Eduardo Ceolin Momesso - Doutorando - SEL/EESC/USP

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado Rogério Andrade Flauzino

Este trabalho é dedicado aos meus pais, José Roberto Martins Rocha e Juraci Linhares por sempre darem apoio e sempre acreditarem no meu potencial nessa jornada que foi a Graduação.

AGRADECIMENTOS

Primeiramente, agradeço a Deus pela oportunidade de crescimento envolvida

À Escola de Engenharia de São Carlos e seu corpo docente, diretoria e a administração por oferecer um curso completo, de alta qualidade e um ambiente agradável e amigável de estudos.

Ao professor Eduardo Nobuhiro Asada, pelo auxílio prestado em seu feedback durante o desenvolvimento do trabalho e por ter sido o professor a introduzir um dos temas centrais do trabalho, a de análise de sistemas de potência.

Aos meus pais, por sempre me darem suporte necessário.

Ao professor Fernando Santos Osório por ser o orientador em minhas primeiras iniciações cientificas de 2014 até 2015 e me introduzir ao tema do machine learning e da pesquisa acadêmica.

Minha namorada, Aline Boni Minetto pelo carinho e suporte forte e muito importante dado nesse momento da graduação.

E as demais pessoas que indiretamente influenciaram no desenvolvimento deste trabalho

"Agradeço todas as dificuldades que enfrentei
não fosse por elas, eu não teria saído do lugar."
Chico Xavier

RESUMO

ROCHA, L. V. R. **Utilização de redes neurais para determinação de tensão nodal em barras não observáveis**. 2018. 98p. Monografia (Trabalho de Conclusão de Curso) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2018.

Com a introdução de novos equipamentos em redes de energia, novos meios de monitoramento também são desenvolvidos. PMUs são medidores capazes de obter informação em tempo real das tensões nodais de uma rede de potência com precisão, mas também possuem alto custo de instalação e manutenção. Utilizando-se de redes neurais artificiais, pode-se estimar o valor de magnitude e ângulo de fase de barras não observáveis. Compara-se, então, a alocação de medidores para tornar uma rede totalmente observável com a alocação para tornar a mesma rede não observável de nível 1. Utilizando-se das redes padrões IEEE de 14 e 30 barras, alocações foram feitas por meio de árvore de possibilidade, diminuindo em até 50% o número de PMUs instalados no sistema. Para o treinamento da rede neural, casos de exemplo foram gerados aplicando-se fatores aleatórios de consumo de potência e obtendo-se os valores de tensões nodais pela resolução do fluxo de potência por Gauss-Seidel. Conclui-se que uma rede com 2 camadas escondidas de 25 neurônios é suficiente para o problema com 14 barras e uma rede de 2 camadas escondidas de 35 neurônios é suficiente para o problema com 30 barras. Ambas obtiveram erros nos valores estimados menores que 10^{-4} com base em valores medidos com erros de 10^{-5}, mostrando serem capazes de substituir PMUs.

Palavras-chave: Rede neural artificial. Aprendizado de máquina. Estimação. Gauss-Seidel. Rede de potência. Unidade de medida fasorial. Fluxo de potência. Árvore de possibilidade.

ABSTRACT

ROCHA, L. V. R. **Use of neural network for determining voltage on non-observable bars**. 2018. 98p. Monografia (Trabalho de Conclusão de Curso) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2018.

With the introduction of new equipment in power grids, new monitoring devices are also developed. PMUs are meters that can accurately obtain real-time information on nodal voltages of a power grid, but it also has a high cost of installation and maintenance. Using artificial neural networks, the value of magnitude and phase angle of unobservable buses can be estimated. The allocation of meters to make a network fully observable is compared with the allocation to make the same network of level 1 unobservable. Using the IEEE standard networks of 14 and 30 bus the installations were made through a tree of possibilities, reducing the number of PMUs installed in the system by up to 50 %. Random operational load points were generated and the resulting bus voltage for each bus in system was obtained through power flow resolution using Gauss-Seidel method, generating the data set for neural network training. It was concluded that a network of 2 hidden layers of 25 neurons is suficient the problem of 14 bars and a network of 35 neurons meets that of 30 bars. Both obtained precision of estimated values greater than 10^{-4} based on measured data with 10^{-5} of precision.

Keywords: Artificial neural network. Machine learning. Estimation. Gauss-Seidel. Power Grid. Phasor measurement units. Power flow. Possibility tree.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos
EESC	Escola de Engenharia de São Carlos
USP	Universidade de São Paulo
CSV	Comma-separated values (Valores separados por vírgula)
CDF	Common data format
PMU	Phasor Measurement Unit
RNA	Rede neural artificial
PMC	Perceptron multicamada
EQM	Erro quadrático médio
EAM	Erro absoluto médio
TDD	Desenvolvimento guiado por teste

LISTA DE SÍMBOLOS

π	Pi
Δ	Delta
θ	Teta
ϕ	Phi
$\sum$	Somatório

SUMÁRIO

1 INTRODUÇÃO

Para o melhor controle sobre os sistemas de potência é de grande importância saber o estado da rede sendo controlada, no que se refere ao consumo de potência pelas cargas, as tensões e correntes nodais nos barramentos e as correntes entre os ramos. Assim, é possível identificar situações de falta, calcular perdas, planejar manutenção preventiva e realizar novas expansões.(WU, 1990)

Porém, a determinação de todos fasores de tensão das barras da rede de potência em tempo real, de forma a monitorar o sistema, não é trivial, com exigência de equipamentos de medição, sistema de transmissão, armazenamento e processamento dos dados, o que muitas vezes eleva o custo, não compensando economicamente a sua utilização para sistemas de grande porte.(NUQUI, 2001)

Esta é a dificuldade da instalação de medidores modernos e precisos como as *Phasor measurement units*(T.L. et al., 1993), ou PMUs, que conseguem medir fasores de barramentos com sincronização via satélite entre as outras PMUs instaladas no sistema com precisão e em tempo real(PHADKE, 1993).

Métodos clássicos de modelagem para a determinação do estado de regiões não observáveis na rede em tempo real já existem (SCHWEPPE; WILDES, 1970a), porém, são muito dependentes em utilização de pseudo-medidas, ou seja, estimações com base no histórico da região para o dado momento, o que introduz erros significantes no modelo em sí.

Logo, sistemas inteligentes(SILVA; SPATTI; FLAUZINO, 2010g) podem ser utilizados de forma a determinar estados de barras que não são observados diretamente por medidores, diminuindo-se a necessidade de instalação de novos medidores e mantendo a confiabilidade das informações com a vantagem na flexibilidade em relação às situações em que a rede de potência pode se encontrar.

Logo, este trabalho estuda a utilização de redes neurais artificiais para mitigar o custo de instalação de medidores, havendo a estimação de fasores de tensão em barras cuja medição não ocorre de forma direta, dessa forma, permitindo a diminuição no número de medidores a serem instalados sem perder o monitoramento em tempo real do estado e do fluxo de potência da rede.

Este documento apresenta 5 capítulos: O capítulo 2 apresenta a bibliografia utilizada para o desenvolvimento da pesquisa, o capítulo 3 descreve a metodologia utilizada, o capítulo 4 apresenta os resultados obtidos e o capítulo 5 apresenta as considerações finais obtida nos resultados e na experiência adquirida com o processo desenvolvido.

2 REVISÃO BIBLIOGRÁFICA

2.1 Descrição do modelo de sistema de potência

Um sistema de potência em regime permanente pode ser representado por um diagrama unifilar, ou seja, um grafo de barras conectados por ramos. Cada barra, com seu tipo e características elétricas, representando um ou mais centros de geração, de consumo ou uma subestação de intersecção de linhas de transmissão. Os ramos representam as linhas de transmissão entre as barras ou transformadores de nível de tensão ou defasadores (STEVENSON, 1986d).

2.1.1 Modelo de barras

As barras podem se apresentar de diversos tipos, mas os mais importantes e mais utilizados para representação são as barras *Slack*, PV e PQ (STEVENSON, 1986c):

- Barras *Slack*: Valor de tensão e ângulo de fase fixos, barra de referencia angular. Condiz com uma barra com grande potencial e controle de geração de energia, capaz de compensar e satisfazer as variações de necessidades de potência da rede.
- Barras PQ: Valor de potência ativa e reativa fixos, são comumente barras de consumo.
- Barras PV: Magnitude de tensão e potência ativa fixa, são barra de geração controlada ou compensadores de potência reativa.

A convenção do sinal de potência utilizada é o de positivo para barra geradora e o negativo para barras consumidoras.(STEVENSON, 1986c)

Possuem em suas características, as variáveis elétricas:

- Magnitude de tensão, V;
- Ângulo de fase da tensão, θ;
- Potência ativa, P;
- Potência reativa, Q.

E um parâmetro, B_{sh}, representando a susceptância na barra paralela à carga.

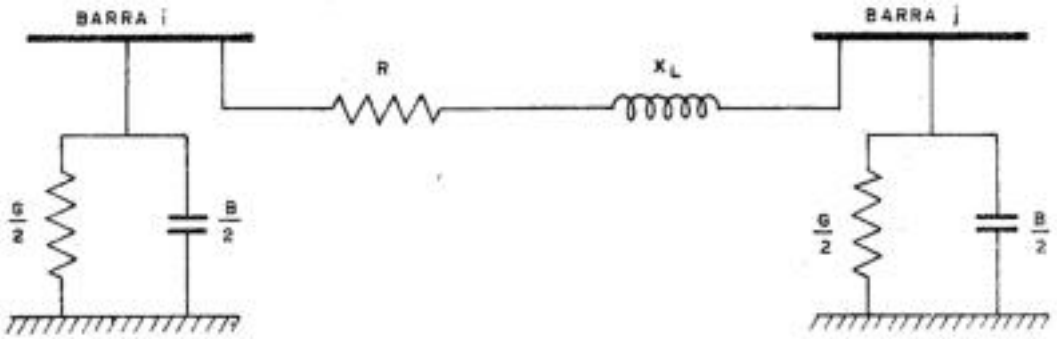
2.1.2 Modelo de transmissão π

O modelo de transmissão π é utilizado quando a conexão existente entre duas barras é uma linha de transmissão. Para modelagens gerais são utilizados modelos equivalentes de longa distância, podendo ser do modelo π ou T (STEVENSON, 1986f).

Para utilização em sistemas de potência de grande escala, são mais utilizados os modelos de transmissão longa π em relação aos modelos T, por não necessitar a inserção de uma barra extra para cada linha de transmissão.

O modelo π de transmissão é apresentado na figura 1, onde a admitância da linha em relação aos isoladores é representado por duas admitâncias, uma em cada ponta.

Figura 1: Representação do modelo π de linha de transmissão



As características elétricas principais dos ramos representando linhas de transmissão são:

- Resistência da linha, R_{km};
- Reatância da linha, X_{km};
- Susceptância *shunt* da linha, B_{km}^{sh}.

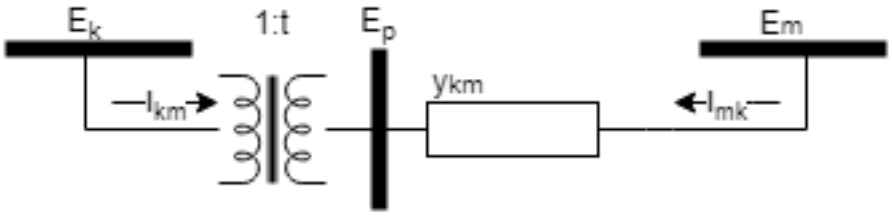
2.1.3 Modelo de transformador

O modelo de transformador é utilizado quando a conexão entre duas barras é um transformador (STEVENSON, 1986b). Simplifica-se a estrutura em um transformador ideal com tap e defasagem configurados e uma impedância equivalente série, como mostra a figura 2, onde $t = ae^{j\phi}$ sendo que $a \in \mathbb{R}^+$, representa a relação de transformação, e $\phi \in [-\frac{\pi}{2}, \frac{\pi}{2}]$, representa a defasagem.

As características elétricas dos ramos representando transformadores são:

- Impedância equivalente, $Z_{km} = R_{km} + jX_{km}$
- Tap do transformador, a_{km}

Figura 2: Modelo de transformador



- Defasagem do transformador, ϕ_{km}

2.1.4 Modelo generalizado de ramo

Com as descrições em 2.1.2 e em 2.1.3, pode-se generalizar o modelo de ramos com as seguintes características elétricas:

- Resistência equivalente, R_{km};
- Reatância equivalente, X_{km};
- Susceptância *shunt* da linha, B^{sh}_{km};
- Tap do transformador, a_{km};
- Defasagem do transformador, ϕ_{km}.

Aplicando-se as condições:

- $a_{km} = 1$ e $\phi_{km} = 0$ para linhas de transmissão;
- $B^{sh}_{km} = 0$ para transformadores.

2.2 Cálculo de correntes

As correntes nos ramos podem ser deduzidas a partir da utilização da lei de Ohm nos modelos assumidos.

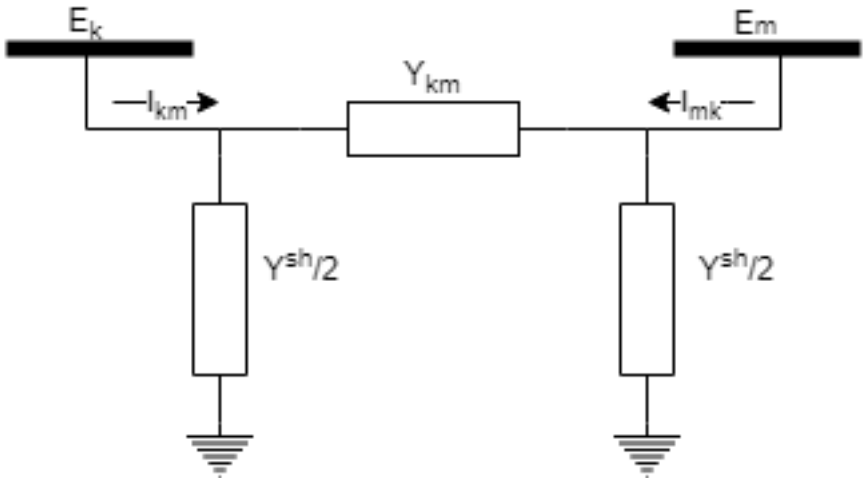
2.2.1 Linhas de transmissão π

As correntes em uma linha de transmissão do modelo π são apresentadas na figura 3.

Dessa forma, considerando que $Y^{sh} = jB^{sh}_{km}$, é possível determinar que a corrente I_{km} pela equação 2.1.

$$I_{km} = y_{km}(E_k - E_m) + jB^{sh}_{km}E_k = (y_{km} + jB^{sh}_{km})E_k - y_{km}E_m \tag{2.1}$$

Figura 3: Corrente em linhas de transmissão π



2.2.2 Transformadores

As correntes em um transformador foram apresentados na figura 2, onde usa-se de uma barra fictícia P para auxílio da formulação.

Partindo-se da formulação do transformador ideal, ficam determinadas as equações 2.2 e 2.3 (STEVENSON, 1986i).

$$\frac{E_k}{E_p} = \frac{1}{t} \tag{2.2}$$

$$S_{kp} = S_{pk} \tag{2.3}$$

As potências trifásicas S_{kp} e S_{pk} são dadas por 2.4 e 2.5, respectivamente (STEVENSON, 1986h).

$$S_{kp} = E_k * I_{km}^* \tag{2.4}$$

$$S_{pk} = E_p * I_{mk}^* \tag{2.5}$$

Logo, substituindo-se 2.4, 2.5 e 2.2 em 2.3, obtém-se a relação entre as correntes I_{km} e I_{mk} dadas por 2.6

$$\frac{I_{km}}{I_{mk}} = (\frac{E_p}{E_k})^* = -t^* \tag{2.6}$$

Considerando-se então a admitância equivalente do transformador, determina-se I_{mk} dado por 2.7.

$$I_{mk} = y_{km}(E_m - E_p) = y_{km}(E_m - t * E_k) = -t * y_{km} * E_k + y_{km}E_m \tag{2.7}$$

Substituindo 2.6 em 2.7, determina-se I_{km} como na equação 2.8

$$I_{km} = |t|^2 y_{km} E_k - t^* y_{km} E_m \tag{2.8}$$

2.2.3 Formulação geral

É possível resumir as equações 2.1 e 2.8 na equação 2.9

$$I_{km} = (|t|^2 y_{km} + j B_{km}^{sh}) E_k - t^* y_{km} E_m \tag{2.9}$$

Respeitando-se as restrições dos parâmetros:

- $a_{km} = 1$ e $\phi_{km} = 0$ para linhas de transmissão
- $B_{km}^{sh} = 0$ para transformadores

2.3 Matriz admitância

É a matriz que representa os dados da rede de forma matricial, simplificado trabalhos de formulação de resoluções baseados na rede (STEVENSON, 1986a). Relaciona as correntes nodais com as tensões, ou seja, já forma matricial 2.10, tem-se que.

$$I = Y * E \tag{2.10}$$

Onde:

- I é o vetor das correntes nodais;
- E é o vetor das tensões nodais;
- Y é a matriz admitância total, composta por Y = G+jB.

De forma geral, com os dados de barra e de ramos definidos, os valores da matriz admitância podem ser definidos por 2.11 ao introduzir os efeitos de possíveis transformadores descritos em (STEVENSON, 1986b).

$$\begin{cases} Y_{km} = -a_{km} * e^{-j\phi_{km}} * y_{km} & \text{fora da diagonal} \\ Y_{mk} = -a_{km} * e^{j\phi_{km}} * y_{km} & \text{fora da diagonal} \\ Y_{kk} = j b_k^{sh} + \sum_{m \in \Omega_k} (j b_{km}^{sh} + a_{km}^2 y_{km}) & \text{na diagonal} \end{cases} \tag{2.11}$$

Vale notar que, caso não houver transformador defasador na rede, a matriz admitância é simétrica.

2.4 Métodos de resolução de fluxo de potência

Os métodos clássicos de fluxo de potência são importantes para a realização de análises de perfil de carga, correntes em ramos e perdas do sistema. Para aplicação em tempo real, porém, são limitados pois requerem dados de potência em cada carga, assim como considera que os parâmetros dos ramos da rede não mudam com o tempo.

Os métodos clássicos são descritos em livros didáticos(STEVENSON, 1986c) e os principais utilizados são os métodos de Gauss, Gauss-Seidel, Newton-Raphson e Newton-Raphson desacoplado.

2.4.1 Potência nodal

Refere-se ao balanço que relaciona o consumo ou geração de potência ativa e reativa de barras com as correntes e tensões nodais (STEVENSON, 1986c).

De forma geral, a potência nodal complexa é dado por 2.12. Utilizando a informação da equação 2.10, temos a equação 2.13.

$$S_k^* = E_k^* * I_k \tag{2.12}$$

$$S_k^* = E_k^* * (\sum_{m \in K} Y_{km} E_m) \tag{2.13}$$

Substituindo as informações da equação 2.11 em 2.13 e separando a parte real da imaginária, é possível determinar as relações de potência ativa e reativa de cada barra, resultando em 2.14 e 2.15.

$$P_k = V_k \sum_{m \in K} V_m (G_{km} cos(\theta_{km}) + B_{km} sen(\theta_{km})) \tag{2.14}$$

$$Q_k = V_k \sum_{m \in K} V_m (G_{km} sen(\theta_{km}) - B_{km} cos(\theta_{km})) \tag{2.15}$$

2.4.2 Gauss

2.4.2.1 Formulação geral

De forma geral, dado o sistema definido por Ax=b, a relação das matrizes representa a equação 2.16. Logo, x_i é definido por 2.17 para uma primeira iteração, assim, o valor de x_i para uma iteração $m+1$ é definido por 2.18, definindo as próximas iterações.

$$\sum_{j=1}^{n} A_{ij} * x_j = b_i, i = 1, ..., n \tag{2.16}$$

$$x_i = \frac{1}{A_{ij}} * (b_i - \sum_{j=1, j\neq i}^{n} A_{ij} x_j), i = 1, .., n \tag{2.17}$$

$$x_i^{m+1} = \frac{1}{A_{ij}} * (b_i - \sum_{j=1, j\neq i}^{n} A_{ij} x_j^m), i = 1, .., n \tag{2.18}$$

Caso o estado inicial for suficientemente próximo à solução desejada, haverá convergência a esta.

2.4.2.2 Formulação para o sistema de potência

Para a resolução do fluxo de potência do sistema, temos a equação 2.19, onde, isolando o termo E_k obtemos a equação 2.20 (STEVENSON, 1986e).

$$S_k^* = E_k^* * I_k = E_k^* * \sum_{n\in K} Y_{kn} E_n = E_k^* * \sum_{n\in \Omega_k} Y_{kn} E + E_k^* * Y_{kk} E_k \tag{2.19}$$

$$E_k^{m+1} = \frac{1}{Y_{kk}} * (\frac{S_k^*}{E_k^{*(m)}} - \sum_{n\in \Omega_k} Y_{kn} E_n^m) \tag{2.20}$$

As barras são então atualizada segundo a seguinte lógica:

- Para uma barra PQ, S_k é definido, logo, a iteração ocorre diretamente;
- Para uma barra PV, Q_k é calculado a partir de 2.15 com o valor de tensão atual, a iteração ocorre e apenas o ângulo de fase é calculado;
- Para uma barra *Slack*, não há atualização dos valores de tensão pois já estão definidos.

Quando a iteração convergir, os valores de potência ativa e reativa das barras *Slacks* e de potência reativa das barras PV são calculados diretamente pelas equações 2.14 e 2.15.

Para facilitar a convergência, pode ser escolhido o valor inicial, da magnitude de tensão das barras PQ, igual a 1 Pu e ângulo de fase nulo das barras PQ e PV.

2.4.3 Gauss-Seidel

Uma variação do método do Gauss, utiliza os valores já calculados de $x_i, i = 1, ..., j-1$ na presente iteração para calculo de x_j. Ao utilizar tais valores, a convergência é garantida e ocorre de forma mais rápida, porém elimina a oportunidade de tornar o processo paralelo.

A relação geral de Gauss-Seidel é dada por 2.21 e para o sistema de potência é dada por 2.22.

$$x_i^{m+1} = \frac{1}{A_{ij}} * (b_i - \sum_{j=1}^{i-1} A_{ij} x_j^{m+1} - \sum_{j=i+1}^{n} A_{ij} x_j^m), i = 1, .., n \tag{2.21}$$

$$E_k^{m+1} = \frac{1}{Y_{kk}} * (\frac{S_k^*}{E_k^{*(m)}} - \sum_{n=1}^{k-1} Y_{kn} E_n^{m+1} - \sum_{n=k+1}^{nb} Y_{kn} E_n^m) \tag{2.22}$$

2.4.4 Newton-Raphson

2.4.4.1 Formulação geral

O método de resolução por Newton-Raphson se baseia na determinação da raiz por meio de derivada (STEVENSON, 1986g).

De modo geral, para uma função f(x), uma raiz da função pode ser determinada a partir de iterações apresentadas na equação 2.23.

$$x^{m+1} = x^m - \frac{f(x^m)}{f'(x^m)} \tag{2.23}$$

Para um sistema de n equações e n incógnitas, o método de Newton passa a ser a 2.24, onde J é a matriz Jacobiana da função, representando a derivada de 2.23, dado por 2.25.

$$x^{m+1} = x^m - [J(x^m)]^{-1} * g(x^m) \tag{2.24}$$

$$J = \frac{\partial}{\partial x} g(x) = \begin{bmatrix} \frac{\partial}{\partial x_1} g_1 & \frac{\partial}{\partial x_2} g_1 & \dots & \frac{\partial}{\partial x_n} g_1 \\ \frac{\partial}{\partial x_1} g_2 & \frac{\partial}{\partial x_2} g_2 & \dots & \frac{\partial}{\partial x_n} g_2 \\ \dots & \dots & \dots & \dots \\ \frac{\partial}{\partial x_1} g_n & \frac{\partial}{\partial x_2} g_n & \dots & \frac{\partial}{\partial x_n} g_n \end{bmatrix} \tag{2.25}$$

Semelhante ao método de Gauss, caso o estado inicial for suficientemente próximo à solução desejada, haverá convergência a esta.

2.4.4.2 Situações

Como o método de Newton se baseia em derivadas, existem certos problemas com relação à convergência que podem ser encontrados em alguns casos, são eles problemas de pontos com derivada igual a zero e múltiplas raízes.

Para um ponto com derivada zero ou próxima de zero, a matriz Jacobiana tende à zero, logo, sua inversa tende ao infinito resultando em uma divergência indesejada para a resolução do problema.

Para múltiplas raízes, pode-se atingir uma diferente solução baseando-se em um ponto inicial diferente, logo, há a importância de se escolher o ponto inicial o mais próximo da solução desejada.

Por conta da quantidade de vezes que o fluxo de potência será aplicado para a geração dos dados de exemplo, podem ocorrer várias situações de não convergência, logo, não será utilizado para o trabalho.

2.5 Estimação de estado de uma rede de potência

A estimação de estado da rede de potência é de determinação do vetor $\vec{V}$ de magnitudes e tensões das barras da rede em tempo real, mesmo sem todas as medidas aferidas da rede. O consumo de potência da área não observável da rede é estimada com base no registro histórico coletados por meios básicos de coleta de dados, denominada pseudo-medidas.

Na formulação do problema, para cada medida sendo considerada no modelo, tem-se que a formulação da medição é a apresentada na equação 2.26, onde z é a medida aferida ou considerada como pseudo-medida, f(x) é a medida real e μ é o erro gerado pela medida.

$$z = f(x_{real}) + \mu \tag{2.26}$$

o vetor de estimação$\hat{x}$ é então definido como o valor que minimiza a expressão 2.27, onde θ é uma matriz diagonal cujos elementos são as covariâncias de cada medida ou pseudo-medida i (SCHWEPPE; WILDES, 1970a).

$$J(x) = [z - f(x)]'\theta^{-1}[z - f(x)]. \tag{2.27}$$

Para encontrar o valor de x é aplicado o método de Newton utilizando-se da expansão da série de Taylor desta equação, considerando apenas a primeira ordem, ou seja, supondo uma estimativa inicial x_o, a função f(x) é dado por 2.28. Onde $F(x_o)$ é a matriz jacobiana $\delta f_{k1}(x)/\delta x_{k2}$ para $x = x_o$.

$$f(x) = f(x_o) + F(x_o)[\Delta x] + ... \tag{2.28}$$

O estimativa é então feita por um método iterativo, cujos passos são determinados com a substituição de 2.28 em 2.27, quando o valor do método convergir, o valor é então considerado estimado. Em efeito, o processo aplica uma ponderação entre os valores de medição considerados, com base no fator de covariância.

Linearizações também podem ser feitas de forma a tornar o método mais direto e simplificado em relação às medições (SCHWEPPE; WILDES, 1970b; G.; BISWAS; MAHA-

LANABIS, 2007), porém, com perda de precisão por conta do aumento de considerações pré-concebidas que são inseridas no modelo.

2.5.1 Desvantagens

A utilização de pseudo-medidas atribui ao modelo uma quantidade inerente de erro grande em relação às medidas reais da rede, por conta de imprecisões nos métodos de coleta de dados.

Outra desvantagem é a dependência de processos iterativos baseados em matrizes jacobianas para se obter estimações mais precisas, o que gera problemas semelhantes à resolução de fluxo de potência com método de Newton, como a possibilidade de haver divergência nas iterações e a convergência para um mínimo local e não o ponto de solução definitiva.

2.5.2 Alternativas

Um método alternativo, que é o proposto por este trabalho é de diminuir a dependência de valores assumidos, mas se baseia em simulações completas previamente realizadas e com capacidade de ter sua eficiência comprovada antes de entrar em operação, esse é o caso da utilização da redes neurais.

2.6 Redes neurais artificiais

Redes neurais são estruturas de processamento de dados baseado em redes neurais biológicas, capazes de classificar informações, generalizar funções e prever informações futuras(SILVA; SPATTI; FLAUZINO, 2010g), como por exemplo, resolução de fluxo de potência(KUMAR et al., 1991; EFE S.B, 2013).

2.6.1 Rede perceptron

A rede perceptron é a versão mais simples e didática das redes neurais(SILVA; SPATTI; FLAUZINO, 2010d) ela é utilizada para problemas de classificação simples.

Composta de um número arbitrário de entradas $x_1, x_2, ..., x_n$ e apenas uma saída. As entradas são ponderadas através de pesos e é subtraído deste resultado um valor, denominado limiar de ativação, resultando no valor u, como apresentado na equação 2.29.

$$u = \sum_{i=1}^{n} x_i * w_i - b \tag{2.29}$$

O resultado dessa operação é então atribuída a uma função, que gera a saída do neurônio, podendo ser esse a função degrau bipolar ou degrau, apresentadas em 2.30 e

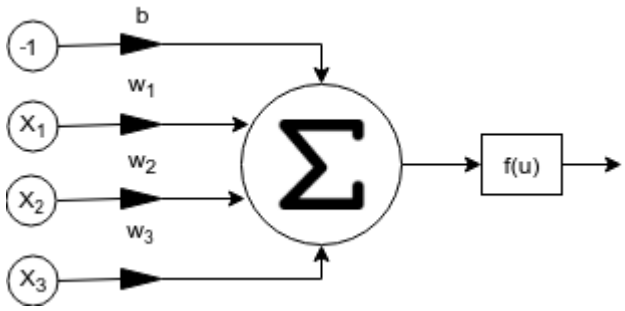
2.31, respectivamente.

$$y(u) = \begin{cases} -1 & \text{se } u < 0 \\ 0 & \text{se } u = 0, \\ 1 & \text{se } u > 0 \end{cases} \tag{2.30}$$

$$y(u) = \begin{cases} 0 & \text{se } u < 0 \\ 1 & \text{caso contrário} \end{cases} \tag{2.31}$$

Um esquema gráfico do neurônio perceptron, com 3 entradas, está apresentado na figura 4.

Figura 4: Rede perceptron



2.6.2 Rede perceptron multicamadas (PMC)

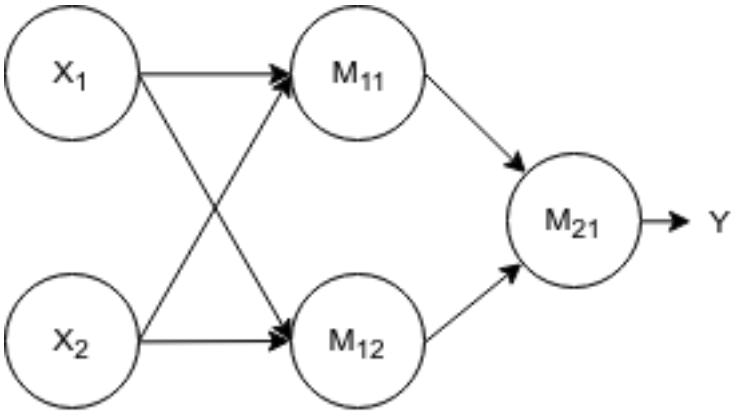
A rede perceptron multicamadas, ou PMC é uma generalização da rede perceptron, onde pode haver n camadas com m_i número de neurônios em cada camada i. As quantidade de entradas dos neurônios de uma camada i são iguais a quantidade de saída dos neurônios da camada i-1, assim, o resultado gerado por uma camada será utilizada como entrada para a camada seguinte.(SILVA; SPATTI; FLAUZINO, 2010f)

Um exemplo de PMC é apresentado na figura 5, com uma camada escondida, duas entradas e 1 saída. Tal configuração é conhecida por ser suficiente para executar como uma porta ou-exclusiva.

Para tal topologia, por conta do processo de treino a ser utilizado, a função de ativação dos neurônios deve ser funções contínuas e diferenciáveis, são elas as funções linear, sigmoide e tangente hiperbólico apresentadas em 2.32, 2.33 e 2.34, respectivamente.

As funções binárias, degrau, sigmoide e tangente hiperbólico são as mais utilizadas para classificação de padrões, sendo que as duas últimas tem a vantagem de serem

Figura 5: Rede perceptron multicamadas com uma camada escondida



diferenciáveis. Já as funções linear e linear limitada são mais utilizadas para problemas de generalização de função, com a linear sendo diferenciável.

$$y(u) = u \tag{2.32}$$

$$y(u) = \frac{1}{1 + e^{-u}} \tag{2.33}$$

$$y(u) = \frac{2}{1 + e^{-2u}} - 1 \tag{2.34}$$

Redes neurais com duas camadas escondidas são suficientes para aproximar qualquer função não linear, ou ainda serem utilizadas em qualquer problema de classificação.(IRIE B. MIYAKE, 1993)(FUNAHASHI, 1989)

2.6.3 Treino de rede neural para classificação e aproximação de função com PMC

A resolução de problemas de classificação e aproximação de função com redes PMC se baseia no ajuste dos valores dos pesos dos neurônios da rede(SILVA; SPATTI; FLAUZINO, 2010h).

O treino de uma rede PMC para tais problemas deve ser supervisionado, ou seja, com dados de exemplos de entrada com as saídas desejadas previamente definidas(SILVA; SPATTI; FLAUZINO, 2010i). Logo, deve-se ter vetores de entrada $x_t = \{x_1, x_2, ..., x_n\}$ e os vetores de saída $y_{desejado} = \{y_{1d}, y_{2d}, ..., y_{nd}\}$ correspondente de cada caso.

A rede, então, é alimentada com os dados de entrada dos exemplos, gerando as saída $y_{atual} = \{y_{1a}, y_{2a}, ..., yna\}$. Assim, o erro quadrático médio, ou EQM, da rede em relação às saídas desejadas é dado pela equação 2.35(SILVA; SPATTI; FLAUZINO, 2010b).

$$EQM = \sum_{i=1}^{N} \frac{(y_{id} - y_{ia})^2}{N} \tag{2.35}$$

O objetivo central do treino é a diminuição do valor resultante de 2.35, ajustando os valores dos pesos w_{ij} de cada neurônio de cada camada a cada exemplo de entrada e saída fornecidos a rede. Chama-se uma época quando todos os exemplos do conjunto de dados de treino foram fornecidos à rede para ajuste dos pesos.

2.6.3.1 Treino por batelada

Uma alternativa para o treino é a realização dos ajustes a cada quantidade definida de pontos de exemplo, nomeado batelada. Dessa forma é possível aproveitar mais de processamento paralelo, executando múltiplas operações similares simultaneamente utilizando núcleos de processamento diferentes e, por fim, utilizando o valor médio de erro obtido ao final de cada batelada para a realização da correção.

Problemas com grande quantidade de dados a serem processados podem se beneficiar desse método de treino, tornando a execução de uma época mais rápida, porém sacrifica precisão na atualização a cada época, necessitando de maior número de épocas para alcançar um mesmo valor de EQM que é atingido realizando ajustes dos pesos a cada exemplo.

2.6.3.2 Problema de aproximação de função com rede neural

Para o problema de aproximação de função $\mathbb{R}^N \rightarrow \mathbb{R}^M$, os valores de entrada da rede são os valores conhecidos do conjunto $\mathbb{R}^N$, logo, possuí N entradas. Já os valores de saída são os valores a serem estimados no conjunto $\mathbb{R}^M$, logo, possuí M saídas.

Para aproximação de função deseja-se a liberdade para a saída de valores contínuos, assim, utiliza-se a função linear na saída.(SILVA; SPATTI; FLAUZINO, 2010a).

2.6.3.3 Normalização de dados

De forma a garantir a eficiência da rede neural em relação à estimação, que trabalha parâmetros de pesos e saída de neurônios que variam de 0 a 1, os valores de entrada e saída de operação da rede devem ser normalizados em relação ao conjunto de dados utilizado no treino.

Ou seja, para cada valor x_{ij} da entrada e saída i do exemplo j, o valor a ser utilizado no treino da rede é x^o_{ij} calculado por 2.36.

$$x^o_{ij} = \frac{(x_{ij} - \bar{x_i})}{\mu_i} \tag{2.36}$$

Sendo $\bar{x_i}$ representando a média de valor dos dados de treino e definido por 2.37 e

μ_i representando o desvio padrão definido por 2.38.

$$\bar{x_i} = \frac{\sum_{j=1}^{N} x_{ij}}{N} \tag{2.37}$$

$$\mu_i = \sqrt{\frac{\sum_{j=0}^{N} (x_{ij} - \bar{x_i})^2}{N-1}} \tag{2.38}$$

Assim, na operação da rede neural após o treino, cada valor de entrada x_i é normalizado utilizando a média e desvio padrão já calculados em 2.37 e 2.38, enquanto que, para cada saída y_i^o, deve haver o processo reverso definido em 2.39.

$$y_i = (y_i^o + \bar{y_i}) * \mu_i \tag{2.39}$$

Dessa forma, o erro e_i^o para cada valor de saída i da rede neural deve ser multiplicado pelo mesmo fator de desvio padrão de forma a representar o erro real em relação ao valor sendo estimado e_i, como mostra a equação 2.40.

$$e_i = e_i^o * \mu_i \tag{2.40}$$

2.6.3.4 Validação da rede

Para validação da rede neural treinada, deve-se separar parte dos exemplos para testes para não serem utilizados no ajuste dos pesos, dessa forma, é possível aferir o erro de estimação da rede em relação a estes pontos e garantir que a a precisão da rede neural se mantém para pontos diferentes dos utilizados no treino (SILVA; SPATTI; FLAUZINO, 2010j).

2.6.4 Cuidados com redes neurais

Redes neurais artificiais também exigem cuidados com algumas questões na hora de escolha de topologia e no treino.

2.6.4.1 Mínimos locais

Um método de diminuição do EQM pode resultar em um ponto de mínimo da função que não represente o mínimo global da função, assim, mesmo a rede encontrando um ponto de mínimo, este ponto ainda pode não ser o ideal(SILVA; SPATTI; FLAUZINO, 2010c).

Para contornar tal problema é necessário então realizar mais de um ciclo de treino da mesma topologia com valores de pesos variáveis entre si, e, assim, escolher a rede que obteve o menor erro na fase de validação.

2.6.4.2 *Overfitting*

Um numero grande de neurônios nas camadas escondidas não significam grande precisão na saída em relação ao problema generalizado, pois pode ocorre o problema do *overfitting*, quando a rede atinge erro pequeno para o *dataset* de treino mas atinge erro alto para os dados de validação(SILVA; SPATTI; FLAUZINO, 2010e).

Para evitar tal problema, três medidas podem ser aplicadas, gerar grande número de dados de treino, o que aumenta o tempo de treino, realizar o treino sobre diferentes topologias, identificando a melhor rede que gera o menor erro em relação aos dados de teste e também acompanhar a evolução do erro em relação aos dados de validação, parando o treino quando um dos erros monitorados não melhorar com várias épocas em seguida.

2.6.4.3 Taxa de aprendizagem inapropriada

A taxa de aprendizagem influencia diretamente o processo de treino e a forma com que o ponto ótimo da solução é encontrado. Com uma taxa de aprendizagem alta, o ponto ótimo do sistema pode não ser atingido por conta dos passos grandes dados nas correções em cada iteração de treino. Com uma taxa de aprendizagem baixa, o ponto ótimo pode ser alcançado com melhor facilidade, porém, aumenta-se a quantidade de épocas necessários para convergir. Portanto, é importante a observação e estudo com testes preliminares para determinar a taxa de aprendizagem ideal para o problema.

2.6.5 Paralelismo

Como o funcionamento da rede neural envolve multiplicações e somatórias repetitivas, o processo pode ser paralelizado por mais de uma CPU ou GPU, principalmente para treinos em batelada, acelerando o processo de treino para quantidade grande de testes(LIAO, 2011).

Porém, se tal método de treino não for utilizado, ou o número de pontos por batelada for reduzido, a utilização de uma GPU pode aumentar o tempo de treino, pois a transferência de dados entre a CPU e GPU a cada etapa de treino pode gastar grande quantidade de tempo em relação ao tempo de processamento da informação pela GPU em si. Portanto, deve-se analisar também qual a melhor abordagem de treino para a aplicação desejada levando em consideração o erro desejado, o tipo de rede neural, a quantidade de dados de treino e o tempo disponível para treino.

2.7 Unidade de Medidas Fasoriais

Phasor Measurement Units, ou PMUs, são capazes de medir fasores de tensão em barras com o sincronismo via satélite, fornecendo em tempo real dados de tensão e fase das barras onde o medidor é instalado. PMUs também são capazes de medirem o valor

das correntes dos ramos conectados a barra, possibilitando o cálculo de tensões de barras vizinhas(PHADKE, 1993).

A base fundamental do cálculo dos valores de tensão é a transformada discreta de Fourier aplicada ao sinal amostrado, dado pela equação 2.41, onde N é o número de medidas em um período, X é o fasor resultante e x_k o k-ésimo valor medido dentro do período.

$$X = \frac{\sqrt{2}}{N} \sum_{k=1}^{N} x_k e^{-j2k\pi/N} \tag{2.41}$$

A utilização de PMUs porém envolve alto custo, não apenas de equipamento, mas também do sistema de transmissão, armazenamento dos dados e de manutenção, logo, há a necessidade de se otimizar o número de medidores em um sistema(T.L. et al., 1993).

2.8 Conceito de nível de observabilidade e não observabilidade de barras

2.8.1 Observabilidade

A observabilidade de uma barra é definida através da possibilidade de se observar as características elétricas, ou seja, fasor de tensão, de forma direta ou indireta(PHADKE, 1993).

A alocação de uma PMU gera observabilidade ao fato que (PHADKE, 1993):

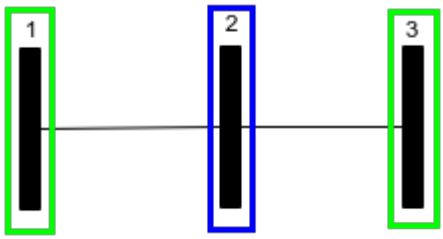
- O fasor de tensão da barra ao qual o medidor está instalado é conhecido;
- Os fasores de correntes dos ramos que conectam a barra também são conhecidos, resultando na medida indireta do fasor de tensão das barras vizinhas;
- O fasor de corrente entre duas barras com fasor de tensão conhecidas é também conhecido.

Se todos os fasores de tensão de uma rede são conhecidos, o sistema é dito como totalmente observável, assim como o sistema apresentado na figura 6, onde a PMU se encontra na barra 2, marcada em azul e as barras vizinhas são então marcadas como barras observáveis de forma indireta, marcadas em verde. No geral de um terço a um quarto das barras deve ter PMU instaladas para tornar um sistema totalmente observável(T.L. et al., 1993).

2.8.2 Nível de não observabilidade

Barras não observáveis são barras que não podem ter os fasores de tensão calculados com as medidas disponíveis. Quando uma barra não observada é cercada por duas barras

Figura 6: Sistema totalmente observável com 3 barras e 1 PMU na barra 2



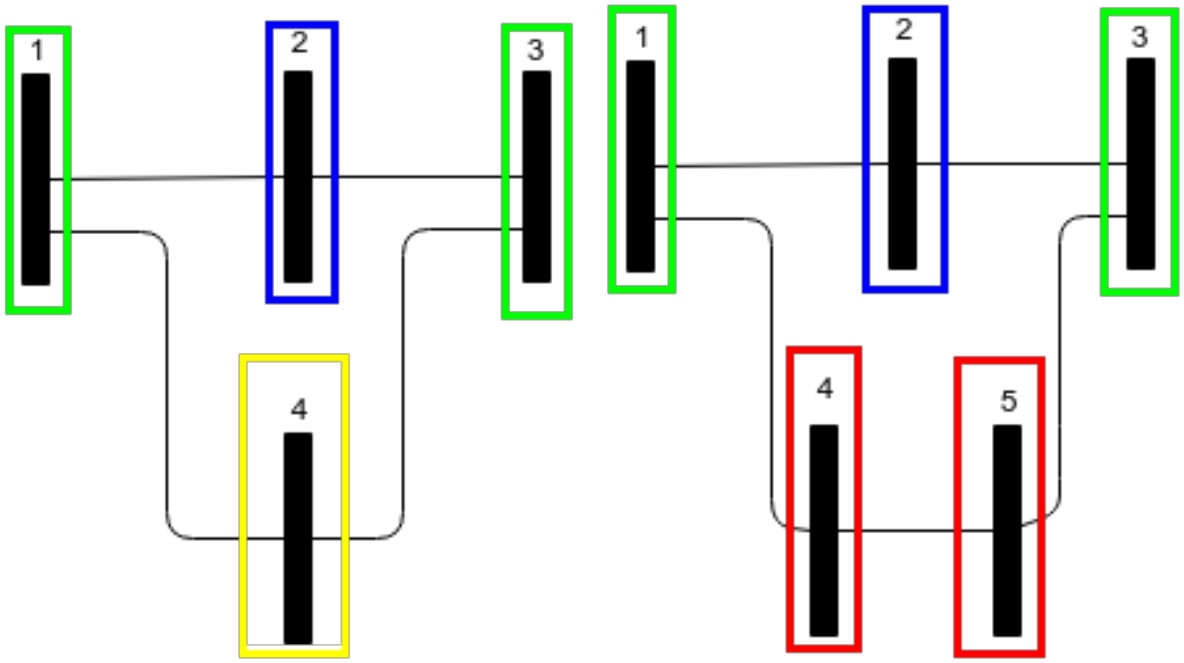
observáveis indiretamente, ela é dita ter um nível de não observabilidade, assim como apresentado na figura 7a, onde a barra 4, está em conexão com outras duas barras calculadas(NUQUI, 2001).

Quando existem duas barras não observáveis entre barras observáveis indiretamente, tais barras são ditas como barras de segundo nível de não observabilidade, assim como o apresentado na figura 7b. O conceito pode então ser estendido para outros níveis.

Figura 7: Comparação de nível do não observabilidade de sistema de potência

(a) Nível de não observabilidade 1

(b) Nível de não observabilidade 2



Para barras não observáveis também é possível a determinação da tensão de forma aproximada através de sistemas de controle estimadores(ARISTOY G., 2015; WU, 1990).

2.9 Ferramentas computacionais

2.9.1 Biblioteca *Boost*

A biblioteca *Boost* (BOOST..., 2005) é uma biblioteca extensa de código aberto para programas escritos em c++ desenvolvido por uma comunidade de programadores profissionais, ela possui uma grande gama de ferramentas que facilitam o trabalho de um programador(MEYERS, 2011).

2.9.1.1 *Boost unit test framework*

A biblioteca de testes de unidade tem facilitadores com questão de realização de testes de unidade em c++.

Ela se utiliza de macros e *templates* para gerar códigos, que realizam os testes descritos e, em caso de falha, gerar relatório de falha, indicando local onde ocorre falhas de memória caso houver.

2.9.2 Tensorflow

Desenvolvida pela *Google*, base dos sistemas de inteligencia artificial da empresa e de outras empresas grandes como NVIDIA, Intel e Coca-Cola, é definida oficialmente como:

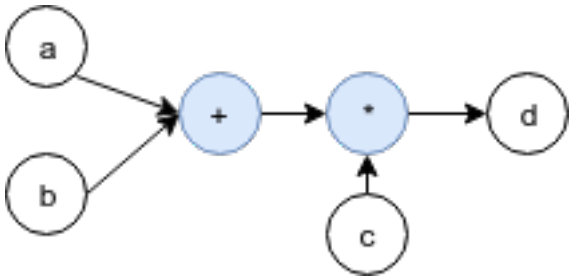
> *Tensorflow* é uma biblioteca de código aberto para computação numérica utilizando o fluxo de dados em grafos. Nodos nos grafos representam operações matemáticas, enquanto as bordas representam matrizes de dados multidimensionais (tensores) que fluem pelos grafos (TENSORFLOW, 2017).

Pela característica de código aberto, questões de compatibilidade de sistemas e licença de utilização podem ser melhor resolvidos em relação a programas proprietários como *Matlab*, podendo ainda ser especializada para cada estação.

Possuí uma forma de programação onde os processos são definidos anteriormente, formando um grafo de operações que é executado quando uma sessão é iniciada. Assim, é possível otimizar os processos envolvidos, como uso de memória e ordem de processos, trazendo vantagens para o processamento de grande quantidade de dados(ABADI et al., 2016b).

Um exemplo de operação do tipo "$d = (a + b) * c$"é apresentada na figura 8, nota-se também que as variáveis de entrada podem ser matrizes, não apenas escalares.

Figura 8: Grafo da operação "$d = (a + b) * c$"no *tensorflow*



2.9.2.1 Paralelismo

A estrutura de programação oferecida pelo *Tensorflow* também permite a realização de processos paralelos por mais de uma CPU ou GPU. A grande vantagem da utilização do *Tensorflow* é o suporte a utilização de GPU, o que aumenta a velocidade de processamento, necessário para aplicações de monitoramento em tempo real(LIAO, 2011; ABADI et al., 2016a).

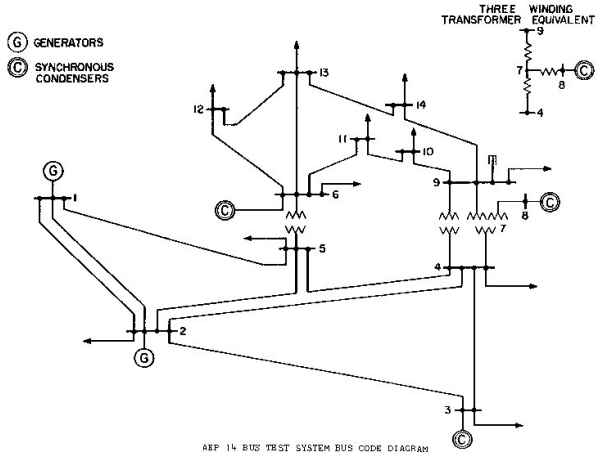
3 METODOLOGIA APLICADA

3.1 Rede padrão de teste

Para a análise dos métodos aplicados serão utilizados os sistemas de barras de teste disponíveis pela Universidade de Washington(UW. . . , 1999) no formato *IEEE common data format*(PIERCE et al., 1973). Nesses dados são especificados os perfis de carga comuns a serem utilizados para teste de sistema com 14, 30, 57, 118 e 300 barras. Para efeito desse projeto será utilizado como base o sistema de 14 e 30 barras apresentados nas figuras 9a e 9b

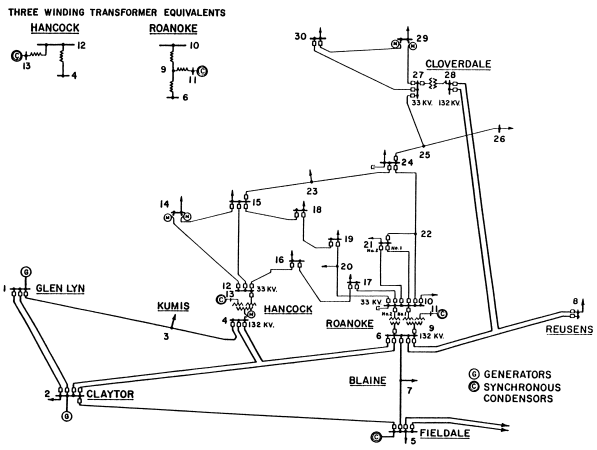
Figura 9: Redes de potência utilizados para análise

(a) Rede de potência IEEE 14 barras



FONTE: Sistema. . . (1999a)

(b) Rede de potência IEEE 30 barras



FONTE: Sistema. . . (1999b)

3.2 Alocação de medidores

A alocação de medidores será feita por meio de árvore de possibilidade, limitando as possibilidades para as mais efetivas em um presente estado da rede, ou seja, caso a rede não esteja no estado desejado, as possibilidades serão novamente calculadas a alocação de uma PMU nova na rede.

Cada PMU irá fornecer o fasor de tensão da barra em que está conectada, assim como os fasores de corrente nos ramos conectados a ela. Considerando os valores de impedância da rede fixos e conhecidos, a presença de um medidor torna a própria barra observável de forma direta e as barras vizinhas observáveis de forma indireta, com o fasor

de tensão sendo determinado isolando E_m da equação 2.9 resultando em 3.1.

$$E_m = \frac{(|t|^2 y_{km} + jB_{km}^{sh})E_k - I_{km}}{-t^* y_{km}} \tag{3.1}$$

Serão feitas duas formas de alocações diferentes, uma focando em tornar a rede totalmente observável e outra com o objetivo de tornar a rede não observável de 1º nível.

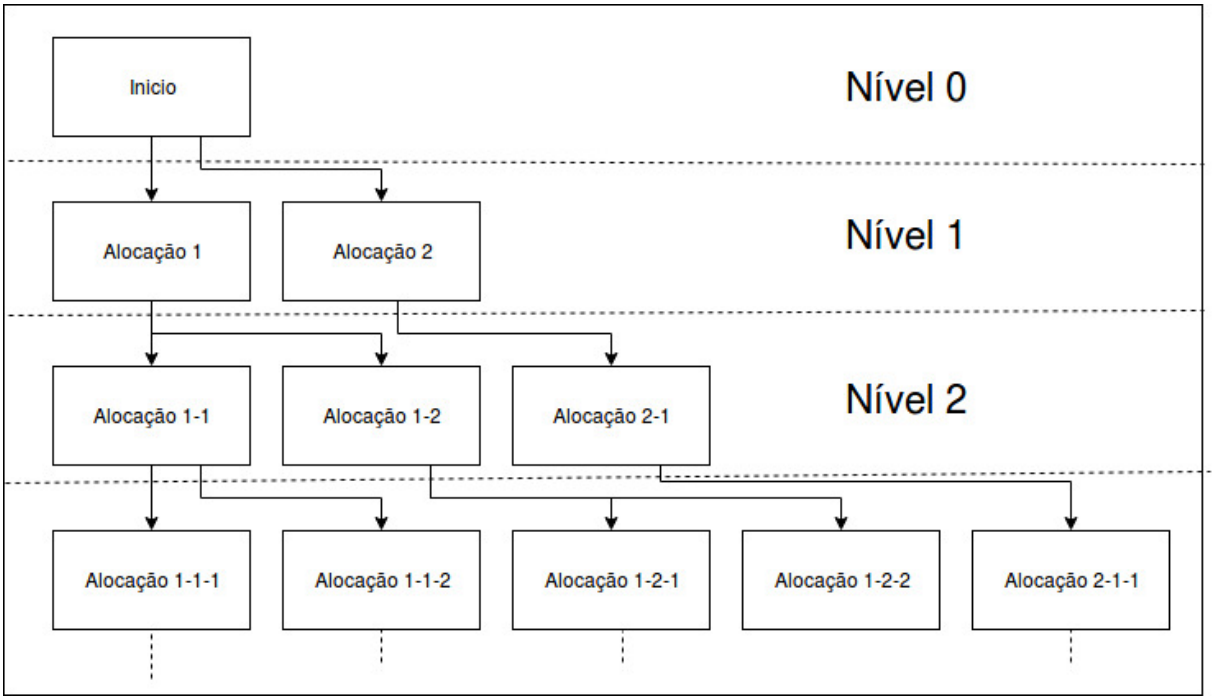
3.2.1 Busca por árvore de possibilidade

A busca por configuração poderia ser feita de forma bruta, ou seja, testando a observabilidade da rede dentro de todas as possibilidades de alocação de PMU, representado por um vetor com comprimento igual ao número de barras da rede com valor 0, representando a ausência de PMU ou 1 representando a presença do mesmo.

A grande desvantagem desse método de força bruta é a forma como ela escala para sistemas grandes. Ou seja, para sistemas de 14 barras as possibilidades a serem testadas são $2^{14} = 16.384$ possibilidades, já para o de 30 barras são de $2^{30} = 1.073.741.824$, logo para sistemas grandes o método é impraticável.

Para diminuir o tempo de busca, a cada inserção, buscam-se as novas possibilidades para as próximas inserções, como é apresentado na figura 10.

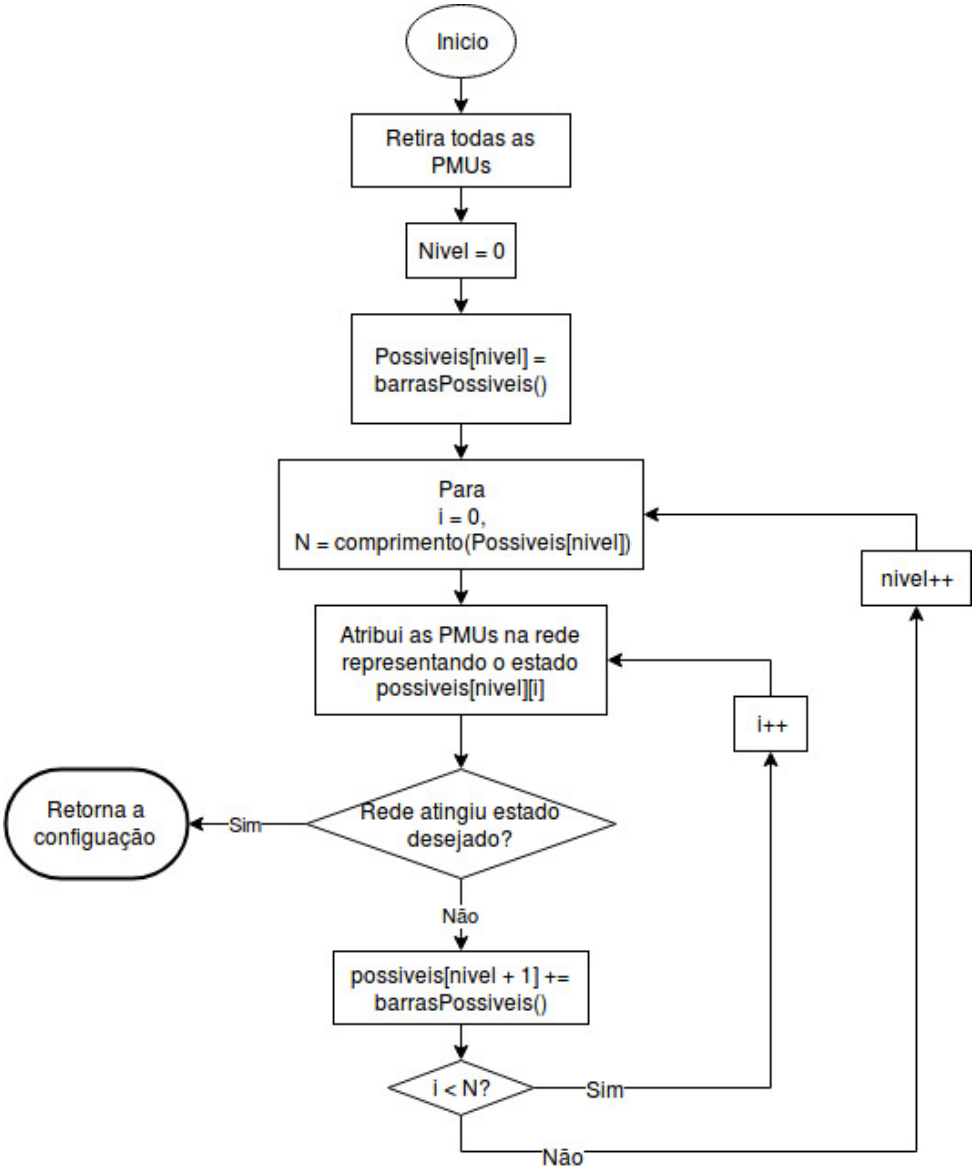
Figura 10: Exemplo de árvore de possibilidades



Realizando a busca a cada nível, a primeira solução encontrada também representa a solução com menor número de PMU. O algoritmo de busca generalizado utilizado para

alocação de PMUs em redes é o apresentado na figura 11, onde pode-se variar as funções de estado desejado da rede e as possibilidades de alocação para um dado estado da rede.

Figura 11: Algoritmo generalizado de alocação de PMU em uma rede de potência



3.2.2 Alocação para observabilidade total da rede

Para que a rede se torne totalmente observável com o mínimo de PMUs, deve-se primeiro considerar as barras não observáveis com apenas uma conexão entre barras não observáveis, pois não há outra forma de torná-las observáveis senão alocando uma PMU na barra em si ou na barra vizinha. De forma a tornar mais barras observáveis de forma indireta além dessa, torna-se prioridade alocar medidor na barra vizinha.

Caso nenhuma barra se encontre na primeira condição, serão considerados como possibilidades as barras não observáveis com o maior número de conexões entre barras não observáveis.

Por exemplo, na figura 12a, a barra número 8, em destaque azul, possui apenas uma conexão, sendo ela com a barra 7. Logo observa-se que as únicas possibilidades de torná-la observável é alocando um medidor na barra 7, marcada em vermelho, ou na própria barra 8. Para aproveitar das medições de corrente, é mais conveniente instalar a PMU na barra 7.

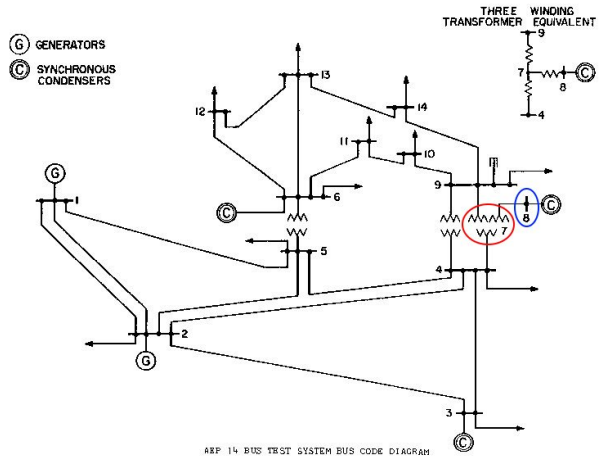
A lógica segue para as demais barras, pois esta alocação torna as barras 14, 10 e 3 na mesma situação da barra 8 ao alocar a primeira PMU na barra 7, como mostra a figura 12b. Destaca-se, em vermelho, a barra com PMU, em azul, as barras observáveis pela PMU e, em cinza, as barras não observáveis com apenas uma conexão entre barra não observável.

Tal estado indica que devem ser alocados PMUs nas barras 13, 11 e 2, tornando todo o sistema observável.

Caso todas as barras não observáveis se encontrem com mais de uma conexão entre outras barras na mesma condição, as novas possibilidades passam a ser as barras com maior numero de conexões.

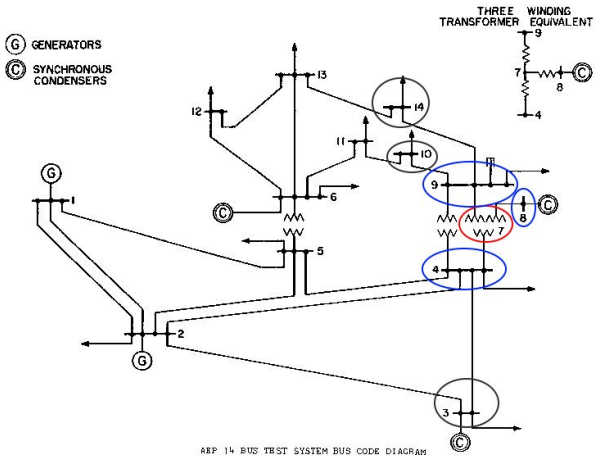
Figura 12: Alocação de PMUs na rede de 14 barras

(a) Primeira alocação



FONTE: Sistema... (1999a)

(b) Novas barras com apenas 1 conexão



FONTE: Sistema... (1999b)

3.2.3 Alocação de PMU para sistema não observável de nível 1

Para tornar o sistema não observável de nível 1, não é necessário a preocupação com as barras de apenas uma conexão com outra barra assim como o explicado na sessão 3.2.2. Logo, pode-se aplicar uma abordagem mais básica de limitação de possibilidade da árvore.

Para cada alocação de medidor, serão consideradas as barras não observáveis com o maior número de conexões com outras barras não observáveis para a alocação do próximo medidor.

Como exemplo, para o sistema de 14 barras, a relação inicial de cada barra não observada com outras barras é apresentada na tabela 1, logo, observa-se que as possibilidades iniciais são limitadas a apenas a barra 4, local onde primeira alocação deve ser feita.

Tabela 1: Número de ramos para cada barra no sistema de 14 barras

Nº barra	**Nº ramos**
1	2
2	4
3	2
4	5
5	4
6	4
7	3
8	1
9	4
10	2
11	2
12	2
13	3
14	2

3.3 Geração de dados de treino

Feita a alocação de PMUs na rede, é necessário a coleta de diferentes pontos de operação da rede.

3.3.1 Geração de pontos de operação

Para a geração dos dados de operação da rede de potência, foi aplicado fatores de aleatoriedade no perfil de carga da rede utilizando como base inicial os dados registrados no arquivo original.

São dois fatores de aleatoriedade principais que são aplicados a cada parâmetro relevante das barras. O fator aleatório geral da rede, ΔS, representando a variação geral na carga da rede, aplicado em todos parâmetros de potência das barras e calculado apenas uma vez, e o fator aleatório de cada barra, podendo ser ΔP para potência ativa de barras PQ, ΔQ para potência reativa de barras PQ ou ΔPV para potência ativa de barras PV, calculado diferente para cada barra.

O processo recebe como parâmetros principais:

- $\Delta S_{min} \in \mathbb{R}^+$
- $\Delta S_{max} \in \mathbb{R}^+$

- $\Delta P_{min} \in \mathbb{R}^+$
- $\Delta P_{max} \in \mathbb{R}^+$
- $\Delta Q_{min} \in \mathbb{R}^+$
- $\Delta Q_{max} \in \mathbb{R}^+$
- $\Delta PV_{min} \in \mathbb{R}^+$
- $\Delta PV_{max} \in \mathbb{R}^+$

Representando os limites de variação negativa e positiva, de cada parâmetro. Por exemplo, $S_{min} = 0,2$ e $S_{max} = 0,2$ representam uma variação de -20% a 20% no valor de consumo geral da rede.

As variações para cada exemplo de treino a ser gerado são obtidas a partir de um fator aleatório $a \in [0, 1]$ e convertendo a escala de 0 a 1 para X_{min} a X_{max}, portanto, são calculados como:

- $\Delta S = a_s * (\Delta S_{max} + \Delta S_{min}) - \Delta S_{min}$
- $\Delta P_i = a_{ip} * (\Delta P_{max} + \Delta P_{min}) - \Delta P_{min}$
- $\Delta Q_i = a_{iq} * (\Delta Q_{max} + \Delta Q_{min}) - \Delta Q_{min}$
- $\Delta PV_i = a_{ipv} * (\Delta PV_{max} + \Delta PV_{min}) - \Delta PV_{min}$

Sendo a_s, o fator aleatório de consumo geral da rede de potência, a_{ip} e a_{iq}, os fatores aleatórios individuais de potência ativa e reativa, respectivamente, da i-ésima barra PQ e a_{ipv} o fator aleatório individual de potência ativa da i-ésima barra PV.

Portanto, os valores de potência, consumidas ou geradas, para cada barra i são dados como:

- barras PQ:
  - $P_i = P_{0i} * (1 + \Delta P_i) * (1 + \Delta S)$
  - $Q_i = Q_{0i} * (1 + \Delta Q_i) * (1 + \Delta S)$
- barras PV:
  - $P_i = P_{0i} * (1 + \Delta PV_i) * (1 + \Delta S)$

Sendo P_{0i} e Q_{0i} os valores nominais de potência ativa e reativa para cada barra i.

3.3.2 Resolução do fluxo de potência

Com os dados aleatórios gerados, o fluxo de potência para o perfil estabelecido do sistema é resolvido, de forma a obter a informação de todas as barras. Para se obter melhor garantia de convergência no fluxo, o método de Gauss-Seidel foi utilizado com precisão de 10^{-5}.

3.3.3 Definição de entrada e saída da RNA e dos dados de treino

Com os dados calculados pela resolução do fluxo de potência, resta armazenar os dados para serem utilizados pelo treino da rede neural. Os dados são salvos no formato CSV em dois arquivos diferentes, um arquivo com os dados de medição disponíveis na rede de potência e um segundo arquivo com o valor calculado dos parâmetros de tensão e ângulo de fase das barras não observadas.

Define-se então que o arquivo com dados de medições, que serão utilizadas na entrada da rede neural, contém, para cada linha de exemplo, as colunas de dados:

- Caso barra com PMU
  - P_i
  - Q_i
  - V_i
  - Θ_i
  - $I_{ij}, j \in \Omega_i$
  - $I_{ij}, j \in \Omega_i$
- Caso barra *Slack* ou PV sem PMU
  - P_i
  - V_i

Considerando Ω_i o conjunto de barras conectadas a barra i e que as barras PQ são não controladas, sem medições.

O arquivo de dados das barras não observáveis, a serem utilizados como exemplo de saída da rede neural durante o seu treinamento, contém, para cada linha de exemplo, a coluna de dados:

- Caso barra PQ sem Slack:
  - V_i

  - Θ_i

- Caso barra PV sem Slack:

  - V_i

Dados de potência ativa e reativa, fasores de tensão das barras observáveis de forma indireta e fluxo de potência nos ramos, podem ser calculados diretamente por equações sem necessidade de um processo iterativo.

3.4 Determinação da topologia da rede neural

3.4.1 Número de camadas

Como o problema não apresenta linearidade em sua resolução, será utilizado uma rede com quatro camadas, ou seja, uma de entrada, duas escondidas e uma de saída.

3.4.2 Função de ativação

Para a ativação dos neurônios da camada escondida será utilizado a função sigmoide, que é do tipo $\mathbb{R} \rightarrow \mathbb{R}^+$, esta característica é importante porque, para cada valor de peso de um neurônio vale que $w_{ij} \in \mathbb{R}$, limitando a saída da função de ativação evita a equivalência entre multiplicação entre dois números positivos e a multiplicação entre dois números negativos, limitando uma possibilidade redundante de rede neural.

Para a ativação dos neurônios de saída, será utilizado a função linear, pois é um problema de generalização de função.

3.4.3 Número de neurônios por camada

De forma a se obter a melhor topologia de rede neural a resolver o problema de determinação de tensões nodais da rede nas barras não observadas, serão testados diferentes topologias e escolher o melhor, utilizando 80% dos dados para treino e 20% dos dados para teste. Dessa forma, a cada época de treino é verificado a evolução do erro, tanto nos dados de treino quanto nos dados de teste, de forma a evitar o *overfitting* devido ao número excessivo de épocas de treino.

A topologia inicial a ser treinada é a topologia de dez neurônios na primeira camada escondida e dez neurônios na segunda camada escondida (10-10). Ela é então treinada cinco vezes, limitando a um certo número de épocas, e a rede que obteve o melhor resultado em relação à média dos erros quadráticos médios com os dados de treino e os dados de teste é selecionada.

O processo inteiro se repete aumentando a quantidade em cinco neurônios para ambas as camada, até ser encontrada a topologia em que o erro não diminui satisfatoriamente, ou por conta do aparecimento de *overfitting* ou por pouco ganho de precisão.

O erro utilizado para a comparação será a média do erro absoluto médio de cada saída, dado pela equação 3.2, onde y_i é o valor desejado na i-ésima amostra e y_i^* é a saída da rede obtida. O processo será encerrado quando não houver redução de mais de 5% do EAM em relação à topologia anterior, valor calculado pela equação 3.3.

$$EAM = \sum_{i=1}^{N} \frac{|y_i - y_i^*|}{N} \tag{3.2}$$

$$Red_i\% = \frac{EAM_{i-1} - EAM_i}{EAM_{i-1}} * 100 \tag{3.3}$$

3.4.4 Método de treino

O treino a ser utilizado será o método de Gradiente Descente para PMC por ser um método mais conservador em relação à implementação matemática, com apenas a taxa de aprendizagem como parâmetro de ajuste.

3.4.5 Treino por lotes

Será também analisado a necessidade de treino por lotes, se há vantagem em sua utilização e também se a utilização de uma GPU com CUDA pode influenciar positivamente no processamento. Será analisado três treino simples da rede 10-10 para o problema de 14 barras para quatro configurações diferentes:

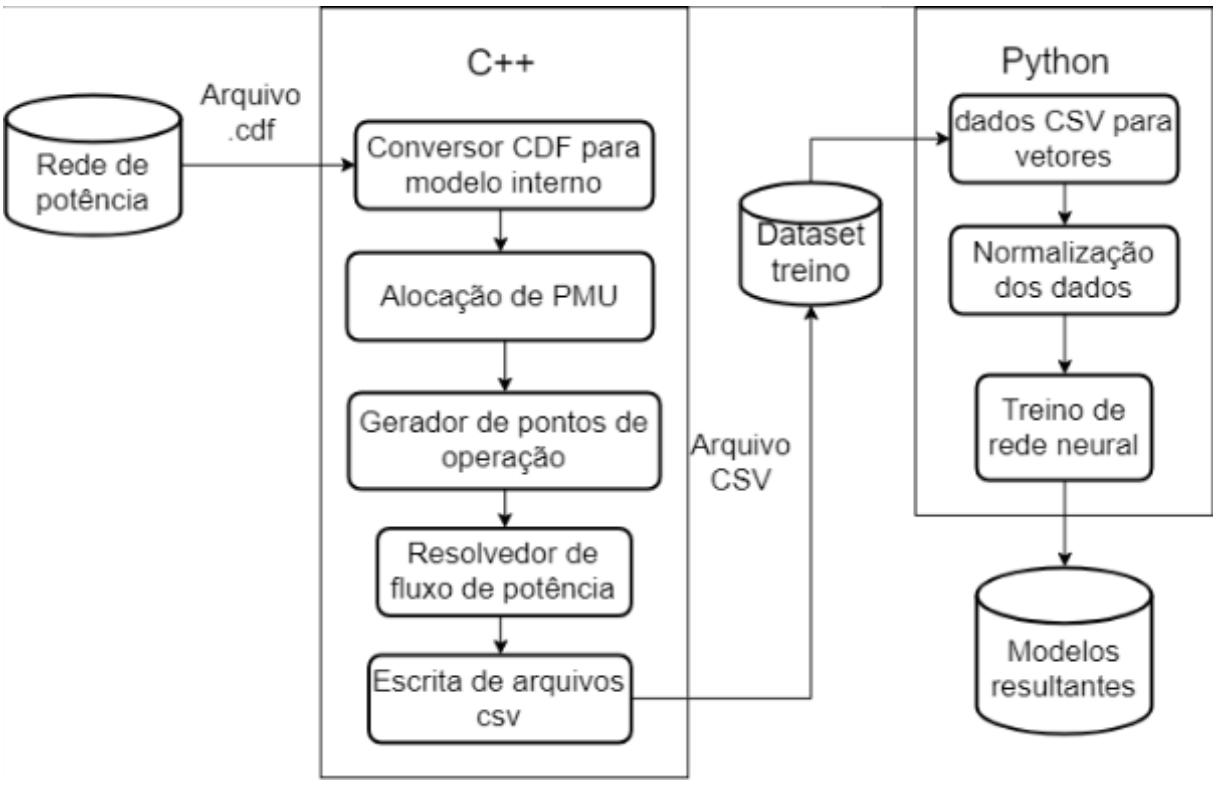
- CPU com 1 item por lote;
- GPU com 1 item por lote;
- CPU com 256 itens por lote;
- GPU com 256 itens por lote.

Coletando a média de épocas e do tempo atingido na análise, é possível determinar o melhor método a ser utilizado para o resto do treino.;

3.5 Fluxograma processo

Em resumo, o processo de treino sendo pesquisado pode ser resumido no fluxograma apresentado na figura 13.

Figura 13: Fluxograma processo geração de rede neural para estimação de estado em barras não observáveis



O processo se inicia com a descrição da rede de potência em estudo no formato .cdf, este arquivo é alimentado ao programa em c++, responsável pela geração dos dados para o treino da rede neural e estes são alimentados à um *script* Python, onde será utilizada a interface do *tensorflow* para geração e treino da rede neural.

3.6 Metodologia de programação

Como o projeto envolve a programação em linguagem C++ e python e, como não se tem muito conhecimento prévio da estrutura final do programa, será utilizado métodos de desenvolvimento ágil de software.

O código de geração de dados de treino para a rede neural será escrito em c++ utilizando o paradigma de programação orientada a objetos (DEITEL H.M. DEITEL, 2001), de forma a se obter a melhor qualidade de escrita de programa e se obter melhor aproveito da performance atribuída ao programa compilado com otimizações feitas pelo compilador.

O desenvolvimento do programa em C++, por ser extenso deve também ser aplicado o conceito do TDD(JANZEN, 2005), de forma manter fluxo de desenvolvimento fluido com flexibilidade para modificações.

O código de treino da rede neural será escrito em Python, pois a biblioteca *tensorflow*

oferece melhor interface de utilização nesta linguagem, apesar da linguagem interpretada, a performance não é afetada pois a biblioteca sendo instalada direto do código fonte, cuja base de processamento é escrita em C++, otimizações por processador podem ser feitas. O paradigma escolhido para esta sessão foi o procedural utilizando interfaces dos objetos pré-definidos em pacotes externos.

3.6.1 Hardware utilizado

Para o processamento e comparação de tempos de execução será utilizada uma estação com as seguintes configurações:

- Processador intel i5-8400 2,800GHz
- 8GB de memória ram DDR5 2,400GHz
- Placa de vídeo Nvidia GTX 1070ti com 8GB de ram dedicada
- Sistema operacional Ubuntu 16.04.5 LTS
- Python 3.5.2
- *tensorflow* 1.10.0
- GNU GCC-5 (usado para compilação de binários do *tensorflow*)

4 RESULTADOS

4.1 Alocação de medidores

4.1.1 14 Barras

Para o caso de 14 barras, o algoritmo de alocação de PMUs pôde ser realizado sem a utilização do algoritmo computacional, algo que também confirma e valida a implementação do algoritmo no programa, cujo resultado pode ser utilizado para implementar teste unitário.

4.1.1.1 Observabilidade completa

Como citado em 3.2.2, a árvore de possibilidade para alocação das barras no sistema é o mostrado na figura 14, onde os diferentes ramos após a primeira colocação da PMU foram simplificados em apenas um passo em que se aloca as PMUs em cada possibilidade.

4.1.1.2 Não-observabilidade de 1º grau

A árvore de possibilidades para o sistema de 14 barras é apresentado na figura 15, logo, pode-se determinar que as posições ideais para alocação de PMUs são as barras 4 e 6.

Na alocação do segundo medidor, as barras 6 e 13 são considerados pois, apesar da barra 6 possuir conexões com 4 outras barras, uma delas já é observável de forma indireta, logo, não é contabilizado, totalizando o número de conexões para três para cada uma das barras.

4.1.2 30 Barras

Para o caso de 30 barras, as possibilidades são maiores e a árvore mais complexa, portanto, elas não foram feitas de forma manual, deixando este trabalho para o computador.

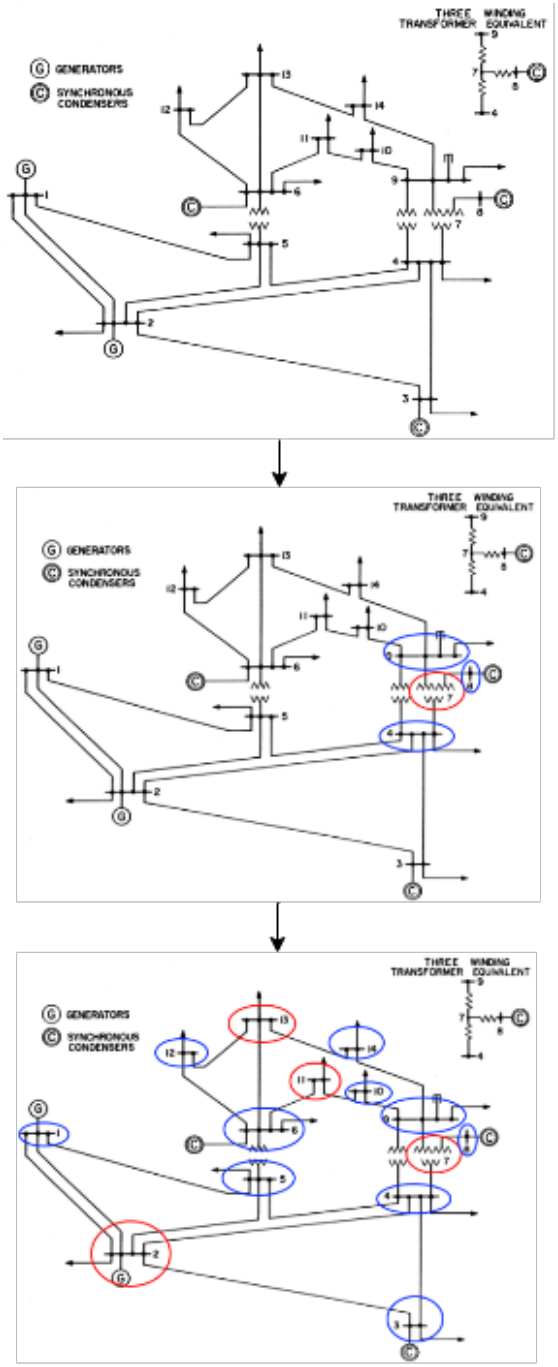
As barras para observabilidade total são 3, 5, 9, 12, 16, 19, 22, 24, 25, 27 e 28 e estão apresentados na figura 16a, onde barras com marcação vermelha são barras com medidores e azuis são barras observáveis de forma indireta.

As barras para não-observabilidade de 1º grau são as barras 1, 6, 12, 19, 24 e 27 e estão apresentados na figura 16b, onde as barras 5, 11, 17, 21 e 26 se mantém como não observáveis.

4.1.3 Comparação

Comparando os resultados lado a lado, em relação ao número de PMUs utilizados, resulta na tabela 2, onde percebe-se que o método aplicado tem potencial de diminuição

Figura 14: Arvore de possibilidade para alocação de PMUs na rede IEEE de 14 barras, observabilidade total

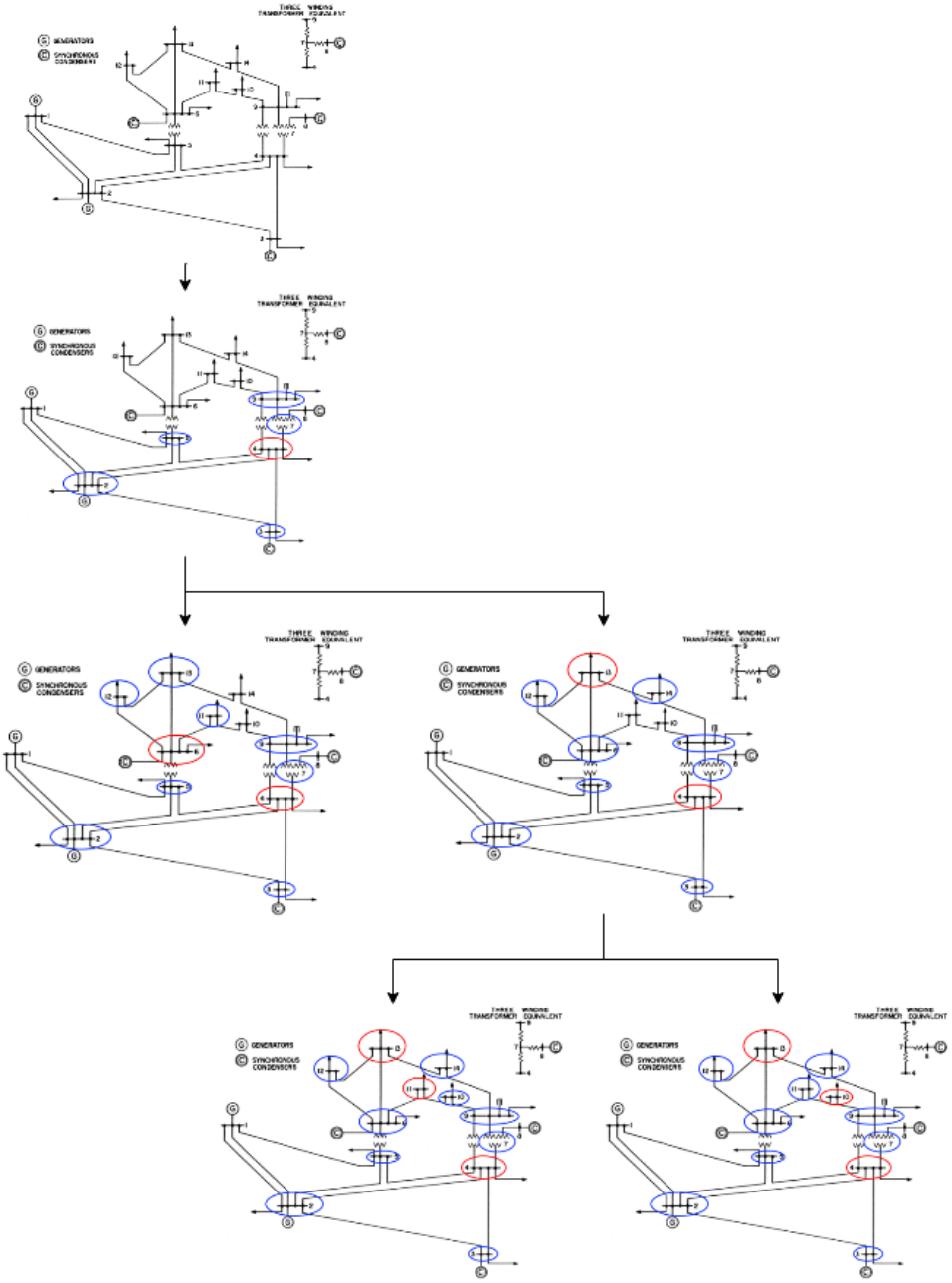


da utilização de PMU em até 50%, dependendo da estrutura da rede de potência.

Tabela 2: Comparação do número de PMUs instalados

Nº de barras	Nº PMUs observabilidade total	Nº PMUs não-observabilidade nivel 1	Redução
14	4	2	50%
30	11	6	45.45%

Figura 15: Árvore de possibilidade para alocação de PMUs na rede IEEE de 14 barras, não observabilidade de 1º grau



4.2 Geração de dados de treino

4.2.1 Geração de pontos de operação

Os parâmetros utilizados para a geração de dados para ambas as redes estão listados na tabela 3.

Figura 16: Alocação de PMU para o sistema IEEE 30 barras

(a) Observabilidade total (b) Não-observabilidade de 1º grau

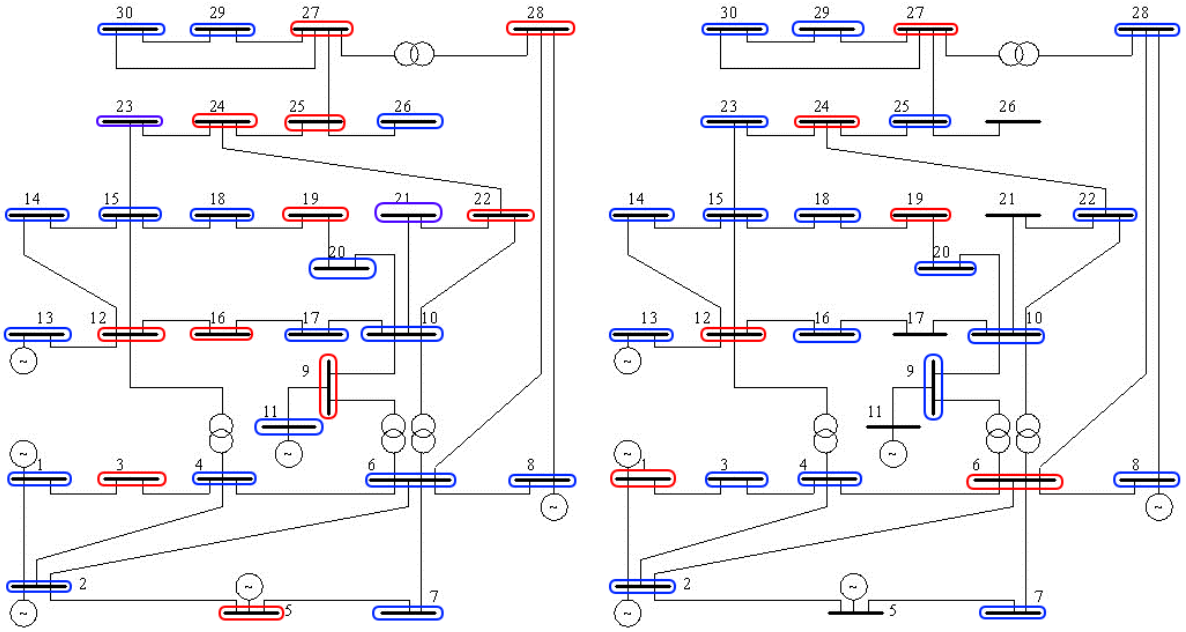


Tabela 3: Parâmetros de geração de pontos de operação da rede

Parâmetro	Valor
ΔS_{min}	0.3
ΔS_{max}	0.03
ΔP_{min}	0.02
ΔP_{max}	0.01
ΔQ_{min}	0.01
ΔQ_{max}	0.05
ΔPV_{min}	0.04
ΔPM_{max}	0.01

4.2.2 Geração de arquivos com dados

A definição dos dados de entrada e saída das redes neurais, assim como a média e o desvio padrão obtida do conjunto de treino a ser aplicado na normalização dos mesmos, estão apresentados nas tabelas 4 para o problema de 14 barras e 5 para o problema de 30 barras, onde $I_{i.j}$ indica módulo da corrente da barra i para a barra j e $\theta_{I_{i.j}}$ indica o ângulo de fase.

Para tanto, foram utilizados 100.000 exemplos onde 80.000 serão utilizados para treinamento e 20.000 utilizados para validação.

Observa-se um grande valor de desvio padrão para o ângulo de fase da corrente da barra 4 para 2 no caso de 14 barras, após análise dos dados observou-se que a grande

variação se deu pela aproximação do valor com o valor π que dependendo do perfil, apresentava atraso ou adiantamento próximo a π.

Tabela 4: Dados para treino de rede neural, 14 barras

Saída/Entrada	**Variável**	**Média**	**Desvio Padrão**
Entrada	P_1	1,975	229,572E-03
	V_1	1,060	1,432E-12
	P_2	155,807E-03	17,330E-03
	V_2	1,045	1,461074E-12
	P_3	-801,9789E-03	89,216E-03
	V_3	1,010	1,406653E-12
	P_6	-95,343E-03	10,616E-03
	Q_6	51,973E-03	36,988E-03
	V_6	1,070	2,542E-12
	θ_6	-223,943E-03	26,134E-03
	$I_{6.5}$	368,785E-03	34,464E-03
	$\theta_{I_{6.5}}$	2,615	39,798E-03
	$I_{6.11}$	82,400E-03	11,221E-03
	$\theta_{I_{6.11}}$	-1,019	43,971E-03
	$I_{6.12}$	67,532E-03	7,816E-03
	$\theta_{I_{6.12}}$	-585,886E-03	31,143E-03
	$I_{6.13}$	159,929E-03	18,841E-03
	$\theta_{I_{6.13}}$	-699,075E-03	33,889E-03
	P_8	0,000	0,000
	V_8	1,090	2,402E-12
	P_4	-411,071E-03	45,484E-03
	Q_4	33,623E-03	3,712E-03
	V_4	1,032	2,633E-03
	θ_4	-154,901E-03	17,922E-03
	$I_{4.2}$	457,926E-03	51,005E-03
	$\theta_{I_{4.2}}$	2,615	1,692
	$I_{4.3}$	205,075E-03	16,889E-03
	$\theta_{I_{4.3}}$	-349,514E-03	111,041E-03
	$I_{4.5}$	547,187E-03	57,987E-03
	$\theta_{I_{4.5}}$	-2,946	28,336E-03
	$I_{4.7}$	297,179E-03	20,112E-03
	$\theta_{I_{4.7}}$	493,731E-03	76,364E-03

Continua na próxima página

Tabela 4 – continuação da página anterior

Saída/Entrada	Parâmetro	Média	Desvio Padrão
	Entrada $I_{4.9}$	151,303E-03	10,999E-03
	$\theta_{I_{4.9}}$	348,297E-03	87,217E-03
Saída	θ_2	-72,974E-03	9,013E-03
	θ_3	-185,417E-03	22,996E-03
	θ_8	-203,306E-03	23,442E-03
	V_5	1,031	2,436E-03
	θ_5	-131,466E-03	15,373E-03
	V_7	1,048	2,653E-03
	θ_7	-203,307E-03	23,442E-03
	V_9	1,044	4,138E-03
	θ_9	-227,917E-03	26,283E-03
	V_10	1,042	4,140E-03
	θ_{10}	-231,432E-03	26,738E-03
	V_{11}	1,053	2,438E-03
	θ_{11}	-229,493E-03	26,624E-03
	V_{12}	1,056	1,654E-03
	θ_{12}	-236,451E-03	27,536E-03
	V_{13}	1,051	2,317E-03
	θ_{13}	-236,950E-03	27,557E-03
	V_{14}	1,031	5,150E-03
	θ_{14}	-246,994E-03	28,631E-03

Tabela 5: Dados para treino de rede neural, 30 barras

Saída/Entrada	Parâmetro	Média	Desvio Padrão
Entrada	P_1	2,214	259,748E-03
	Q_1	-245,701E-03	41,937E-03
	V_1	1,060	1,432E-12
	P_1	0,000	0,000
	$I_{1.2}$	1,388	166,676E-03
	$\theta_{I_{1.2}}$	116,867E-03	15,719E-03
	$I_{1.3}$	713,419E-03	81,529E-03
	$\theta_{I_{1.3}}$	96,029E-03	12,539E-03
	P_2	155,907E-03	17,389E-03

Continua na próxima página

Tabela 5 – continuação da página anterior

Saída/Entrada	Parâmetro	Média	Desvio Padrão
Entrada	V_2	1,043	1,422E-12
	P_5	-802,443E-03	89,477E-03
	V_5	1,010	1,407E-12
	P_8	-255,578E-03	28,512E-03
	V_8	1,010	1,407E-12
	P_{11}	0,000	0,000
	V_{11}	1,082	1,555E-12
	P_{13}	0,000	0,000
	V_{13}	1,071	385,448E-15
	P_6	0,000	0,000
	Q_6	0,000	0,000
	V_6	1,022	1,930E-03
	θ_6	-164,686E-03	19,755E-03
	$I_{6.2}$	491,579E-03	55,396E-03
	$\theta_{I_{6.2}}$	3,027	1,972E-03
	$I_{6.8}$	325,021E-03	12,260E-03
	$\theta_{I_{6.8}}$	-836,514E-03	163,219E-03
	$I_{6.4}$	638,182E-03	63,500E-03
	$\theta_{I_{6.4}}$	2,843	52,564E-03
	$I_{6.7}$	322,851E-03	35,398E-03
	$\theta_{I_{6.7}}$	-193,672E-03	14,687E-03
	$I_{6.9}$	282,467E-03	19,839E-03
	$\theta_{I_{6.9}}$	356,283E-03	90,742E-03
	$I_{6.10}$	144,826E-03	11,735E-03
	$\theta_{I_{6.10}}$	174,367E-03	101,209E-03
	$I_{6.28}$	172,469E-03	17,868E-03
	$\theta_{I_{6.28}}$	-357,625E-03	4,822E-03
	P_{12}	-96,380E-03	10,688E-03
	Q_{12}	-64,702E-03	7,158E-03
	V_{12}	1,020	3,361E-03
	θ_{12}	-231,570E-03	27,775E-03
	$I_{12.13}$	364,468E-03	24,005E-03
	$\theta_{I_{12.13}}$	1,339	27,776E-03
	$I_{12.4}$	419,862E-03	33,739E-03
	$\theta_{I_{12.4}}$	-2,756	85,144E-03
	$I_{12.14}$	63,635E-03	8,038E-03

Continua na próxima página

Tabela 5 – continuação da página anterior

Saída/Entrada	Parâmetro	Média	Desvio Padrão
Entrada	$\theta_{I_{12.14}}$	-471,429E-03	44,821E-03
	$I_{12.15}$	141,658E-03	18,369E-03
	$\theta_{I_{12.15}}$	-478,541E-03	59,406E-03
	$I_{12.16}$	50,540E-03	7,141E-03
	$\theta_{I_{12.16}}$	-410,742E-03	105,271E-03
	P_{19}	-81,753E-03	9,068E-03
	Q_{19}	-29,332E-03	3,245E-03
	V_{19}	999,807E-03	7,126E-03
	θ_{19}	-258,508E-03	30,741E-03
	$I_{19.18}$	17,211E-03	2,279E-03
	$\theta_{I_{19.18}}$	1,004	2,826
	$I_{19.20}$	72,506E-03	7,185E-03
	$\theta_{I_{19.20}}$	2,421	10,981E-03
	P_{24}	-74,867E-03	8,304E-03
	Q_{24}	-57,801E-03	6,395E-03
	V_{24}	1,003	7,829E-03
	θ_{24}	-255,079E-03	29,976E-03
	$I_{24.22}$	47,030E-03	6,205E-03
	$\theta_{I_{24.22}}$	2,625	73,485E-03
	$I_{24.23}$	8,998E-03	1,466E-03
	$\theta_{I_{24.23}}$	-2,443	434,469E-03
	$I_{24.25}$	26,335E-03	2,287E-03
	$\theta_{I_{24.25}}$	2,525	34,706E-03
	P_{27}	0,000	0,000
	Q_{27}	0,000	0,000
	V_{27}	1,023	5,818E-03
	θ_{27}	-240,785E-03	28,588E-03
	$I_{27.25}$	62,227E-03	6,570E-03
	$\theta_{I_{27.25}}$	-744,376E-03	33,914E-03
	$I_{27.28}$	175,666E-03	20,104E-03
	$\theta_{I_{27.28}}$	2,567	31,895E-03
	$I_{27.29}$	53,767E-03	6,421E-03
	$\theta_{I_{27.29}}$	-500,180E-03	32,019E-03
	$I_{27.30}$	61,074E-03	7,311E-03
	$\theta_{I_{27.30}}$	-466,607E-03	32,364E-03
Saída	θ_2	-78,057E-03	9,767E-03

Continua na próxima página

Tabela 5 – continuação da página anterior

Saída/Entrada	Parâmetro	Média	Desvio Padrão
Saída	θ_5	-206,724E-03	25,509E-03
	θ_8	-172,543E-03	21,557E-03
	θ_{11}	-215,321E-03	25,512E-03
	θ_{13}	-231,569E-03	27,775E-03
	V_3	1,039	2,322E-03
	θ_3	-114,282E-03	13,337E-03
	V_4	1,033	2,551E-03
	θ_4	-140,183E-03	16,505E-03
	V_7	1,011	1,999E-03
	θ_7	-190,174E-03	22,941E-03
	V_9	1,030	3,180E-03
	θ_9	-215,322E-03	25,512E-03
	V_{10}	1,020	5,312E-03
	θ_{10}	-241,555E-03	28,580E-03
	V_{14}	1,009	5,023E-03
	θ_{14}	-245,390E-03	29,399E-03
	V_{15}	1,006	5,587E-03
	θ_{15}	-247,036E-03	29,450E-03
	V_{16}	1,013	4,956E-03
	θ_{16}	-240,389E-03	28,587E-03
	V_{17}	1,014	5,722E-03
	θ_{17}	-244,478E-03	29,005E-03
	V_{18}	1,001	6,760E-03
	θ_{18}	-256,150E-03	30,493E-03
	V_{20}	1,004	6,756E-03
	θ_{20}	-255,194E-03	30,304E-03
	V_{21}	1,010	6,613E-03
	θ_{21}	-248,639E-03	29,407E-03
	V_{22}	1,011	6,564E-03
	θ_{22}	-248,458E-03	29,372E-03
	V_{23}	1,002	6,921E-03
	θ_{23}	-252,856E-03	29,963E-03
	V_{25}	1,011	7,141E-03
	θ_{25}	-248,789E-03	29,426E-03
	V_{26}	995,575E-03	8,973E-03
	θ_{26}	-255,158E-03	30,231E-03

Continua na próxima página

Tabela 5 – continuação da página anterior

Saída/Entrada	Parâmetro	Média	Desvio Padrão
Saída	V_{28}	1,016	2,195E-03
	θ_{28}	-173,904E-03	20,863E-03
	V_{29}	1,006	7,887E-03
	θ_{29}	-259,236E-03	30,874E-03
	V_{30}	996,267E-03	9,081E-03
	θ_{30}	-272,428E-03	32,553E-03

4.3 Treino de rede neural

4.3.1 Comparação de tempo

Os tempos resultantes para a análise preliminar de treino para atingir erro de $10^{-}2$ estão apresentados na tabela 6 enquanto que a quantidade de épocas está apresentado na tabela 7.

Tabela 6: Tempo (s) médio de processamento da rede neural, erro médio absoluto = 10^{-2}

Lote / Processador	1 por lote	256 por lotes
GPU	367	255
CPU	175	173

Tabela 7: Número de épocas para atingir erro médio absoluto = 10^{-2}

Lote / Processador	1 por lote	256 por lotes
GPU	5	733
CPU	5	755

Observou-se que a utilização da GPU não traz benefícios ao processamento, por conta do tamanho dos dados de treino não ser grande o suficiente para que uma vantagem seja notável e também o problema tem melhor convergência com utilizando CPU.

Observou-se uma leve vantagem do treino por lotes com relação ao tempo, porém, buscando melhor precisão no resultado final, foi escolhido realizar o treino com 1 dado por lote.

4.3.2 Comparação de topologias

Levando em consideração que não será utilizado o treino por lotes, cada etapa de treino foi limitado a 100 épocas de forma a limitar o tempo por etapas, considerando-se também que levou-se apenas 5 épocas para atingir erro menor que 10^{-2}.

Os resultados da evolução do erro para o treino da rede neural a medida que o número de neurônios nas camadas aumenta estão apresentados nas tabela 8 e 9 para o problema com 14 e 30 barras, respectivamente.

Tabela 8: Evolução do erro quadrático médio da rede neural para problema de 14 barras

Topologia	**EAM**	**EAM (validação)**	**Média**	**Redução (%)**
10/10	4,850E-03	4,670E-03	4,760E-03	0,000
15/15	3,815E-03	3,600E-03	3,708E-03	22,111
20/20	3,439E-03	3,286E-03	3,363E-03	9,305
25/25	3,149E-03	3,052E-03	3,101E-03	7,792
30/30	3,141E-03	3,007E-03	3,074E-03	0,855

Tabela 9: Evolução do erro quadrático médio da rede neural para problema de 30 barras

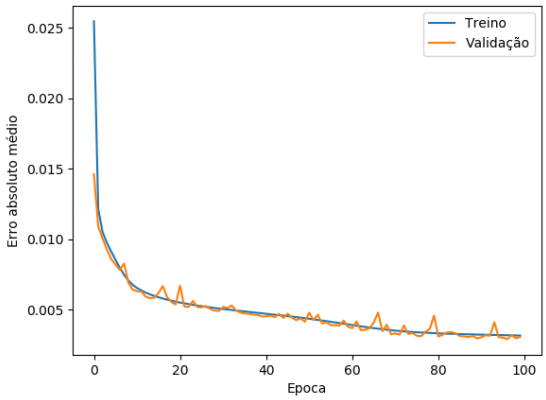
Topologia	**EAM**	**EAM (validação)**	**Média**	**Redução (%)**
10/10	4,494E-03	4,376E-03	4,435E-03	0,000
15/15	3,874E-03	3,624E-03	3,749E-03	15,468
20/20	3,222E-03	3,103E-03	3,163E-03	15,644
25/25	2,855E-03	2,720E-03	2,788E-03	11,858
30/30	2,538E-03	2,427E-03	2,483E-03	10,942
35/35	2,413E-03	2,307E-03	2,360E-03	4,935

Com tais resultados, foi concluído que para o problema de 14 barras, a topologia de 25 neurônios em cada camada escondida deve ser utilizado, pois não há melhoria significativa para o uso da topologia com 30 neurônios, enquanto que para o problema de 30 neurônios, a topologia de 35/35 deve ser utilizada. As evoluções do erro absoluto médio no treino de cada uma dessas topologias está apresentado na figura 17.

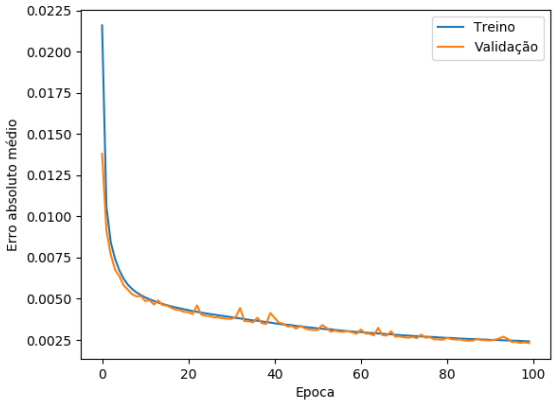
Os valores dos pesos que representam tais topologias são grandes, sendo 2569 valores para a rede do problema de 14 barras e 5643 para a rede de 30 barras, portanto, fica inviável o registro em papel e será armazenado junto com o código fonte em um repositório github.

Figura 17: Evolução do erro médio absoluto por época

(a) 14 barras



(b) 30 barras



4.3.3 Análise final de erro

Os erros (EAM) para cada saída da rede do problema de 14 barras e o problema de 30 barras, junto com os seus respectivos valores multiplicados ao desvio padrão ($EAM * \mu$, estão apresentados na tabela 10 e na tabela 11, respectivamente.

Tabela 10: Erro absoluto médio da previsão de barras não observáveis da rede de 14 barras

Variável	**EAM**	**EAM*μ**
θ_2	1,310E-03	11,810E-06
θ_3	1,208E-03	27,769E-06
θ_8	1,010E-03	23,684E-06
V_5	2,687E-03	6,545E-06
θ_5	1,787E-03	27,469E-06
V_7	3,261E-03	8,649E-06
θ_7	1,019E-03	23,881E-06
V_9	4,138E-03	17,122E-06
θ_9	1,110E-03	29,177E-06
V_{10}	5,244E-03	21,712E-06
θ_{10}	1,457E-03	38,953E-06
V_{11}	4,802E-03	11,709E-06
θ_{11}	1,561E-03	41,550E-06
V_{12}	2,118E-03	3,503E-06
θ_{12}	1,426E-03	39,275E-06
V_{13}	3,051E-03	7,069E-06
θ_{13}	1,438E-03	39,623E-06
V_{14}	15,616E-03	80,421E-06
θ_{14}	3,759E-03	107,625E-06

Tabela 11: Erro Absoluto médio da previsão de barras não observáveis da rede de 30 barras

Variável	**EAM**	**EAM*μ**
θ_2	2,110E-03	20,610E-06
θ_5	1,909E-03	48,702E-06
θ_8	2,005E-03	43,219E-06
θ_{11}	1,022E-03	26,062E-06
θ_{13}	1,371E-03	38,074E-06
V_3	2,756E-03	6,401E-06
θ_3	1,648E-03	21,985E-06
V_4	2,350E-03	5,995E-06
θ_4	1,668E-03	27,534E-06
V_7	3,527E-03	7,051E-06
θ_7	1,632E-03	37,446E-06
V_9	1,543E-03	4,906E-06
θ_9	1,119E-03	28,554E-06
V_{10}	1,652E-03	8,774E-06
θ_{10}	1,554E-03	44,408E-06
V_{14}	6,173E-03	31,008E-06
θ_{14}	1,639E-03	48,187E-06
V_{15}	2,920E-03	16,312E-06
θ_{15}	1,391E-03	40,964E-06
V_{16}	4,978E-03	24,668E-06
θ_{16}	1,571E-03	44,898E-06
V_{17}	4,815E-03	27,550E-06
θ_{17}	1,624E-03	47,105E-06
V_{18}	3,376E-03	22,822E-06
θ_{18}	1,591E-03	48,526E-06
V_{20}	2,918E-03	19,714E-06
θ_{20}	1,811E-03	54,876E-06
V_{21}	3,409E-03	22,544E-06
θ_{21}	1,574E-03	46,273E-06
V_{22}	3,077E-03	20,199E-06
θ_{22}	1,616E-03	47,458E-06
V_{23}	3,988E-03	27,603E-06
θ_{23}	1,423E-03	42,647E-06
V_{25}	1,802E-03	12,868E-06
θ_{25}	1,695E-03	49,874E-06
V_{26}	4,928E-03	44,218E-06
θ_{26}	2,087E-03	63,097E-06
V_{28}	1,364E-03	2,994E-06
θ_{28}	1,501E-03	31,308E-06
V_{29}	2,702E-03	21,314E-06
θ_{29}	1,343E-03	41,457E-06
V_{30}	2,178E-03	19,782E-06
θ_{30}	1,670E-03	54,363E-06

O gráficos de barras representativo do resultado apresentado das tabelas 10 e 9 estão apresentados nas figuras 18 e 19, respectivamente. É possível observar uma média de erro próximo a $3 * 10^{-5}$ para ambos os casos, onde o erro não ultrapassa 10^{-4}, com apenas uma exceção para o problema de 14 barras e mesmo esse erro continua não sendo significativo.

Figura 18: Gráfico de erro por saída, problema 14 barras

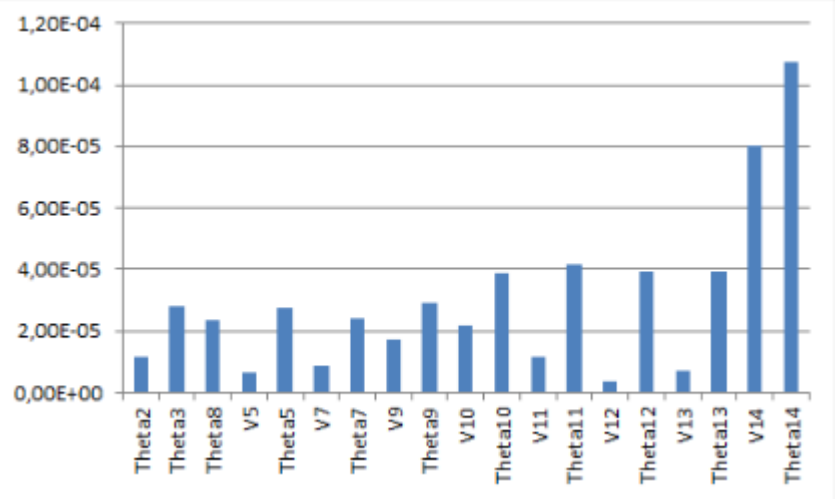
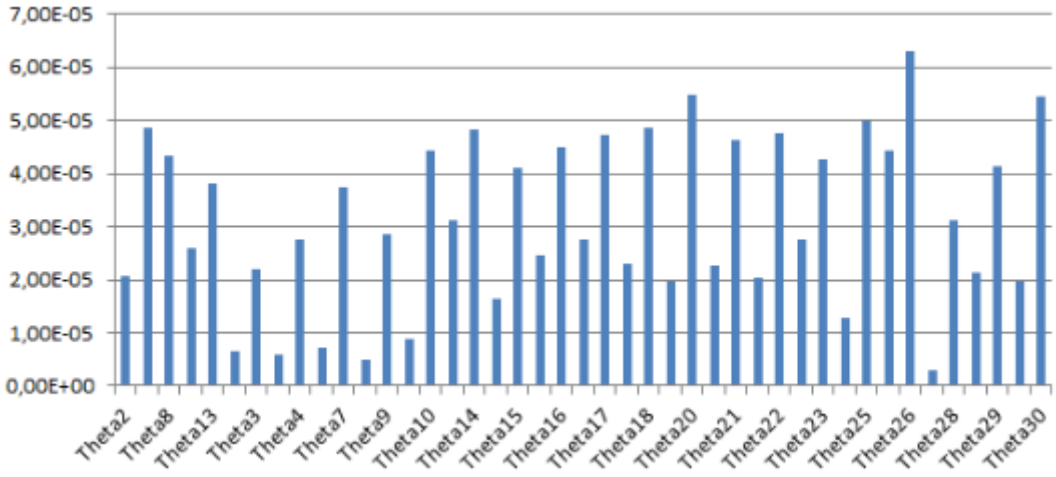


Figura 19: Gráfico de erro por saída, problema 30 barras



4.4 Código fonte, arquivos binários e CSV

Todos os arquivos de código fonte desenvolvidos no projeto foram hospedados na internet no Github, plataforma de compartilhamento utilizado de código fonte para desenvolvedores, podendo ser acessados via o link <https://github.com/BigsonLvrocha/NeuralPowerFlow>.

Os arquivos principais para a pesquisa estão no apêndice, sendo o gerador de dados de treino para as redes o apêndice A, o *script* de comparação de tempo de execução o apêndice B e o *script* de treino de rede neural no apêndice C.

Os arquivos CSV e binários gerados pelos programas, isso inclui os dados utilizados no treino e valores resultantes de peso das redes neurais obtidas, também se encontram no mesmo repositório.

5 CONCLUSÃO

Com base na análise feita e os resultados obtidos, é possível concluir que, para um caso de redução de custos em relação a instalação de PMUs, para obter dados em tempo real, uma rede neural pode ser utilizado sem perda significativa de precisão. Segundo as tabelas 10 e 11, para dados de medições obtidas com 10^{-5} de precisão, uma rede neural é capaz de determinar o valor das barras não observadas com precisão acima de 10^{-4}.

O caso também pode potencialmente ser aplicado em caso de falha de medidor em redes de potência com observabilidade total, caso tal falha a rede mantenha pelo menos o 1º nível de não-observabilidade, redes neurais treinadas para tal situação podem substituir tal medidor no monitoramento.

Também é necessário destacar que a precisão dos valores calculados através das redes neurais dependem diretamente da integridade e representatividade dos dados de treino, quanto mais representativo é a geração de pontos de operação da rede, melhor é a performance de tal método de estimação.

Nota-se também que a dependência dos dados de teste é um limitante para a utilização de tal método, onde, para considerar uma situação diferente, é necessário realizar o treinamento da rede neural mais uma vez e, como foi observado, é uma operação que demanda tempo.

Porém a grande vantagem do processo é a utilização da modelagem simplificada para geração dos dados de treino, ou seja, as situações são previamente simulados e o processo de treino da rede neural realiza o trabalho de definição da função de previsão. Ou seja, um processo empírico que possuí eficiência comprovada e determinada antes de entrar em operação. Não é um modelo que utiliza de pseudo-medidas que introduzem grande quantidade de erro, nem dependem de utilização de matrizes jacobianas, na estimação como ocorre nos métodos tradicionais.

Uma outra possível utilização é a reutilização da rede neural previamente treina sobre um conjunto de treino para a previsão de uma nova situação sob a mesma rede de potência, realizando o processo de treino com os pesos iniciais iguais à rede anterior, possivelmente diminuindo o tempo do segundo treino da rede.

Tal método, porém, se baseia em configuração fixa a regime permanente, não é projetada para operar com variações de impedância de linha nem faltas, porém, pesquisa posterior pode ser feita explorando tais situações.

REFERÊNCIAS

ABADI, M. et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. **CoRR**, abs/1603.04467, 2016. Disponível em: <http://arxiv.org/abs/1603.04467>.

_____. Tensorflow: A system for large-scale machine learning. **CoRR**, abs/1605.08695, 2016. Disponível em: <http://arxiv.org/abs/1605.08695>.

ARISTOY G., I. S. F. E. G. A. Power network state estimation algorithms and their use with synchrophasors. **IEEE PES Innovative Smart Grid Technologies Latin America**, 2015.

BOOST library. 2005. Disponível em: <http://www.boost.org/>. Acesso em: 2017.12.5.

DEITEL H.M. DEITEL, P. Programação orientada a objeto. In: **C++ Como programar**. 3. ed. Porto Alegre: Bookman, 2001. cap. 6, p. 396–397.

EFE S.B, C. M. Power flow analysis by artificial neural network. **International Journal Energy and Power Engineering**, v. 2, n. 6, p. 204–208, 2013.

FUNAHASHI. On the approximate realization of continuous mappings by neural networks. **Neural Network**, v. 2, n. 3, p. 183–192, 1989.

G., S.; BISWAS, K. K.; MAHALANABIS, A. K. **Statistical linearization approach to power-system state estimation**. 2007. 411-418 p.

IRIE B. MIYAKE, S. Capabilities of three-layered perceptrons. In: **IEEE 1988 International Conference on Neural Networks**. [S.l.]: IEEE, 1993. v. 1, p. 641–648.

JANZEN, D. S. Test-driven development concepts, taxonomy, and future direction. **Computer**, v. 38, n. 9, p. 43–50, 9 2005.

KUMAR, N. et al. **aplication of artificial neural networks to load flow solutions**. Kampur, 1991.

LIAO, Y. **Power Flow Analysis on CUDA-based GPU**. 2011. Dissertação (Mestrado) — Faculty of the Worcester Polythechnic Institute, 2011.

MEYERS, S. Biblioteca boost. In: **C++ Eficaz: 55 maneiras de aprimorar seus programas e projetos**. 3. ed. Porto Alegre: Bookman, 2011. cap. 9, p. 289–292.

NUQUI, R. **State Estimation and voltage security monitoring using synchronized phasor measurements**. 7 2001. Tese (Doutorado) — Vignia Polythechnic Intitute and State University, Blacksburg, Virginia, 7 2001.

PHADKE, A. Synchronized phasor measurements in power systems. **IEEE Computer Applications in Power**, v. 6, n. 2, p. 10–15, 4 1993.

_____. Synchronized phasor measurements in power systems. **IEEE Computer Applications in Power**, v. 6, n. 2, p. 10–15, 1993.

PIERCE, H. E. J. et al. Common format for exchange of solved load flow data. **IEEE Transactions on Power Apparatus and Systems**, PAS-92, n. 6, p. 1916 – 1925, 1973.

SCHWEPPE, F. C.; WILDES, J. Power system static-state estimation, part i: Exact model. **TRANSACTIONS ON POWER APPARATUS AND SYSTEMS**, n. 1, p. 120–125, 1 1970.

_____. Power system static-state estimation, part ii: Approximate model. **TRANSACTIONS ON POWER APPARATUS AND SYSTEMS**, n. 1, p. 125–130, 1 1970.

SILVA, I. N.; SPATTI, D. H.; FLAUZINO, R. A. Aproximação por rede neural artificial. In: **Redes Neurais Artificiais para engenharia e ciências aplicadas**. 1. ed. São Paulo: Artliber, 2010. cap. 5, p. 132–137.

_____. Erro quadrático médio. In: **Redes Neurais Artificiais para engenharia e ciências aplicadas**. 1. ed. São Paulo: Artliber, 2010. cap. 5, p. 99–100.

_____. Minimos locais. In: **Redes Neurais Artificiais para engenharia e ciências aplicadas**. 1. ed. São Paulo: Artliber, 2010. cap. 5, p. 157–158.

_____. Neuronio perceptron. In: **Redes Neurais Artificiais para engenharia e ciências aplicadas**. 1. ed. São Paulo: Artliber, 2010. cap. 5, p. 59–63.

_____. overfitting. In: **Redes Neurais Artificiais para engenharia e ciências aplicadas**. 1. ed. São Paulo: Artliber, 2010. cap. 5, p. 153–155.

_____. Perceptron multicamadas. In: **Redes Neurais Artificiais para engenharia e ciências aplicadas**. 1. ed. São Paulo: Artliber, 2010. cap. 5, p. 90–95.

_____. Rede neural artificial. In: **Redes Neurais Artificiais para engenharia e ciências aplicadas**. 1. ed. São Paulo: Artliber, 2010. cap. 1, p. 21–24.

_____. Treino de pmc. In: **Redes Neurais Artificiais para engenharia e ciências aplicadas**. 1. ed. São Paulo: Artliber, 2010. cap. 5, p. 94–95.

_____. Treino supervisionado. In: **Redes Neurais Artificiais para engenharia e ciências aplicadas**. 1. ed. São Paulo: Artliber, 2010. cap. 5, p. 51–52.

_____. Validação de pmc. In: **Redes Neurais Artificiais para engenharia e ciências aplicadas**. 1. ed. São Paulo: Artliber, 2010. cap. 5, p. 147–151.

SISTEMA IEEE 14 barras. 1999. Disponível em: <http://labs.ece.uw.edu/pstca/pf14/14bus600.tif>. Acesso em: 2018.10.22.

SISTEMA IEEE 30 barras. 1999. Disponível em: <http://labs.ece.uw.edu/pstca/pf30/30bus600.tif>. Acesso em: 2018.10.22.

STEVENSON, W. D. Admitância. In: **Elementos de análise de sistemas de potência**. 2. ed. São Paulo: McGraw-Hill, 1986. cap. 7, p. 179–182.

_____. Admitância de transformadores. In: **Elementos de análise de sistemas de potência**. 2. ed. São Paulo: McGraw-Hill, 1986. cap. 8, p. 228–235.

_____. Conceito de barras. In: **Elementos de análise de sistemas de potência**. 2. ed. São Paulo: McGraw-Hill, 1986. cap. 8, p. 206–207.

_____. Diagrama unifilar. In: **Elementos de análise de sistemas de potência**. 2. ed. São Paulo: McGraw-Hill, 1986. cap. 7, p. 165–167.

_____. Gauss-siedel. In: **Elementos de análise de sistemas de potência**. 2. ed. São Paulo: McGraw-Hill, 1986. cap. 8, p. 207–209.

_____. Modelo de linha. In: **Elementos de análise de sistemas de potência**. 2. ed. São Paulo: McGraw-Hill, 1986. cap. 5, p. 95–112.

_____. Newton-raphson. In: **Elementos de análise de sistemas de potência**. 2. ed. São Paulo: McGraw-Hill, 1986. cap. 8, p. 209–216.

_____. Potencia complexa. In: **Elementos de análise de sistemas de potência**. 2. ed. São Paulo: McGraw-Hill, 1986. cap. 8, p. 19.

_____. Transformador ideal. In: **Elementos de análise de sistemas de potência**. 2. ed. São Paulo: McGraw-Hill, 1986. cap. 8, p. 147–151.

TENSORFLOW. 2017. Disponível em: <https://www.tensorflow.org/>. Acesso em: 2017.12.5.

T.L., B. et al. Power system observability with minimal phasor measumerement placement. **IEEE Transactions on Power Systems**, v. 8, n. 2, p. 707–715, 5 1993.

UW Power System Test Case Archive. 1999. Disponível em: <http://labs.ece.uw.edu/pstca/>. Acesso em: 2018.10.22.

WU, F. Power system state estimation: a survey. **International Journal of Electrical Power and Energy Systems**, v. 12, n. 2, p. 80–87, 1990.

Apêndices

APÊNDICE A – FUNÇÃO MAIN DE GERAÇÃO DE DADOS DA REDE

```
#include <ctime>
#include <string>
#include <iostream>
#include "rede/netDatasetGenerator.hpp"

using std::string;
using std::cout;
using std::endl;

int main(int argc, char const *argv[]) {
    if (argc < 4) {
        cout << "Program usage: bin [netFile] [inputDataFile] [solutionDa
        << endl;
        return 2;
    }
    int points = 10000;
    double dS   = 0.03;
    double dS_  = 0.30;
    double dP   = 0.01;
    double dP_  = 0.02;
    double dQ   = 0.005;
    double dQ_  = 0.01;
    double dPV  = 0.01;
    double dPV_ = 0.04;
    uint64_t seed = (uint64_t) std::time(NULL);
    string inputNetFile(argv[1]);
    string outInputFile(argv[2]);
    string outSolutionFile(argv[3]);
    for (int i = 4; i + 1 < argc ; i += 2) {
        string option(argv[i]);
        string value(argv[i + 1]);
        if (option == "--points") {
            points = std::stoi(value);
            continue;
        } else if (option == "--ds") {
            dS = std::stof(value);
```

```
            continue;
        } else if (option == "--ds-") {
            dS_ = std::stof(value);
            continue;
        } else if (option == "--dp") {
            dP = std::stof(value);
        } else if (option == "--dp-") {
            dP_ = std::stof(value);
        } else if (option == "--dq") {
            dQ = std::stof(value);
        } else if (option == "--dq-") {
            dQ_ = std::stof(value);
        } else if (option == "--dpv") {
            dPV = std::stof(value);
        } else if (option == "--dpv-") {
            dPV_ = std::stof(value);
        } else if (option == "--seed") {
            seed = std::stoi(value);
        }
    }
    ifstream net(inputNetFile.c_str());
    ofstream input(outInputFile.c_str());
    ofstream solution(outSolutionFile.c_str());
    std::default_random_engine eng(seed);
    neuralFlux::NetDatasetGenerator gen;
    std::vector<int> pmus = gen.generateFromFile(
        net,
        input,
        solution,
        &eng,
        points,
        dS,
        dS_,
        dP,
        dP_,
        dQ,
        dQ_,
        dPV,
        dPV_,
```

```
        std::cout);
    cout << points << " pontos gerados com pmus nas barras: "
    << endl;
    for (int i = 0 ; i < pmus.size() ; i++) {
        cout << pmus[i] << ",";
    }
    cout << endl;
    return 0;
}
```

APÊNDICE B – SCRIPT DE COMPARAÇÃO DE TEMPOS DE TREINO DE REDE NEURAL

```
import sys
import csv
import numpy as np
import tensorflow as tf
import math
from tensorflow import keras
import os
import time

trainDataPath = sys.argv[1]
labelDataPath = sys.argv[2]
layer1 = int(sys.argv[3])
layer2 = int(sys.argv[4])
withGpu = sys.argv[5] == '1'
epochs1 = 150
epochs2 = 10000
countEpochs1 = 0
countEpochs2 = 0
batch1 = 1
batch2 = 256
time1 = 0
time2 = 0
validationSplit = 0.2
minMae = 0.001
early_stop1 = keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience = 10
)
early_stop2 = keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience = 10
)
earlyStopError = 0.01
```

```
class EarlyStoppingByLossVal(keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        global earlyStopError
        current = logs.get('mean_absolute_error')
        if current < earlyStopError:
            self.model.stop_training = True

def loadDataFromFile(datasetPath):
    with open(datasetPath, 'r', newline='') as csvFile:
        num_lines = sum(1 for line in csvFile)

    with open(datasetPath, 'r', newline='') as csvFile:
        csvReader = csv.reader(csvFile, delimiter=",", quotechar='"')
        header = next(csvReader)
        columnCount = len(header) - 1

    data = np.zeros((num_lines - 1, columnCount))
    with open(datasetPath, 'r', newline='') as csvFile:
        csvReader = csv.reader(csvFile, delimiter=",", quotechar='"')
        header = next(csvReader)
        for i in range(num_lines - 1):
            row = next(csvReader)
            for j in range(columnCount):
                data[i][j] = float(row[j])
    mean = data.mean(axis=0)
    std = data.std(axis=0)
    data = data - mean
    data = np.divide(data, std, out=np.zeros_like(data), where=std!=0)
    return data

def build_model(layer1, layer2, inputData, labelData):
    model = keras.Sequential([
        keras.layers.Dense(layer1, activation=tf.nn.sigmoid,
                           input_shape=(inputData.shape[1],)),
        keras.layers.Dense(layer2, activation=tf.nn.sigmoid),
        keras.layers.Dense(labelData.shape[1])
    ])
    optimizer = tf.keras.optimizers.SGD(lr=0.1)
    model.compile(loss='mse',
```

```
                optimizer=optimizer,
                metrics=['mae'])
    return model

if (not withGpu):
    os.environ['CUDA_VISIBLE_DEVICES'] = '-1'

trainData = loadDataFromFile(trainDataPath)
labelData = loadDataFromFile(labelDataPath)

for i in range(3):
    model = build_model(layer1, layer2, trainData, labelData)
    initTime = time.time()
    history1 = model.fit(
        trainData,
        labelData,
        epochs=epochs1,
        validation_split=validationSplit,
        verbose=1,
        callbacks=[early_stop1, EarlyStoppingByLossVal()],
        batch_size=batch1
    )
    time1 += time.time() - initTime
    countEpochs1 += len(history1.history["val_mean_absolute_error"])

    model = build_model(layer1, layer2, trainData, labelData)
    initTime = time.time()
    history2 = model.fit(
        trainData,
        labelData,
        epochs=epochs2,
        validation_split=validationSplit,
        verbose=1,
        callbacks=[early_stop2, EarlyStoppingByLossVal()],
        batch_size=batch2
    )
    time2 += time.time() - initTime
    countEpochs2 += len(history2.history["val_mean_absolute_error"])
```

```
print("\nResultado final\n\n")
print("batch = " + str(batch1) + "\n")
print("epocas = " + str(countEpochs1/3) + "\n")
print("tempo =  " + str(time1/3) + "\n\n")
print("batch = " + str(batch2) + "\n")
print("epocas = " + str(countEpochs2/3) + "\n")
print("tempo =  " + str(time2/3) + "\n\n")
```

APÊNDICE C – SCRIPT DE TREINO DA REDE NEURAL

```
import sys
import csv
import numpy as np
import tensorflow as tf
import math
import pickle
from tensorflow import keras
import os
os.environ['CUDA_VISIBLE_DEVICES'] = '-1'

def loadDataFromFile(datasetPath):
    with open(datasetPath, 'r', newline='') as csvFile:
        num_lines = sum(1 for line in csvFile)

    with open(datasetPath, 'r', newline='') as csvFile:
        csvReader = csv.reader(csvFile, delimiter=",", quotechar='"')
        header = next(csvReader)
        columnCount = len(header) - 1

    data = np.zeros((num_lines - 1, columnCount))
    with open(datasetPath, 'r', newline='') as csvFile:
        csvReader = csv.reader(csvFile, delimiter=",", quotechar='"')
        header = next(csvReader)
        for i in range(num_lines - 1):
            row = next(csvReader)
            for j in range(columnCount):
                data[i][j] = float(row[j])
    mean = data.mean(axis=0)
    std = data.std(axis=0)
    data = data - mean
    data = np.divide(data, std, out=np.zeros_like(data), where=std!=0)
    return data

def build_model(layer1, layer2, inputData, labelData):
    model = keras.Sequential([
        keras.layers.Dense(layer1, activation=tf.nn.sigmoid,
```

```
                                input_shape=(inputData.shape[1],)),
        keras.layers.Dense(layer2, activation=tf.nn.sigmoid),
        keras.layers.Dense(labelData.shape[1])
    ])
    optimizer = tf.keras.optimizers.SGD(lr=0.1)
    model.compile(loss='mse',
                  optimizer=optimizer,
                  metrics=['mae'])
    return model

def trainModel(model, inputData, labelData, split, patience = 10):
    EPOCHS = 100
    # The patience parameter is the amount of epochs to check for improvement
    early_stop = keras.callbacks.EarlyStopping(
        monitor='val_loss',
        patience = patience
    )

    # Store training stats
    history = model.fit(
        train_data,
        label_data,
        epochs=EPOCHS,
        validation_split=split,
        verbose=1,
        callbacks=[early_stop],
        batch_size=1
    )
    return history


datasetPath = sys.argv[1]
labelPath = sys.argv[2]
modelPath = sys.argv[3]
historyPath = sys.argv[4]

validationSplit = 0.2
train_data = loadDataFromFile(datasetPath)
label_data = loadDataFromFile(labelPath)
```

```
layers = [10, 10]
errorBest = 0
errorNow = 0
patience = 5
netBestMeanError = 0
netBestValMeanError = 0
netBestErrorTotal = 0
netBestEpoch = 0
errors = []

while (1 == 1):
    netBestMeanError = 0
    netBestValMeanError = 0
    netBestErrorTotal = 0

    print('layers: ')
    print(layers)

    for i in range(patience):
        model = build_model(layers[0], layers[1], train_data, label_data)
        history = trainModel(model, train_data, label_data, 0.2)
        meanError = history.history['mean_absolute_error'][-1]
        valMeanError = history.history['val_mean_absolute_error'][-1]
        errorTotal = meanError + valMeanError

        print('\ni: ' + str(i) + '\n\n')
        print('mean_absolute_error: ' + str(meanError) + '\n')
        print('val_mean_absolute_error: ' + str(valMeanError) + '\n')
        print('errorTotal: ' + str(errorTotal) + '\n')

        if (errorTotal < netBestErrorTotal or netBestErrorTotal == 0):
            netBestMeanError = meanError
            netBestValMeanError = valMeanError
            netBestErrorTotal = errorTotal
            net = model
            netHistory = history
            netBestEpoch = len(history.epoch)
```

```
    errors.append([
        netBestMeanError,
        netBestValMeanError,
        netBestErrorTotal,
        netBestEpoch
    ])

    if netBestErrorTotal < errorBest or errorBest == 0:
        errorBest = netBestErrorTotal
        bestModel = net
        bestHistory = netHistory
        histPath = historyPath + '_' +  str(layers[0]) + '_' + str(layers[1])
        modPath = modelPath + '_' + str(layers[0]) + '_' + str(layers[1])
        with open(histPath, 'wb') as historyFile:
            pickle.dump(bestHistory.history, historyFile)
        bestModel.save(modPath)
        layers[0] += 5
        layers[1] += 5
    else:
        break


print(errors)
```

APÊNDICE D – LEIA-ME DO REPOSITÓRIO

Pasta com arquivos utilizados no trabalho de conclusão de curso de Luiz Victor Linhares Rocha para a obtenção do titulo de Engenheiro eletricista pela Escola de Engenharia de São Carlos na Universidade de São Paulo (EESC-USP)

Resumo das pastas:

- dataset/ arquivos CSV com os dados de treino e de validação, cada arquivo é um csv conta com 100.001 linhas, sendo o primeiro o cabeçãlho indicando qual o parâmetro da coluna

- modelOuput/ arquivos de saída dos scripts de treino de rede

  Possuem o formato:

  - errors.ods - Planilha de análise de erro para o treino de diferentes topologias.
  - history[N]_[a1]_[a2] - arquivo binário com dados do histórico de treino da rede neural da resolução do problema com N barras possuindo a1 neurônios na primeira camada e a2 neurônios na segunda camada.
  - model[N]_[a1]_[a2] - arquivo binário com dados do modelo da rede neural da resolução do problema com N barras possuindo a1 neurônios na primeira camada e a2 neurônios na segunda camada.
  - outputSummary[N]_[a1]_[a2].csv - Planilha, no formato csv, de resultado do erro quadrático médio para a rede neural da resolução do problema com N barras possuindo a1 neurônios na primeira camada e a2 neurônios na segunda camada.
  - summary[N] - Resumo sobre os dados de entrada e saída para o treino da rede neural da resolução do problema com N barras possuindo a1 neuronios na primeira camada e a2 neurônios na segunda camada.
  - weights[N]_[a1]_[a2].csv - Pesos resultantes que definem a rede neural da resolução do problema com N barras possuindo a1 neuronios na primeira camada e a2 neurônios na segunda camada.

- netData/ arquivos cdf descrevendo as redes de potências utilizada como base de estudo.

- python/ scripts python para treino de rede neural.

  - inputDataInfoPrinter.py: Script de geração de arquivos .csv com informação generalizada de média e desvio padrão de arquivos de treino geradas pelo programa em c++.
  - netToCsv.py: Script de conversão de arquivo binário .h5, salvo pelo tensorflow, para .csv, descrevendo os pesos resultantes.
  - netTrainer.py: Script principal de treino de rede neural, comparando diferentes topologias e seus erros.
  - outputDataInfoPrinter.py: Script de geração de arquivos .csv com informação de erro, normalizado e não-normalizado de cada saída da rede neural.
  - singleNetTrainer.py: Script de treino de apenas 1 rede, utilizado para estudos.
  - speedTestGPU.py: Script de comparação de performance entre utilização de CPU e GPU.

- rede/ código fonte c++ criado para geração de dados e resolução de fluxo de potência.
- test/ código de teste c++ para teste das classes c++.
- util/ código de utilidades para o desenvolvimento do programa c++.

resumo dos arquivos:

- generatePoints.cpp: arquivo com função main para ser compilado em um programa que recebe um arquivo CDF, aloca PMUs e salva arquivos de entrada e saída para treino de uma rede neural
- CMakeLists.txt: Arquivo de instruções de compilação pelo CMake

Observações: Para abrir arquivos .CSV no excel ou openoffice é importante atribuir o tipo de número de ponto flutuante para Ingles americano.