Universidade Federal do Piauí
Centro de Ciências da Natureza
Programa de Pós-Graduação em Ciência da Computação

Uma Otimização do Perceptron Multicamadas Utilizando Aproximação Quadrática e Método de Ponto Proximal

Felipe Monte Machado Magalhães de Sousa

Número de Ordem PPGCC: M001

Teresina-PI, Fevereiro de 2016

Felipe Monte Machado Magalhães de Sousa

Uma Otimização do Perceptron Multicamadas Utilizando Aproximação Quadrática e Método de Ponto Proximal

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da UFPI (Área de concentração: Sistemas de Computação), como parte dos requisitos necessários para a obtenção do Título de Mestre em Ciência da Computação.

Universidade Federal do Piauí – UFPI

Centro de Ciências da Natureza

Programa de Pós-Graduação em Ciência da Computação

Orientador: Paulo Sérgio Marques dos Santos

Coorientador: João Xavier da Cruz Neto

Teresina-PI

Fevereiro de 2016

Felipe Monte Machado Magalhães de Sousa

Uma Otimização do Perceptron Multicamadas Utilizando Aproximação Quadrática e Método de Ponto Proximal/ Felipe Monte Machado Magalhães de Sousa. – Teresina-PI, Fevereiro de 2016-

51 p. : il. (algumas color.) ; 30 cm.

Orientador: Paulo Sérgio Marques dos Santos

Dissertação (Mestrado) – Universidade Federal do Piauí – UFPI
Centro de Ciências da Natureza
Programa de Pós-Graduação em Ciência da Computação, Fevereiro de 2016.

1. Redes neurais artificiais. 2. Perceptron multicamadas. 3. Otimização. 4. Programação não-linear. I. Paulo Sérgio Marques dos Santos. II. Universidade Federal do Piauí. III. Centro de Ciências da Natureza. IV. Doutor

CDU 02:141:005.7

Felipe Monte Machado Magalhães de Sousa

Uma Otimização do Perceptron Multicamadas Utilizando Aproximação Quadrática e Método de Ponto Proximal

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da UFPI (Área de concentração: Sistemas de Computação), como parte dos requisitos necessários para a obtenção do Título de Mestre em Ciência da Computação.

Trabalho aprovado. Teresina-PI, 29 de Fevereiro de 2016:

Prof. Dr. Paulo Sérgio Marques dos Santos
Orientador

Prof. Dr. João Xavier da Cruz Neto
Co-Orientador

Prof. Dr. André Macedo Santana
Convidado 1

Prof. Dr. Roberto Cristovão Mesquita Silva
Convidado 2

Teresina-PI
Fevereiro de 2016

À Leilane, Biju, Yumi, Nikita e Loki,
meus amigos, amores e companheiros de todas as horas.

Agradecimentos

Agradeço à minha esposa, Leilane, por toda a compreensão e apoio, principalmente, durante o desenvolvimento deste trabalho.

Agradeço à minha família o companheirismo e carinho em cada fase da minha vida.

Agradeço aos professores da UFPI pela força e pelo bom ambiente de estudo.

Agradeço aos professores Harilton Araújo, Ricardo Lira e André Macedo pelo apoio, desde a graduação, fundamental para o meu preparo para o mestrado.

Agradeço ao professor Arnaldo Brito pela dedicação e compartilhamento de conhecimento.

Finalmente, agradeço, especialmente, ao professor Paulo Santos pelo apoio, companheirismo, dedicação, esforço e, principalmente, paciência.

"Somos todos sonhadores."
(Gackt)

Resumo

Por conta de sua alta capacidade de generalização, predição e auto-ajuste, as redes neurais artificiais têm sido amplamente usadas em muitas áreas de pesquisa e mercado. Dentre essas redes, o perceptron multicamadas é uma das mais usadas. Baseado em modelos de otimização e no algoritmo backpropagation, o Perceptron Multicamadas pode solucionar qualquer problema que possa ser mapeado em seus sinais de entrada. Entretanto, o processo de treinamento do Perceptron Multicamadas pode convergir antes de alcançar os valores ótimos, o que resultaria em um pequeno erro em seu aprendizado. Este trabalho propõe o uso de modelos de otimização para cálculo do tamanho da taxa de aprendizagem e determinação de uma direção de descida para tornar o Perceptron Multicamadas mais preciso, fazendo com que o aprendizado do mesmo se aproxime mais da solução ótima, diminuindo assim o erro.

Palavras-chaves: redes neurais artificiais. perceptron multicamadas. algoritmo do ponto proximal. programação não-linear.

Abstract

Due to its high ability to generalize, predicting and self-tuning, the artificial neural networks have been widely used in many research and market areas . Among these networks, the Multilayer Perceptron is one of the most used. Based on optimization models and backpropagation algorithm, the Multilayer Perceptron can solve any problem that can be mapped to it's input signals. However, the Multilayer Perceptron's training process may converge before it reachs the optimums values, that result in a small error in his learning. This paper proposes the use of optimization models to calculate the size of the rate of learning and determination of a descent direction to make the Multilayer Perceptron more accurate, making the learning of it get closer to the optimum solution, reducing the error.

Keywords: artificial neural networks. multilayer perceptron. proximal point algorithm. nonlinear programming.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

IA	Inteligência Artificial
RNAs	Redes Neurais Artificiais
PMC	Perceptron Multicamadas
MLP	Multilayer Perceptron (sigla em inglês para PMC)
INPPA	Inexact Nonmonotone Proximal Point Algorithm

Sumário

Introdução

Cada vez mais o tempo para a realização de tarefas tem sido otimizado. Seja para tarefas simples e sem necessidade de muita perícia, como lavar um carro, até tarefas de alta complexidade, como realizar um mapeamento genético, o uso de sistemas automatizados é mais comum.

Para que um sistema automatizado opere com bom funcionamento, este precisa contar com uma capacidade de adaptação ao ambiente em que está sendo utilizado. A capacidade de se adaptar e raciocinar sobre um determinado problema, em sistemas computacionais, é chamada de inteligência artificial.

Existem diversas formas de tornar um sistema computacional inteligente, tais como redes neurais artificiais, lógica fuzzy, computação evolutiva, entre outras.

Uma das formas de IA mais comuns em vários campos de pesquisa são as redes neurais artificiais (RNAs). As RNAs são modelos matemáticos inspirados no cérebro humano e contam com conceitos de otimização.

Mas não basta um sistema ser inteligente. Ele precisa realizar a tarefa ao qual foi encarregado com o melhor resultado e no menor tempo possíveis. Diversos estudos têm sido realizados nessa área com objetivo de tornar os sistemas inteligentes mais precisos e diminuir seu tempo de convergência.

Um modelo específico de RNA é o Perceptron Multicamadas (PMC). Este modelo possui uma capacidade de generalização e classificação muito alta, e por isso foi escolhido como caso de estudo e pesquisa deste trabalho.

Com o objetivo de melhorar o desempenho do PMC, alguns métodos de otimização serão apresentados e aplicados no mesmo.

Motivação

Tendo em vista que a cada dia mais tarefas são automatizadas contando com o apoio de decisões de sistemas artificialmente inteligentes, faz-se necessário que o algoritmo responsável pela tomada de decisão seja eficiente. Para que um algoritmo seja considerado eficiente, não basta que esse convirja, isso deve ocorrer em um tempo aceitável. Além disso, sua resposta deve ser precisa e específica para a aplicação a qual o mesmo está sendo submetido.

Uma vez que o Perceptron Multicamadas é uma das redes neurais mais utilizadas, decidiu-se focar este trabalho no mesmo a fim de torná-lo ainda mais preciso melhorando

sua capacidade de generalização através da combinação de métodos de otimização à seu algoritmo clássico.

Objetivos

Este trabalho tem como principal objetivo otimizar o Perceptron Multicamadas a fim de tornar sua capacidade de generalização mais precisa. Além disso, pretende-se realizar testes comparativos com o modelo clássico do PMC, focando na precisão dos resultados fornecidos por ambos, variação dos resultados e tempo de convergência na fase de treinamento.

Parte I

Preparação da pesquisa

1 Preliminares

1.1 Redes Neurais Artificiais

As redes neurais artificiais (RNA) são modelos computacionais baseados nas redes neurais biológicas. As RNAs, também chamadas de neurocomputação, podem ser definidas como um conjunto de unidades de processamento interligado por um grande número de interconexões. As unidades de processamento, assim como as redes biológicas, são os neurônios, os quais são os elementos básicos de sua constituição e as interconexões são sinapses artificiais.

Como abordado por Silva (SILVA; SPATTI; FLAUZINO, 2013), as RNAs possuem capacidade de aquisição e manutenção do conhecimento e sua função é processar estímulos advindos do meio e emitir ou não uma resposta com base em seu processamento. Possuem alta capacidade de generalização, bom desempenho em problemas mal definidos e alta resistência a ruídos inseridos em sua entrada. O modelo matemático mais simples das redes neurais foi proposto por McCulloch e Pitts em 1943 (McCulloch; Pitts, 1943), sendo ainda o modelo mais usado, chamado de combinador linear, conforme (1.1).

$$u = \sum_{i=0}^{n} w_i * x_i \tag{1.1}$$

Nesta equação, x representa os sinais de entrada da rede, sendo estes advindos do meio externo; w representa os pesos sinápticos, também chamados de termos de ponderação, os quais são valores para ponderar cada uma das variáveis de entrada do neurônio e u representa o potencial de ativação, como apresentado em (HAYKIN et al., 2008).

Para que a rede neural artificial produza uma saída, esta precisa operar sobre o resultado calculado pela mesma. O modelo matemático que representa esta operação é representado logo a seguir.

$$y = g(u) \tag{1.2}$$

Nesta equação, y é o sinal de saída produzido pela rede e é calculado através de uma função de ativação (como as funções degrau, linear, tangente hiperbólica, logística, entre outras) $g(u)$. Dependendo da arquitetura da rede, o sinal produzido pela mesma (y) pode ser utilizado como sinal de entrada para outros neurônios. Um modelo genérico de redes neurais, como exibido em (HAYKIN et al., 2008), (SILVA; SPATTI; FLAUZINO, 2013) e outros, é apresentado a seguir.

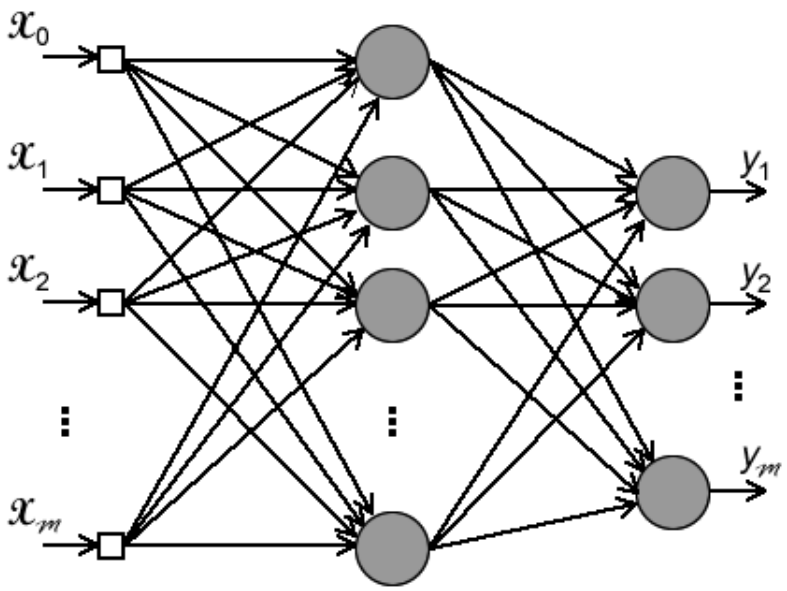

Figura 1 – Modelo genérico de uma rede neural artificial
Fonte: próprio autor

Na equação apresentada, os x representam as entradas advindas do meio e esta camada é chamada camada de entrada; nela não é realizada qualquer computação, os n representam os neurônios, sendo que os neurônios são distribuídos em camadas chamadas camadas neurais escondidas e a última é chamada de camada neural de saída. Os segmentos de retas representam as interconexões neurais e os y representam as saídas da rede.

A arquitetura de uma rede é a forma com que os neurônios são dispostos nas mesmas. Simon (HAYKIN et al., 2008) explica que de acordo com a arquitetura da rede, ela pode ser classificada como feedforward ou recorrentes e redes de camada simples ou múltiplas camadas.

Nas redes feedforward, o fluxo de informação segue uma única direção, sendo esta da camada de entrada para a camada de saída, ou seja, não existe realimentação (ciclo) de informação na rede. Nas redes recorrentes, as saídas computadas pelos neurônios são reutilizadas na rede como sinais de entrada para outros neurônios, ou seja, existe uma realimentação de informação.

Nas redes de camada simples existe a presença de apenas uma camada neural, sendo esta a camada de saída. Nas redes de múltiplas camadas existem duas ou mais camadas neurais, sendo a camada de saída e pelo menos uma camada neural escondida.

Além da arquitetura, uma rede neural artificial difere das outras pela sua topologia, a qual é uma composição estrutural que uma determinada arquitetura pode assumir. O número de neurônios de cada camada e a função de ativação utilizada são exemplos de topologias que podem ser especificadas em cada rede.

Um dos principais problemas referente à topologia de uma rede é a expecificação do número de neurônios utilizados para compor a mesma. Neste cenário existem três possibilidades: o primeiro é escolher um número adequado de neurônios e a rede se comportar de meneira adequada, ou seja, ter sucesso em seu aprendizado; o segundo é escolher um número inferior ao mínimo necessário para o aprendizado da rede e a rede não conseguir abstrair o padrão de aprendizagem. Este problema é chamado de "underfitting"; o terceiro é escolher um número de neurônios muito superior ao necessário para o aprendizado e a rede conseguir abstrair o padrão de treinamento, mas associá-lo a um padrão bem mais complexo que o real, ou seja, o padrão aprendido se enquadra apenas a uma parte do padrão real. Este problema é chamado "overfitting". Um exemplo de cada um dos três casos é apresentado na Figura 2.

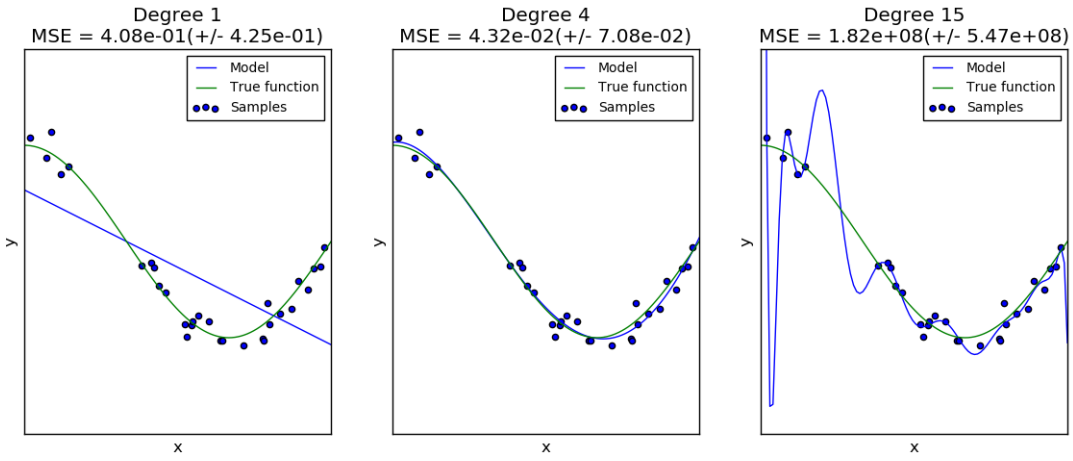

Figura 2 – Exemplo de "underfitting"e "overfitting"
Fonte: http://scikit-learn.org

Na Figura 2, no centro, tem-se uma abstração quase perfeita do padrão em questão. Esta seria uma situação ideal em um processo de aprendizagem de uma rede. À esquerda, o padrão aprendido pela rede foi o de uma função de grau 1, não conseguindo assim abstrair nem mesmo o padrão de aprendizagem. Este é um comportamento comum em uma situação de "underfitting". À direita, o padrão aprendido pela rede foi o de uma função de grau 15 e, ainda que a abstração do padrão de teste tenha se dado de maneira satisfatória, o padrão aprendido não se enquadra ao padrão real. Este é um comportamento comum em uma situação de "overfitting".

As redes neurais diferem na maneira em que as mesmas são treinadas. As formas de treinamento de uma RNA são chamadas paradigmas de aprendizagem e alguns dos principais paradigmas de aprendizagem são aprendizado supervisionado, aprendizado não-supervisionado ou aprendizado por reforço.

No aprendizado supervisionado existe a presença de um professor que fornece as saídas esperadas para as amostras de treinamento. No aprendizado não-supervisionado,

as saídas desejadas não são informadas à rede. A informação fornecida para uma dada entrada é se a saída calculada é satisfatória ou não, podendo ser vista como um caso particular de aprendizado supervisionado.

Não existe uma organização ideal de uma rede neural. Cabe ao pesquisador encontrar qual a arquitetura, topologia e paradigma de aprendizagem mais indicado para problema que será submetido à rede neural artificial.

1.1.1 Redes perceptron

Como mostrado em (HAYKIN et al., 2008), as redes Perceptron são as redes neurais artificiais mais simples. São constituídas de um único neurônio, por consequência, possuem uma única camada neural e não possuem qualquer tipo de realimentação, ou seja, a arquitetura da rede é feedforward de camada simples.

Por possuírem apenas um neurônio, a topologia da rede não difere com relação ao número de neurônios. Quanto à função de ativação, são normalmente usadas a função degrau (1.3) ou a função degrau bipolar (1.4).

$$f(u) = \begin{cases} 1, & se \quad u \geq 0 \\ 0, & se \quad u < 0 \end{cases} \tag{1.3}$$

$$f(u) = \begin{cases} 1, & se \quad u > 0 \\ 0, & se \quad u = 0 \\ -1, & se \quad u < 0 \end{cases} \tag{1.4}$$

O ajuste dos pesos sinápticos é feito proporcionalmente às entradas e o erro, como mostrado a seguir:

$$w = w + \eta(d - y)x \tag{1.5}$$

sendo w os pesos sinápticos, η a taxa de aprendizagem, d a saída desejada, y a saída produzida e x as entradas da rede.

Como paradigma de treinamento, as redes Perceptron utilizam o aprendizado supervisionado. Suas entradas devem ser valores reais, seus pesos sinápticos valores reais e as saídas produzidas pela rede são binárias.

O Perceptron é considerado um discriminador linear, isso por sua limitação ao responder apenas problemas linearmente separáveis. Como a maioria dos problemas do mundo real são problemas não linearmente separáveis, o perceptron só resolveria problemas teóricos e de simples resolução. Devido à sua limitação, Minsky e Pappert publicam duras

críticas às redes perceptron, em 1969. Após esta publicação, o interesse em estudar as redes perceptron caiu drasticamente.

1.1.2 Redes Perceptron Multicamadas (PMC)

As redes neurais ficaram praticamente esquecidas durante mais de uma década, até que em 1982 Hopfield publica seus resultados com um novo conceito, as redes recorrentes (HOPFIELD, 1982). A partir das redes recorrentes, em 1986 Rumelhart, Hinton e Williams propõem um algoritmo de retropropagação do erro (RUMELHART; HINTON; WILLIAMS, 1986), que faria com que as camadas intermediárias do Perceptron pudessem ser corrigidas. Esse algoritmo é conhecido como "error-backpropagation".

Com o advento das redes recorrentes, a principal limitação do perceptron poderia ser contornada e este estaria apto a resolver problemas mais robustos e do mundo real.

Ao se acrescentar retroalimentação às redes perceptron, e aumentar o número de neurônios e camadas utilizado nas mesmas, o perceptron tornou-se capaz de resolver qualquer problema do mundo real que possa ser mapeado em entradas da rede. A este novo perceptron foi dado o nome de Perceptron Multicamadas.

Assim como no Perceptron, no PMC as saídas da rede são resultado da combinação linear das entradas com os pesos. Entretanto, as entradas das camadas escondidas são as saídas das camadas anteriores e assim por diante até a camada de saída.

Uma vez que se tenha encontrado a saída da rede, o erro é calculado e com base neste, os pesos da ultima camada são corrigidos. O erro é encontrado usando a função do erro quadrático, exibida logo a seguir.

$$E(w) = \frac{1}{2} * \sum_{k=0}^{p} (d_k - y_k^l)^2 \tag{1.6}$$

sendo $E(w)$ o erro quadrático, p o número de neurônios na última camada, d a saída desejada do k-ésimo neurônio, l a última camada neural e y a saída real do k-ésimo neurônio da l-ésima camada.

As redes Perceptron Multicamadas são caracterizadas por possuírem, no mínimo, uma camada neural escondida, ou seja, possuem pelo menos duas camadas neurais. As saídas computadas nos neurônios de uma camada servem de entradas para os neurônios da camada seguinte, até que cheguem a ultima camada. Utilizam o algoritmo backpropagation, ou seja, uma vez que a saída produzida pela rede não é satisfatória ao problema, o sinal é propagado da saída para o começo da rede, ajustando os valores dos pesos sinápticos.

Como as saídas produzidas pelas camadas escondidas são desconhecidas, o ajuste do erro das mesmas deve ser feito por meio da retro-propagação do erro, (HAYKIN et al., 2008), ou seja, o erro da camada de saída é passado para as camadas intermediárias,

uma por vez, da mais próxima da camada de saída para a mais próxima da camada de entrada. Para isso, calcula-se o gradiente local da camada de saída, que nada mais é que a combinação do erro calculado com a direção que minimiza a função do erro a partir do corrente ponto, ou seja:

$$\delta = (d - y) * g'(I) \tag{1.7}$$

sendo δ o gradiente local, d a saída desejada, y a saída real, I o combinador linear e $g^{'}(I)$ o gradiente da função de ativação usada na rede.

Uma vez conhecido o gradiente local, os pesos da última camada devem então ser atualizados:

$$w = w + \eta\delta y \tag{1.8}$$

sendo w o vetor de pesos sinápticos e η a taxa de aprendizagem.

Depois de calculados os pesos da última camada, deve-se calcular o gradiente local de cada neurônio da camada anterior, para isso, faz-se:

$$\delta^{l-1} = \sum_{i=1}^{n} (\delta_n^l * w_n^l) * g'(I) \tag{1.9}$$

sendo δ o gradiente local da l-ésima camada e n o número de neurônios na l-ésima camada.

O processo do ajuste dos pesos da camada escondida repete iterativamente até a camada inicial.

As funções de ativação mais comumente usadas no PMC são a função linear, função logística (1.10) e a função tangente hiperbólica (1.11).

$$f(I) = \frac{1}{1 + e^{-\beta I}} \tag{1.10}$$

$$f(I) = \frac{1 - e^{-\beta I}}{1 + e^{-\beta I}} \tag{1.11}$$

nas quais $\beta \in (0, 1)$ e I é o combinador linear.

Assim como o Perceptron, o Perceptron Multicamadas possui como paradigma de treinamento o aprendizado supervisionado. Suas entradas, pesos sinápticos e saídas produzidas pela rede são valores reais.

1.2 Noções de Otimização

Problemas de otimização são encontrados nas mais diversas áreas de pesquisa. O problema da otimização, geralmente, tem como objetivo maximizar ou minimizar uma função através da substituição de variáveis por valores dentro de um determinado conjunto viável. Em uma definição mais formal, Ribeiro e Karas (RIBEIRO; KARAS, 2011) "otimização consiste em encontrar pontos de mínimo ou de máximo de uma função real sobre um conjunto $\Omega \subset R^n$".

Minimizar um problema é equivalente a maximizar o seu oposto, ou seja, minimizar $f(x) =$ maximizar $-f(x)$, portanto, neste trabalho, será feita referência apenas ao termo minimizar. Considerando-se isso, muitos autores, dentre os quais Friedlander (FRIEDLANDER, 1994), Apolinário (APOLINARIO, 2014), Silva (SILVA, 2013) e outros, apresentam a formulação matemática genérica dos problemas de otimização da seguinte forma:

$$\begin{array}{lll} \text{Minimizar} & & f(x) \\ \text{sujeito} & \text{a} & x \in \Omega. \end{array} \tag{1.12}$$

Na qual $f : \mathbb{R}^n \to \mathbb{R}$ e $\Omega \subset \mathbb{R}^n$

Dados o conjunto Ω, chamado conjunto factível, e as propriedades das funções objetivo, os problemas de otimização podem ser diferentes, por exemplo, as funções envolvidas no problema podem ser contínuas ou não, diferenciáveis ou não, lineares ou não.

Geralmente, modelos lineares se limitam a aproximações de modelos reais enquanto fenômenos físicos ou econômicos, devido ao mapeamento de seus fenômenos serem melhores representados em funções não lineares, são modelados em problemas não-lineares.

1.2.1 O problema de otimização

Considerando-se o modelo (1.12), e tomado um $x^* \in \Omega \subset R^n$, diz-se que x^* é um minimizador local de f em Ω se existe $\delta > 0$, tal que $f(x^*) < f(x)$, para todo $x \in B(x^*, \delta) \cap \Omega$. Se $f(x^*) < f(x)$, para todo $x \in \Omega$, x^* é chamado de minimizador global de f em Ω.

Como citado por Silva (SILVA, 2013), em problemas de programação linear, o minimizador local é sempre o minimizador global. Em problemas de programação não-linear, o minimizador local está em um ponto de mínimo local da função, enquanto o minimizador global está em uma área de mínimo global.

1.2.1.1 Condições de otimalidade

Considerando-se o seguinte problema restrito:

$$\begin{array}{lll} \text{Minimizar} & & f(x) : \mathbb{R}^n \to \mathbb{R} \\ \text{sujeito a:} & & \\ g_i(x) \leq 0 & \text{para todo} & i = 1, 2, ..., m \\ h_j(x) = 0 & \text{para todo} & j = 1, 2, ..., n. \end{array} \tag{1.13}$$

A condição de otimalidade de Karush-Kuhn-Tucker garante que para x^* ser um minimizador local do problema deve existir um vetor $\lambda^* \in R^m$ e um $\eta^* \in R^n$ com $\lambda^* \geq 0$, tal que

$$\nabla f(x^*) + \sum_{i=1}^{m} \lambda_i \nabla g_i(x^*) + \sum_{j=1}^{n} \eta_j \nabla h_j(x^*) = 0 \tag{1.14}$$

$\lambda_i g_i(x^*) = 0$ para todo i = 1, 2,..., m

As componentes do vetor λ^* são chamadas de Multiplicadores de Lagrange e as condições acima são suficientes em problemas convexos.

Um resultado importante nesta direção é o teorema de Bolzano-Weierstrass (FRIEDLANDER, 1994) que diz que uma função real contínua f, definida em um conjunto fechado e limitado $S \subset R^n$, admite um minimizador global em S.

Além dos teoremas apresentados, existem outros teoremas e condições que garantem a presença de um minimizador, local ou global, em f, como os apresentados em Ribeiro e Karas (RIBEIRO; KARAS, 2011) e Friedlander (FRIEDLANDER, 1994).

1.2.2 Algoritmos

Em problemas de otimização, principalmente problemas de programação não-linear, dificilmente a solução do mesmo pode ser encontrada diretamente. Nestes casos, o método de resolução utilizado é um método iterativo que tem como objetivo encontrar uma sequência de valores para x, que converjam a um número, ou seja, o minimizador de $f(x)$.

Esta sequência pode ser descrita como:

$$f(x^{k+1}) < f(x^k), \quad \text{para todo} \quad k = 1, 2, \ldots, m. \tag{1.15}$$

Ribeiro e Karas (RIBEIRO; KARAS, 2011) ressaltam que esses algoritmos possuem três aspectos característicos. Primeiramente, a construção do algoritmo propriamente dita, que deve considerar todas as características envolvidas no problema, como sua estruturação. Segundo, a sequência construída pelo algoritmo, que deve convergir para uma solução do

problema. Terceiro, a velocidade de convergência da solução, que não basta convergir, deve fazê-lo em um tempo razoável.

1.2.2.1 Algoritmos de descida

De modo geral, para que o valor de $x^{(k+1)}$ seja menor que o valor de x^k, deve-se dar um passo, a partir de x^k, seguindo em uma direção que faça com que $f(x)$ decresça e $t_k > 0$ (ZHANG; HAGER, 2011). Qualquer direção que $f(x)$ decresça é chamada direção de descida.

Deste modo, pode-se observar:

$$x^{k+1} = x^k + td^k \tag{1.16}$$

sendo $t \neq 0$, x^k o ponto atual, $x^{(k+1)}$ o próximo ponto da sequência, t o tamanho do passso tomado na direção de descida (por se tratar de um problema de minimização) d^k. Se $\bigtriangledown f(x)^T d < 0$, então d é uma direção de descida para f, a partir de x.

O modelo mais simples de algoritmo de descida é um algoritmo que realiza uma busca linear para determinar o tamanho do passo dado em cada iteração. O mesmo é descrito logo a seguir.

Data: $x^0 \in \mathbb{R}^n$
$k = 0$
while $\bigtriangledown f(x^k) \neq 0$ **do**
  Calcule d^k tal que $\bigtriangledown f(x^k)^T d^k < 0$
  Escolha $t_k > 0$ tal que $f(x^k + t_k d^k) < f(x^k)$
  Faça $x^{k+1} = x^k + t_k d^k$
  $k = k + 1$
end

Algorithm 1: Algoritmo elementar de minimização

O algoritmo apresentado permanece em loop até encontrar um x^k cujo $\bigtriangledown f(x^k) = 0$, ou seja, um ponto estacionário. Enquanto não encontra, o mesmo gera uma sequência a qual f decresce. Entretanto, não é possível afirmar se esta sequência possui algum ponto de acumulação ou não.

Ainda sobre o algoritmo apresentado, nota-se que o tamanho do passo t_k é obtido sem atender qualquer método de busca, deste modo, é possível que o algoritmo não venha a convergir em um tempo razoável, ou se quer convergir, o que pela terceira característica apresentada por Ribeiro e Karas (RIBEIRO; KARAS, 2011), o torna um algoritmo irrelevante para a maioria das aplicações.

1.2.2.2 Método da seção áurea

De modo geral, o método da seção áurea consiste em dividir o domínio de uma função, iterativamente, até o espaço particionado se resumir ao mínimo da função. Entretanto, para que este método possa ser aplicado, a função a ser minimizada deve ser contínua e unimodal.

Para facilitar o entendimento, a Figura 3 mostra o gráfico de uma função unimodal e os quatro pontos que a particionam.

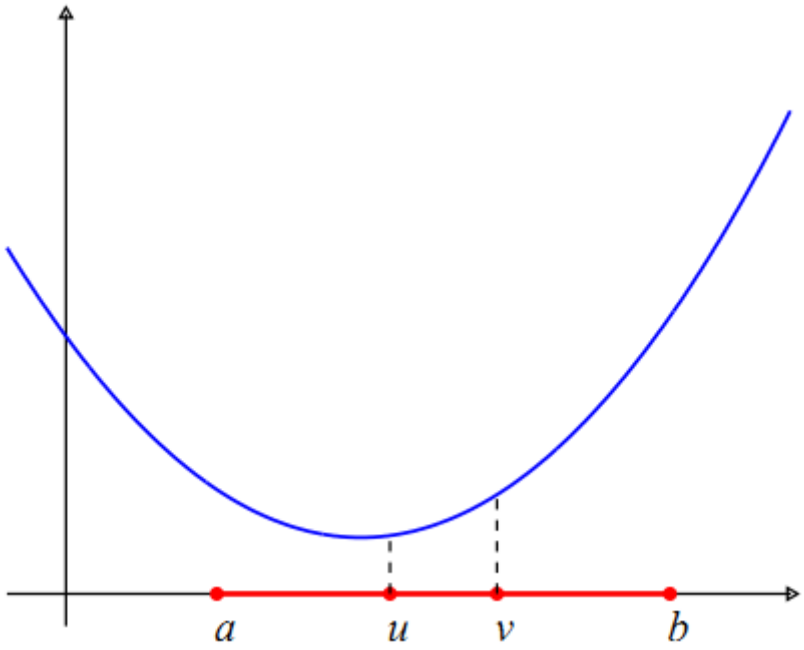

Figura 3 – Particionamento de uma função contínua unimodal
Fonte: (RIBEIRO; KARAS, 2011)

Supondo que um minimizador de f pertence ao intervalo $[a, b]$, atribui-se f e v, tal que $a < u < v < b$. Então, compara-se $f(u)$ e $f(v)$, caso $f(u) < f(v)$, o intervalo $[v, b]$ não pode conter um minimizador, caso contrário o intervalo $[a, u]$ não pode conter um minimizador. Deste modo, o intervalo que não contém o minimizador deve ser descartado e o intervalo restante deve ser particionado novamente até encontrar o minimizador de f.

A fim de otimizar o processo de calcular $f(x)$ e de particionar o intervalo, divide-se o intervalo $[a, b]$ de forma que a razão entre o maior segmento e o segmento todo é igual à razão entre o menor e o maior dos segmentos. Esta razão é conhecida como o número de ouro e é igual a $\frac{(\sqrt{5}-1)}{2}$.

Sendo assim, tem-se:

$$\frac{b-u}{b-a} = \frac{u-a}{b-u} \quad e \quad \frac{v-a}{b-a} = \frac{b-v}{v-a} \tag{1.17}$$

1.2.2.3 Método de Armijo

Como a busca exata no processo de minimização de uma função pode ser extremamente lento e complexo, existem métodos de busca inexata, como o método de Armijo, que em vez de realizarem uma busca exata, eles apenas tentam encontrar uma aproximação das minimizações da função. Além disso, o método de Armijo não possui a limitação de usabilidade apenas em funções unimodais.

O modelo proposto por armijo objetiva encontrar um tamanho de passo $t > 0$, tal que

$$f(x + td) \leq f(x) + \eta t \bigtriangledown f(x)^T d \tag{1.18}$$

no qual $\eta \in (0, 1)$.

Desta forma, o algoritmo de Armijo pode ser escrito como:

Data: $x \in \mathbb{R}^n$, $d \in \mathbb{R}^n$, γ, $\eta \in (0, 1)$
$t = 1$
while $f(x + td) > f(x) + \eta t \bigtriangledown f(x)^T d$ **do**
| $t = \gamma * t$
end

Algorithm 2: Algoritmo elementar de minimização

Apesar de simples, o algoritmo de Armijo é bastante eficiente e pode encontrar uma aproximação do mínimo de f em poucas iterações.

2 Treinamento de Redes Neurais Usando Algoritmos de Otimização

Como foi abordado anteriormente, as redes neurais artificiais possuem a capacidade de aprendizagem, através da auto-organização de seus pesos sinápticos. Esta é, certamente, a característica mais importante de uma rede neural artificial. Para que a rede seja apta a realizar tal procedimento, é necessário que a mesma minimize a função do erro, iterativamente, até que o erro seja suficientemente pequeno.

Para que o processo de minimização da função do erro seja feito de maneira satisfatória, deve-se levar em consideração a direção que será tomada a cada iteração do processo e o tamanho do passo nesta direção. Desta maneira, pode-se garantir que os pesos sinápticos irão convergir para uma solução do problema.

Serão abordados a seguir, alguns métodos de minimização que podem ser usados no processo de auto-ajuste dos pesos sinápticos das redes neurais artificiais.

2.1 Método do gradiente

O gradiente (tomado neste trabalho como g_k) é um vetor que indica a direção e o sentido de maior crescimento, a partir de um ponto, do valor de uma grandeza. O módulo do vetor gradiente (denotado por $||g_k||$) indica a taxa de variação de grandeza escalar, ou seja, o tamanho do vetor, com relação à distância movida, de um ponto a outro, quando se desloca na direção e sentido deste vetor.

Tendo em mente que o vetor gradiente aponta para a direção e sentido de maior crescimento da função, uma vez que o objetivo seja minimizar a função, deve-se tomar a direção oposta ao gradiente, ou seja:

$$d^k = -g_k \tag{2.1}$$

Dessa maneira, a equação (1.16) pode ser reescrita como segue:

$$x^{k+1} = x^k - tg_k \tag{2.2}$$

A equação (2.2) é chamada de algoritmo ou método do gradiente.

De acordo com Yu e Wilamowski (YU; WILAMOWSKI, 2011), "O processo de treinamento do algoritmo do gradiente é de convergência assintótica. Na proximidade

da solução, todos os elementos do vetor gradiente seriam muito pequenos e haveria uma mudança muito pequena nos pesos".

2.1.1 Estratégias para determinação da taxa de aprendizagem

A taxa de aprendizagem, usada pelas redes neurais artificiais, nada mais é que o tamanho do passo dado, a partir do ponto atual, seguindo a direção de descida adotada.

No modelo tradicional do perceptron multicamadas, a taxa de aprendizagem escolhida é um valor fixo, aleatório e pequeno. Entretanto, adotar este valor aleatoriamente pode fazer com que a convergência do algoritmo se torne bastante lenta. Além disso, com o tamanho do passo fixo, duas situações podem ocorrer:

A primeira, caso a taxa de aprendizagem seja muito pequena, o algoritmo iria convergir para a solução de uma maneira tão lenta, que não alcançaria a solução em um tempo viável, como demonstrado na Figura 4, à esquerda.

A segunda, caso a taxa de aprendizagem seja muito grande, o algoritmo poderia saltar a solução ideal em cada iteração e, da mesma forma, levar muito tempo para alcançar a solução, como demonstrado na Figura 4, à direita. Em um pior caso, o algoritmo poderia alcançar o chamado efeito ping-pong e nem mesmo chegar a uma solução.

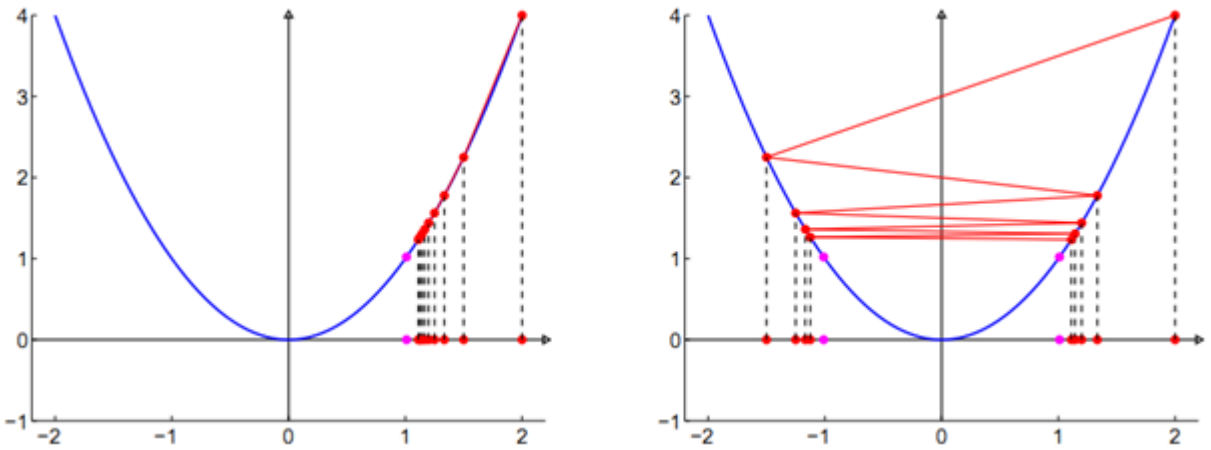

Figura 4 – À esquerda, adotando o tamanho do passo pequeno; à direita, grande
Fonte: (RIBEIRO; KARAS, 2011)

Uma forma de contornar estes problemas é calcular o tamanho do passo para cada iteração do algoritmo e, a partir de cada ponto, andar na direção de descida pelo tamanho do passo calculado.

Na literatura são conhecidos métodos capazes de calcular o tamanho do passo para alcançar uma solução exata ou aproximada, tais como o método de Armijo, o método de Euler, o método poligonal melhorado, o método de Heun, entre outros apresentados em Farias (FARIAS, 2012), Kuhl (KUHL, 2010), Ribeiro e Karas (RIBEIRO; KARAS, 2011).

2.2 Método de Newton

O método de Newton é um dos métodos de otimização mais importante da literatura, não apenas por sua implementação, mas por que serviu de inspiração para muitos métodos derivados do mesmo. Referências ao método de Newton podem ser encontradas em diversos trabalhos, como os quais Yu e Wilamowski (YU; WILAMOWSKI, 2011), Miao (MIAO, 2002), Farias (FARIAS, 2012), Ribeiro e Karas (RIBEIRO; KARAS, 2011).

Apesar de ser um método robusto, o método de Newton tem algumas limitações. Uma delas é que este pode ser aplicado apenas em funções que são soma de quadrados de funções não-lineares. Estas funções podem ser generalizadas da seguinte forma:

$$E(w) = \sum_{i=1}^{m} r_i^2(w). \tag{2.3}$$

Como pode ser visto em (1.6), a equação do erro quadrático se enquadra nesta limitação e, portanto, o método de Newton pode ser aplicado.

Yu e Wilamowski (YU; WILAMOWSKI, 2011) salientam que o método de Newton presume que todos os pesos são linearmente independentes e as componentes do vetor gradiente são funções do peso, como demonstrado a seguir:

$$\begin{cases} g_1 &= E_1(w_1, w_2, ..., w_n) \\ g_2 &= E_2(w_1, w_2, ..., w_n) \\ \dots & \\ g_n &= E_n(w_1, w_2, ..., w_n) \end{cases} \tag{2.4}$$

nas quais $E_1, E_2, \dots, E_n$ são relações não lineares entre os pesos e componentes do grafiente relacionado.

Derivando g_k em (2.4), pela série de Taylor, encontrada em Ribeiro e Karas (RIBEIRO; KARAS, 2011), e usando a aproximação de primeira ordem, tem-se:

$$\begin{cases} g_1 \approx g_{1,0} + \frac{\partial g_1}{\partial w_1}\triangle w_1 + \frac{\partial g_1}{\partial w_2}\triangle w_2 + ... + \frac{\partial g_1}{\partial w_n}\triangle w_n \\ g_2 \approx g_{2,0} + \frac{\partial g_2}{\partial w_1}\triangle w_1 + \frac{\partial g_2}{\partial w_2}\triangle w_2 + ... + \frac{\partial g_2}{\partial w_n}\triangle w_n \\ \dots \\ g_n \approx g_{n,0} + \frac{\partial g_n}{\partial w_1}\triangle w_1 + \frac{\partial g_n}{\partial w_2}\triangle w_2 + ... + \frac{\partial g_n}{\partial w_n}\triangle w_n \end{cases} \tag{2.5}$$

Combinando-se a definição de gradiente com o gradiente da função do erro em relação à w, obtem-se:

$$\frac{\partial g_i}{\partial w_j} = \frac{\partial(\frac{\partial E}{\partial w_j})}{\partial w_j} = \frac{\partial^2 E}{\partial w_i \partial w_j} \tag{2.6}$$

Ao inserir (2.6) em (2.5), tem-se:

$$\begin{cases} g_1 \approx g_{1,0} + \frac{\partial^2 E}{\partial^2 w_1}\triangle w_1 + \frac{\partial^2 E}{\partial w_1 \partial w_2}\triangle w_2 + ... + \frac{\partial^2 E}{\partial w_1 \partial w_n}\triangle w_n \\ g_2 \approx g_{2,0} + \frac{\partial^2 E}{\partial w_2 \partial w_1}\triangle w_1 + \frac{\partial^2 E}{\partial^2 w_2}\triangle w_2 + ... + \frac{\partial^2 E}{\partial w_2 \partial w_n}\triangle w_n \\ \dots \\ g_n \approx g_{n,0} + \frac{\partial^2 E}{\partial w_n \partial w_1}\triangle w_1 + \frac{\partial^2 E}{\partial w_n \partial w_2}\triangle w_2 + ... + \frac{\partial^2 E}{\partial^2 w_n}\triangle w_n \end{cases} \tag{2.7}$$

Comparando com o método do gradiente, as derivadas de segunda ordem da função do erro total precisam ser calculadas para cada componente do vetor gradiente. Com intuito de minimizar a função do erro, cada elemento do gradiente deveria ser zero, portanto, os lados esquerdos das equações (2.7) são todos zeros, logo:

$$\begin{cases} 0 \approx g_{1,0} + \frac{\partial^2 E}{\partial^2 w_1}\triangle w_1 + \frac{\partial^2 E}{\partial w_1 \partial w_2}\triangle w_2 + ... + \frac{\partial^2 E}{\partial w_1 \partial w_n}\triangle w_n \\ 0 \approx g_{2,0} + \frac{\partial^2 E}{\partial w_2 \partial w_1}\triangle w_1 + \frac{\partial^2 E}{\partial^2 w_2}\triangle w_2 + ... + \frac{\partial^2 E}{\partial w_2 \partial w_n}\triangle w_n \\ \dots \\ 0 \approx g_{n,0} + \frac{\partial^2 E}{\partial w_n \partial w_1}\triangle w_1 + \frac{\partial^2 E}{\partial w_n \partial w_2}\triangle w_2 + ... + \frac{\partial^2 E}{\partial^2 w_n}\triangle w_n \end{cases} \tag{2.8}$$

Ao combinar as equações (2.8) com o gradiente do erro em relação à w, tem-se:

$$\begin{cases} -\frac{\partial E}{\partial w_1} = -g_{1,0} \approx \frac{\partial^2 E}{\partial^2 w_1}\triangle w_1 + \frac{\partial^2 E}{\partial w_1 \partial w_2}\triangle w_2 + ... + \frac{\partial^2 E}{\partial w_1 \partial w_n}\triangle w_n \\ -\frac{\partial E}{\partial w_2} = -g_{2,0} \approx \frac{\partial^2 E}{\partial w_2 \partial w_1}\triangle w_1 + \frac{\partial^2 E}{\partial^2 w_2}\triangle w_2 + ... + \frac{\partial^2 E}{\partial w_2 \partial w_n}\triangle w_n \\ \dots \\ -\frac{\partial E}{\partial w_n} = -g_{n,0} \approx \frac{\partial^2 E}{\partial w_n \partial w_1}\triangle w_1 + \frac{\partial^2 E}{\partial w_n \partial w_2}\triangle w_2 + ... + \frac{\partial^2 E}{\partial^2 w_n}\triangle w_n \end{cases} \tag{2.9}$$

que em forma de matriz fica:

$$\begin{bmatrix} -g_1 \\ -g_2 \\ \dots \\ -g_n \end{bmatrix} = \begin{bmatrix} -\frac{\partial E}{\partial w_1} \\ -\frac{\partial E}{\partial w_2} \\ \dots \\ -\frac{\partial E}{\partial w_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial^2 E}{\partial^2 w_1}\triangle w_1 & \frac{\partial^2 E}{\partial w_1 \partial w_2}\triangle w_2 & ... & \frac{\partial^2 E}{\partial w_1 \partial w_n}\triangle w_n \\ \frac{\partial^2 E}{\partial w_2 \partial w_1}\triangle w_1 & \frac{\partial^2 E}{\partial^2 w_2}\triangle w_2 & ... & \frac{\partial^2 E}{\partial w_2 \partial w_n}\triangle w_n \\ \dots & & & \\ \frac{\partial^2 E}{\partial w_n \partial w_1}\triangle w_1 & \frac{\partial^2 E}{\partial w_n \partial w_2}\triangle w_2 & ... & \frac{\partial^2 E}{\partial^2 w_n}\triangle w_n \end{bmatrix} \times \begin{bmatrix} \triangle w_1 \\ \triangle w_2 \\ \dots \\ \triangle w_n \end{bmatrix} \tag{2.10}$$

Nota-se que a matriz Hessiana é:

$$H = \begin{bmatrix} \frac{\partial^2 E}{\partial^2 w_1}\triangle w_1 & \frac{\partial^2 E}{\partial w_1 \partial w_2}\triangle w_2 & \dots & \frac{\partial^2 E}{\partial w_1 \partial w_n}\triangle w_n \\ \frac{\partial^2 E}{\partial w_2 \partial w_1}\triangle w_1 & \frac{\partial^2 E}{\partial^2 w_2}\triangle w_2 & \dots & \frac{\partial^2 E}{\partial w_2 \partial w_n}\triangle w_n \\ \dots & & & \\ \frac{\partial^2 E}{\partial w_n \partial w_1}\triangle w_1 & \frac{\partial^2 E}{\partial w_n \partial w_2}\triangle w_2 & \dots & \frac{\partial^2 E}{\partial^2 w_n}\triangle w_n \end{bmatrix} \tag{2.11}$$

Ao combinar (2.11) com o gradiente do erro em relação à w e (2.10), obtem-se:

$$-g = H\triangle w \tag{2.12}$$

Portanto:

$$\triangle w = -H^{-1}g \tag{2.13}$$

Finalmente, a regra de atualização dos pesos segundo o método de Newton é:

$$w_{k+1} = w_k - H_k^{-1}g_k \tag{2.14}$$

Apesar de eficiente, o cálculo da matriz Hessiana torna o método de Newton um pouco lento quando comparado a outros métodos de minimização de mesma ordem. Ainda assim, o método de Newton serve de referência para uma gama de outros métodos.

2.3 INPPA

INPPA é um algoritmo inexato não-monótono de ponto proximal desenvolvido por Santos e Silva (SANTOS; SILVA, 2014) para problemas de minimização suave sem restrições. O INPPA utiliza o método do ponto poximal aliado ao algoritmo CG-Steihaug para encontrar uma direção de descida e controlar o tamanho do passo nesta direção.

Através do parâmetro proximal $(t_k) \subset \mathbb{R}_+$, um método de ponto proximal aplicado na resolução de um problema de minimização genérico gera uma sequência de pontos (x^k) que tendem a uma solução ótima (x^*). Ao associá-lo à solução do sub-problema do ponto proximal, mostrado também em (APOLINARIO, 2014) e (MACEDO, 2009), tem-se:

$$\min_{x\in\mathbb{R}^n} f(x) + \frac{1}{2t_k}||x - x_k||^2. \tag{2.15}$$

E o modelo quadrático para f próximo de x_k é dado por

$$q_k(x) := f_k + g_k^T(x - x_k) + \frac{1}{2}(x - x_k)^T B_k(x - x_k) \tag{2.16}$$

na qual, $B_k \in \mathbb{R}^{n\times n}$ é qualquer matriz simétrica que aproxime a Hessiana de $f(x)$.

Assim, a fim de resolver (2.15), adota-se a aproximação quadrática de $f(x)$:

$$\min_{x\in\mathbb{R}^n} q_k(x) + \frac{1}{2t_k}||x - x_k||^2 \tag{2.17}$$

Para provar a eficiência do INPPA. Santos e Silva realizaram testes de resolução de problemas selecionados a partir da coleção CUTEr, referenciado pelos mesmos. Seus resultados mostram uma boa performance na resolução de problemas degenerados, além de alta eficiência quando comparado a métodos de filtro de região de confiança.

3 Materiais e Métodos

Dentre as aplicações do Perceptron Multicamadas, uma das principais é a classificação de padrões. Para avaliar as alterações propostas no algoritmo de treinamento do PMC, o mesmo foi submetido a testes de classificação de padrões de portas lógicas "AND", "OR"e "XOR"e identificação de tumor mamário, à partir de valores extraídos de telerradiografias.

Para fins comparativos do modelo proposto, tanto o PMC convencional como o PMC proposto foram submetidos ao mesmo conjunto de padrões de entrada, pesos sinápticos inicias, topologia e conjunto de testes, a cada comparação efetuada. Tanto a topologia quanto os pesos sinápticos iniciais adotados em cada teste são geradas aleatoriamente.

A implementação de ambos os algoritmos foi feita em scilab, utilizando a IDE Scilab 5.3, executado em um computador modelo desktop de processador Intel(R) Core(TM) i5-4440, 3.10 GHz, 8GB de memória RAM DDR3 1333MHz e sistema operacional Windows 7 64 bits.

Parte II

Proposta

4 Sistema Proposto

Este trabalho tem como proposta combinar um método de otimização ao modelo clássico do Perceptron Multicamadas a fim de aumentar a sua capacidade de generalização de conhecimento, fazendo com que o mesmo aumente sua capacidade de abstração, tornando seus resultados mais precisos quando aplicados a problemas de classificação e generalização. Para alcançar tal objetivo foram avaliadas técnicas de minimização de funções e seu comportamento frente a solução de problemas.

Em linhas gerais, os problemas de minimização são focados em dois pontos, encontrar uma direção de descida e calcular o tamanho do passo. De modo geral, um bom algoritmo de minimização encontra uma combinação destes dois pontos, alcançando a maior minimização a cada iteração.

Como no algoritmo clássico do PMC a direção de descida é calculada, sendo essa a direção oposta ao gradiente, mas o tamanho do passo, chemado no PMC de taxa de aprendizagem, não é calculado e sim atribuído um valor aleatório pequeno, buscou-se um método que pudesse calcular âmbos.

Dentre os métodos pesquisados, um algoritmo de minimização baseado em região de confiança e método do ponto proximal se destacou. Batizado de INPPA (Inexact Nonmonotone Proximal Point Algorithm) por Santos e Silva (SANTOS; SILVA, 2014), seus criadores, este se mostrou bastante eficiente.

A partir do algoritmo clássico do PMC e dos resultados mostrados por Santos e Silva, propõe-se a combinação do algoritmo clássico do PMC com o INPPA, em sua fase de treinamento, a fim de tornar a rede mais precisa em sua fase de classificação.

Parte III

Parte Final

5 Resultados e Discussão

Para validar o sistema proposto, foram realizados alguns testes comparando o resultado obtido pelo mesmo com o modelo clássico do PMC.

Os testes tiveram como foco avaliar três categorias do método proposto, sendo essas:

- Precisão das saídas;
- Tempo de convergência;
- Intervalo das respostas apresentadas.

Como base de dados foram utilizadas duas classes de problemas:

- Classificação de saídas das portas lógicas AND, OR e XOR;
- Identificação de tumor mamário, à partir de valores extraídos de telerradiografias.

O resultado dos testes realizados são exibidos a seguir.

5.1 Resultados numéricos

Para obtenção dos resultados deste trabalho, foram realizados testes de classificação de padrões, tanto pelo PMC clássico quanto pelo modelo proposto, chamado aqui de PMC INPPA. Cada grupo de testes é composto por 100 execuções do algoritmo e cada execução inicializa novos valores para os pesos sinápticos iniciais e uma nova topologia. Tanto a topologia quanto os pesos iniciais utilizados em um modelo do PMC são os mesmos utilizados pelo outro modelo na corrente iteração da comparação.

A inicialização dos pesos sinápticos se dá de maneira aleatória, atribuindo aos mesmos valores que variam de -2 a 2, com uma precisão de seis casas decimais. Da mesma forma, a escolha da topologia utilizada é feita de maneira aleatória, sendo que a mesma varia de uma a duas camadas neurais escondidas e de dois a cinco neurônios em cada uma destas camadas. A camada de saída sempre possui um único neurônio. As funções de ativação de cada camada neural são escolhidas aleatoriamente entre a função logística e a função tangente hiperbólica, de modo que a função de ativação seja a mesma para todos os neurônios da mesma camada, mas não necessariamente seja a mesma de outras camadas neurais.

Para facilitar a interpretação dos resultados, os mesmos foram apresentados em forma de gráficos. Os eixos dos gráficos representam a atual iteração de teste que foi realizada com novos valores iniciais (eixo horizontal) e o número de acertos de cada algoritmo em sua fase de classificação (eixo vertical). Cada ponto colorido representa o número de acertos da atual iteração do teste, sendo em azul o PMC clássico, e em vermelho o PMC INPPA. Mesmo que os resultados sejam discretos, foi passada uma linha de continuidade, da mesma cor dos pontos, entre os pontos para facilitar a interpretação.

Para considerar que uma resposta dada por um dos algoritmos está correta, adotou-se uma taxa de tolerância de 0.2, para mais ou para menos, com relação ao valor de resposta ideal.

Também é apresentado o comparativo do tempo médio de execução decorrido na fase de treinamento de cada algoritmo. Este tempo é apresentado em milissegundos.

A taxa de aprendizagem da fase de treinamento adotada no PMC clássico foi 0.5 para todos os grupos de teste. Também para todos os grupos de teste, a precisão requerida foi de 10^{-6}. Os outros dados pertinentes à cada rede são exibidos logo antes dos respectivos gráficos.

Figura 5: porta lógica AND; 4 entradas; 10 padrões de aprendizagem e 6 padrões de classificação.

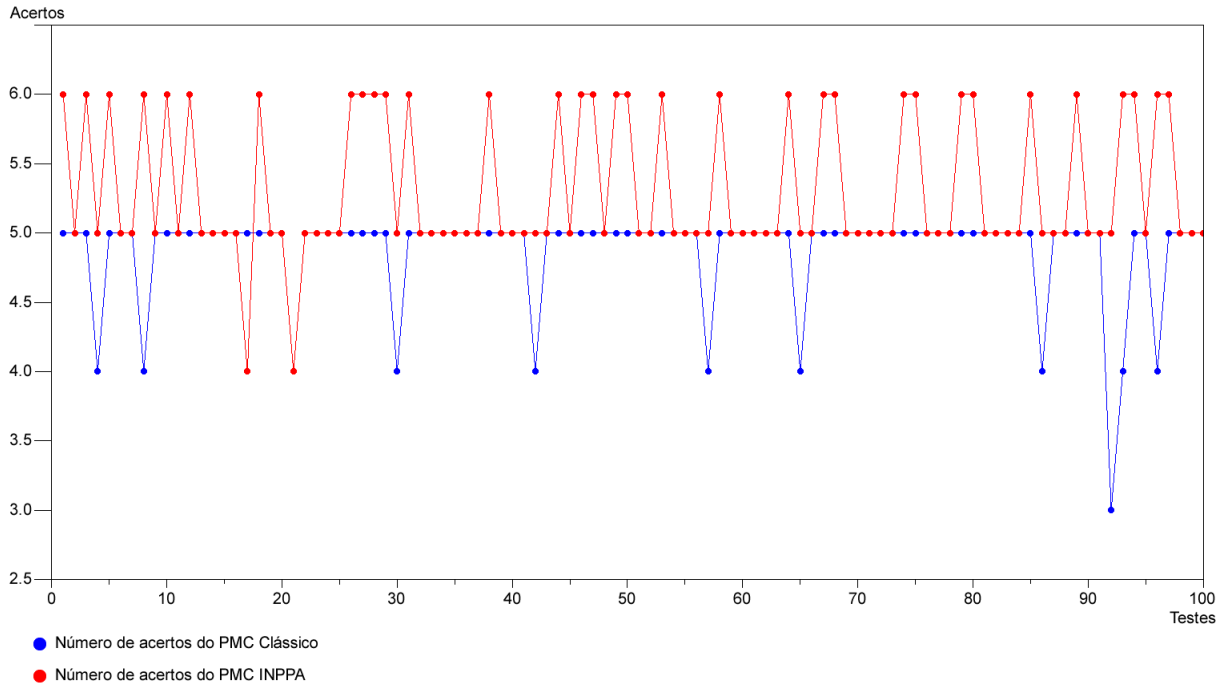

Figura 5 – Gráfico comparativo dos acertos na classificação da porta lógica AND 4 entradas
Fonte: próprio autor

Como pode-se notar na Figura 5, o PMC clássico apresentou como melhor resultado 5 acertos, 89 resultados dentre os 100 testados, enquanto o PMC INPPA apresentou como

Tabela 1 – Comparação dos tempos de execução para a porta lógica AND 4 entradas

	PMC Clássico	PMC INPPA
Tempo de execução	101.33482	165.80878

melhor resultado 6 acertos, 33 resultados dentre os 100. Além disso, somando os resultados para 5 e 6 acertos do PMC INPPA obtêm-se 98 acertos neste intervalo. Já o tempo de execução do PMC Clássico foi de 101.33482 e do PMC INPPA de 165.80878.

Figura 6: porta lógica OR; 4 entradas; 10 padrões de aprendizagem e 6 padrões de classificação.

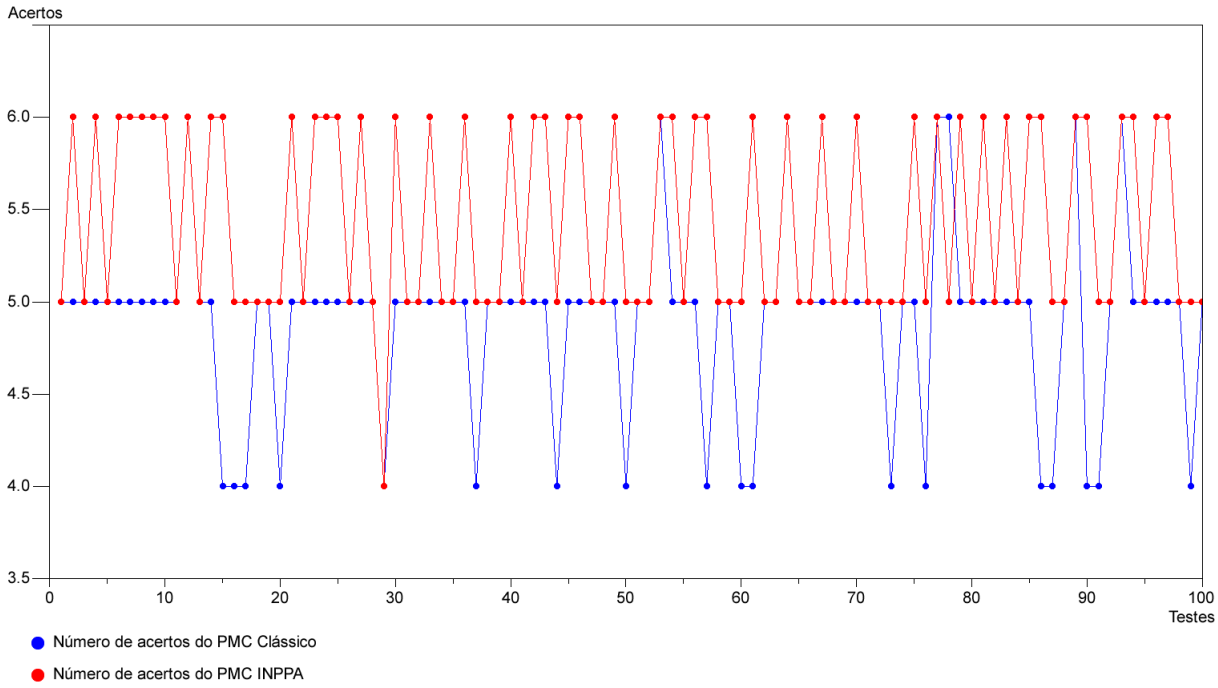

Figura 6 – Gráfico comparativo dos acertos na classificação da porta lógica OR 4 entradas
Fonte: próprio autor

Tabela 2 – Comparação dos tempos de execução para a porta lógica OR 4 entradas

	PMC Clássico	PMC INPPA
Tempo de execução	88.688441	154.40449

A Figura 6 apresenta como melhor resultado do PMC clássico e do PMC INPPA 6 acertos, 7 resultados do clássico e 45 do INPPA, enquanto o resultado de âmbos para 5 acertos foi de 75 para o PMC clássico e 54 para PMC INPPA, ficando evidente a variação muito grande dentre os resultados do PMC clássico. Novamente, observando o intervalo de 5 e 6 acertos, o PMC clássico obteve 82 resultados contra 99 do PMC INPPA. O tempo de execução do PMC Clássico foi de 88.688441 e do PMC INPPA de 154.40449.

Figura 7: porta lógica XOR; 4 entradas; 10 padrões de aprendizagem e 6 padrões de classificação.

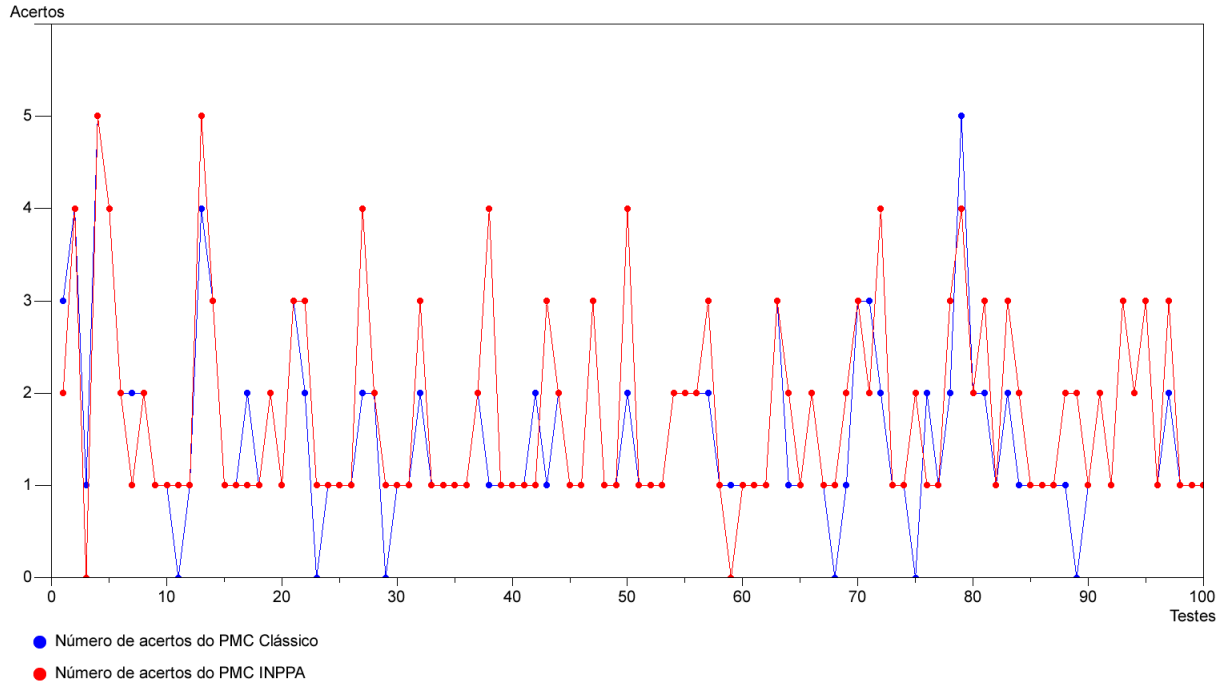

Figura 7 – Gráfico comparativo dos acertos na classificação da porta lógica XOR 4 entradas
Fonte: próprio autor

Tabela 3 – Comparação dos tempos de execução para a porta lógica XOR 4 entradas

	PMC Clássico	PMC INPPA
Tempo de execução	538.769036	274.61305

Na Figura 7, tanto o PMC clássico quanto o PMC apresentaram como melhor resultado 5 acertos, apenas 2 resultados dentre os 100 testados. Analisando o intervalo de 3, 4 e 5 acertos, nota-se uma ligeira vantagem do PMC INPPA sobre o PMC clássico, 24 contra 14 resultados. O tempo de execução do PMC Clássico foi de 538.769036 e do PMC INPPA de 274.61305.

Figura 8: porta lógica AND; 5 entradas; 22 padrões de aprendizagem e 10 padrões de classificação.

Tabela 4 – Comparação dos tempos de execução para a porta lógica AND 5 entradas

	PMC Clássico	PMC INPPA
Tempo de execução	160.86563	141.40776

A Figura 8 apresenta como melhor resultado do PMC clássico 7 acertos, 1 dos 100 testados, e do PMC INPPA 10 acertos, sendo 65 resultados dos 100 testados. O tempo de execução do PMC Clássico foi de 160.86563 e do PMC INPPA de 141.40776.

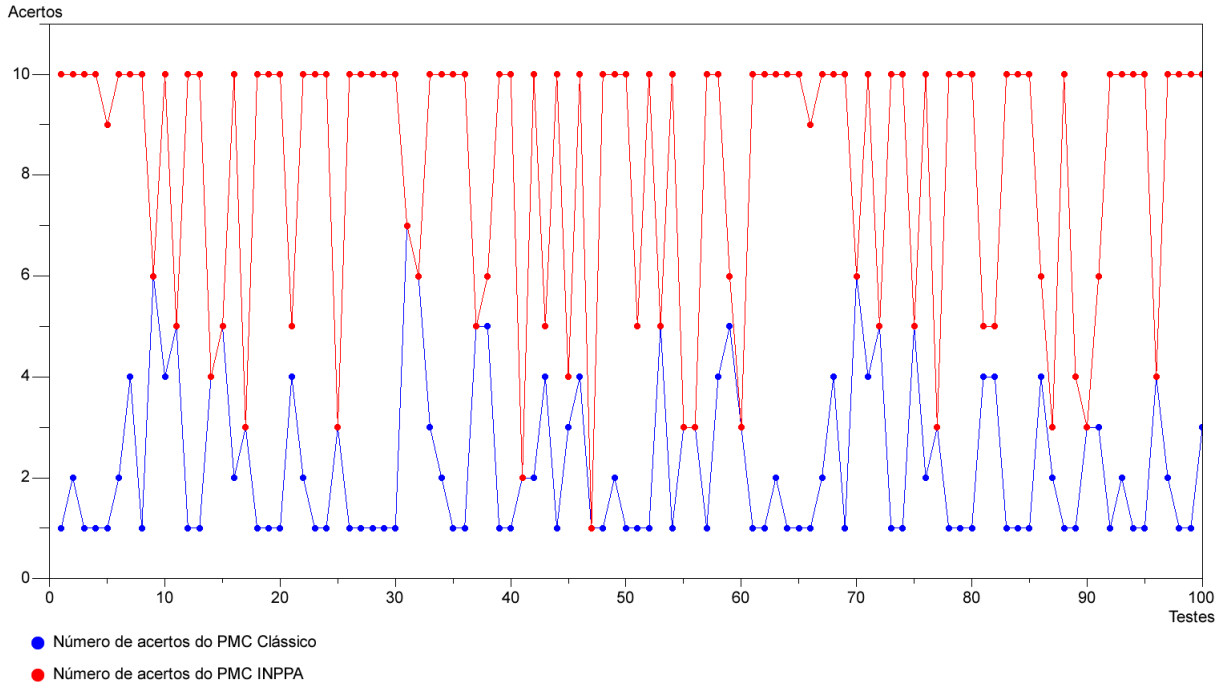

Figura 8 – Gráfico comparativo dos acertos na classificação da porta lógica AND 5 entradas
Fonte: próprio autor

Figura 9: porta lógica OR; 5 entradas; 22 padrões de aprendizagem e 10 padrões de classificação.

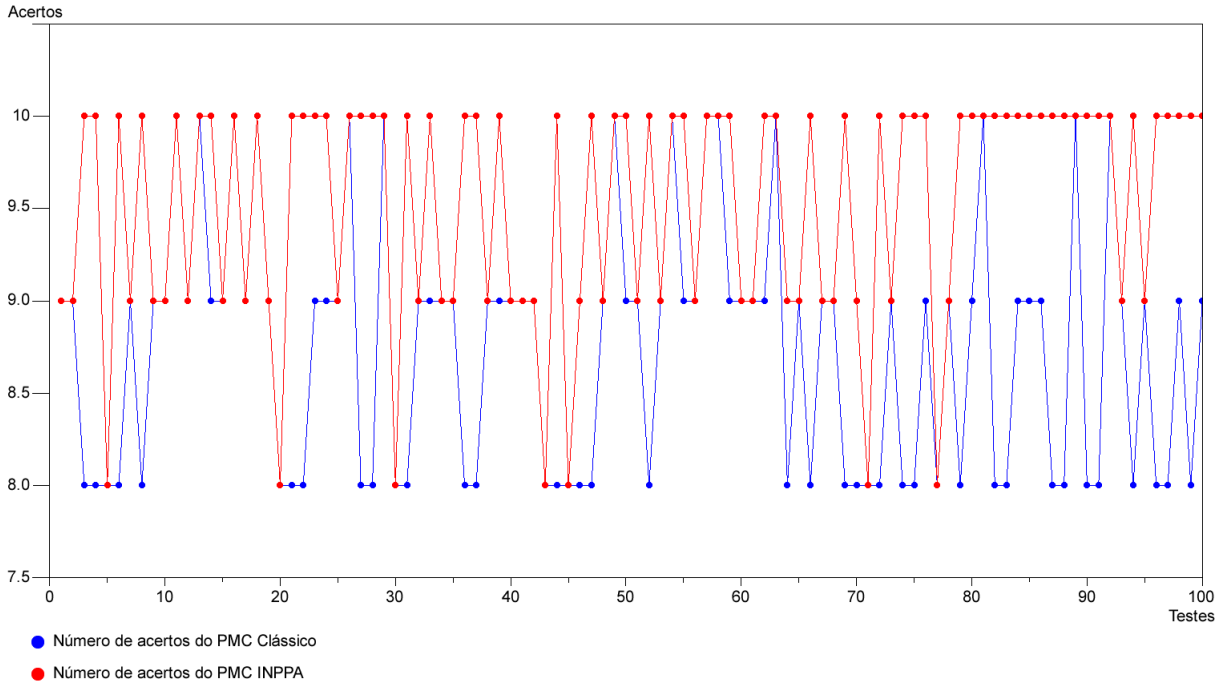

Figura 9 – Gráfico comparativo dos acertos na classificação da porta lógica OR 5 entradas
Fonte: próprio autor

Tabela 5 – Comparação dos tempos de execução para a porta lógica OR 5 entradas

	PMC Clássico	PMC INPPA
Tempo de execução	154.06789	191.09472

Como pode-se notar na Figura 9, tanto o PMC clássico quanto o PMC INPPA apresentaram como melhor resultado 10 acertos, sendo 14 resultados do clássico e 60 do INPPA. Além disso, somando os resultados para 9 e 10 acertos do PMC INPPA obtêm-se 93 resultados contra 60 resultados do clássico neste intervalo. O tempo de execução do PMC Clássico foi de 154.06789 e do PMC INPPA de 191.09472.

Figura 10: porta lógica XOR; 5 entradas; 22 padrões de aprendizagem e 10 padrões de classificação.

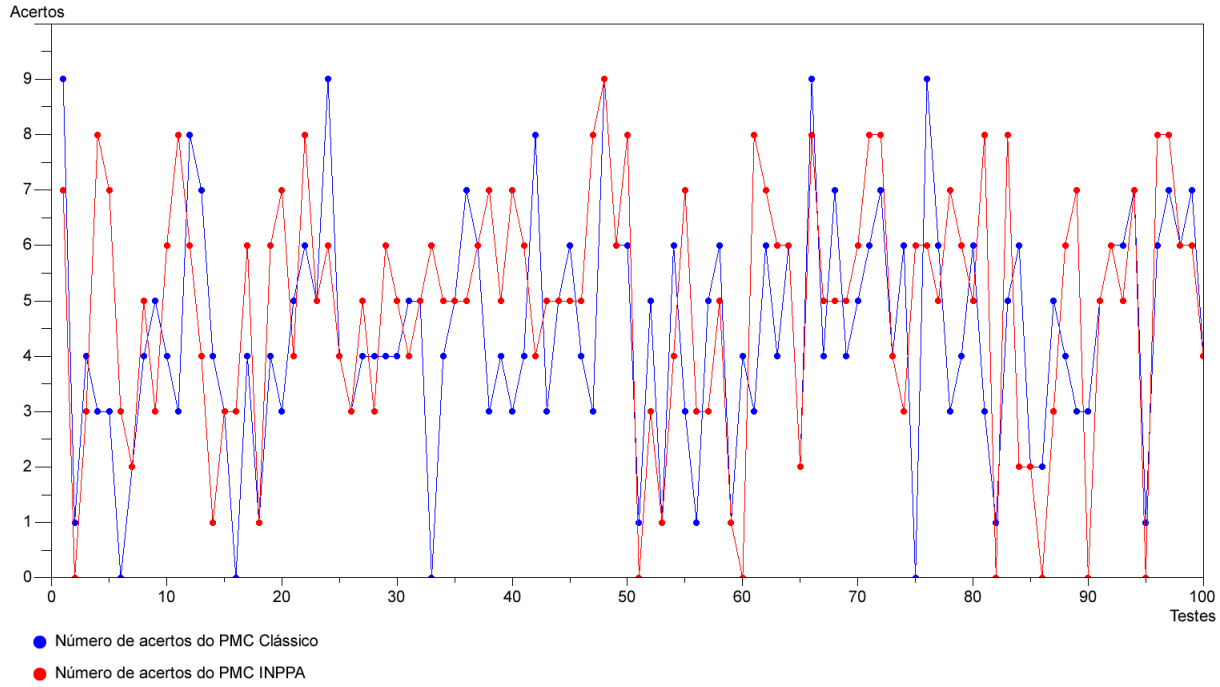

Figura 10 – Gráfico comparativo dos acertos na classificação da porta lógica XOR 5 entradas
Fonte: próprio autor

Tabela 6 – Comparação dos tempos de execução para a porta lógica XOR 5 entradas

	PMC Clássico	PMC INPPA
Tempo de execução	735.96332	458.07679

Na Figura 10 pode-se notar que o melhor resultado do PMC clássico e do PMC INPPA foi de 9 acertos, sendo 5 do clássico e 1 do INPPA. Entretanto, ao analisar o intervalo de 7, 8 e 9 acertos, obtêm-se 24 resultados do PMC INPPA contra 14 do PMC clássico. Já o tempo de execução do PMC Clássico foi de 735.96332 e do PMC INPPA de 458.07679.

Figura 11: identificação de tumor mamário por telerradiografia; 4 entradas; 100 padrões de aprendizagem e 50 padrões de classificação.

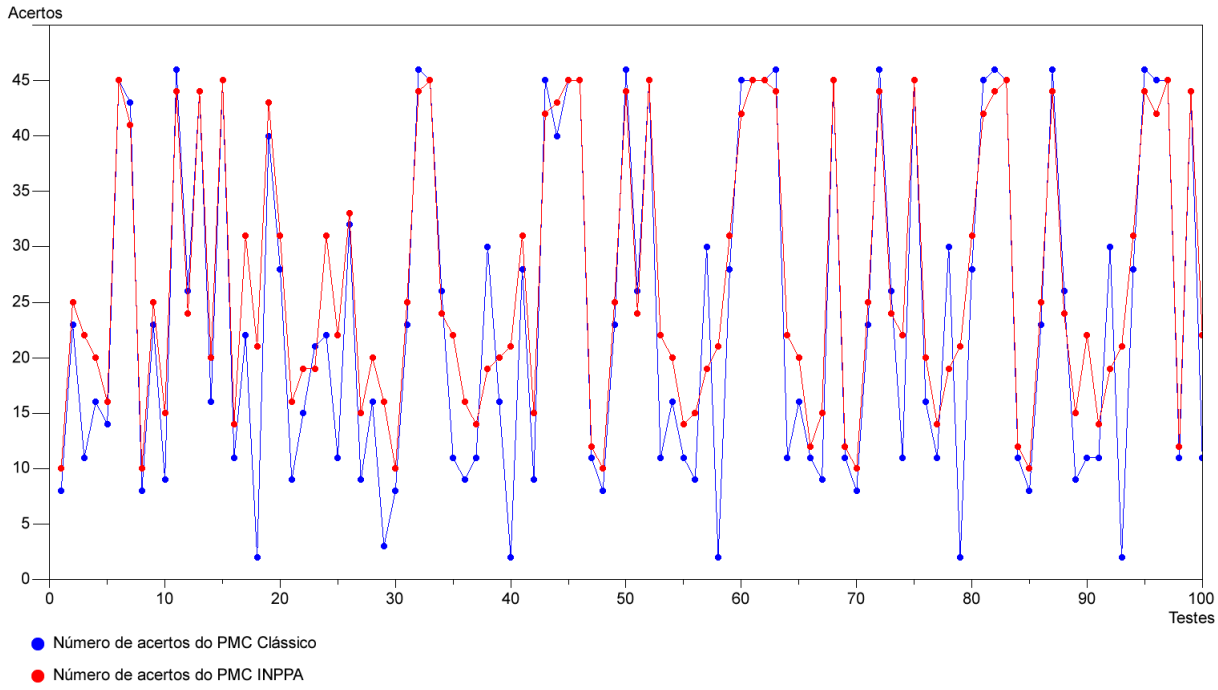

Figura 11 – Gráfico comparativo dos acertos na classificação de tumor mamário
Fonte: próprio autor

Tabela 7 – Comparação dos tempos de execução para a classificação de tumor mamário

	PMC Clássico	PMC INPPA
Tempo de execução	3897.4276	1363.4799

A Figura 11 apresenta como melhor resultado do PMC clássico 46 acertos, 8 dos 100 testados, e do PMC INPPA 45 acertos, sendo 12 dos 100 testados. Ao observar o intervalo de 40 a 46 acertos, âmbos somam 29 resultados, entretanto, ao analizar o intervalo abaixo de 20 acertos, nota-se que o PMC clássico soma 47 resultados contra 32 resultados do PMC INPPA, ou seja, uma maior concentração de resultados em uma região de poucos acertos se comparado o clássico ao INPPA. O tempo de execução do PMC Clássico foi de 3897.4276 e do PMC INPPA de 1363.4799.

5.2 Base de Dados

Para a realização dos testes de padrões a partir de telerradiografias mamárias deste trabalho foram utilizados dados adiquiridos do banco de dados "Digital Database for Screening Mammography"(DDSM), desenvolvido pela "University of South Florida"(HEATH M.AND BOWYER et al., 2001) e (HEATH et al., 1998).

5.3 Considerações Finais

Como visto nos dados numéricos exibidos anteriormente, pôde-se observar que a proposta deste trabalho se mostrou satisfatória.

Em todos os casos observados, o PMC INPAA apresentou resultados mais precisos (maior número de acertos) que o PMC clássico. Além disso, destaca-se também que o tempo de execução da fase de classificação dos dois algoritmos foi bem competitivo em problemas simples, como classificação das portas lógicas OR e AND. Já nos problemas mais complexos, como a classificação da porta lógica XOR de 5 entradas e a classificação de tumor mamário, o PMC INPPA apresentou tempos bem menores que o PMC Clássico. Vale ainda ressaltar que na fase de treinamento do INPPA, muitas operações com matrizes são realizadas, e no algoritmo desenvolvido para a avaliação deste trabalho, nenhuma otimização de código foi realizada nessas operações, o que fez com que o tempo de convergência fosse ainda maior.

Além disso, ao comparar individualmente os resultados de cada iteração dos testes realizados, fica ainda mais evidente a vantagem do PMC INPPA sobre o PMC clássico, exceto em poucas iterações isoladas.

Conclusões e Trabalhos Futuros

Este trabalho se dispôs a pesquisar um método de minimização que pudesse ser combinado ao modelo clássico do PMC, a fim de torná-lo mais eficiente e eficaz, aumentando sua capacidade de generalização, tornando seus resultados mais precisos. Desta forma, viu-se no INPPA uma boa alternativa para alcançar tais propósitos, uma vez que o mesmo se mostrou muito eficiente em testes numérico realizados por seus desenvolvedores.

Como mostrado nos experimentos numéricos realizados neste trabalho, ficou claro que a proposta de combinar o INPPA ao PMC clássico alançou as espectativas propostas. Não só o algoritmo proposto alcançou uma taxa de acertos maior que o algoritmo clássico, como o fez em menos tempo, e quando não, pelo menos em um tempo competitivo.

Em síntese, o modelo proposto se mostra um bom método para resolução de problemas de classificação e predição de valores em redes neurais, além de abrir uma gama de novas propostas de pesquisa.

Trabalhos Futuros

Como propostas de trabalhos futuros pretende-se buscar novos métodos de minimização, a fim de combiná-los aos algoritmos utilizados neste trabalho com intuito de alcançar uma capacidade de generalização ainda maior pela rede, tornando seus resultados mais precisos. Ainda como proposta para um futuro trabalho, observar o comportamento sistemático do algoritmo e encontrar um padrão de topologias adequado ao método a fim de evitar problemas de ”overfitting”e ”underfitting”.

Pretende-se encontrar alternaivas para aumentar a velocidade de convergência do algoritmos, como evitar o cálculo da matriz Hessiana, buscando métodos que possam substituir este cálculo com igual eficiência e buscar diminuir o número de operação com matrizes, que demandam muito mais tempo que operações com valores escalares.

Outra pretenção é verificar a viabilidade de adequar o INPPA a outros modelos de redes neurais, como as redes de Hopfield e Kohonen, uma vez que o método se mostrou eficiente para o PMC.

Pretende-se também encontrar novas bases de dados, tanto de classificação como de generalização e predição de valores, para realizar mais testes comparativos. Além disso, realizar comparações com outros métodos combinados ao PMC encontrados na literatura.

Referências

APOLINARIO, H. C. F. Método de Ponto Proximal para Minimização Multiobjetivo Quase-Convexa. *Tese de Doutorado - PESC/UFRJ*, 2014.

FARIAS, T. S. M. C. Metodologia para reconstrução 3d baseada em imagens. *Journal Desconhecido*, 2012.

FRIEDLANDER, A. Elementos de programação não-linear. *Journal Desconhecido*, 1994.

HAYKIN, S. Redes Neurais Princípios e prática. *ARTMED Editora S. A., Porto Alegre, RS*, 2008.

HEATH, M. et al. Digital mammography. *Proceedings of the Fourth International Workshop on Digital Mammography, 457-460, Kluwer Academic Publishers*, 1998.

HEATH M.AND BOWYER, K. et al. No title. *Proceedings of the Fifth International Workshop on Digital Mammography, M.J. Yaffe, 212-218. Medical Physics Publishing. ISBN 1-930524-00-5.*, 2001.

HOPFIELD, J. J. Neural Network and physical systems with Emergent collective Computational Abilities. *Division of Chemistry and Biology.*, 1982.

KUHL, N. Métodos de passo simples para equaçõoes diferenciais ordinárias. *Journal Desconhecido*, 2010.

MACEDO, A. C. Método de Ponto Proximal para Otimização. *Journal Desconhecido*, 2009.

MCCULLOCH, W. S.; PITTS, W. H. A Logical Calculus of the Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics*, 1943.

MIAO, G. A comparison of backpropagation and conjugate gradient algorithms forefficient training of multi-layer perceptron networks. *Journal Desconhecido*, 2002.

RIBEIRO, A. A.; KARAS, E. W. Um curso de otimização. *Journal Desconhecido*, 2011.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. *Institute of Cognitive Science*, 1986.

SANTOS, S. A.; SILVA, R. C. M. An inexact and nonmonotone proximal method for smooth unconstrained minimization. *Journal of Computational and Applied Mathematics, 2014*, 2014.

SILVA, I. N.; SPATTI, D. H.; FLAUZINO, R. A. Redes Neurais Artificiais Para Engenharia e Ciências Aplicadas *Artliber, São Paulo, SP*, 2010.

SILVA, R. B. Método do Ponto Proximal Interior para Minimização Quase-Convexa. *Journal Desconhecido*, 2013.

YU, H.; WILAMOWSKI, B.M. Levenberg-marquardt training-industrial electronics handbook. *vol. 5 - Intelligent Systems, chapter 12, pp. 12-1 to 12-15*, 2011.

ZHANG, H.; HAGER, W. W. A Nonmonotone Line Search Technique and its Application to Unconstrained Optimization. *vol. 14 - Society for Industrial and Applied Mathematics, pp. 1043 to 1056*, 2004.

Apêndices

APÊNDICE A – Funções de Apoio

Por não se relacionarem diretamente com o escopo deste trabalho, algumas das funções utilizadas nos algoritmos CG-Steihaug e INPPA não foram apresentadas ao longo do mesmo. Com o intuito facilitar a interpretação desses algoritmos, essas funções serão apresentadas aqui.

$$g := g_k \tag{A.1}$$

$$B := B_k + \frac{1}{t_k} I \tag{A.2}$$

$$\triangle := t_k ||g_k|| \tag{A.3}$$

$$m(s) := g^T s + \frac{1}{2} s^T B s \tag{A.4}$$

$$\lim_{k->\infty} \eta_k = 0 \tag{A.5}$$

$$||s_k|| = t_k ||g_k|| \tag{A.6}$$

$$||(B_k + \frac{1}{t_k} I)s_k + g_k|| \leq \eta_k ||g_k|| \tag{A.7}$$

$$s_k^T \hat{B}_k s_k = s_k^T B_k s_k + i||s_k||^2 > 0 \tag{A.8}$$

$$\sigma_k := \frac{-g_k^T s_k}{s_k^T \hat{B}_k s_k} \tag{A.9}$$

$$f(x_k + \alpha^{(k)} s_k) \leq C_k + \gamma_1 m_k(\alpha^{(k)} s_k) \tag{A.10}$$

$$Q_{k+1} = \xi_k Q_k + 1 \tag{A.11}$$

$$C_{k+1} = \frac{\xi_k Q_k C_k + f(x_{k+1})}{Q_{k+1}} \tag{A.12}$$

Anexos

ANEXO A – Algoritmo CG-Steihaug

O algoritmo CG-Steihaug, apresentado a seguir, foi retirado de (SANTOS; SILVA, 2014).

Dados: $g \in \mathbb{R}^n$, $B \in \mathbb{R}^{nxn}$, $\Delta > 0$, $\epsilon > 0$ e $\eta > 0$, calcular $s \in \mathbb{R}^n$ de tal forma que $||s|| = \Delta$ ou $||Bs + g|| \leq \eta||g||$:

Passo 1. Fazer $s_0 = 0$, $r_0 = -g$, $d_0 = r_0$, $\delta_0 = r_0^T r_0$ e $i = 0$.

Passo 2. Calcular $q_i = Bd_i$ e $\gamma_i = d_i^T q_i$.

Se $\gamma_i \leq \epsilon\delta_i$ então calcular $\tau > 0$ de modo que $||s_i + \tau d_i|| = \Delta$,fazer $s = s_i + \tau d_i$ e retornar.

Passo 3. Calcular $\alpha_i = r_i^T r_i/\gamma_i$ e $s_{i+1} = s_i = \alpha_i d_i$.

Se $||s_{i+1} \geq \Delta$ então calcular $\tau > 0$ de modo que $||s_i + \tau d_i|| = \Delta$, fazer $s = s_i + \tau d_i$ e retornar.

Passo 4. Calcular $r_{i+1} = r_i - \alpha_i q_i$.

Se $||r_{i+1}|| \leq \eta||g||$ então fazer $s = s_{i+1}$ e retornar.

Passo 5. Calcular $\beta_i = r_{i+1}^T r_{i+1}/r_i^T r_i$, $d_{i+1} = r_{i+1} + \beta_i d_i$ e $\delta_{i+1} = r_{i+1}^T r_{i+1} + \beta_i^2 \delta_i$. Fazer $i = i + 1$ e voltar ao Passo 2.

ANEXO B – Algoritmo INPPA (Algoritmo de Ponto Proximal Não-Monótono Inexato)

O algoritmo INPPA, apresentado a seguir, foi retirado de (SANTOS; SILVA, 2014).

Dados: $\varepsilon > 0$, γ_0, γ_1, β, $\theta \in (0,1)$, $\gamma_2 > 1$, $0 < t_{min} < t_{max} < +\infty, 0 \leq \xi_{min} \leq \xi_{max} \leq 1$, $x_0 \in \mathbb{R}^n$ e $t_0 \in [t_{min}, t_{max}]$, calcular g_0 e B_0, fazer $k = 0$, $Q_0 = 1$, $C_0 = f(x_0)$ e definir $\{\eta_k\} \subset \mathbb{R}_+$ satisfazendo (A.5).

Enquanto $||g_k|| > \varepsilon$ fazer:

Passo 1. Resolver (2.17), aproximadamente, obtendo um passo s_k tal que $||s_k|| \leq t_k||g_k||$ e ou (A.6) ou (A.7) seja verdadeiro.

Passo 2. Se $g_k^T s_k > -\theta||g_k||||s_k||$ então

$x_{k+1} = x_k$, $t_{k+1} = \gamma_0 \frac{||s_k||}{||g_k||}$, $Q_{k+1} = Q_k$, $C_{k+1} = C_k$ e $k = k + 1$.

Caso contrário,

2.1 Fazer $\alpha^{(k)} = 1$.

2.2 Se $f(x_k + \alpha^{(k)} s_k) > C_k + \gamma_1 m_k(\alpha^{(k)} s_k)$ então calcular i satisfazendo (A.8) e $\alpha^{(k)}$ o maior α em $\{\sigma_k, \beta\sigma_k, \beta^2\sigma_k, ...\}$ para o qual (A.10) é verificado.

2.3 Fazer $x_{k+1} = x_k + \alpha^{(k)} s_k$, calcular g_{k+1} e B_{k+1}, fazer $t^{(k)} = \frac{||\alpha^{(k)} s_k||}{||g_k||}$ e $t_{k+1} = \min\{t_{max}, \max\{t_{min}, \gamma_2 t^{(k)}\}\}$, atualizar Q_{k+1}, C_{k+1} usando (A.11) e (A.12) e fazer $k = k + 1$.