UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

ESCOLA DE ENGENHARIA

DEPARTAMENTO DE ENGENHARIA ELETRÔNICA E COMPUTAÇÃO

REDES NEURAIS APLICADAS À SEPARAÇÃO ESTRELAS-GALÁXIAS EM PLACAS FOTOGRÁFICAS

Autor: _____

Martin Seefelder de Assis

Orientador: _____

Prof. Luiz Pereira Calôba, Dr.Ing.

Co-Orientador: _____

Prof. Márcio Antônio Geimba Maia, D.Sc.

Examinador: _____

Prof. Luiz Wagner Pereira Biscainho, D.Sc.

Examinador: _____

Dr. César Augusto Caretta, D.Sc.

DEL

MAIO 2003

"Este globo inteiro, este astro, não sendo sujeito à morte, e a dissolução e a aniquilação sendo impossíveis na Natureza, de tempos em tempos renova a si próprio, mudando e alterando todas as suas partes. Não existe o superior e o inferior absoluto, como pensava Aristóteles; não existe posição absoluta no espaço, mas a posição de um corpo celeste é relativa à de outros. Em todo lugar há mudanças na posição através do Universo, e o observador está sempre no

Resumo

Neste trabalho desenvolvemos uma rede neural capaz de classificar os objetos após a digitalização de uma placa fotográfica nas categorias de estrelas, galáxias e outros objetos.

Definindo como classificação confiável aquela que apresenta um erro menor do que 30%, podemos considerar que a rede é confiável até a magnitude $\boldsymbol{m_J}$ = 22,3 para todas as classes, sendo que no caso particular das galáxias, a classificação se mantém confiável mesmo para as galáxias acima de magnitude $\boldsymbol{m_J}$ = 22,5.

Palavras-Chave

REDES NEURAIS

CLASSIFICAÇÃO

LINGUAGEM C++

ESTRELA

GALÁXIA

Índice

RESUMO.....III

PALAVRAS-CHAVE.....IV

ÍNDICE.....V

1 INTRODUÇÃO.....1

1.1 OBJETIVO.....3

2 REDES NEURAIS.....7

2.1 REDES NEURAIS BIOLÓGICAS [2].....7

2.1.1 Divisões do Sistema Nervoso.....7

2.1.2 Neurônio Biológico.....8

2.2 REDES NEURAIS ARTIFICIAIS.....10

2.2.1 Histórico [8].....10

2.3 CARACTERÍSTICAS E ELEMENTOS [8].....11

2.3.1 Funções de ativação.....12

2.4 ALGORITMO DE RETROPROPAGAÇÃO [8].....13

3 IMAGENS ASTRONÔMICAS.....19

3.1 INTRODUÇÃO.....19

3.2 PLACAS FOTOGRÁFICAS E DETECTORES DE ESTADO SÓLIDO.....21

3.3 DIGITALIZAÇÃO DE IMAGENS ASTRONÔMICAS.....23

3.3.1 Processo de Digitalização [3].....24

3.3.2 Parametrização [3].....24

4 VISUAL NNET.....**26**

4.1 INTRODUÇÃO.....26

4.2 CLASSES.....26

4.2.1 Classe DataHandler.....27

4.2.2 Classe ToolBox.....28

4.2.3 Classe Weight.....29

4.2.4 Classe Neuron.....30

4.2.5 Classe Layer.....31

4.2.6 Classe Network.....32

4.3 INTERFACE GRÁFICA.....35

4.3.1 Interface de Configuração e Funções Especiais.....36

4.3.2 Interface de Treinamento.....37

4.3.3 Interface de Teste.....39

4.3.4 Interface de execução.....39

5 DESENVOLVIMENTO.....**41**

5.1 INTRODUÇÃO.....41

5.2 TESTES DE CORRELAÇÃO.....41

5.3 DEFINIÇÃO DA CONFIGURAÇÃO DA REDE.....45

5.3.1 Topologia.....45

5.3.2 Configuração dos parâmetros de treinamento.....47

5.3.2.1 Regra delta.....47

5.3.2.2 Treinamento por Época.....48

5.3.2.3 Treinamento Batch.....49

5.3.2.4 Conclusão.....50

5.4 TREINAMENTO DA REDE.....51

5.4.1 Processo de Seleção dos Dados.....51

5.4.2 Processo de Treinamento.....54

6 ANÁLISE DOS RESULTADOS.....62

6.1 INTRODUÇÃO.....62

6.2 ANÁLISE.....62

6.3 COMPARAÇÃO.....67

6.4 APLICAÇÃO PRÁTICA.....71

7 CONCLUSÃO.....72

8 BIBLIOGRAFIA.....73

APÊNDICE A – CD.....A

DADOS.....A

DOCUMENTAÇÃO.....A

Artigos.....A

Projeto.....A

PROGRAMAS.....A

Código Fonte.....A

MATLAB.....A

Executável (Windows).....A

1 Introdução

As redes neurais começaram a ser desenvolvidas no década de 50 com inspiração nos modelos nervosos biológicos. Elas têm como principais características o poder de aprendizado e a capacidade de reconhecimento de padrões e de abstração.

Desde seu desenvolvimento, as redes neurais, assim como outras técnicas de inteligência artificial como a lógica Difusa (*Fuzzy Logic*) e mais recentemente os algoritmos genéticos, vêm conquistando espaço na solução de problemas nos mais variados campos de atuação. Dentre alguns destes setores podemos citar:

- Aeroespacial

Piloto automático de alto desempenho, detecção de danos em aeronaves.

- Financeiro

Determinação do grau de confiabilidade que pode ser atribuído a um cliente.

- Defesa

Rastreamento de alvos, discriminação de objetos, reconhecimento facial, identificação de imagens e sinais.

- Industrial

Predição da quantidade e o tipo de gás expelido em um determinado processo industrial [substituindo equipamentos caros utilizados para este propósito no passado].

- Médico

Desenvolvimento de próteses, diagnóstico de doenças.

- Robótica

Controle de trajetórias, sistemas de visão.

- Voz

Reconhecimento e compressão de voz, conversão de texto para voz.

- Telecomunicações

Compressão de dados, tradução em tempo real, enlace Terra-Satélite.

- Transporte

Sistema de rotas.

Porém, muitos ainda vêem as redes neurais com certa descrença. Isto se deve ao fato de muito dinheiro ter sido investido no desenvolvimento de sistemas de inteligência artificial sem um retorno satisfatório. Isto foi especialmente verdadeiro nos anos 70 quando as pesquisas com redes neurais atravessaram seus momentos mais obscuros.

Todavia, grande parte dos insucessos com sistemas de inteligência artificial, principalmente fora do âmbito universitário, podem ser atribuídos a um pobre modelamento do problema a ser abordado, em associação com a falta de estudo adequado sobre as propriedades estruturais dos dados disponíveis.

Apesar das dificuldades, um campo no qual as redes neurais sempre se destacaram foi o de reconhecimento de padrões. Esta valiosa capacidade das redes neurais tem incentivado pesquisas e investimentos, em especial nas áreas de reconhecimento de padrões. Problemas que antes só poderiam ser resolvidos através dos olhos experientes de especialistas ganharam um poderoso aliado computacional que, além de ser indiscutivelmente mais rápido, pode às vezes chegar a ter uma eficiência superior à dos seus equivalentes humanos.

Esta propriedade ímpar das redes neurais faz delas um candidato natural para a questão de separação de estrelas e galáxias em imagens astronômicas. Embora haja uma grande diferença de tamanho entre os dois tipos de objetos (galáxias são agrupamentos de bilhões de estrelas), as grandes distâncias cósmicas fazem com que estes dois objetos acabem por se confundir, principalmente para as galáxias mais distantes.

Porém, são justamente estas galáxias que mais interessam aos astrônomos, pois através do mapeamento de sua distribuição pelo Universo podemos determinar com maior exatidão como foi o início e qual será o destino do cosmos.

É neste contexto que as redes neurais podem se apresentar como uma ferramenta mais eficiente que proporcionará uma classificação mais confiável para os astrônomos. Alguns

programas já foram desenvolvidos neste sentido, sendo um dos mais utilizados entre eles o SExtractor.

Este *software* utilizou-se de milhões de objetos simulados em computador para treinar uma rede neural capaz de identificar e separar estrelas de galáxias [1].

Porém, por ter sido treinado com imagens simuladas de CCD (*Charged Coupled Device*), este programa não é capaz de identificar alguns objetos comuns em placas fotográficas.

E embora os CCDs apresentem grandes vantagens, alguns imageamentos ainda são feitos com placas fotográficas para manter-se a compatibilidade com dados já obtidos, como por exemplo o estudo dos movimentos próprios de estrelas.

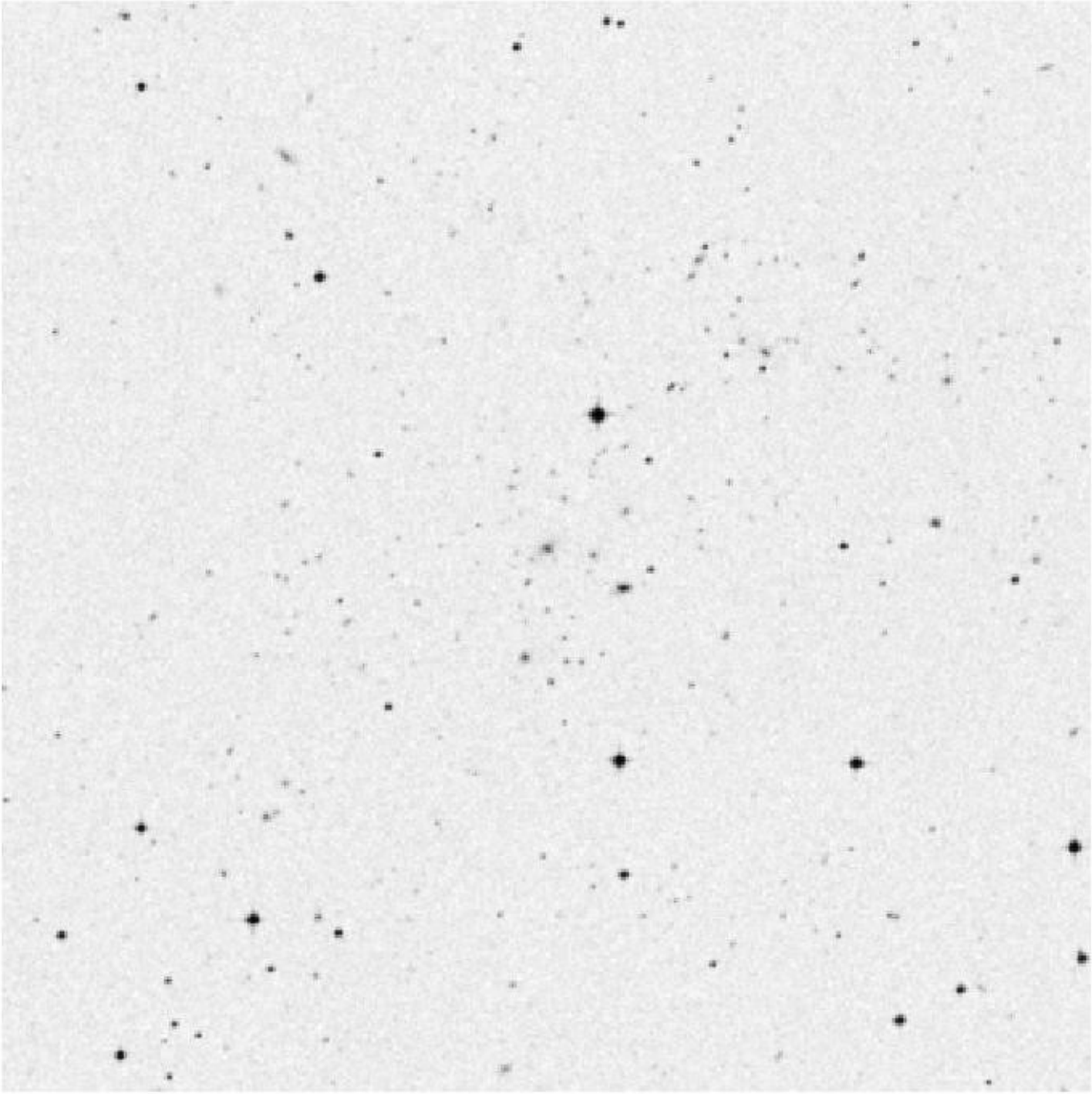
1.1 Objetivo

O objetivo deste trabalho será desenvolver uma rede neural capaz de classificar os objetos identificados após a digitalização de uma placa fotográfica nas categorias de estrelas, galáxias e outros objetos de forma eficiente e confiável.

Mais do que gerar um *software*, este trabalho visa a comprovar a aplicabilidade das redes neurais ao problema de separação de estrelas e galáxias em placas fotográficas de menor resolução do que as imagens de CCD. Pretendemos demonstrar também que as redes neurais são capazes de identificar e agrupar objetos indesejáveis em uma classe separada, como 'riscos' e imagens 'fantasma' de reflexos internos ao telescópio..

Para o treinamento da rede iremos utilizar imagens de uma região do céu que contém dois aglomerados de galáxias, os aglomerados A2534 e A2536. As imagens foram digitalizadas e pré-processadas pelo pacote *FOCAS* (*Faint Objects Classification and Analysis System*).

Figura 1 – Aglomerado A2534 [5]



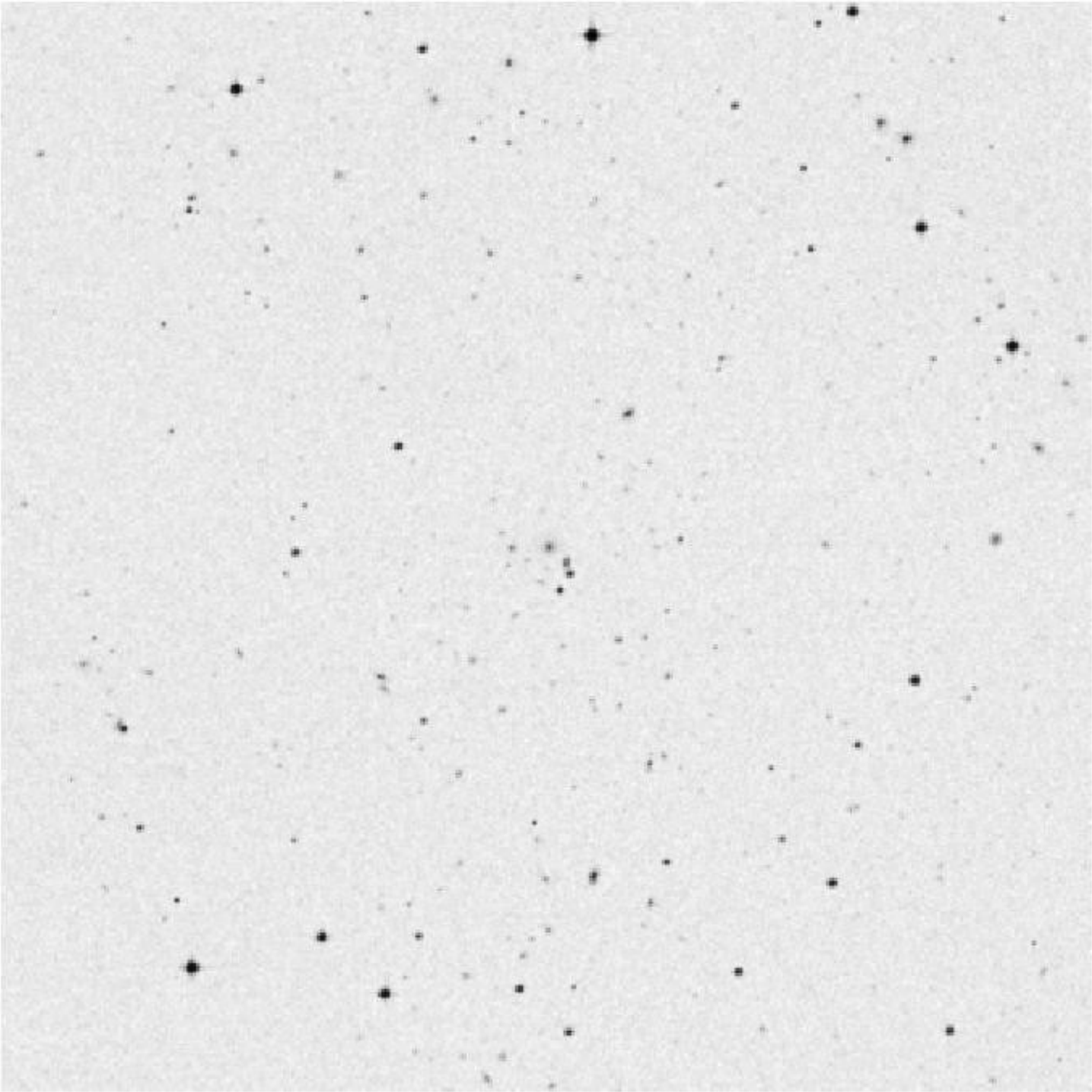


Figura 2 - Aglomerado A2536 [5]

Para aumentar a confiabilidade dos dados usaremos as informações cruzadas de três arquivos.

O primeiro arquivo contém os parâmetros e classificações obtidas ao se processar as imagens digitalizadas utilizando-se o FOCAS. O segundo arquivo contém um subconjunto do primeiro que teve a classificação analisada visualmente por astrônomos do Observatório Nacional / MCT. O terceiro arquivo contém dados obtidos pelo processamento de imagens de CCD de uma área menor da mesma região do céu.

2 Redes Neurais

2.1 Redes Neurais Biológicas [2]

O sistema nervoso controla todas as atividades físicas conscientes e inconscientes. Ele é formado por bilhões de células nervosas, os neurônios, que captam informações vindas do interior e do exterior do corpo do indivíduo.

No nascimento, as conexões entre as células nervosas ainda são imaturas, daí a incapacidade das crianças de realizar muitos atos e comportamentos, como controlar órgãos internos (como a bexiga), lembrar-se de fatos remotos etc. Enquanto o sistema nervoso vai se desenvolvendo, a criança vai desenvolvendo uma ampla variedade de habilidades.

2.1.1 Divisões do Sistema Nervoso

O sistema nervoso se divide em duas partes; o sistema nervoso central (SNC) e o sistema nervoso periférico (SNP). O SNC é a região responsável pela recepção de estímulos, comando e desencadeamento de respostas. O SNP é formado pelas vias que conduzem os estímulos ao SNC ou que levam as ordens emanadas do sistema central para os órgãos efetuadores.

Os processos biológicos pelos quais nos movemos, pensamos, percebemos, aprendemos, lembramos etc. são reflexos das funções cerebrais.

O cérebro é feito de neurônios (ou células nervosas) e células glia. As células nervosas comandam a motricidade, a sensibilidade e a consciência; as células glia sustentam e mantêm vivos os neurônios. O cérebro integra informação sensorial e dirige respostas motoras.

Através de uma proeminente ranhura chamada fissura longitudinal, o cérebro é dividido em duas metades chamadas hemisférios. Na base desta fissura encontra-se um espesso feixe de fibras nervosas, chamado corpo caloso, o qual fornece um elo de comunicação entre os hemisférios. O hemisfério esquerdo controla a metade direita do corpo e vice-versa, em razão de um cruzamento de fibras nervosas no bulbo.

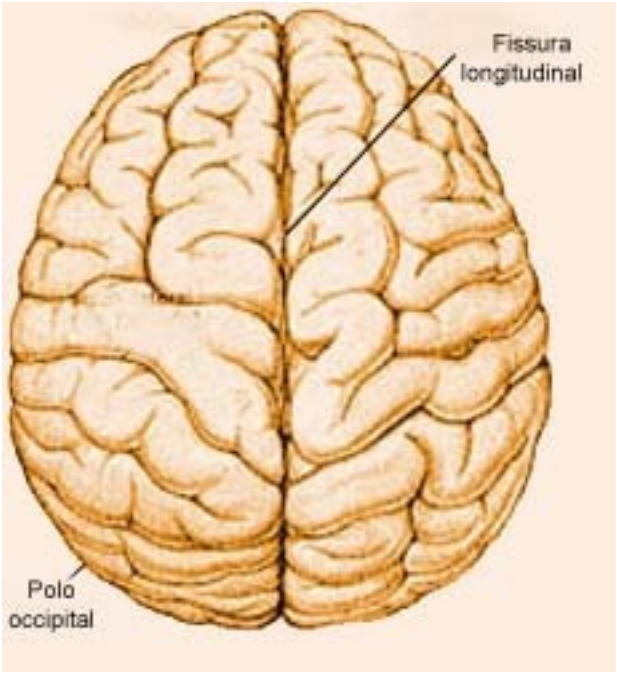


Figura 3 – Hemisférios do cérebro humano

Ainda que os hemisférios direito e esquerdo pareçam ser uma imagem em espelho um do outro, existe uma importante distinção funcional entre eles. Na maioria das pessoas, por exemplo, as áreas que controlam a fala estão localizadas no hemisfério esquerdo, enquanto áreas que governam percepções espaciais residem no hemisfério direito.

2.1.2 Neurônio Biológico

Neurônios são células especializadas. Eles são feitos para receber certas conexões específicas, executar funções apropriadas e passar suas decisões de um evento particular a outros neurônios que estão relacionados com esse evento. Estas especializações incluem uma membrana celular, que é especializada para transportar sinais nervosos como pulsos eletroquímicos; o dendrito, (do grego dendron, ou árvore) que recebe e libera os sinais, o axônio (do grego axoon, ou eixo), o "cabo" condutor de sinais, e pontos de contatos sinápticos, onde a informação pode ser passada de uma célula a outra.

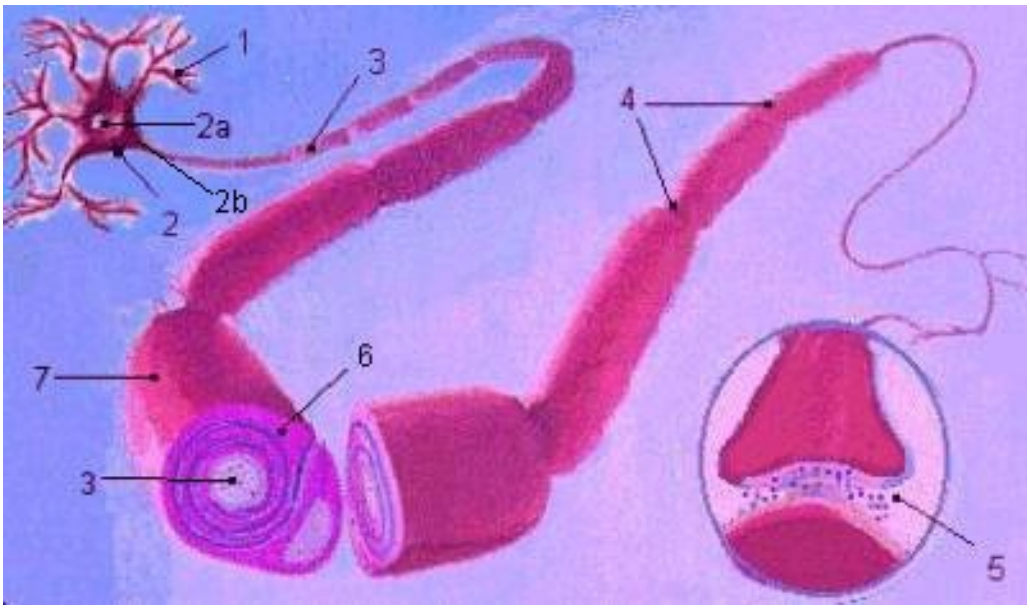


Figura 4 – Neurônio Biológico

Neurônios recebem sinais nervosos de axônios de outros neurônios. A maioria dos sinais é liberada aos dendritos (1). Os sinais gerados por um neurônio são enviados através do corpo celular (2), que contém o núcleo (2a), o "armazém" de informações genéticas. Axônios (3) são as principais unidades condutoras do neurônio. O cone axonal (2b) é a região na qual os sinais das células são iniciados. Células de Schwann (6), as quais não são partes da célula nervosa, mas um dos tipos das células glia, exercem a importante função de isolar neurônios por envolver as regiões membranosas ao redor do axônio formando a bainha de mielina (7), uma substância gordurosa que ajuda os axônios a transmitirem mensagens mais rapidamente. A mielina é quebrada em vários pontos pelos nodos of Ranvier (4), de forma que em uma secção transversal o neurônio se parece como um cordão de salsichas. Ramos do axônio de um neurônio (o neurônio pré-sináptico) transmitem sinais a outro neurônio (o neurônio pós-sináptico) através de uma estrutura chamada sinapse (5). Os ramos de um neurônio podem formar sinapses com até 1000 outros neurônios.

2.2 Redes Neurais Artificiais

2.2.1 Histórico [8]

O modelo de redes neurais moderno surgiu no início década de 40 com a publicação do artigo de McCulloch e Pitts[10] que apresentava o trabalho que vinha sendo desenvolvido por um grupo de pesquisadores da Universidade de Chicago. Neste arquivo eles unificaram os estudos de neurofisiologia e de lógica matemática e demonstraram que com um conjunto de modelos matemáticos que representavam os neurônios e com conexões sinápticas, era possível modelar-se qualquer função computável.

Durante a década de 50 foram surgindo as primeiras teorias de sistemas adaptativos e de aprendizagem, e foi no final desta década que foi publicado o artigo sobre o percéptron, que era um novo método de aprendizagem supervisionada. A partir do percéptron várias redes foram desenvolvidas e apareceram as redes neurais em camadas treináveis com múltiplos elementos.

A teoria de redes neurais vinha se desenvolvendo bem até o final dos anos 60, quando em livro Minsky e Papert[11] demonstraram que havia limites para o que os percéptrons de camada única poderiam representar e supuseram erroneamente que não havia nenhuma razão para que estas limitações fossem superadas pela versão de múltiplas camadas.

Como conseqüência destas limitações e devido a outros fatores como a limitação tecnológica da época, os anos 70 presenciaram um resfriamento no interesse pelo desenvolvimento de redes neurais. Nesta época apenas alguns poucos pioneiros na área de engenharia continuaram a devotar o seu tempo ao estudo das redes neurais.

Se os anos 70 podem ser vistos como a era das trevas para as redes neurais, os anos 80 podem ser comparados à renascença. Uma profusão de novos trabalhos e a revisão de velhos conceitos inundaram a comunidade científica, porém a maior contribuição desta época foi provavelmente a consolidação do algoritmo de retropropagação, *backpropagation*, (Rumelhart, Hilton e Williams,1986)[13] que se destacou como o algoritmo mais popular para o treinamento de redes de múltiplas camadas. Este algoritmo foi definido paralelamente em vários trabalhos e sua idéia básica remonta ao trabalho de Werbos[16] em 1974, mas foi apenas com o trabalho de Rumelhart, Hilton e Williams que sua utilização foi proposta para o

aprendizado de máquinas e se mostrou como isto poderia funcionar. O algoritmo de retropropagação será analisado em maiores detalhes em uma seção adiante por ser o algoritmo utilizado para o desenvolvimento deste trabalho.

2.3 Características e Elementos [8]

Redes neurais são modelos matemáticos que utilizam funções lineares para sintetizar funções não lineares. Como seu próprio nome sugere, foram inspiradas na organização das células do cérebro, os neurônios.

As redes neurais artificiais são constituídas de uma ou mais camadas, contendo elementos que receberam, à semelhança das células cerebrais, o nome de neurônios. Estes neurônios são unidades de processamento de informação que utilizam os valores do vetor de entrada para calcular um valor de saída utilizando-se de uma função de ativação não linear.

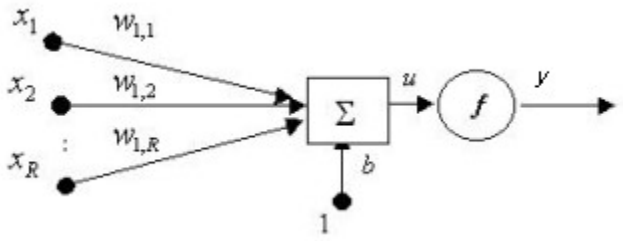


Figura 5 Representação esquemática de um neurônio artificial

Podemos identificar na figura acima três elementos básicos constituintes do neurônio. O primeiro elemento, as sinapses, são os elos de conexão entre os diversos neurônios de uma rede. Associado a cada sinapse existe um valor real chamado de peso ***(w)***, que irá multiplicar o valor de entrada da sinapse para gerar o valor de saída. Sendo assim, um valor x_j que entra na sinapse conectada à camada k será multiplicado pelo peso w_{kj}.

O segundo elemento que pode ser identificado é o somador que soma todos os sinais de entrada ponderados pelas suas respectivas sinapses.

Já o terceiro elemento é a função de ativação ***(f)***, que restringirá a amplitude de saída do neurônio. Os intervalos de saída são tipicamente [0,1] ou [-1,1] .

O neurônio pode ser descrito matematicamente pelas funções abaixo:

$$u_k = \sum_{j=1}^{m} w_{kj} x_j + b_k$$

$$y_k = f(u_k), \qquad \textbf{(1)}$$

onde:

$\boldsymbol{u_k}$ = campo local induzido;

$\boldsymbol{w_{kj}}$ = peso associado à sinapse que conecta o valor da entrada ***j*** com o neurônio ***k***

$\boldsymbol{y_k}$ = valor na saída do neurônio ***k;***

$\boldsymbol{b_k}$ = polarização ou *bias* do neurônio ***k***.

Podemos notar na figura e nas funções acima a presença de uma polarização constante ou *bias*. Sua função é de aumentar ou diminuir o valor de entrada no somador, conforme ele seja positivo ou negativo.

2.3.1 Funções de ativação

Existem vários tipos de funções de ativação que são usadas por neurônios em uma rede neural. Dentro de uma mesma rede, podemos ter várias camadas cada uma com sua própria função de ativação.

Alguns exemplos de funções de ativação utilizadas serão descritas abaixo:

- Função de Limiar. Esta função pode ser descrita como uma função lógica que é regida pela seguinte lei:

$$y_j : \begin{cases} 1, \text{ se } u_j > 0 \\ 0, \text{ se } u_j \leq 0 \end{cases}$$

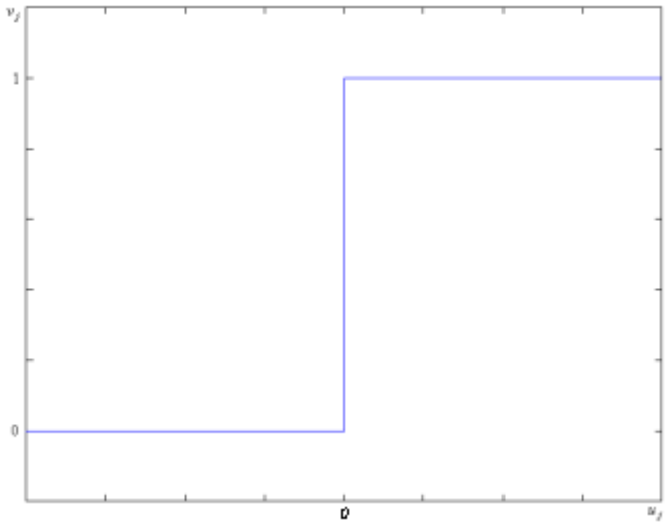


Figura 6 – Função de Limiar

- Funções Sigmóide. Estas funções são as mais utilizadas na construção de redes neurais e seus gráficos apresentam o a forma de "s". A função de ativação utilizada neste trabalho será a função tangente hiperbólica, que é uma função sigmóide compreendida no intervalo [-1,1] .

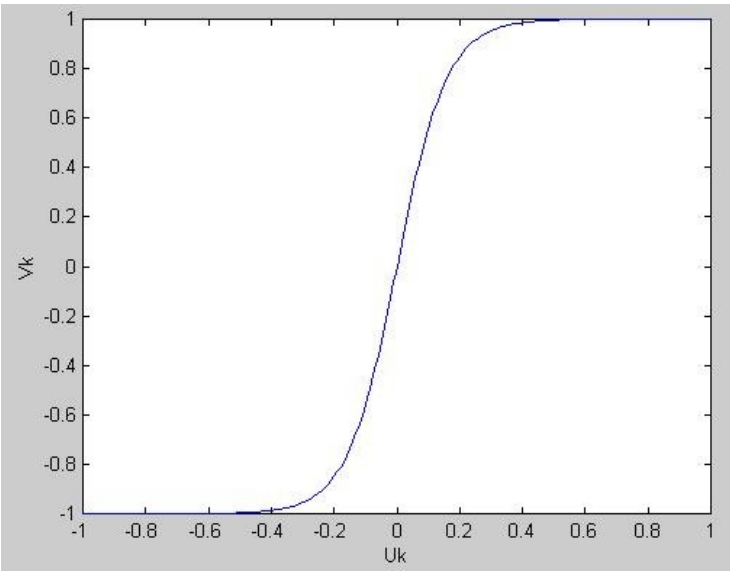


Figura 7 - Função Tangente Hiperbólica

2.4 Algoritmo de Retropropagação [8]

Como vimos anteriormente, existem vários algoritmos de treinamento. Porém, o mais utilizado dentre eles, e o utilizado neste trabalho, é o algoritmo de retropropagação, que veremos a seguir com mais detalhes.

Neste algoritmo, o erro obtido na saída da rede é retropropagado através das camadas da rede, atualizando os pesos que ligam cada uma dessas camadas. O erro na saída da rede é definido como:

$$e_k(n) = a_k(n) - y_k(n), \qquad \textbf{(2)}$$

onde:

$\boldsymbol{a_k(n)}$ = valor-alvo para o neurônio ***j,*** dado o vetor de entrada ***n;***

$\boldsymbol{y_k(n)}$ = saída do neurônio ***j,*** dado o vetor de entrada ***n.***

O valor instantâneo da energia do erro para o neurônio ***k*** $(\varepsilon_k(n))$ é dado por:

$$\varepsilon_k(n) = \frac{1}{2} e_k^2(n). \qquad \textbf{(3)}$$

Já o valor instantâneo para a energia total do erro, $\varepsilon(n)$, é dado pela soma das energias dos neurônios, ou seja:

$$\varepsilon(\mathrm{n}) = \frac{1}{2} \sum_{k=0}^{T} e_k^2(n). \qquad \textbf{(4)}$$

Onde ***T*** é o número total de neurônios na camada de saída da rede.

A energia média do erro total, ε_{med}, é obtida ao fazermos a média das energias instantâneas totais para todos os vetores de entrada da rede, logo é definida como:

$$\varepsilon_{med} = \frac{1}{N} \sum_{n=1}^{N} \varepsilon(n). \qquad \textbf{(5)}$$

Para um dado conjunto de treinamento, ε_{med} representa uma medida do desempenho de aprendizagem da rede. O objetivo do processo de aprendizagem é ajustar os parâmetros livres da rede, isto é, os pesos sinápticos e os níveis de *bias* para minimizar ε_{med}.

Para isso, usamos um método de procura através de um gradiente que aponta na direção do mínimo na superfície de erro.

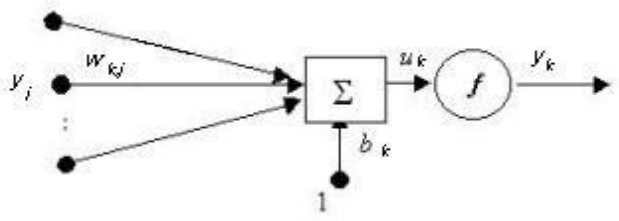


Figura 8 – Modelo do neurônio de saída k

Consideremos, então, a figura 8 acima. Neste caso, o campo local induzido u_k pode ser calculado da seguinte forma:

$$u_k = \sum_{j=0}^{m} w_{kj}(n) y_j(n), \qquad \textbf{(6)}$$

onde m é o número total de entradas (além do *bias*) aplicadas ao neurônio k, y_j é a saída do neurônio j e w_{kj} é o peso sináptico que conecta os neurônios k e j. Assim, o valor na saída do neurônio na n-ésima interação, $y_k(n)$, pode ser descrita como:

$$y_k(n) = f_k(u_k(n)). \qquad \textbf{(7)}$$

O algoritmo de retropropagação aplica uma correção $\Delta w_{kj}(n)$ ao peso sináptico $w_{kj}(n)$ que é proporcional à derivada parcial $\partial \mathcal{E}(n)/\partial w_{kj}$. Usando a regra da cadeia, podemos expressar este gradiente como:

$$\frac{\partial \varepsilon(n)}{\partial w_{kj}(n)} = \frac{\partial \varepsilon(n)}{\partial e_k(n)} \frac{\partial e_k(n)}{\partial y_k(n)} \frac{\partial y_k(n)}{\partial u_k(n)} \frac{\partial u_k(n)}{\partial w_{kj}(n)}. \qquad \textbf{(8)}$$

Diferenciando ambos os lados da equação (4) obtemos:

$$\frac{\partial \varepsilon(n)}{\partial e_k(n)} = e_k(n). \qquad \textbf{(9)}$$

Diferenciando ambos os lados da equação (2) em relação a $y_k(n)$, obtemos:

$$\frac{\partial e_k(n)}{\partial y_k(n)} = -1. \qquad \textbf{(10)}$$

Em seguida, diferenciando a equação (7) em relação a $u_k(n)$, obtemos:

$$\frac{\partial y_k(n)}{\partial u_k(n)} = f_k^{'}(u_k(n)) . \qquad \textbf{(11)}$$

Por último, diferenciamos a equação (6) em relação a w_{kj}, o que resulta em:

$$\frac{\partial u_k(n)}{\partial w_{kj}(n)} = y_j(n) . \qquad \textbf{(12)}$$

Juntando as equações de (8) a (12), teremos, então:

$$\frac{\partial \varepsilon(n)}{\partial w_{kj}(n)} = -e_k(n) f_k^{'}(u_k(n)) y_j(n) . \qquad \textbf{(13)}$$

A correção $\Delta w_{kj}(n)$ é definida pela regra delta:

$$\Delta w_{kj}(n) = -\eta \frac{\partial \varepsilon(n)}{\partial w_{kj}(n)} , \qquad \textbf{(14)}$$

onde η é a taxa de aprendizagem do algoritmo de retropropagação. O sinal negativo na equação 14 indica a descida do gradiente no espaço de pesos, ou seja, a busca pela mudança de pesos que diminua o valor de $\mathscr{E}(n)$. Substituindo-se a equação (13) na equação (14) resulta:

$$\Delta w_{kj}(n) = \eta \delta_k(n) y_j(n) , \qquad \textbf{(15)}$$

onde o gradiente local $\delta_k(n)$ é definido por:

$$\delta_k(n) = -\frac{\partial \varepsilon(n)}{\partial u_k(n)}$$

$$\delta_k(n) = -\frac{\partial \varepsilon(n)}{\partial e_k(n)} \frac{\partial e_k(n)}{\partial y_k(n)} \frac{\partial y_k(n)}{\partial u_k(n)}$$

$$\delta_k(n) = e_k(n) f_k^{'}(u_k(n)) . \qquad \textbf{(16)}$$

Como podemos ver nas equações (15) e (16), um elemento essencial para o ajuste de peso $\Delta w_{kj}(n)$ é o erro $e_k(n)$ na saída do neurônio. Isto não representa problema algum quando estamos nos referindo aos neurônios presentes na última camada da rede, que possuem um valor alvo com o qual podem ser comparados. Mas, como fazer para calcular este erro para um neurônio da camada oculta da rede?

Embora estes neurônios não tenham um alvo definido, eles são também responsáveis pelo erro final, e por isso devem ter seus pesos atualizados. É neste ponto em que o algoritmo de retropropagação entra em ação.

O sinal de erro de um neurônio oculto ***j*** deve ser determinado com base nos sinais de erro de todos os neurônios a ele conectados. Vejamos como fica o valor de $\Delta w_{ji}(n)$, que é a atualização do peso conectado aos neurônios ***i*** e ***j*** nas camadas ocultas da rede.

A equação (16) do gradiente local $\delta_j(n)$ pode ser reescrita da seguinte forma:

$$\delta_j(n) = -\frac{\partial \varepsilon(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial u_j(n)}. \qquad \textbf{(17)}$$

Substituindo-se, agora, a equação (11) na equação (17) obtemos:

$$\delta_j(n) = -\frac{\partial \varepsilon(n)}{\partial y_j(n)} f_j'(u_j(n)). \qquad \textbf{(18)}$$

Para calcular a derivada parcial $\partial \mathscr{E}(n)/\partial y_j$ podemos diferenciar a equação (4), obtendo o seguinte resultado:

$$\frac{\partial \varepsilon(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial y_j}. \qquad \textbf{(19)}$$

Usando a regra da cadeia para a derivada parcial $\partial e_k(n)/\partial y_j(n)$, podemos reescrever a equação (19) da seguinte forma:

$$\frac{\partial \varepsilon(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial u_k(n)} \frac{\partial u_k(n)}{\partial y_j(n)}. \qquad \textbf{(20)}$$

Podemos, ainda, reescrever a equação (2) da seguinte forma:

$$e_k(n) = a_k(n) - f_k(u_k(n)). \qquad \textbf{(21)}$$

Sendo assim:

$$\frac{\partial e_k(n)}{\partial u_k(n)} = -f_k'(u_k(n)). \qquad \textbf{(22)}$$

Se diferenciarmos a equação (6) em relação a $y_j(n)$ teremos:

$$\frac{\partial u_k(n)}{\partial y_j(n)} = w_{kj}(n). \qquad \textbf{(23)}$$

Substituindo as equações (22) e (23) na equação (20), obteremos:

$$\frac{\partial \varepsilon(n)}{\partial y_j(n)} = -\sum_k e_k f_k^{'}(u_k(n)) w_{kj}(n), \qquad \textbf{(24)}$$

que, com base na equação (16), pode ainda ser reescrita como:

$$\frac{\partial \varepsilon(n)}{\partial y_j(n)} = -\sum_k \delta_k(n) w_{kj}(n). \qquad \textbf{(25)}$$

Por fim, substituindo a equação (25) na equação (18) teremos:

$$\delta_j(n) = \sum_k \delta_k(n) w_{kj}(n) f_j^{'}(u_j(n)). \qquad \textbf{(26)}$$

Sendo assim, a atualização $\Delta w_{ji}(n)$ pode ser expressa da seguinte forma:

$$\Delta w_{ji}(n) = \eta \sum_k \delta_k(n) w_{kj}(n) f_j^{'}(u_j(n)) y_i(n). \qquad \textbf{(27)}$$

Note que desta forma conseguimos definir o valor de $\Delta w_{ji}(n)$ em função do somatório dos gradientes locais da camada de saída, que já estão definidos, e não mais em relação ao gradiente local $\delta_j(n)$ do neurônio ***j***.

Este processo deve ser repetido a fim de se atualizar todos os pesos da camada em que se encontra o neurônio ***j*** e todas as camadas que o antecedem, ficando, assim, definido o treinamento para todos os pesos da rede.

3 Imagens Astronômicas

3.1 Introdução

A questão da separação e classificação de objetos celestes presentes em uma imagem é muito delicada e essencial em astronomia. Através do levantamento da distribuição espacial das galáxias pelo cosmo, os astrônomos são capazes de tirar informações sobre a origem, o estado atual e o destino de nosso universo.

Estrelas são, muito essencialmente, imensas esferas de gás formadas dentro de nuvens com dezenas de anos-luz de extensão, como pode ser visto na figura 9, que, devido à sua grande densidade central, dão início a um processo termonuclear que transforma elementos mais simples em outros mais complexos através do processo de fusão nuclear. No nosso Sol, que é uma estrela de pequeno porte da seqüência principal, este processo ocorre em sua forma mais básica, transformando átomos de Hidrogênio em átomos de Hélio, e liberando energia no processo, em acordo com as equações deduzidas por Albert Einstein.

Figura 9 - Pilares da criação na nebulosa da Águia[6]

Galáxias, por sua vez, são grandes conjuntos de estrelas, contendo em geral algumas centenas de bilhões de astros espalhados por regiões de dezenas de milhares de anos-luz de diâmetro. As galáxias possuem várias formas, podendo ser elípticas, irregulares ou espirais, este último sendo o tipo ao qual pertence a galáxia na qual nosso Sistema Solar se encontra, a Via Láctea.

A imagem abaixo mostra como seria vista a Via Láctea por um observador externo.

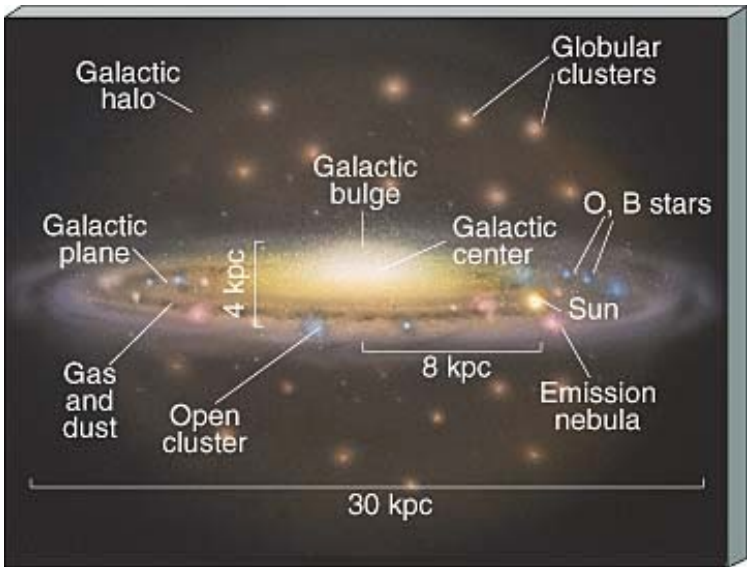


Figura 10 – Aparência da Via Láctea para um observador externo[6]

A Via Láctea faz parte de uma estrutura ainda maior, que é conhecida como o Grupo Local, do qual também faz parte a galáxia de Andrômeda, que se encontra em curso de colisão com a nossa própria galáxia.

O Grupo Local é um exemplo de um aglomerado de galáxias que se mantêm ligadas gravitacionalmente.

O Grupo Local, é constituído por aproximadamente 50 galáxias. O Aglomerado de Virgo, "próximo" ao Grupo Local, por sua vez, possui cerca de 2500 objetos. Os astrônomos acreditam que os aglomerados de galáxias formem superaglomerados, as maiores estruturas encontradas no Universo, cujos membros ainda se encontram em processo de relaxamento dinâmico.

Com uma diferença tão grande de ordem de grandeza entre uma estrela e uma galáxia, pode parecer não ser tão difícil identificar estes objetos em uma imagem. O problema é que a maioria das galáxias encontram-se a bilhões de anos-luz de distância, e a esta distância seu brilho fica tão reduzido que elas passam a ser facilmente confundidas com estrelas presentes na imagem, que por sua vez encontram-se dentro de nossa galáxia.

Para solucionar este problema, os astrônomos lançam mão da experiência e de programas de computador que sejam capazes de identificar o grande número de objetos nas mais variadas magnitudes. O problema se torna especialmente complexo para maiores magnitudes (menor intensidade luminosa), onde os programas de computador têm um desempenho muito aquém do esperado, forçando a uma reclassificação manual.

3.2 Placas Fotográficas e Detectores de Estado Sólido

Até meados da década de 70, estas imagens eram feitas utilizando-se apenas filmes fotográficos com emulsões especiais que ficavam expostas por períodos de tempo da ordem de 60 minutos, para que fosse possível coletar fótons suficientes dos objetos mais distantes.

São usados filtros para coletar-se energia em algum intervalo específico do espectro eletromagnético. Assim, através de medidas em vários filtros (chamados de cores), podemos, por exemplo, ajudar na classificação de alguns objetos astronômicos.

Ainda hoje as placas fotográficas são muito utilizadas por motivo de limitações financeiras, pela necessidade de se manter a compatibilidade de dados, mas principalmente porque uma placa é capaz de abranger amplas regiões do céu.

Porém, em 1974 a NASA começou a utilizar uma tecnologia desenvolvida pelos laboratórios Bell, o CCD (*Charged Coupled Device*).

O CCD é um detector de estado sólido que converte a luz em sinais elétrons que são armazenados e acumulados, até serem lidos e convertidos em sinal elétrico. Os CCDs utilizados nas câmeras de vídeo caseiras requerem muito menos energia, são muito mais sensíveis à luz e muito mais leves do que os tubos a vácuo que eram utilizados [7].

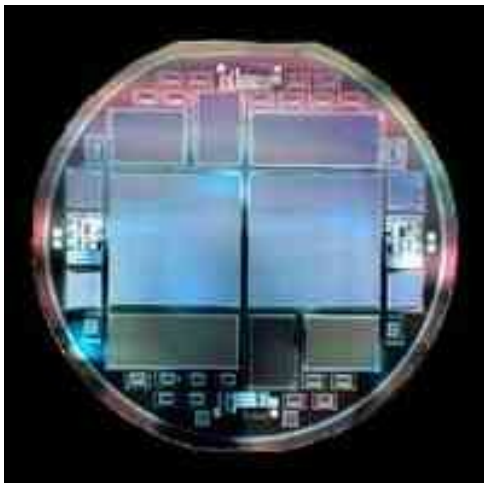


Figura 11 – Matriz de CCDs utilizada em astronomia[12]

Imagens obtidas com CCD são mais práticas, e a fotografia está sendo gradualmente substituída. Por este motivo, os CCDs são cada dia mais utilizados em pesquisas astronômicas e estão presentes em diversas sondas espaciais, como a Galileu, e no telescópio espacial Hubble.

Abaixo podemos ver uma comparação entre imagens digitalizadas de placas fotográficas (esquerda) e obtidas por CCD (direita). A resolução diferente se deve ao fato de as imagens terem sido obtidas em telescópios diferentes.

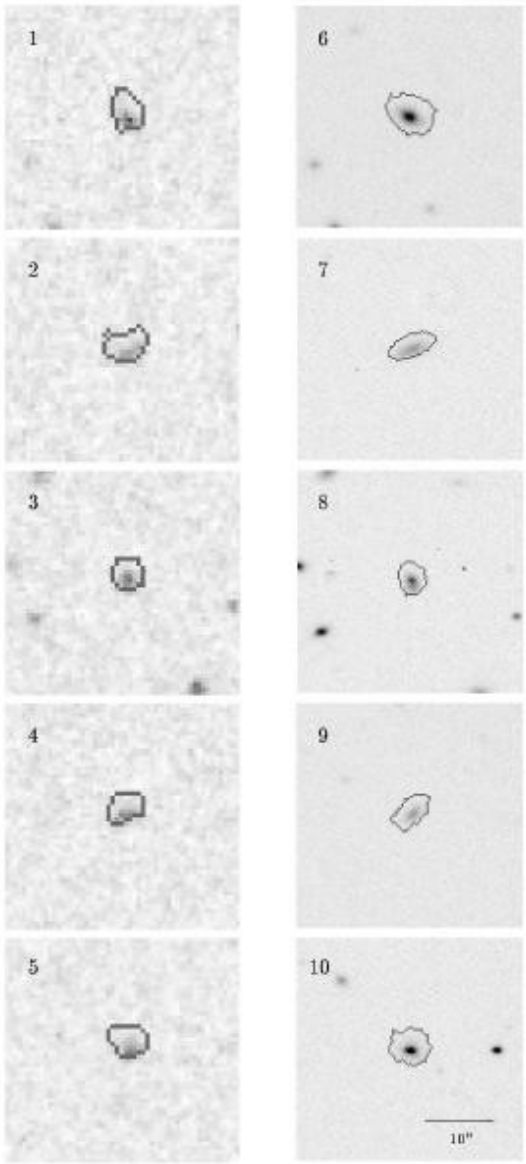


Figura 12- Comparação entre imagens [4]

Os contornos são isofotas limites determinadas pelo FOCAS e correspondem aos contornos dos objetos com intensidade de luz 2,5 vezes o ruído do sinal do céu. Estes objetos foram classificados erroneamente como estrelas quando a análise foi feita baseada nas imagens da placa fotográfica. [4]

3.3 Digitalização de Imagens Astronômicas

Com a evolução dos instrumentos de observação e com o aumento do número de objetos observados, tornou-se impraticável para os astrônomos a análise convencional das imagens obtidas.

Porém, quando as imagens estão em placas fotográficas, estas precisam ser digitalizadas para que os objetos nelas contidos possam ser analisados computacionalmente. Devido à grande complexidade deste processo, optou-se neste trabalho por utilizar dados previamente digitalizados e pré-processados pelo pacote *FOCAS*. Veremos a seguir um pouco mais sobre este processo.

3.3.1 Processo de Digitalização [3]

As imagens utilizadas neste trabalho foram digitalizadas utilizando-se o microdensitômetro PDS 1010A do Observatório Nacional. Este aparelho mede a densidade de uma emulsão fotográfica utilizando um feixe de luz que passa através da placa fotográfica e é captado por uma fotomultiplicadora.

O PDS é composto por três sistemas: o sistema óptico, o de movimentação do carro e a régua. O sistema óptico é formado por duas partes, sendo a parte inferior a que emite o feixe luminoso que fará a varredura da placa. Já a parte superior é responsável por receber o feixe e transmití-lo à fotomultiplicadora.

Para que uma imagem seja digitalizada é necessário que se faça primeiro uma preparação do PDS. Depois é feita uma calibração das escalas e só então é colocada a placa fotográfica para a localização do campo que se pretende digitalizar.

Para o processo de calibração do aparelho é necessário a definição de vários parâmetros, entre os quais estão o tamanho total da imagem e o tamanho dos píxeis a serem utilizados.

Existe um compromisso na hora de se determinar o tamanho do píxel. Quanto menor ele for, maior será a resolução da imagem digitalizada e assim, maior a capacidade de identificar detalhes do objeto. Por outro lado maior será a sensibilidade da placa ao ruído.

3.3.2 Parametrização [3]

Depois de digitalizada a imagem, o FOCAS extrai vários parâmetros que são utilizados para fazer a classificação do objeto.

Como veremos a seguir, foram aplicados os testes de relevância e anti-relevância para a determinação daqueles parâmetros que mais influenciavam na classificação.

Três destes parâmetros são tipos diferentes de magnitudes.

Segundo o teste de relevância, os parâmetros mais importantes são:

- ***Magnitude Isofotal (mi) :*** Esta magnitude é calculada a partir da luminosidade contida na isofota de detecção (calibrada pelo ponto de magnitude zero).
- ***Magnitude Central (mc):*** É o pico de magnitude. É dada pela luminosidade dos 9 píxeis mais brilhantes no centro da imagem.
- ***Magnitude de Abertura (ma):*** Calculada dentro de uma área circular fixa de 69 píxeis (equivalente a um raio de 5 píxeis).
- ***Fração de Resolução (f):*** É a fração de uma componente difusa gerada pela presença de nebulosidade no objeto.

4 Visual NNet

4.1 Introdução

O Visual NNet foi por nós desenvolvido especialmente para o problema de separação de estrelas e galáxias, porém com um pouco de retrabalho ele pode ser adaptado para outras aplicações. Ele foi desenvolvido na linguagem de programação C++ e é baseado em uma arquitetura orientada a objetos.

A rede escolhida para este abordar o problema de separação de estrelas e galáxias foi uma rede do tipo *Back Propagation*. Essa escolha se deve ao fato das classes serem bem definidas e invariantes no tempo. Com isso será possível obter-se, em teoria, resultados mais precisos.

4.2 Classes

Nesta seção iremos explorar as diversas classes que constituem o programa Visual Nnet, detalhando suas principais funções e variáveis. As primeiras classes que serão exploradas são as de uso geral e que não possuem conexão direta com redes neurais.

A classe *DataHandler* é a responsável pelo tratamento inicial dos dados e carregamento destes na rede. Já a classe *ToolBox* possuí funções de uso geral que são necessárias em diversas outras classes.

As seguintes classes são representantes dos diversos elementos que constituem uma rede. Elas estão organizadas de forma que a classe seguinte é na verdade um conjunto de objetos da classe anterior. Sendo assim começaremos analisando as classes *Neuron* e *Weight,*

que representam os neurônios e os pesos da rede, respectivamente. Depois veremos a classe *Layer,* que é um conjunto de objetos da classe *Neuron* e representa as várias camadas da rede. Os neurônios das camadas se interligam uns aos outros através de objetos da classe *Weight*.

Por último, iremos analisar a classe *Network,* que é um conjunto de objetos da classe *Layer* e que representa a rede em si.

4.2.1 Classe DataHandler

O objetivo desta classe é o de carregar os dados dentro do programa e prepará-los para serem usados na rede. Entre suas funções mais importantes estão *loadData(), parseFile()* e *normalize().*

Abaixo encontramos uma lista de todas as funções pertencentes a essa classe com uma breve descrição sobre suas funcionalidades.

Funções Públicas:

- *getPar(int setNum, int index)* – Retorna o parâmetro associado ao índice *index* do vetor de parâmetros dado pela variável *setNum.*
- *getMiMaxDifference()* – Retorna a maior diferença entre os valores de magnitudes isofotais e sua média.
- *getMiMeanValue()* – Retorna o valor médio da magnitude isofotal.
- *checkMiOK()* - Retorna verdadeiro se a posição um na magnitude isofotal está definida na memória.
- *getMiValue(int index)* – Retorna o valor da magnitude isofotal correspondente à posição de memória dada pela variável *index.*
- *getTestBegin()* - Retornam o índice correspondente ao primeiro conjunto de dados do grupo de teste.
- *getTrainLimiar()* - Retorna o índice correspondente ao último conjunto de dados do grupo de treino.
- *getTotalEvent(int index)* - Retorna o número total de eventos associados à posição de memória determinada pela variável *index*.
- *getResElemNum()* - Retorna número de elementos presentes neste conjunto de resultados
- *getResultSet(int index)* - Retorna o conjunto de resultados associados à posição de memória determinada pela variável *index.*

- *getParSet(int index)* – Retorna o conjunto de dados associados à posição de memória determinada pela variável *index*.
- *loadData(CString FilePath,CString &exclLine, int mi, int ir2)* - Carrega na memória os dados presentes no arquivo determinado pela variável *filePath*, excluindo os parâmetros presente na variável *excLine* e carregando na memória a posição da magnitude isofotal e do "segundo momento de raio de intensidade ponderada" no conjunto de parâmetros dado pela variável *mi.*

Funções Privadas:

- *updateMiPos(FILE *fl)* - Atualiza a posição da magnitude isofotal com base nos parâmetros a serem excluídos.
- *parseLine(CString line)* - Carrega na memória quais os parâmetros que devem ser excluídos.
- *normalize()* – Normaliza o conjunto de parâmetros.
- *parseFile(FILE *fl, int pfMi, int pfIr2)* - Dentro desta função, um arquivo de texto é convertido em dados e carregado no conjunto de parâmetros, assim como os resultados são interpretados e carregados no conjunto de resultados. Ao mesmo tempo, um outro conjunto de dados é preparado para a rotina de execução da rede.
- *countSets(FILE *fl)* - Retorna o número de pares entrada-saída presentes no arquivo apontado pela variável *fl.*
- *countPar(FILE *fl)* - Retorna o número de parâmetros presentes nas linhas do arquivo de dados.

4.2.2 Classe ToolBox

Esta classe foi desenvolvida para servir às outras classes. Ela uma classe estática, o que significa que não precisa de objeto para ser invocada. Assim, qualquer outra classe pode chamar uma função na classe *ToolBox* sem que para isso tenha que associar-lhe um objeto. As principais funções desta classe são *castBallot()*, *getRandom()* e *atod().*

Abaixo encontramos uma lista de todas as funções pertencentes a essa classe com uma breve descrição sobre suas funcionalidades.

Funções Públicas:

- *closeToolBox()* – Finalizador da classe *ToolBox.*
- *saveVSse(double vSse)* - Salva o erro quadrático do conjunto de validação .
- *sigmoid(double value)* - calcula a função sigmóide tendo como entrada a variável *value.*
- *castBallot()* – retorna um número sorteado de um intervalo de valores predeterminado. O número sorteado não será retornado novamente até que a votação seja reiniciada.
- *resetBallot()* - reinicia a votação.
- *startBallot(int begin, int end)* - inicia um sorteio, determinando o início do intervalo de votação através da variável *begin* e o término deste intervalo através da variável *end.*
- *getRandom()* - retornam um número aleatório que varia entre –0,3 e +0,3.
- *saveSse(double sse)* - Salva o erro quadrático do conjunto de treinamento .
- *closeSse()* – Fecha o arquivo de erros quadráticos.
- *openSse()* – Abre o arquivo de erros quadráticos.
- *atod(CString numString)* – Converte uma *string* de texto em um número de formato *double*.

Funções Privadas:

- *initalize()* - sendo a única função privada desta classe, ela é chamada automaticamente quando uma função da classe *ToolBox* é chamada pela primeira vez no programa ou quando é chamada após a função *closeToolBox().*

4.2.3 Classe Weight

Esta classe representa os pesos das conexões sinápticas entre neurônios de duas camadas diferentes. Ela também é responsável pela atualização desses pesos durante o processo de treinamento da rede. As principais funções desta classe são *update()*, *feedBack()* e *execute().*

Abaixo encontramos uma lista de todas as funções pertencentes a essa classe com uma breve descrição sobre suas funcionalidade.

Funções Públicas:

- *getWeightValue()* - Retorna o valor do peso associado a este objeto.
- *undo()* - Desfaz a última alteração feita no valor do peso, restaurando o valor anterior.

- *resetDelta()* - Zera o valor de *delta* associado a este peso.
- *update(double learningRate, double momentum, bool store)* - Atualiza o valor do peso com base nas variáveis de entrada *learningRate, momentum e store*, durante as rotinas de treinamento.
- *feedBack()* - Multiplica o valor do neurônio de saída pelo valor do peso, retro-propagando, assim, o erro pela rede.
- *execute()* - Multiplica o valor do neurônio de entrada pelo peso e coloca este valor no. neurônio de saída.
- *initialize()* – Inicializa o peso com um valor aleatório.
- *initialize(double value)* - Inicializa o peso utilizando o valor da variável *value.*
- *connectOutput(neuron* cell)* - conecta o peso ao neurônio de saída que será um objeto da classe *neuron.*
- *connectInput(neuron* cell)* - conecta o peso ao neurônio de entrada que será um objeto da classe de *neuron.*

Esta classe não possui funções privadas.

4.2.4 Classe Neuron

Esta classe representa os neurônios, que são os elementos básicos de uma rede neural. Nesta classe, é onde se encontra definida a função de ativação. A função mais importante desta classe é a função *execute().*

Funções Públicas:

- *inputBias(double value)* - Carrega o valor do *bias* com valor da variável *value.*
- *getBias()* - Retoma o valor do *bias* associado a este neurônio.
- *undo()* – Desfaz a última alteração feita no valor do *bias*, restaurando o valor anterior.
- *updateBias(double lr, bool store)* - Atualiza o valor do *bias* em função das variáveis de entrada *lr* e *store.*
- *calculateDelta()* - Calcula o *delta* deste neurônio.
- *getDeltaValue()* - Retorna o valor do *delta* associado a este neurônio.
- *inputDeltaValue(double value)* - Carrega a variável *delta* com o valor da variável de entrada *value.*

- *execute()* - calcula o valor na saída do neurônio, sendo que este valor é o resultado da função de ativação, tendo como entrada o valor de entrada do neurônio.
- *inputData(const double &value)* - carrega o valor de entrada do neurônio com o valor da variável *value.*
- *getValue()* - retorna o valor de saída deste neurônio.
- *initialize(int inputs, bool isBiased)* - Inicializa o neurônio, dando como parâmetros de entrada a quantidade de entradas que ele possui, dado pelo valor da variável *inputs*, e dizendo se este neurônio possui bias, informação dada pela variável *isBiased* .
- *setFunction(int n)* - determina a função de ativação deste neurônio com base na variável de entrada ***n.***
- *neuron()* - Construtor da classe *neuron,* que tem como função de ativação padrão a tangente hiperbólica.
- *neuron(int n)* - Construtor da classe neuron, que define a função de ativação com base na variável de entrada ***n.***

Funções Privadas:

- *tanHyp()* - Calcula o valor da tangente hiperbólica com base no valor de entrada do neurônio.

4.2.5 Classe Layer

Esta classe é um conjunto de objetos da classe *neuron* e constitui as camadas da rede. Como visto anteriormente, estas camadas podem ser camadas de entrada, camadas escondidas ou ainda camada de saída.

A função mais importante desta classe é a função *execute().*

Funções Públicas:

- *inputBias(double value,int index)* - Carrega o valor do *bias* do neurônio com o índice dado pela variável *index*, com o valor da variável *value.*
- *getCellBias(int index)* - Retorna o valor do *bias* do neurônio indicado pelo índice *index.*
- *updateNeuronBias(int index, double learningRate, bool accumulate)* - comanda o neurônio indicado pela variável i*ndex* a atualizar o seu *bias* com base no valor das variáveis *learningRate* e *accumulate.*

- *setBias(bool value)* - Através desta função determina-se se a camada em questão possuirá *bias*, com base no valor da variável *value.*
- *isBiased()* - Retorna verdadeiro se esta camada possuí *bias*, e retorna falso em caso contrário.
- *calculateDelta(int index)* - Comanda o neurônio indicado pelo índice *index* a calcular o valor de *delta* .
- *inputError(double value, int index)* - carrega a saída do neurônio indicado pelo índice o *index* com o valor contido na variável *value.*
- *getNeuronValue(int index)* - retorna o valor na saída do neurônio indicado pelo índice *index.*
- *execute(int index)* - comanda o neurônio indicado pelo índice *index* a executar a função *execute().*
- *inputData(double value,int index)* - Carrega a entrada do neurônio indicado pelo índice *Index* com o valor contido na variável *value.*
- *getNeuron(int index)* - Retorna um ponteiro que aponta para o objeto da classe *neuron* indicado pelo índice *index.*
- *getLength()* - Retorna o número de neurônios presentes nesta camada.
- *initialize(int cellNum, int inputNum, bool bias)* - Inicializa a camada com base nos parâmetros *cellNum*, *inputNum* e *bias* que representam o número de células na camada, o número de entradas de cada célula e se esta camada contém *bias*, o respectivamente.

Esta classe não possui funções privadas.

4.2.6 Classe Network

Esta classe representa a rede em si. Ela é composta por dois ou mais objetos da classe *layer.* É com a classe Network que o usuário interage diretamente, pois é nela que são definidas as rotinas este de treinamento, testes e a utilização da rede. É nesta classe também onde estão definidas as funções responsáveis pela avaliação do treinamento e do teste da rede.

As principais funções desta classe são as funções *Train(), Test() e Run().*

Funções Públicas:

- *getNetworkOutput()* – Retorna um vetor com os valores dos neurônios de saída da rede.

- *Run(CString fileName)* – Analisa os dados carregados na memória pela função *loadData* e salva o resultado no arquivo indicado pela variável *fileName* .
- *Dump(int choice)* – Analisa os dados carregados na memória associados com a variável *choice* e salva os valores dos neurônios de saída da rede nos arquivos *out1.dat*, *out2.dat* e *out3.dat.*
- *antirelevanceTest()* – Realiza o teste de anti-relevância.
- *relevanceTest()* – Realiza o teste de relevância.
- *reset()* – Zera a matriz de dados, a matriz de pesos e a matriz de camadas, desinicializando, assim, a rede.
- *save(CString weightFile)* - Salva a matriz de pesos em um arquivo.
- *setSseEdit(CEdit* handler)* – Cria um *Handler* para o campo *Summed Squared Error* na interface gráfica.
- *Train(int trnStps, double lr, double scalingFactor, double epoch, int batch, double momentum, bool isBatch)* – Rotina de treinamento da rede. O treino é realizado utilizando-se os parâmetros dados pelas variáveis de entrada que são o número de passos de treinamento (*trnStps*), o fator de escalonamento do passo de treinamento (*scailingFactor*), a taxa de aprendizado (*lr*), o número de épocas de treinamento (*epoch*), o número de caixas de treinamento (*batch*), o momento (*momentum*) e se o treinamento utilizará o método de atualização por "caixas" (*isBatch*).
- *Test(CString &gOK, CString &sOK, CString &oOK, CString &gError, CString &sError, CString &oError, CString &lessThenJ, CString &moreThenJ)* - Função responsável pela realização dos testes na rede e por passar os resultados para interface gráfica através das variáveis que representam o campo de porcentagem de acerto de galáxias (*gOK*), o campo de porcentagem de acerto de estrelas (*sOK*), o campo de porcentagem de acerto de outros objetos (*oOK*), o campo de porcentagem de erro de galáxias (*gError*), o campo de porcentagem de erro de estrelas (*sError*), o campo de porcentagem de erro de outros objetos (*oError*), além de receber os limites de treinamento através das variáveis de limite inferior de magnitude (*lessThenJ*) e limite superior de magnitude (*moreThenJ*).
- *isReady()* - Retorna verdadeiro se a rede tiver sido corretamente inicializada.

- *initialize(int *layers, bool *layerBias)* - Inicializa rede utilizando o vetor de entrada *layers* para determinar a quantidade de camadas terá a rede e quantos neurônios existem em cada uma dessas camadas, e utiliza o vetor *layerBias* para determinar quais destas camadas terão *bias*.
- *initialize(FILE* cf)* - Inicializa rede utilizando o arquivo de configuração *cf.*

Funções Privadas:

- *checkBias(double* biasBuf,int bufSize)* – Retorna verdadeiro se a rede estiver usando *bias*, falso em caso contrário.
- *readValueLine(FILE* confFile ,double* Buf, int neuronNum)* – Lê uma linha do arquivo de configuração.
- *readFromFile(FILE* theFile,char* element)* – Lê um caractere do arquivo de configuração.
- *restoreValues()* – Restaura os valores de peso e *bias* do passo anterior. (*Não está em uso*).
- *calculateError(double &starMisc, double &galaxyMisc, double &otherMisc)* – Calcula o erro de classificação para as classes. O erro é determinado pelo número de objetos errados em relação ao número total de objetos classificados como pertencentes àquela classe.
- *closeFiles()* – Fecha todos os arquivos que registram o desempenho e evolução da rede.
- *openFiles()* – Abre todos os arquivos que registram o desempenho e evolução da rede.
- *avaliateTraining(int stepNumber)* – Avalia o desempenho da rede.
- *backPropagate(double *result,bool accumulate)* – É a função responsável pela retropropagação do erro pela rede. A variável *accumulate* indica se as variações devem atualizar os pesos imediatamente ou se devem ser acumuladas.
- *calculatePercentual(double &star, int sTotal, double &galaxy, int gTotal, double &other, int oTotal)* – Calcula o percentual de objetos de cada classe que foram corretamente classificados em relação ao número total de objetos efetivamente pertencentes àquela classe.
- *checkResult(double* result)* – Verifica se a rede classificou corretamente o objeto. Esta função é chamada para os elementos do conjunto de teste e pela função *Run()*.

- *checkResult(double* result, double &valSse)* – Verifica se a rede classificou corretamente o objeto. Esta função é chamada para os elementos do conjunto de verificação, e o erro quadrático médio é salvo na variável *valSse.*
- *propagate(double* parSet)* – Propaga os dados do vetor de parâmetros *parSet* pela rede.

4.3 Interface Gráfica

O VisualNNet foi desenvolvido utilizando-se as classes MFC, que possibilitam uma configuração rápida da interface gráfica . Esta interface (figura 13) foi desenvolvida para possibilitar uma maior flexibilidade na escolha da topologia e dos parâmetros de treinamento.

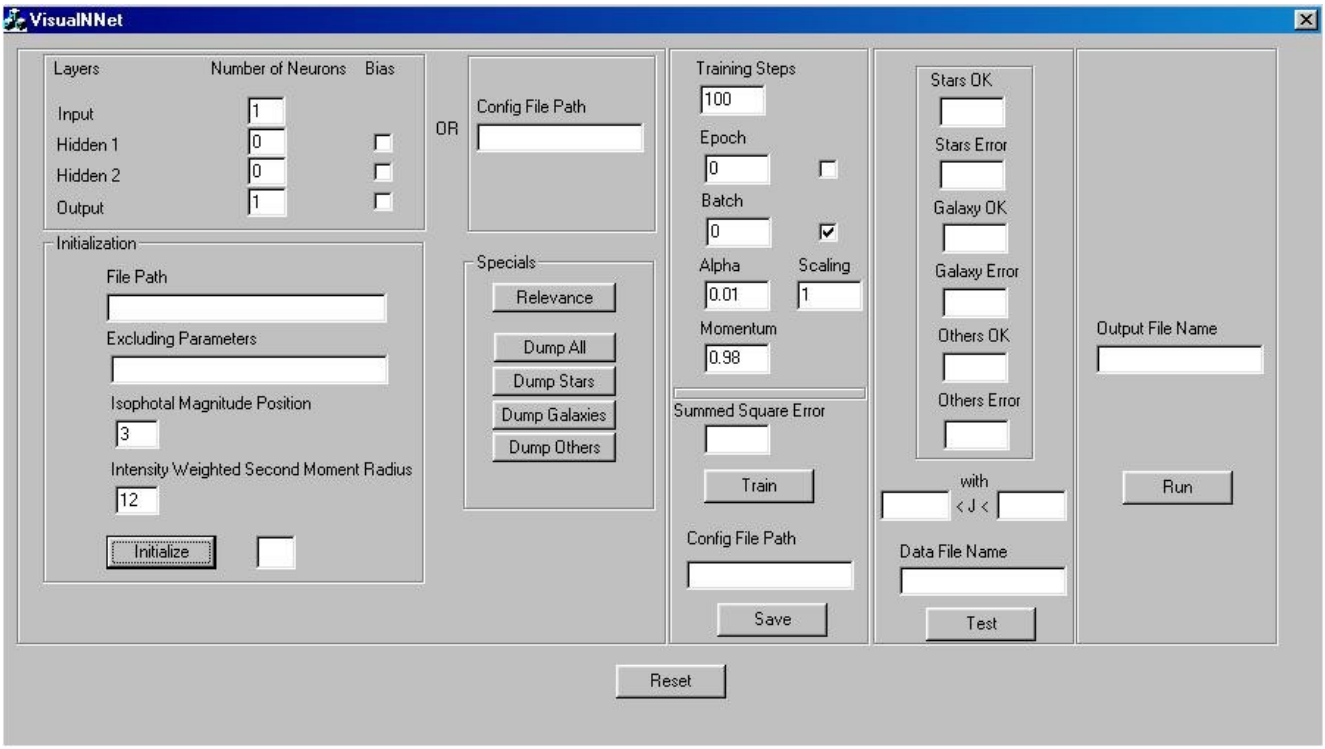


Figura 13 - Interface Gráfica do Visual Nnet

O botão de *Reset* apaga todas as configurações e dados armazenados na memória do programa para que o usuário possa iniciar uma nova seção de treinamento independente da seção anterior.

4.3.1 Interface de Configuração e Funções Especiais

A interface da figura 14 permite ao usuário definir a topologia da rede e o arquivo de dados que será ou usado para o treinamento da rede, ou analisado para a separação de estrelas e galáxias.

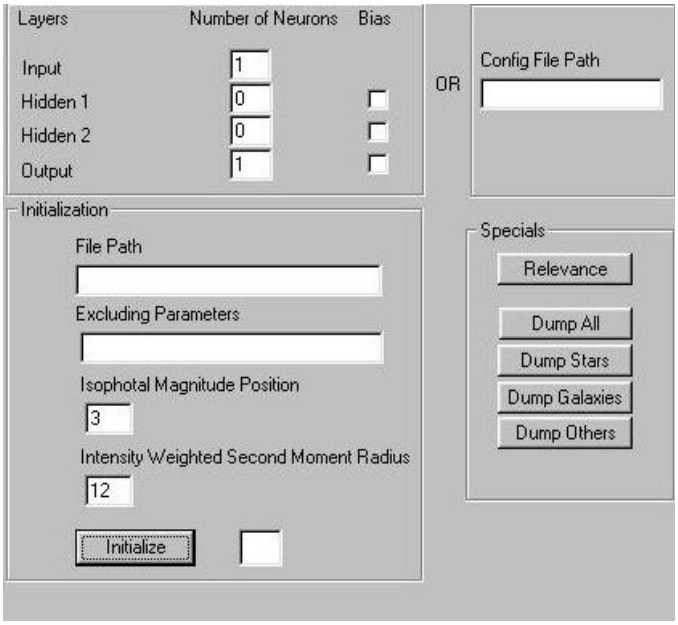


Figura 14 - Interface de entrada de Dados

Nesta interface o usuário tem a possibilidade de determinar quantos neurônios terá cada camada e se os neurônios destas camadas possuirão um *bias*. O usuário pode alternativamente entrar com o nome de um arquivo de configuração que possui todos os dados necessários para a inicialização da rede.

Se o usuário entrar com um arquivo de configuração, o programa irá desconsiderar qualquer dado que tenha sido fornecido com relação à topologia da rede.

Na figura 14 nós podemos notar mais 4 campos:

- **File Path** – Neste campo o usuário irá colocar o caminho do arquivo contendo os dados que serão utilizados para o treinamento da rede
- **Excluding Parameters** – Aqui o usuário dá a posição no arquivo de dados em que se encontram parâmetros que não devem ser usados para o treinamento da rede. Esta posição deve ser dada sem a preocupação de como o programa irá ler o arquivo.

- **Isophotal Magnitude Position** – Este campo indica qual a posição no arquivo de dados em que se encontra a Magnitude isofotal. Este parâmetro é um dos mais importantes que devem ser analisados na classificação de um objeto como estrela ou galáxia e precisa ter sua posição indicada se o usuário pretender testar a rede utilizando faixas de magnitude.
- **Intensity Weighted Second Moment Radius** - Este campo indica qual a posição no arquivo de dados em que se encontra o segundo momento de raio. Este parâmetro é utilizado em algumas análises da separação de estrelas e galáxias.

Nesta interface podemos notar ainda a área de funções especiais. Estas funções têm como finalidade ajudar a analisar a as propriedades do arquivo de dados e da saída da rede. Estas funções são:

- **Relevance** – Este botão executa os testes de relevância e anti-relevância que indicam quais os parâmetros que mais contribuem para um melhor treinamento da rede.
- **Dump All** – Esta função salva em um arquivo o resultado dos neurônios de saída da rede para todos os objetos do arquivo indicado em *File Path.*
- **Dump Stars -** Esta função salva em um arquivo o resultado dos neurônios de saída da rede para todos os objetos rotulados como estrelas no arquivo indicado em *File Path.*
- **Dump Galaxies** - Esta função salva em um arquivo o resultado dos neurônios de saída da rede para todos os objetos rotulados como galáxias no arquivo indicado em *File Path.*
- **Dump Others** - Esta função salva em um arquivo o resultado dos neurônios de saída da rede para todos os objetos rotulados como outros objetos no arquivo indicado em *File Path.*

4.3.2 Interface de Treinamento

A figura abaixo mostra a interface de treinamento do programa.

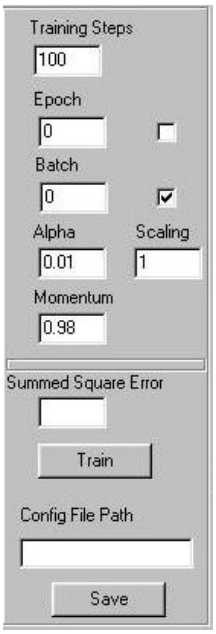


Figura 15 - Interface de treinamento

No campo *Training Steps* é onde o usuário define o número de passos de treinamento que serão dados pela rede. Os demais campos o explicados a seguir:

- **Epoch –** Treinamento por épocas, ou seja, os pesos são atualizados com a média dos deltas acumulados após ***n*** épocas de treinamento.
- **Batch –** Treinamento por caixas. Os pesos são atualizados com a média dos deltas acumulados após ***n*** pares entrada-saída.

Quando a opção selecionada, tanto *Epoch* quanto *Batch*, tiver 0 em seu campo, isto significa que a rede será treinada pela regra delta, ou seja, os pesos são atualizados pelos deltas gerados após cada par entrada-saída.

- **Learning Rate –** Parâmetro que permite ajustar a taxa de aprendizagem da rede. É muito importante, pois evita oscilações que podem inviabilizar o treinamento.
- **Scaling –** O valor indicado neste campo multiplica a taxa de aprendizado da rede no final de cada passo de treinamento para determinar a taxa do próximo passo. Quando o valor é 1, a taxa permanece constante ao longo dos passos de treinamento. Valores menores do que 1 causam um decréscimo na taxa de treinamento até um valor mínimo estipulado em 0,01.
- **Momentum –** Este parâmetro é utilizado para que a rede ganhe "inércia" suficiente para superar vales locais.

O erro médio quadrático final resultante é mostrado no campo *Summed Squared Error* ao final do treinamento. Após o treinamento, o usuário tem a opção de salvar a sua rede em

um arquivo de configuração que poderá ser utilizado para inicializar a rede conforme já visto anteriormente.

4.3.3 Interface de Teste

A figura abaixo mostra a caixa de dialogo que permite a realização de testes com a configuração obtida para a rede após o treinamento.

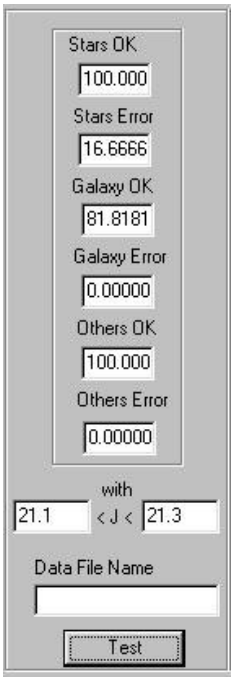


Figura 16 – Interface de Teste

As janelas de magnitude permitem que se escolha a faixa de magnitude em que será realizado o teste. O resultado aparece nas janelas acima e indicam o percentual de acerto e de erro para estrelas, galáxias e outros objetos, respectivamente. Por exemplo, na figura acima podemos ver que todas as estrelas e outros objetos foram corretamente classificados nesta faixa de magnitude. Por outro lado, 81,81% das galáxias foram classificadas corretamente. Os 18,19% restantes, foram classificados erroneamente como estrelas. È importante ainda notar que estas galáxias que foram erroneamente classificadas representam 16,67% do total de objetos que foram classificados como estrelas pela rede.

4.3.4 Interface de execução

Na figura abaixo vemos a interface de execução do programa VisualNNet.

Figura 17 - Interface de Execução

Nesta interface o usuário determina o nome do arquivo onde será salva a análise dos dados que foram carregados na memória e que é executada através do botão "*Run*".

5 Desenvolvimento

5.1 Introdução

Neste capítulo iremos analisar todo o processo de análise de dados e treinamento da rede. Começaremos pelo teste de correlação, passando pelo processo de definição de topologia e por último o treinamento da rede em si.

5.2 Testes de Correlação

Os testes de correlação são utilizados para determinar dentro de um conjunto grande de parâmetros quais aqueles que mais contribuem para a melhora no processo de treinamento.

Isto é muito útil, pois alguns parâmetros não só não contribuem, como também inserem ruído na entrada da rede, prejudicando, assim, o resultado final.

Nosso conjunto inicial de dados era constituído de 30 parâmetros, que são o resultado do pré-processamento das imagens realizado pelo FOCAS. Esses parâmetros, como já foi discutido na seção 3.3.2, representam as características morfológicas do objeto ao qual estão associados.

A fim de determinar quais os parâmetros que deveriam ser utilizados pela rede, foi utilizado o método da relevância e anti-relevância[1], este último desenvolvido na Universidade Federal do Rio de Janeiro para ser aplicado no treinamento de redes neuronais empregadas na classificação de partículas de alta energia.

Para determinarmos a relevância e a anti-relevância de cada parâmetro, primeiramente realizamos uma seção de treinamento da rede utilizando 1000 passos de treinamento e a

metodologia de definição de topologia e configuração que será descrita na seção 5.3. Com a rede resultante deste treinamento é que realizamos os testes de relevância e anti-relevância.

Existem alguns parâmetros que podem ter uma influência circunstancial na classificação, mas que não devem ser levados em conta. Isto pode acontecer, por exemplo, quando há um aglomerado de galáxias em uma região do céu, o que significaria que a ascensão reta, a declinação e o número do objeto teriam uma influência na classificação do objeto para aquela placa. Por isso esses parâmetros foram previamente excluídos, para que não se obtivesse um falso resultado.

O teste de relevância é utilizado para se medir quanto cada parâmetro contribui para o treinamento total da rede. Quanto maior o valor da relevância de um parâmetro ***j***, maior será a sua influência. O valor da relevância de um parâmetro ***j*** pode ser definido pela equação abaixo:

$$R(x_j) = \frac{\sum_{i=1}^{N_P} \| out(x_i) - out(x_i \mid x_{i,j} = < x_j > \|^2}{N_P}, \qquad \textbf{(28)}$$

onde:

out(x$_i$) = Soma dos elementos do vetor de saída para cada conjunto de parâmetros aplicados à entrada;

out(x$_i$ | x$_{i,j}$ = < x$_j$ >) = Mesmo vetor de saída, porém com a componente ***j*** do vetor de parâmetros substituída por seu valor médio;

N$_P$ = Número de vetores de parâmetros disponíveis para a fase de treinamento.

Para o teste de relevância, obtivemos o seguinte resultado:

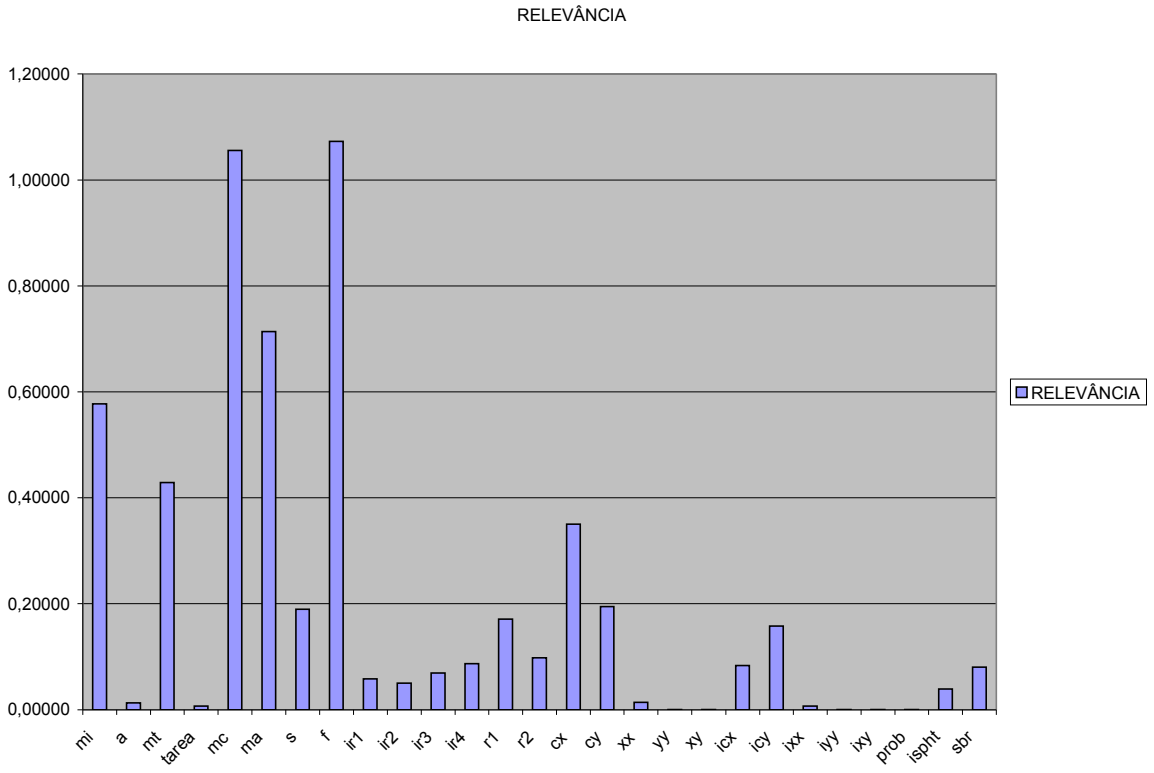


Figura 18 – Resultado do Teste de Relevância

Como critério de decisão, arbitrou-se que os parâmetros escolhidos seriam aqueles que tivessem um índice de relevância superior a 0,5. Utilizando-se este critério, foram escolhidas as magnitudes isofotal, a magnitude do centro, a magnitude de abertura e a fração de resolução como parâmetros a serem utilizados.

Já o teste de anti-relevância tem como finalidade medir o quanto cada parâmetro contribui com o treinamento de cada classe individualmente. Porém, diferentemente do teste de relevância, os parâmetros que mais contribuem são aqueles que possuem a menor Anti-relevância. A anti-relevância para cada parâmetro ***j*** é calculada segundo a fórmula abaixo:

$$AR(x_j) = \sum_{k=1}^{M} \frac{\sum_{i=1}^{N_{kP}} \| out(x_i) - out(x_i \mid x_{i,j} = < x_j > \|^2}{N_{kP}}, \qquad \textbf{(29)}$$

onde:

M = Número total de classes;

out(x$_i$) = Soma dos elementos do vetor de saída para cada conjunto de parâmetros aplicados à entrada;

$out(x_i \mid x_{i,j} = < x_j >)$ = Mesmo vetor de saída, porém com a componente ***j*** do vetor de parâmetros substituída pelo valor médio das componentes ***j*** pertencentes à classe ***M***;

N_{kP} = Número de vetores de parâmetros disponíveis para a fase de treinamento pertencentes à classe ***M***.

O resultado do teste de anti-relevância pode ser visto no gráfico abaixo:

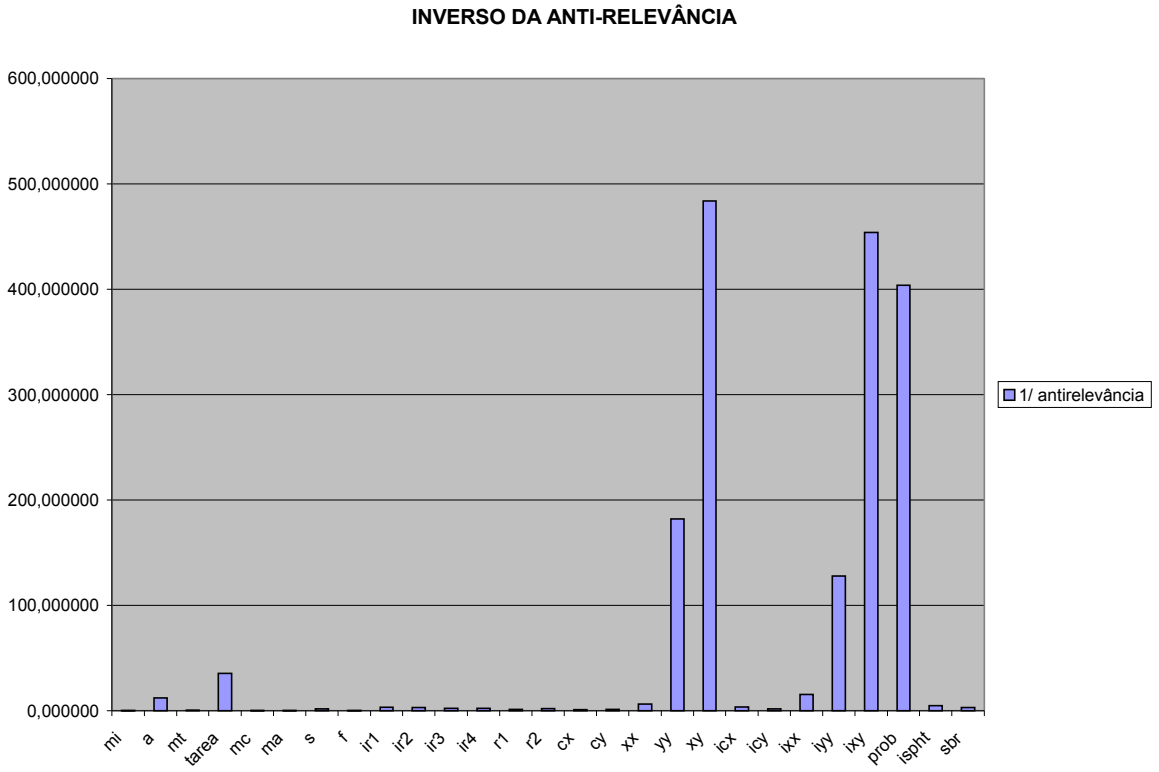


Figura 19 – Resultado do Teste de anti-relevância. A escala vertical vai de 0 a 600,

Para este teste arbitrou-se como critério de decisão que seriam escolhidos aqueles parâmetros cujo valor do inverso de sua anti-relevância fosse maior ou igual a 100, Desta forma os parâmetros escolhidos foram o segundo momento não ponderado ***yy***, o segundo momento não ponderado ***xy***, o segundo momento de intensidade ponderado ***iyy***, o segundo momento de intensidade ponderado ***ixy*** e a probabilidade de resolução (***prob***):

Uma explicação sucinta sobre os parâmetros escolhidos pelo método da relevância pode ser encontrada na seção 3.3.2. Já para os parâmetros identificados pelo método da anti-relevância não fomos capazes de identificar suas definições.

5.3 Definição da Configuração da Rede

5.3.1 Topologia

Primeiramente definiu-se a topologia da rede. Parte desta topologia já é definida pela quantidade de parâmetros de entrada, definido em 9 pelos testes de correlação, e pelo número de classes que determinam o número de neurônios na camada de saída

Deve-se ressaltar que adotaram-se 3 neurônios na camada de saída para se fazer uma separação maximamente espaçada onde a classe das estrelas é representada pelo conjunto de saída (1,-1,-1), a classe de galáxias é representada pelo conjunto (-1,1,-1) e a classe dos outros objetos é representada por (-1,-1,1).

O teste para a definição da topologia consistiu em rodar 100 passos de treinamento, utilizando a regra delta. A cada vez que o teste era executado, mantinham-se constantes o arquivo de dados e os parâmetros de treinamento, variando-se apenas a topologia da rede.

Os parâmetros de treinamento utilizados foram:

- Passo de treinamento (α): 0,01
- Scaling: 1
- Momentum: 0,98

O parâmetro utilizado para comparação foi o erro médio quadrático obtido na saída da rede, e o arquivo de dados analisado foi o Hope.dat. Para se determinar o menor erro realizaram-se 3 rodadas de testes para cada configuração e utilizou-se a média dos erros para a decisão.

Topologia	**1**	**2**	**3**	**Média**
9/3/3	0,281238	0,277338	0,276482	0,278352
9/4/3	0,231851	0,216741	0,270000	0,239531
9/5/3	0,206266	0,207171	0,202928	0,205455
9/6/3	0,196420	0,211880	0,198540	0,202280
9/7/3	0,202686	0,185648	0,203437	0,197257
9/8/3	0,210773	0,200634	0,190170	0,200525
9/9/3	0,189252	0,189044	0,197244	0,191847
bb				
9/5/3	0,123276	0,152145	0,133098	0,136173
9/6/3	0,123947	0,125899	0,124551	**0,124799**
9/7/3	0,128429	0,126251	0,124465	0,126382
9/8/3	0,123546	0,124655	0,129978	0,126060
9/9/3	0,128364	0,126117	0,120324	0,124935

b_				
9/6/3	0,127134	0,122978	0,126096	0,125403
9/7/3	0,130299	0,125393	0,125863	0,127185
9/8/3	0,126086	0,123355	0,125706	0,125049
9/9/3	0,124069	0,128857	0,121996	0,124974
_b				
9/6/3	0,183528	0,179931	0,176502	0,179985

Tabela 1 – Valores do Erro Médio Quadrático para várias topologias com 3 camadas

Topologia	1	2	3	Média
9/6/6/3	0,178995	0,179424	0,174773	0,177731
9/6/5/3	0,194175	0,181965	0,192881	0,189674
9/6/4/3	0,188828	0,180537	0,183862	0,184409
9/6/3/3	0,196532	0,198024	0,219072	0,204543
9/9/9/3	0,166076	0,171628	0,166663	0,168122
9/9/8/3	0,167033	0,171412	0,170236	0,169560
9/9/7/3	0,180371	0,164623	0,167547	0,170847
9/9/6/3	0,175743	0,172678	0,170937	0,173119
9/9/5/3	0,184486	0,172732	0,185182	0,180800
9/9/4/3	0,182360	0,182421	0,175396	0,180059
9/9/3/3	0,218096	0,217515	0,187073	0,207561
b_ _				
9/9/9/3	0,119923	0,114439	0,113790	0,116050
_b_				
9/9/9/3	0,117871	0,116975	0,121076	0,118640
_ _ b				
9/9/9/3	0,149961	0,152175	0,156266	0,152801
bb_				
9/9/9/3	0,114122	0,114295	0,109895	0,112771
_bb				
9/9/9/3	0,126577	0,129006	0,123130	0,126238
bbb				
9/9/9/3	0,112175	0,112663	0,112266	**0,112368**

Tabela 2 – Valores do Erro Médio Quadrático para várias topologias com 4 camadas

Com base nas tabelas acima podemos verificar que a diferença na média dos erros dos dois melhores resultados (0,124799 e 0,112368) é de apenas 0,012431. Por outro lado, a diferença no tempo computacional entre as duas topologias (9/6/3 e 9/9/9/3) é bem significativa. Por este motivo foi escolhida a topologia de 9 neurônios na camada de entrada, 6 na camada intermediária e três na camada de saída, sendo as duas últimas camadas alimentadas com ***bias***.

5.3.2 Configuração dos parâmetros de treinamento

Para definir-se a configuração dos parâmetros de treinamento da rede, foi utilizada uma metodologia semelhante à utilizada para a definição da topologia. Porém, desta vez, são a topologia e o número de passos de treinamento é que são mantidos constantes.

A definição dos parâmetros de treinamento é estreitamente relacionada com a forma de treinamento que será utilizada. Por isso os testes foram divididos em três partes: regra delta, por Época e *Batch*.

5.3.2.1 Regra delta

Na Regra delta, os pesos da rede são atualizados a cada passo de treinamento. O primeiro parâmetro a ser determinado será a taxa de aprendizado (*learning rate* - α) e o fator de escalonamento associado, que multiplica o valor da taxa de aprendizado a cada passo de treinamento.

A taxa máxima de aprendizado utilizada foi 0,2 pois acima disto a rede apresentou um comportamento instável.

α	Scaling	1	2	3	Média
0,005	1	0,140223	0,144155	0,138402	0,140926
0,010	1	0,125856	0,125268	0,120585	0,123903
0,015	1	0,123879	0,121778	0,124998	0,123552
0,020	1	0,122616	0,124847	0,118096	0,121853
0,025	1	0,123676	0,115206	0,121253	0,120045
0,030	1	0,120578	0,121090	0,121304	0,120991
0,030	0,999	0,119488	0,118914	0,121985	0,120129
0,050	1	0,113998	0,117255	0,122300	0,117851
0,1	1	0,124681	0,119615	0,125472	0,123256
0,1	0,999	0,123201	0,128216	0,120923	0,124113
0,1	0,990	0,115452	0,112488	0,112625	0,113522
0,1	0,900	0,119042	0,121154	0,117518	0,119238
0,15	0,990	0,118986	0,113923	0,112734	0,115214
0,15	0,950	0,114468	0,115634	0,112197	0,114099
0,15	0,900	0,117988	0,117792	0,118498	0,118092
0,2	0,95	0,111514	0,110681	0,111448	**0,111214**

Tabela 3 – Erro médio quadrático da rede para diferentes α

Após definidas a taxa de aprendizagem, vamos definir o momento da rede.

momentum	1	2	3	Média
1.00	0,111382	0,112910	0,112513	**0,112268**
0,99	0,110697	0,115237	0,116072	0,114002
0,98	0,114800	0,111794	0,111582	0,112725
0,95	0,114459	0,114535	0,111707	0,113567
0,9	0,116828	0,113692	0,115213	0,115244
0,8	0,114398	0,113480	0,115222	0,114367
0,7	0,115019	0,112550	0,117366	0,114978
0,6	0,116958	0,118046	0,116765	0,117256
0,5	0,117873	0,116164	0,118951	0,117663
0	0,121313	0,122736	0,148500	0,130849

Tabela 4 - Erro médio quadrático da rede para diferentes momentos

Podemos notar pela tabela acima que a rede não é tão sensível à variação no momento quanto é à variação na taxa de aprendizagem.

5.3.2.2 Treinamento por Época

No treinamento por época, os pesos são atualizados com a média do erro depois de ***n*** épocas, isto é, ***n*** passos de treinamento. Repetiremos aqui o mesmo método utilizado na seção anterior utilizando-se apenas 1 época pois acima disto as atualizações se dão de forma muito lenta..

Primeiramente iremos definir a taxa de aprendizagem (*learning rate* - α) e o fator de escalonamento associado.

Época	α	Scaling	1	2	3	Média
1	0,01	1	0,456979	0,437312	0,445598	0,446625
1	0,1	1	0,382549	0,390388	0,389387	0,387440
1	0,2	1	0,378117	0,377513	0,369139	0,374920
1	0,5	1	0,360242	0,362144	0,334099	0,352160
1	1	1	0,281692	0,295546	0,280183	**0,285805**

Tabela 5– Erro médio quadrático da rede para 1 época e diferentes valores de α

Definiremos agora o momento da rede.

momentum	1	2	3	Média
1.00	0,300995	0,284087	0,278358	**0,287813**
0,9	0,310724	0,293693	0,296132	0,300183
0,8	0,296293	0,295071	0,306285	0,299216
0,5	0,352029	0,292075	0,304961	0,316355
0	0,343035	0,354034	0,341816	0,346295

Tabela 6- Erro médio quadrático da rede para diferentes momentos

5.3.2.3 Treinamento Batch

Este tipo de treinamento é na verdade um meio termo entre a regra delta e o treinamento por época. Nele são definidas "caixas" com ***n*** vetores de treinamento. Os pesos da rede são atualizados com o erro médio obtido após a apresentação de cada caixa.

O arquivo *hope.dat* possui 3168 vetores, sendo que destes, 2212 são usados para o treinamento. Portanto, para um treinamento por caixas, teremos:

Nº de caixas	Nº de vetores por caixa
2	1106
5	442
10	221
100	22

Tabela 7 – Relação entre número de caixas e número de vetores por caixa

Para a definição da taxa de aprendizado (*learning rate* - α) teremos então:

Nº de Caixas	α	Scaling	1	2	3	Média
2	0,1	1	0,119836	0,122049	0,121033	0,120973
2	0,4	0,95	0,110856	0,117205	0,113010	**0,113690**
2	0,5	0,90	0,119092	0,114400	0,111124	0,114870
5	0,08	1	0,121012	0,122018	0,127614	0,123548
5	0,1	1	0,121288	0,119780	0,119210	0,120093
5	0,3	0,95	0,121682	0,137804	0,123825	0,127770
10	0,1	1	0,128140	0,125180	0,120646	0,124655
10	0,3	1	0,122355	0,121415	0,122840	0,122203
10	0,4	1	0,118108	0,124083	0,119084	0,120425
100	0,1	1	0,263464	0,263657	0,250619	0,259247
100	0,5	1	0,149292	0,146970	0,177824	0,158029
100	1	1	0,140601	0,137566	0,142088	0,140085

Tabela 8– Erro médio quadrático da rede para diferentes tamanhos de caixas e valores de α

Definiremos agora o momento da rede.

momentum	1	2	3	Média
1.00	0,109786	0,110663	0,110537	**0,110328**
0,9	0,114747	0,107475	0,111674	0,111299
0,8	0,114282	0,110455	0,110612	0,111783
0,5	0,113787	0,114114	0,113685	0,113862
0	0,114512	0,114903	0,114152	0,114522

Tabela 9- Erro médio quadrático da rede para diferentes momentos

5.3.2.4 Conclusão

Pela tabela 5, podemos notar que o treinamento por época tem erros quadráticos médios bem superiores ao treinamento pela regra delta (tabela 3) ou por caixas (tabela 8). Isto se dá porque o treinamento por época evolui de uma maneira muito mais uniforme.

Por outro lado, o treinamento por regra delta tem um início mais rápido em conseqüência do maior número de atualizações feitas. Isto é benéfico no início, porém prejudicial em fases avançadas do treinamento, quando o erro da rede já se encontra próximo de um mínimo da superfície de erro.

O treinamento por caixas é a junção dos dois outros treinamentos, por isso deve servir como ponte entre suas qualidades.

Para a fase de treinamento serão adotadas duas abordagens. Na primeira será realizado o treinamento por caixas, utilizando as seguintes configurações:

- 2 "caixas" (1106 vetores)
- $\alpha = 0,4$
- ***Scaling*** = 0,95
- ***momentum*** = 1

A Segunda abordagem será constituída de um treinamento misto, onde no início o método de treinamento empregado será o da regra delta e após aplicar-se-á o treino por épocas. A configuração usada para a regra delta será a seguinte:

- $\alpha = 0,2$
- ***Scaling*** = 0,95
- ***Momentum*** = 1

A configuração do treinamento por época será definida após a etapa do treinamento por regra delta.

5.4 Treinamento da Rede

O processo de treinamento da rede se dividiu em duas partes principais. Primeiramente foi feita uma análise e seleção dos dados disponíveis para o treinamento da rede.

Depois de refinado o arquivo de treinamento, deu-se início ao processo de treinamento propriamente dito.

5.4.1 Processo de Seleção dos Dados

A questão da seleção dos dados para o treinamento da rede é muito delicada, pois dela depende a confiabilidade da classificação final.

Para este trabalho foram utilizados três arquivos contendo informações sobre a mesma região do céu.

O primeiro arquivo *(dataSC.dat)* contém todos os dados disponíveis, oriundos das placas fotográficas. Porém, a classificação dos objetos presentes neste arquivo foi feita pelo FOCAS e não sofreu nenhum tipo de reclassificação visual. Isto é um problema, pois para grandes magnitudes, a classificação do FOCAS é muito pouco confiável.

O segundo arquivo (*dataClean.dat*) contém uma parte dos objetos presentes no arquivo dataSC.dat. Porém estes objetos sofreram uma reclassificação visual feita por astrônomos do Observatório Nacional. Esta reclassificação visual, embora não possa ser considerada 100% correta, apresenta significativa melhora em relação à classificação original. A tabela abaixo demonstra a concordância entre a classificação do FOCAS e a re-classificação visual realizada no Observatório Nacional.

Faixas de Magnitude m_J	Porcentagem de objetos reclassificados visualmente	Concordância entre a classificação visual e a classificação do FOCAS (%)
< 20,5	100	100
20,5 – 20,7	100	90
20,7 – 21,0	100	80
21,0 – 21,1	100	70
21,1 – 21,3	100	60
21,3 – 21,6	90	50
21,6 – 22,0	75	50
22,0 – 22,3	60	40
22,3 – 22,5	30	25

Tabela 10 – Teste de eficiência do FOCAS na classificação[2]

O terceiro arquivo contém os dados oriundos das imagens de CCD *(Treino.dat)*. Por sua melhor resolução, a classificação desses dados pode ser considerada praticamente correta até pelo menos a magnitude 21,5. Porém, certos parâmetros associados aos objetos não coincidem com os aqueles obtidos com a placa fotográfica. Por isso é necessário que se faça uma comparação entre os dados presentes nos arquivos *DataSC.dat* e *Treino.dat*. Uma vez feita esta comparação poderemos utilizar os parâmetros presentes no arquivo *DataSC.dat*, mas com sua classificação sendo dada pelo arquivo Treino.dat.

Infelizmente, esta comparação não é trivial. Para identificarmos um objeto que esteja presente nos dois arquivos, temos que analisar primeiramente suas coordenadas e como critério de decisão secundário, suas magnitudes. O problema é que um dado objeto possui diferentes valores de ascensão reta e declinação nos dois catálogos. Isto se dá em parte pela diferença de resolução das imagens.

O problema é agravado no tocante à ascensão reta pois a precisão do FOCAS para este parâmetro se limita a 4 casas decimais. Os erros máximos considerados foram de 0,0015 graus para ascensão reta e de 0,0010 graus para a declinação.

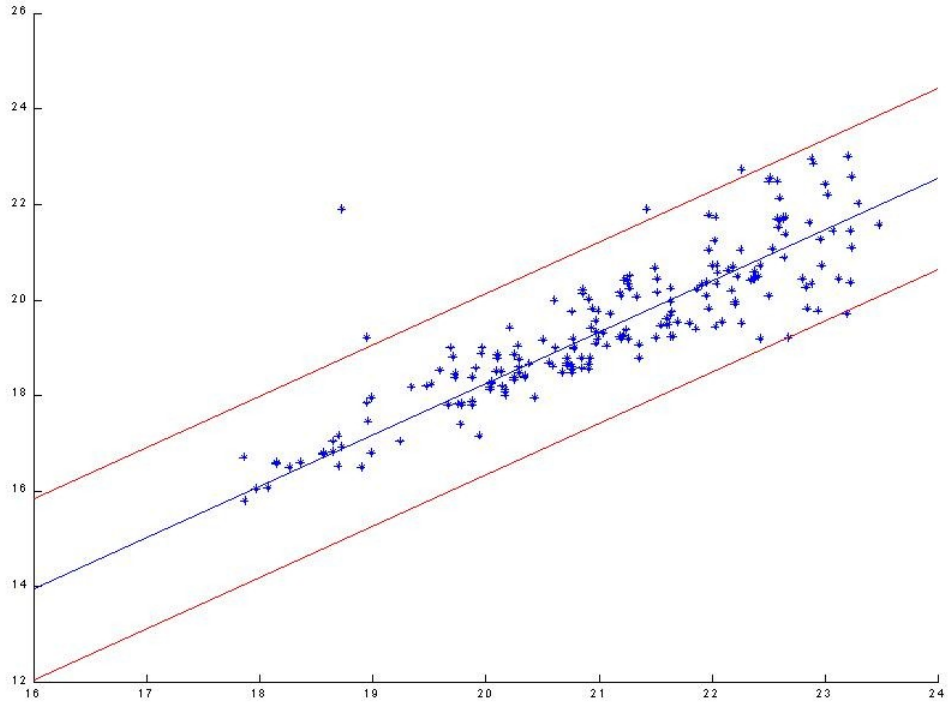
Quanto às magnitudes, para que o mesmo objeto apresentasse o mesmo valor nos dois catálogos, seria necessária uma calibração da imagem, que não foi feita. Porém, a magnitude deve apresentar uma relação linear entre os seus valores nos dois arquivos.

Com base nos dados acima apresentados, foi elaborada uma rotina computacional que tem por função o casamento entre os dados dos dois arquivos. Primeiramente, esta rotina identifica quais objetos possuem coordenadas dentro das faixas de erro especificadas.

Com bastante freqüência, mais de um objeto foi encontrado no arquivo *Treino.dat* que se enquadrava dentro da faixa de erro de um dado objeto do arquivo *dataSC.dat*. Por isso foi necessário lançar mão da magnitude para ajudar na escolha.

Como foi dito anteriormente, as magnitudes correspondentes deveriam ter uma relação linear. Por isso, primeiramente foi necessário determinar-se qual a reta associada a esta relação.

Esta reta foi determinada através de um ajuste linear, plotando-se as magnitudes nos dois catálogos respectivas aos pares de objetos que apresentavam incerteza menor do que 0,001 para suas coordenadas e não encontravam repetições. Com isso obteve-se o gráfico



abaixo:

Figura 20 – Relação Linear entre as magnitudes nos arquivos DataSC.dat e Treino.dat

A reta azul é caracterizada pela equação y = 1,075*x – 3,25. As duas retas vermelhas representam a incerteza correspondente a linearidade e são deslocadas de ± 1,9. A reta

efetivamente usada difere ligeiramente da reta apresentada, mas devido ao cálculo apresentado a seguir não houve diferença significativa no resultado final.

Mesmo utilizando a magnitude como fator de desempate, ainda houve muitos casos em que mais de um objeto se enquadrava dentro dos limites de erro estabelecidos. Por isso, então, foi calculado um erro ponderado cuja formula é dada abaixo:

$$Erro = \frac{Erro_{asc} + 2 * Erro_{dec} + 2 * Erro_{mag}}{5} \qquad \textbf{(30)}$$

Eram considerados pares de objetos associados aqueles que possuíssem menor erro ponderado.

Assim foi criado o arquivo *ParcialLimpo.dat*, que continha apenas os objetos identificados no *CCD*, e o arquivo *ParcialSujo.dat*, que continha o resto dos objetos.

Usando-se, então, o arquivo *ParcialSujo.dat* foi feita então uma nova triagem dos dados. Desta vez utilizou-se o arquivo de objetos reclassificados visualmente pelos astrônomos do Observatório Nacional. Os dados reclassificados que ainda se encontravam no arquivo *ParcialSujo.dat* foram separados no arquivo *reclassificados.dat*.

A última análise feita sobre os dados foi realizada em cima dos dados restantes do arquivo *parcialSujo.dat*. Neste arquivo identificou-se um conjunto de objetos que, com um alto grau de confiabilidade, não eram nem estrelas e nem galáxias. Estes objetos foram separados no arquivo *outros.dat*. O restante dos dados do arquivo *parcialSujo.dat* foram descartados.

Por fim, os três arquivos resultantes desta seqüência de análises foram misturados de forma aleatória no arquivo *hope.dat* que foi o arquivo utilizado para o treinamento da rede.

5.4.2 Processo de Treinamento

Foram utilizados dois processos para o treinamento da rede como descrito anteriormente na seção 5.3.2.4.

Para ambos os processos, rodaram-se 15000 passos de treinamento. Este valor foi determinado de forma empírica, e acima dele não há grandes mudanças nos pesos da rede. Não se adotou nenhum critério de parada. Em vez disso, durante o treinamento a rede verifica

se aquele passo foi o melhor até então. A rede faz esta comparação através de dois parâmetros, a soma e o produto das eficiências.

A soma das eficiências indica os passos de treinamento que conseguiram obter os níveis mais elevados de eficiência. Porém, este parâmetro sozinho pode esconder algumas discrepâncias, onde uma classe é muito bem treinada enquanto as outras obtém resultados medíocres. Por exemplo, se uma classe obtém 100% de eficiência, enquanto as outras obtêm 70%, terá o mesmo resultado na soma de eficiências (240) de um outro caso, preferível, onde todas as 3 classes obtenham 80% de eficiência..

Então, para que um passo seja escolhido como melhor, é levado em consideração também o produto das eficiências. No exemplo anterior teríamos, então:

- Primeiro Caso: 1*0,7*0,7 = 0,49
- Segundo Caso: 0,8*0,8*0,8 = 0,512

Quando um "melhor passo" é encontrado, a rede salva os pesos da rede naquele momento em um arquivo chamado *"bestWeight_X.dat"* onde ***X*** é o passo de treinamento em que a rede se encontrava no momento da gravação.

Para ajudar na visualização do processo de treinamento, foram desenvolvidas algumas rotinas em *MATLAB* que mostram a evolução dos seguintes parâmetros durante o treinamento:

- Erro Médio Quadrático do conjunto de treinamento
- Erro Médio quadrático do conjunto de validação
- Erro percentual na classificação de estrelas, galáxias e outros objetos
- Eficiência para estrelas, galáxias e outros objetos

Para este fim foi elaborada a rotina *"netanalise.m"* que plota os gráficos referentes aos parâmetros acima com base em arquivos salvos pelo programa VisualNNet.

O primeiro método de treinamento que iremos analisar será o treinamento por caixas, utilizando os parâmetros discutidos na seção 5.3.2.4.

Para este método obtivemos as seguintes figuras de treinamento:

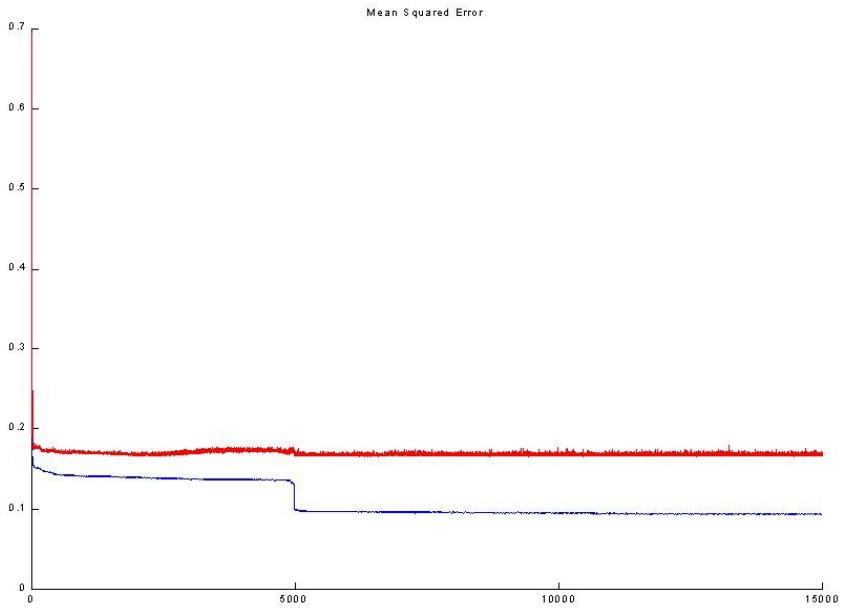


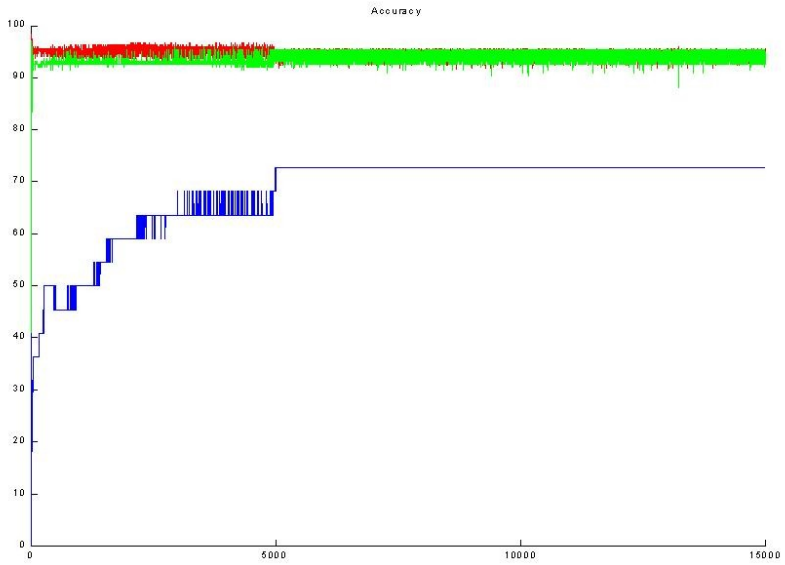
Figura 21 - Erro médio quadrático para o processo de treinamento por caixas

Nesta figura podemos ver a evolução do erro médio quadrático (EMQ) ao longo do processo de treinamento. A curva azul indica o EMQ referente ao conjunto de treinamento, enquanto que a curva vermelha refere-se ao EMQ do conjunto de validação.

O conjunto de validação é um grupo dos dados que não são apresentados à rede durante o processo de treinamento, mas que é usado ao final de cada passo para garantir que o treinamento está sendo capaz de identificar todo o espaço de dados, e não apenas aquele do conjunto de treinamento.

Podemos notar claramente uma queda abrupta no EMQ do conjunto de treinamento por volta do passo 5000, Esta queda indica que a rede estava presa a um mínimo local na superfície de erro, mas conseguiu achar um outro mínimo, mais profundo, que pode ser o mínimo global. Este comportamento do erro é típico de redes que usam o parâmetro "momento", que tem este nome justamente por, comparativamente com a física, dar momento suficiente à rede para vencer mínimos locais.

Figura 22 – Evolução das eficiências de classificação para o método de treinamento por caixas



Na figura 22 podemos acompanhar a evolução nas eficiências de classificação no processo de treinamento por caixas. Esta eficiência mede quantos objetos de uma classe foram classificados corretamente.

A curva verde indica a eficiência na classificação das galáxias, enquanto que a curva vermelha indica a eficiência na classificação das estrelas e a curva azul indica a eficiência de classificação de outros objetos.

Podemos notar nesta figura o efeito positivo da localização de um novo mínimo no processo de treinamento.

Na próxima figura podemos acompanhar a evolução do erro percentual de classificação. Este erro corresponde ao número de objetos que foram classificados erroneamente como sendo de uma certa classe em relação ao número total de objetos classificados pela rede como sendo desta classe.

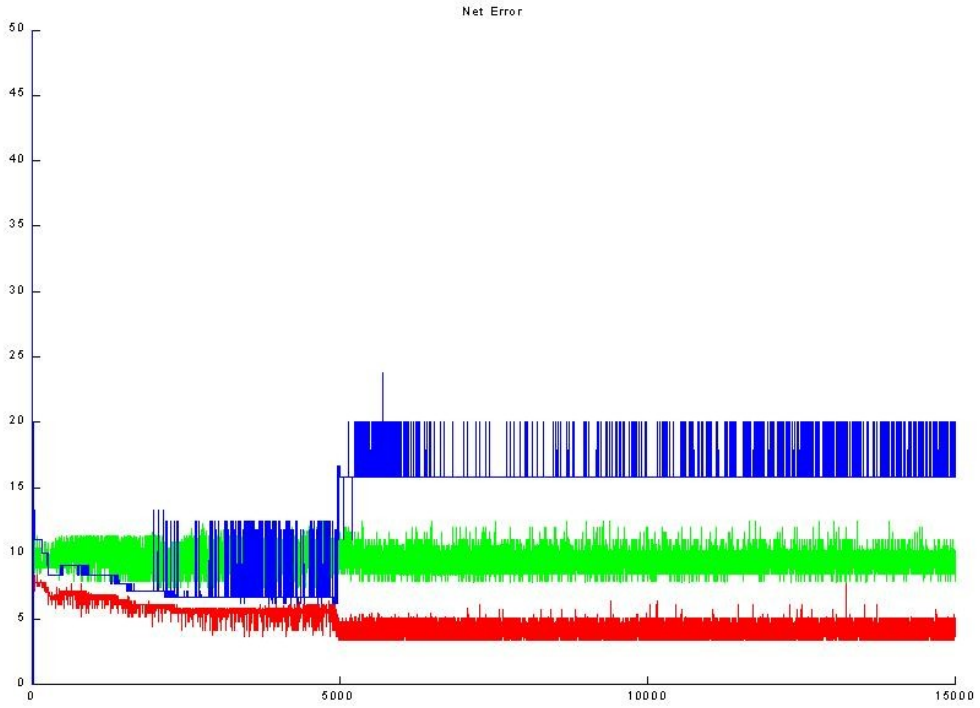


Figura 23 – Evolução do erro de classificação no processo de treinamento por caixas

Aqui evidenciamos um efeito colateral do aumento na eficiência de classificação de outros objetos. Junto com sua eficiência, aumentou também o erro de classificação. Isto se deve ao fato de o separador linear que delimita esta classe ter-se deslocado e avançado sobre outra classe, como podemos ver na figura ilustrativa abaixo

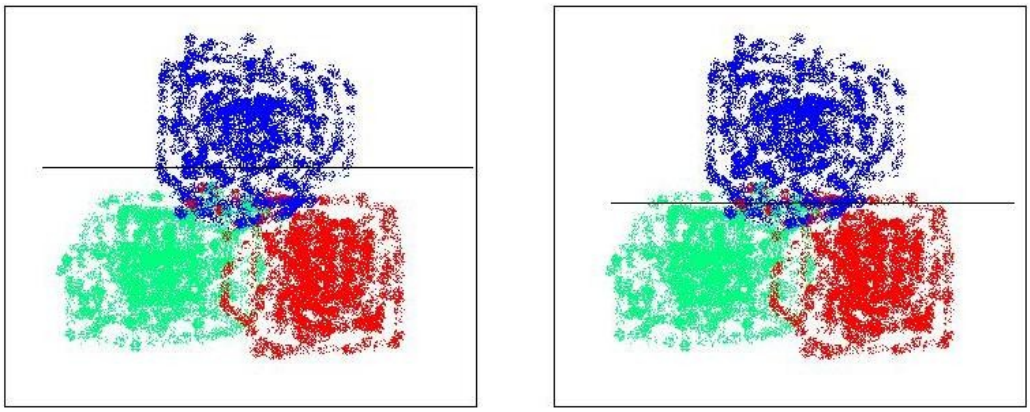


Figura 24 – Exemplos de separadores lineares

Na figura da esquerda vemos que a eficiência de classificação é menor do que a da figura da direita, porém o número de elementos de outras classes que são erroneamente classificados também é menor.

Este processo de treinamento obteve como seu melhor passo de treinamento o passo número 5022, onde se obtiveram os seguintes resultados:

- Soma das eficiências = 262,7273;
- Produto das Eficiências = 0,6563;
- EMQ = 0,0999.

Em contrapartida, o EMQ encontrado ao final do treinamento foi de 0,094157

Analisaremos agora o segundo processo de treinamento, como foi discutido na seção 5.3.2.4. Neste processo utilizou-se a regra delta para o treinamento da rede.

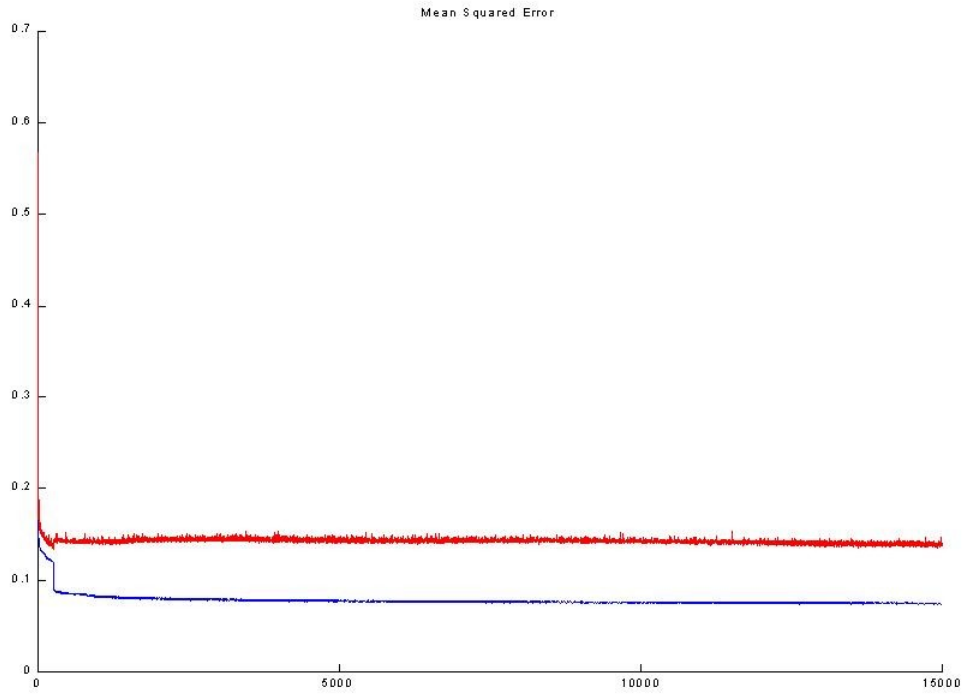
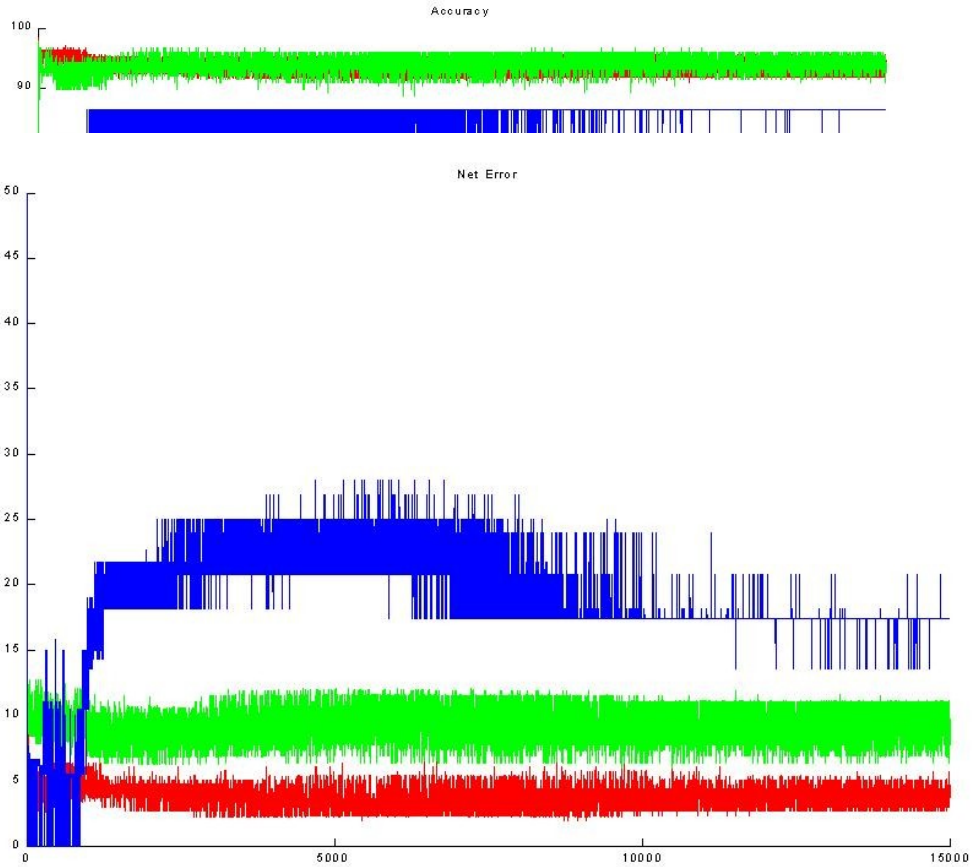


Figura 25 - Erro médio quadrático para o processo de treinamento por regra delta

Na figura 25 podemos notar a mesma queda abrupta que está presente na figura 21. Porém, aqui ela se manifesta muito anteriormente. Isto se deve ao fato de a regra delta proporcionar uma convergência mais rápida. Esta característica é um fator positivo no início do treinamento, mas pode atrapalhar nas fases finais, dificultando a convergência da rede.

Figura 26 - Evolução das eficiências de classificação para a regra delta



Novamente, como no método anterior, podemos notar uma melhora na eficiência de classificação, juntamente com a queda no EMQ.

Figura 27 - – Evolução do erro de classificação no processo de treinamento por regra delta

Como no método anterior, experimentamos um aumento no erro de classificação de outros objetos em decorrência do deslocamento do separador linear desta classe.

Como descrito no item 5.3.2.4, foi realizada uma segunda etapa de treinamento utilizando-se o método de épocas. Porém nenhuma mudança significativa foi percebida. Desta forma, o melhor passo de treinamento para o método da regra delta foi o passo 4378, que obteve os seguintes resultados:

- Soma das eficiências = 275,4773
- Produto das Eficiências = 0,7720
- EMQ = 0,0786.

Pelos resultados exibidos acima, podemos concluir que o melhor método de treinamento foi o da regra delta, que obteve os melhores valores na soma e no produto de suas eficiências.

6 Análise dos Resultados

6.1 Introdução

Nesta seção iremos discutir os resultados obtidos através do treinamento da rede neural pela regra delta, e compará-los com os resultados de outros métodos de classificação como o FOCAS.

6.2 Análise

Nesta seção iremos interpretar os resultados obtidos pela rede e poderemos ter uma noção de quanto a rede foi capaz de se aproximar da realidade.

No gráfico abaixo podemos perceber como a rede separou cada objeto. A situação ideal seria aquela na qual só existissem objetos nos vértices do cubo.

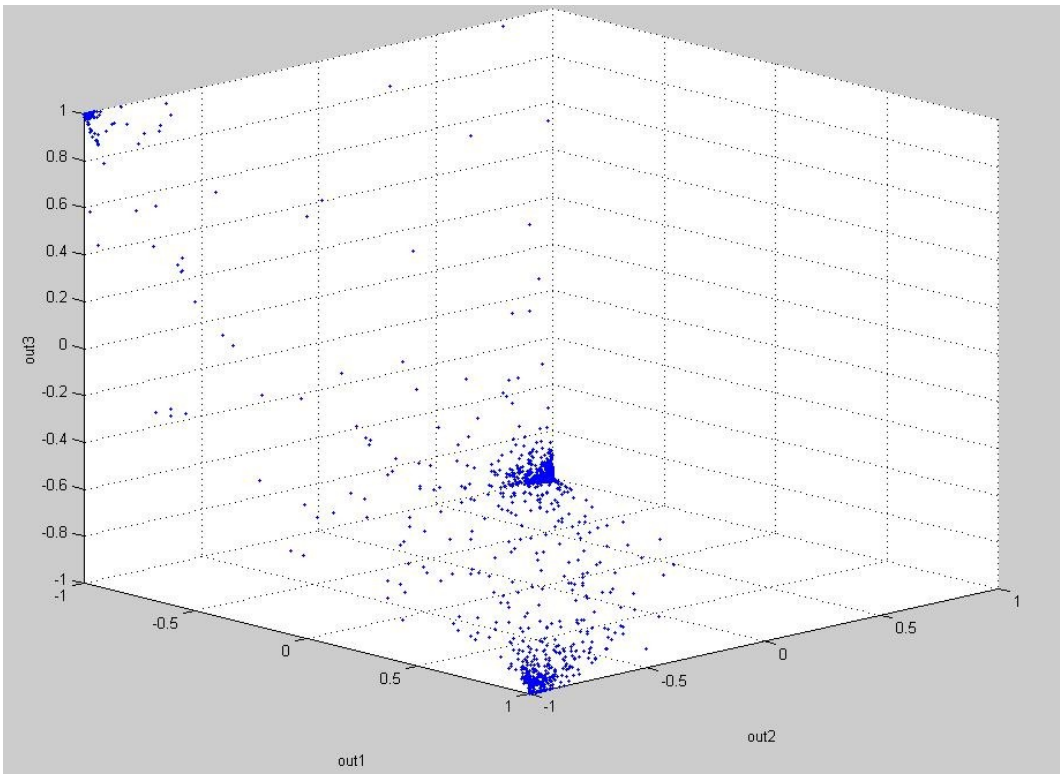


Figura 28 – Distribuição tri-dimensional da saída da rede

A classificação de um objeto é feita baseada nos valores dos neurônios de saída que podem ser interpretados como a probabilidade de o dado objeto pertencer àquela classe. Por isso, quanto mais próximo de um vértice um objeto estiver, maior a certeza que temos sobre sua classificação.

Quando a saída 1 (*out1*) for maior do que as demais, o objeto será classificado como estrela. Quando for a saída 2 (*out2*) a maior, então o objeto será classificado como galáxia, e finalmente quando a saída 3 for a maior, então o objeto será classificado como outro objeto.

A figura acima foi gerada através da rotina "*distribution.m*" para *MATLAB* e pode ser encontrada no Apêndice.

As próximas figuras mostram o comportamento da saída da rede para cada uma das classes. Esta análise é particularmente interessante, pois através dela podemos determinar quais os objetos que foram classificados erroneamente no arquivo de treino, desde que o resto da classificação seja suficientemente confiável (*Seixas et al*[14]).

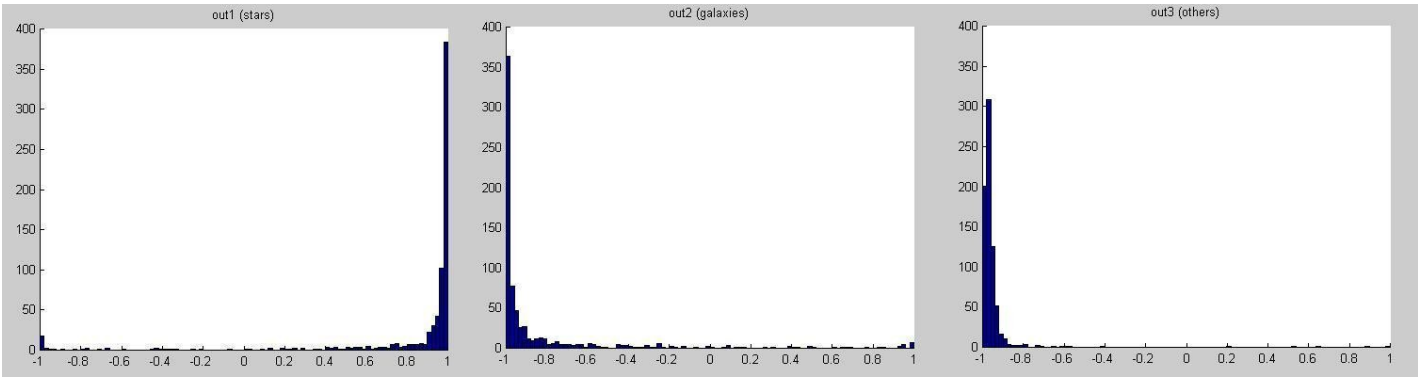
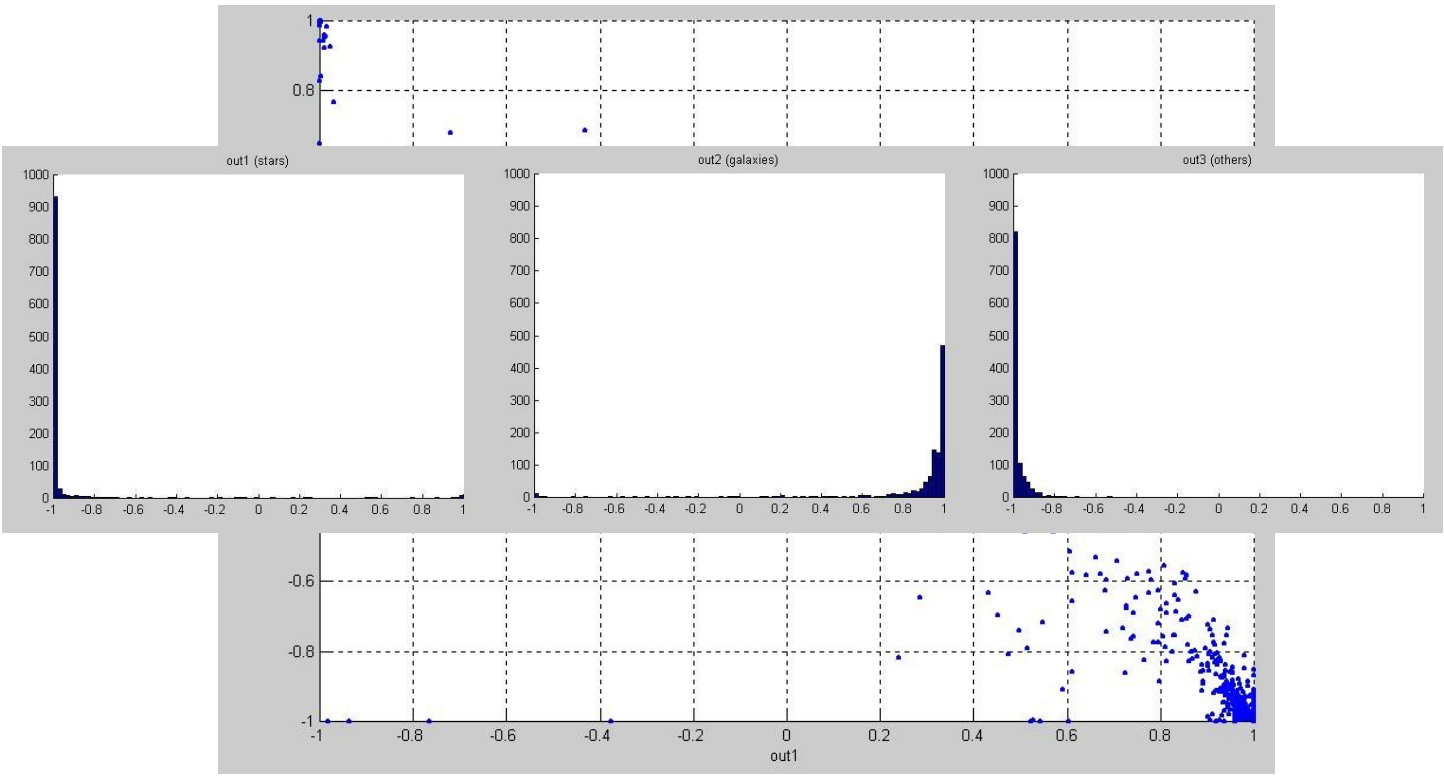


Figura 29 – Histograma de Saída da Rede para estrelas

O primeiro caso que analisaremos será o das estrelas. A figura 29 mostra a distribuição de valores na saída da rede para objetos classificados no arquivo de treinamento como estrelas. Podemos notar que uma pequena parte destes objetos foi classificado pela rede como galáxias. Isto acontece porque provavelmente estes objetos são galáxias que foram rotuladas como estrelas.

A figura abaixo mostra esta distribuição ainda mais claramente. Os objetos que estão no canto superior esquerdo são, provavelmente, galáxias rotuladas como estrelas, ao passo que os objetos que se encontram no canto inferior esquerdo são, provavelmente, outros objetos.

Figura 30 – Dispersão dos objetos rotulados como estrela no arquivo de treinamento



Vejamos agora os objetos rotulados como galáxias. A figura 30 mostra a saída dos nós de saída da rede para estes objetos.

Figura 31- Histograma de Saída da rede para galáxias

Como no caso anterior, pouquíssimos objetos foram classificados como outros objetos, estando a maior parte das classificações errôneas atribuídas a estrelas que foram rotuladas como galáxias.

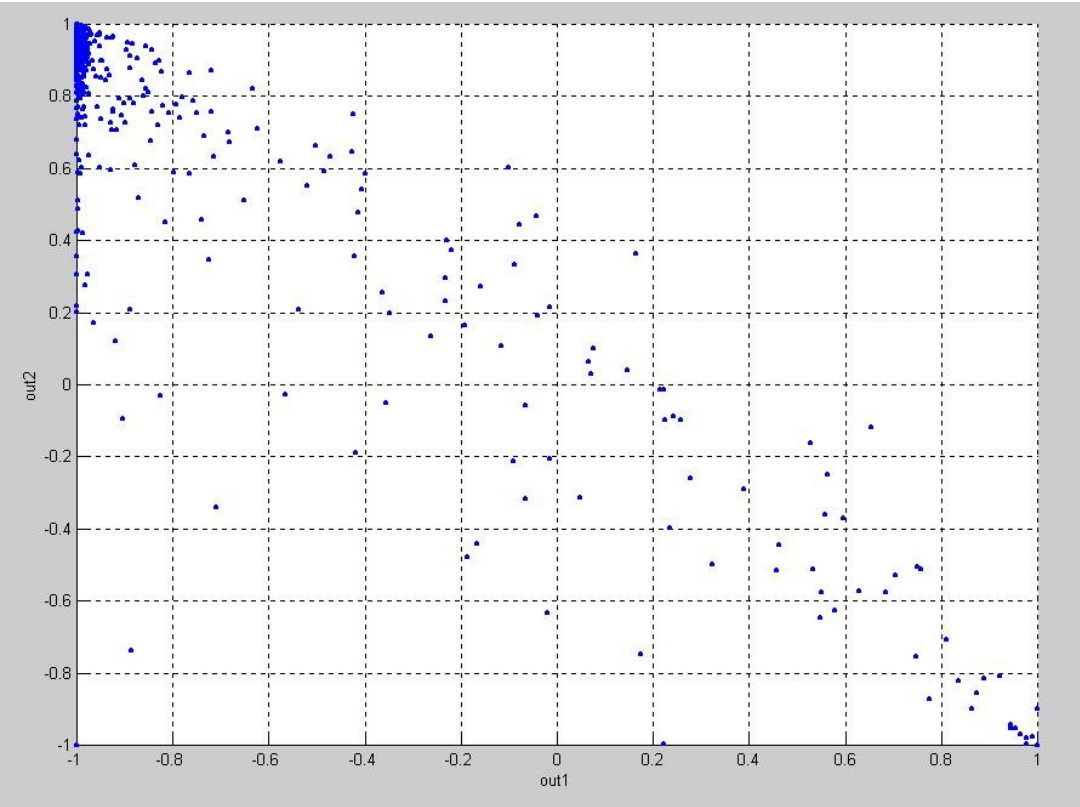


Figura 32 - Dispersão dos objetos rotulados como galáxias no arquivo de treinamento

Neste caso, os objetos que se encontram no canto inferior direito foram rotulados como galáxias quando, provavelmente, na verdade eram estrelas. Apenas alguns poucos objetos são visíveis no canto inferior esquerdo.

A figura abaixo mostra a distribuição de valores nas saídas da rede para a última classe, outros objetos. Esta classe engloba todos os objetos que não são nem estrelas e nem galáxias, podendo ser ruído na imagem, cruzes de difração ou até mesmo rastro de satélites artificiais.

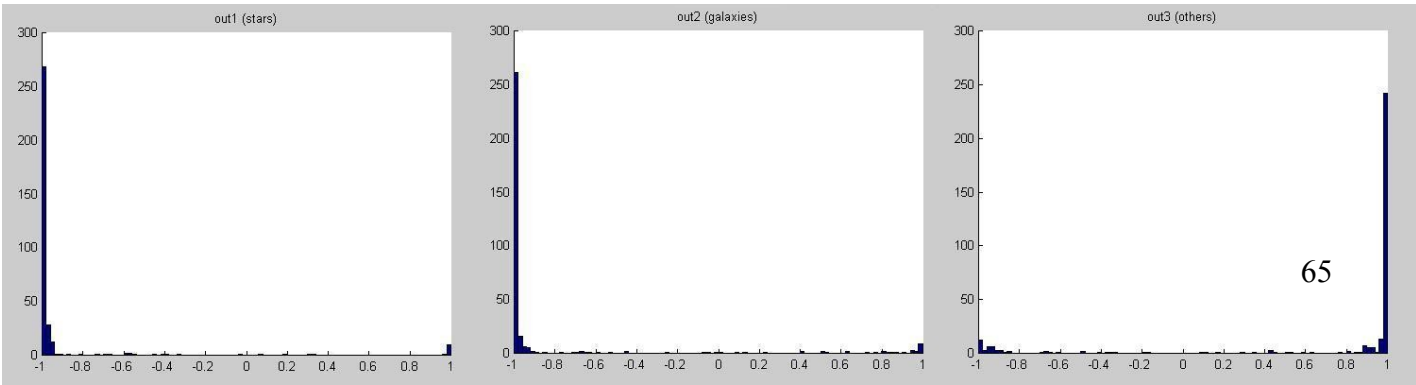


Figura 33 - Histograma de Saída da rede para outros objetos

Podemos notar por estes gráficos que a maior parte dos objetos que foram, provavelmente, erroneamente rotulados eram galáxias que foram confundidas com algum outro objeto. Isto fica ainda mais evidente nas figuras de dispersão abaixo.

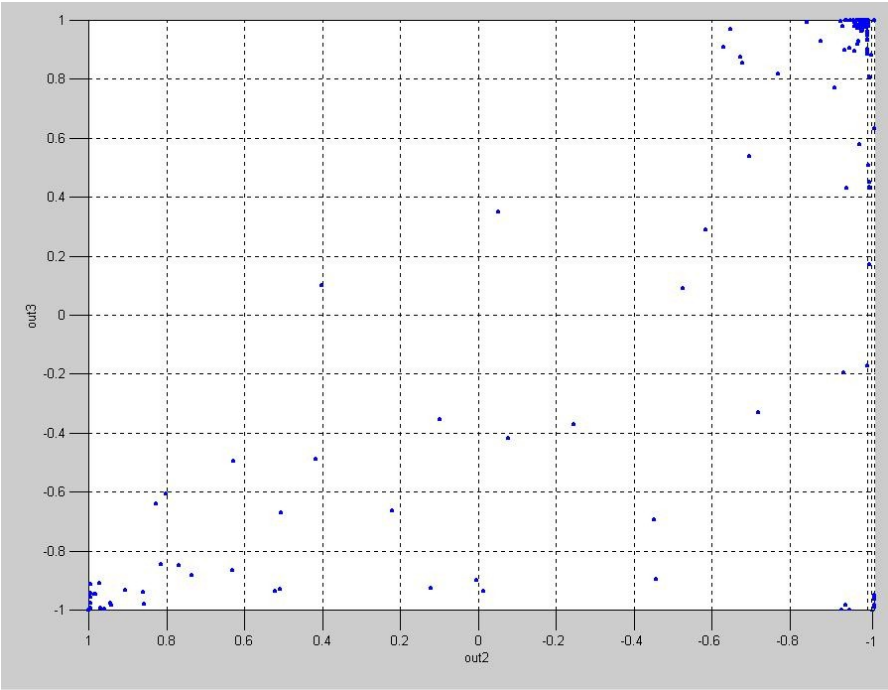
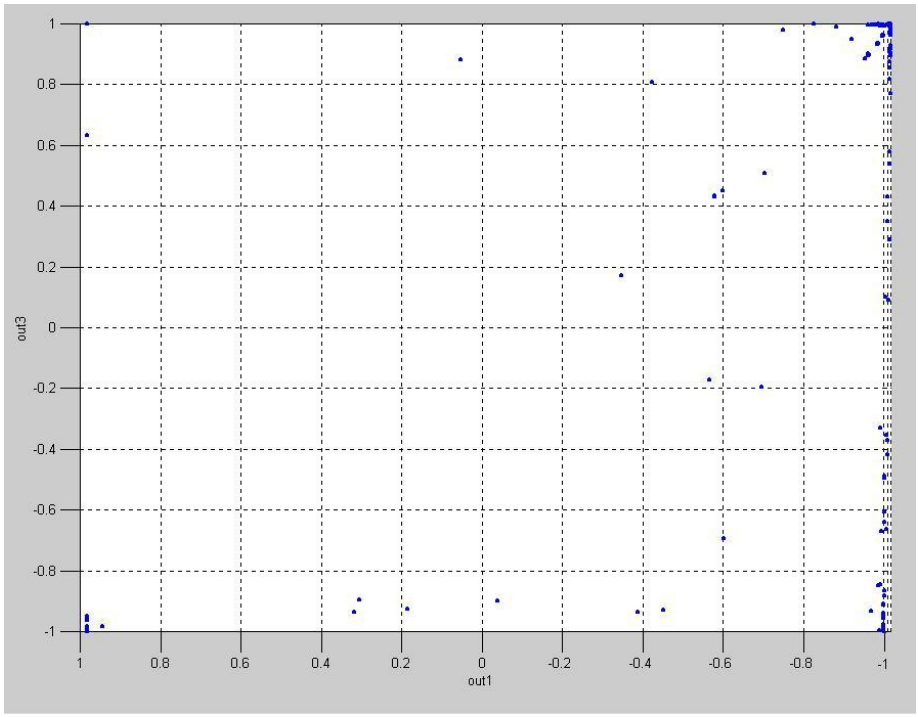


Figura 34 - Dispersão outros objetos-galáxias no arquivo de treinamento

Figura 35 - Dispersão outros objetos-estrelas no arquivo de treinamento



6.3 Comparação

Em termos absolutos, o desempenho da rede para os dados de teste ficou caracterizado da seguinte forma

OBJETO	EFICIÊNCIA (%)	ERRO (%)
Estrelas	92,66	19,65
Galáxias	88,23	10,13
Outros Objetos	81,08	2,17

Tabela 11– Eficiências e Erros totais

Os astrônomos do observatório nacional consideraram a classificação do FOCAS confiável até a magnitude m_J =21,0, pois como podemos ver pela tabela 10 sua concordância com a reclassificação visual foi de até 70% nesta magnitude.

Para melhor visualizar o desempenho da rede, este foi dividido em faixas de magnitudes na tabela abaixo

Faixas de Magnitude	Eficiência Estrelas (%)	Erro Estrelas (%)	Eficiência Galáxias (%)	Erro Galáxias (%)	Eficiência Outros (%)	Erro Outros (%)
< 20,5	100,00	13,24	96,97	0,00	82,60	0,00
20,5 – 20,7	100,00	0,00	100,00	0,00	100,00	0,00
20,7 – 21,0	90,00	0,00	100,00	16,67	87,50	0,00
21 ,0 – 21,1	100,00	0,00	100,00	0,00	100,00	0,00
21,1 – 21,3	100,00	16,67	81,82	0,00	100,00	0,00

21,3 – 21,6	100,00	25,00	90,48	5,00	88,00	0,00
21,6 – 22,0	100,00	16,67	92,68	2,56	94,12	0,00
22,0 – 22,3	86,67	13,33	92,59	19,35	42,85	0,00
22,3 – 22,5	90,91	37,50	70,37	9,52	66,67	50,00
>22,5	58,82	50,00	80,49	23,26	28,57	0,00

Tabela 12 – Eficiências e Erros através das faixas de magnitude

As figuras abaixo são uma representação gráfica da tabela acima.

Figura 36 – Relação entre eficiência e faixas de magnitude

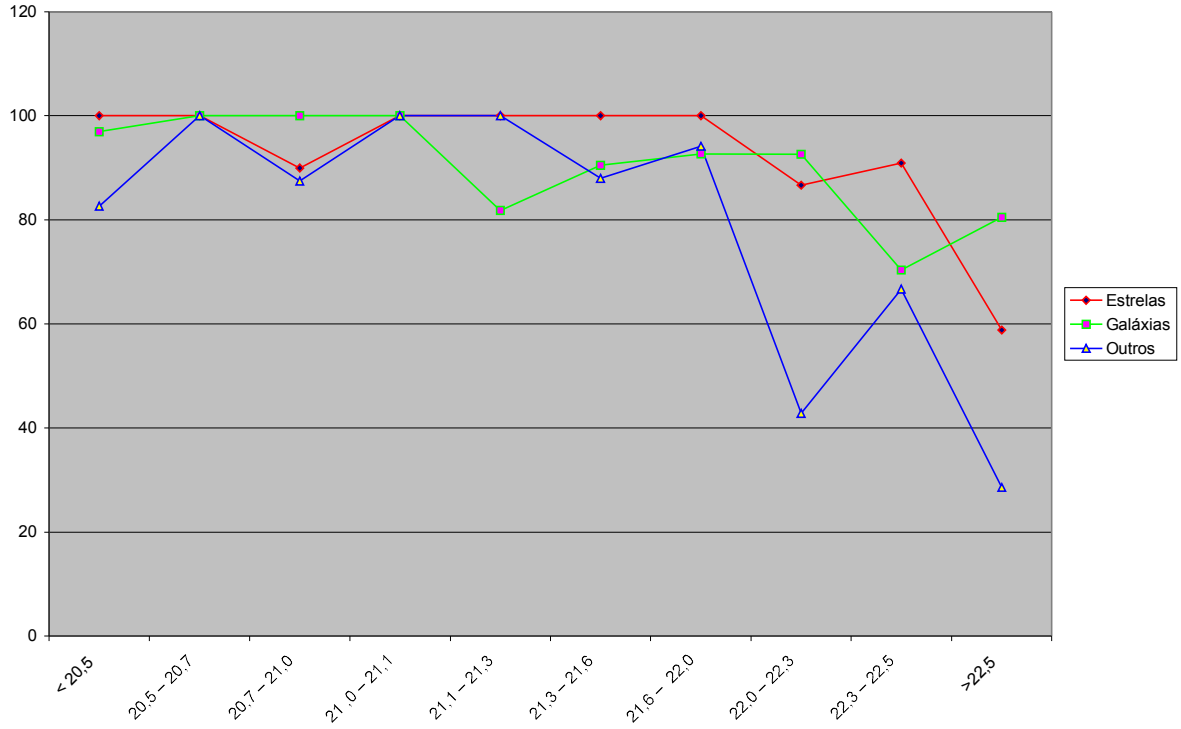
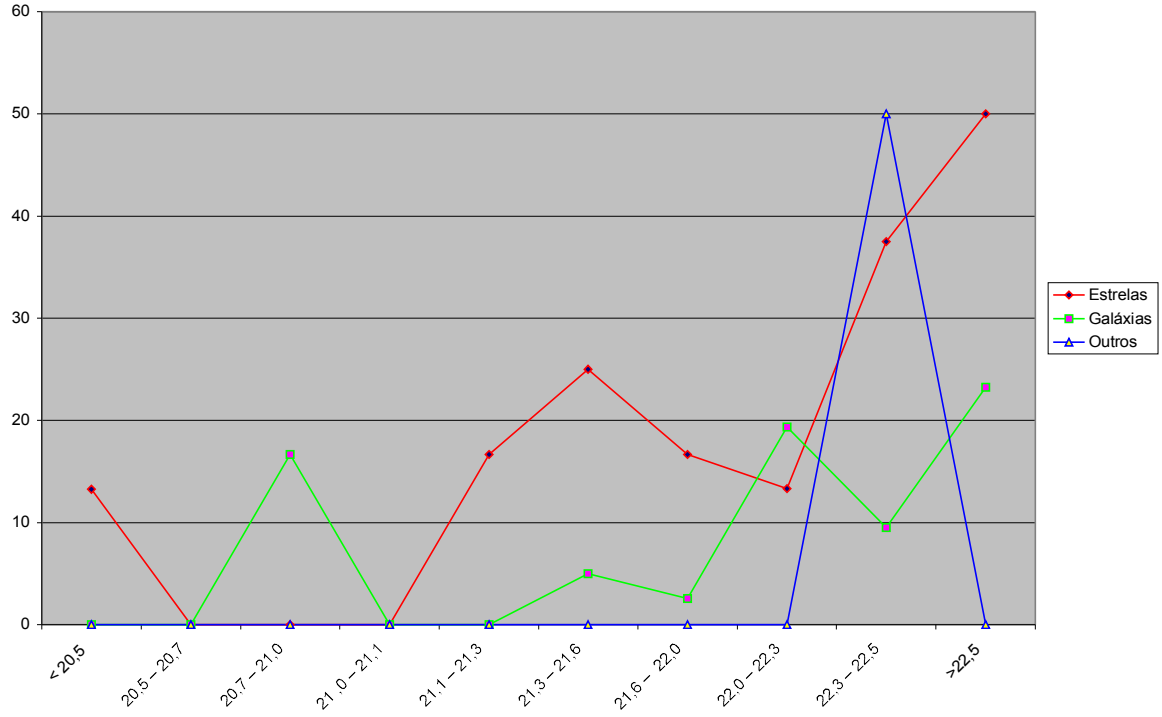


Figura 37 – Relação entre o erro e as faixas de magnitude



Pela tabela e pelas figuras acima, podemos fazer algumas considerações. Como já foi descrito na seção 5.4.2, a eficiência de classificação nos permite visualizar quantos objetos de uma classe a rede foi capaz de identificar.

Embora este parâmetro seja útil para o treinamento da rede, o parâmetro mais importante para se perceber o desempenho da rede seria o erro de classificação. Apesar da eficiência e do erro estarem intrinsecamente ligados (se todas as eficiências fossem 100%, o erro seja 0% para todas as classes), é através do erro que podemos dizer se a rede é confiável ou não.

Uma classe que apresente até 30% de erro para uma dada faixa de magnitude nos diz que, de todos os objetos que a rede nos apresenta como pertencentes àquela classe naquela faixa, 70% deles estão certos.

Se usarmos o mesmo critério utilizado para a determinação da eficiência do FOCAS podemos considerar que a rede é confiável até a magnitude $\boldsymbol{m_J}$ = 22,3 para todas as classes, sendo que no caso particular das galáxias, a classificação se mantém confiável mesmo para as galáxias acima de magnitude $\boldsymbol{m_J}$ = 22.5.

6.4 Aplicação Prática

Os dados utilizados neste trabalho são referentes a dois aglomerados de galáxias que se encontram na região da constelação de Aquárius, A2534 e A2536. A figura abaixo mostra a saída da rede para o arquivo *hope.dat* com a região do aglomerado A2534 marcada pelo retângulo azul e a do aglomerado A2536 marcada pelo retângulo vermelho.

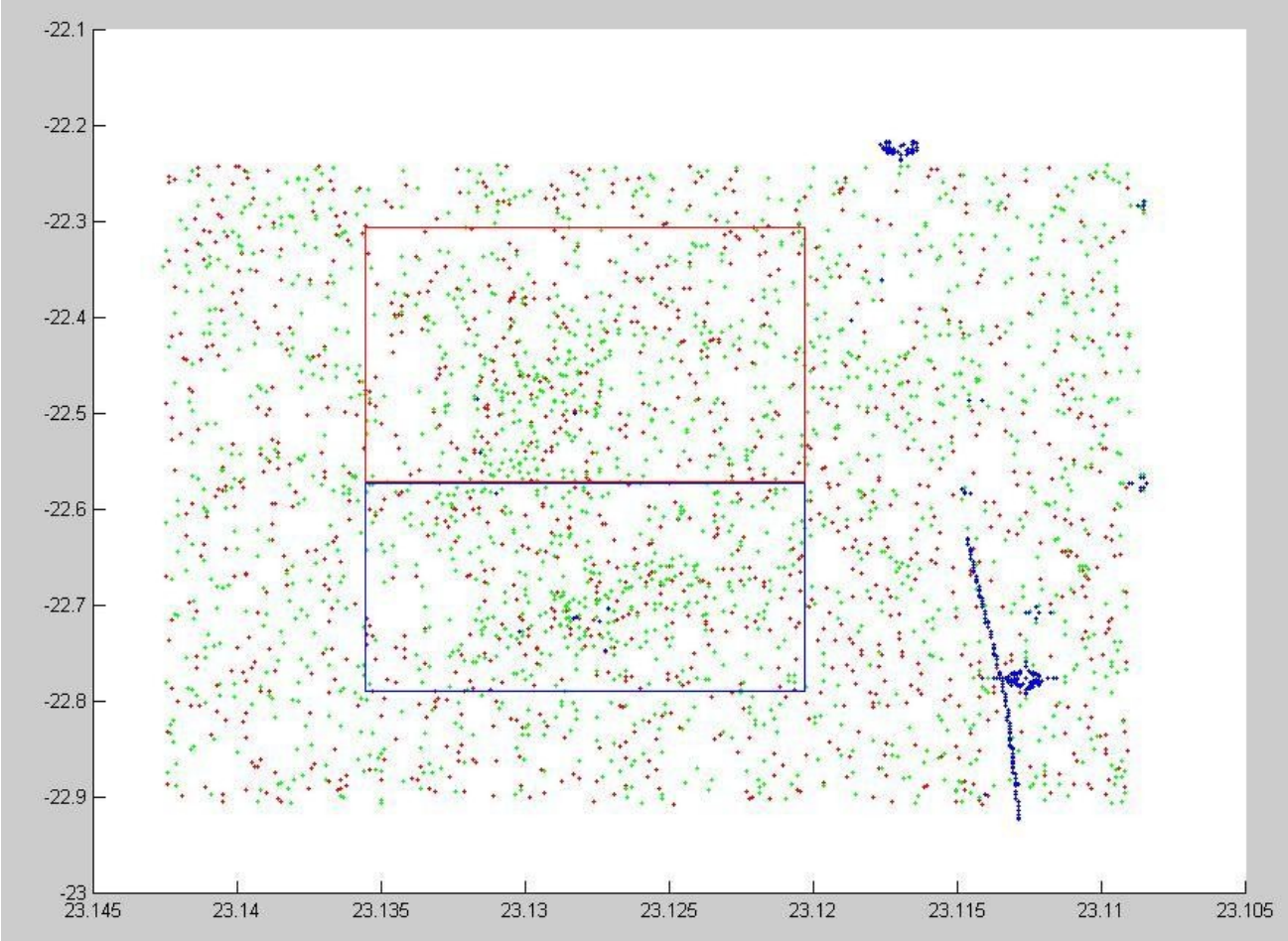


Figura 38 – Saída da rede com a área dos aglomerados identificada

Podemos notar ainda nesta imagem que a rede foi capaz de distinguir alguns outros objetos além das estrelas e galáxias. Um deles é o risco que provavelmente é o rastro de algum satélite que foi captado durante o período de exposição da placa fotográfica. Outro objeto facilmente reconhecível é a cruz de difração de uma estrela.

7 Conclusão

Com base nos resultados obtidos, podemos dizer que a rede neural teve um desempenho superior ao *software* de classificação FOCAS, especialmente para as magnitudes maiores.

Porém, apesar de todo o cuidado na seleção dos dados, é recomendável um inspeção visual para a certificação da classificação da rede, mesmo porque, embora o arquivo de treinamento tenha sido analisado de forma a reduzir ao máximo sua incerteza, este ainda está sujeito a erros que podem estar se refletindo na eficiência da rede. Podemos até cogitar que o baixo desempenho da rede para grandes magnitudes pode ser por culpa dos dados de treinamento, e não da rede em si.

Uma outra forma de tentar melhorar o desempenho da rede seria o de voltar sua atenção para as magnitudes maiores, especializando a rede nesta região. Poderia ser feita ainda uma filtragem nos dados de treinamento com base nos resultados obtidos pela rede, retirando-se do arquivo os objetos que, embora tenham sido previamente rotulados como um objeto, foram classificados com bastante certeza pela rede como pertencente a uma classe diferente.

Esta rede apresenta ainda vantagens sobre o *software* SExtractor[1] por ser capaz de identificar objetos em placas fotográficas com resolução inferior às imagens simuladas. Além disso, por ter sido treinada com base em fotos e não simulações, esta rede é capaz de identificar outros objetos que não sejam estrelas ou galáxias, como rastros de satélites e cruzes de difração, por exemplo, e agrupá-los em uma classe separada.

Seria desejável também uma análise sobre o resultado final para identificar se há algum padrão de erros sistemáticos.

8 Bibliografia

1 - Bertin, E., Arnouts, S. - SExtractor: Software for source extraction, Astronomy & Astrophysics supplement series 117, 1996, 393-404pp;

2 - Cardoso, S. H., Sistema Nervoso, 2000, http://www.nib.unicamp.br/svol/cerebro.html ;

3 - Caretta, C. A. - Estudos dos aglomerados de galáxias A2534 e A2536., Rio de janeiro, Trabalho de obtenção do grau de Astrônomo, Observatório Nacional, 1995;

4 - Caretta, C. A. et al - A quantitative evaluation of the galaxy component of the COSMOS and APM catalogs, The Astronomical Journal 119, 1999, 524-535pp;

5 - Caretta, C. A. - Análise dinâmica dos aglomerados e super aglomerados de galáxias da região de Aquárius, Rio de janeiro, Tese de Doutorado, Observatório Nacional, 2002

6 - Chaisson, E., McMillan, S. - Astronomy Today 2E, Media Edition, 1997;

7 - Charge-Coupled Device (CCD) Imaging Arrays, 1997, http://ranier.hq.nasa.gov/Sensors_page/DD/HST&GLL_CCD.html ;

8 - Haykin, S. - Redes Neurais: princípios e práticas, 2ª Ed., Editora Bookman, 2001;

9 - Jarvis, J. F., Tyson, J. A. - FOCAS: Faint object classification and analysis system, The Astronomical Journal Volume 86, 1981, Número 3;

10 - McCulloch,W.S., Pitts, W. - A logical calculus of the ideas immanent in nervous activity, Bulletin of Mathematical Biophysics Volume 5, 1943, 115-133pp;

11 - Minsky, M.L., Papert, S.A. - Perceptrons, Cambrige, MA, MIT Press, 1969;

12 - Novo sensor digital revoluciona astronomia, 2002, http://www.inovacaotecnologica.com.br/noticias/010110020816.html ;

13 - Rumelhart, D.E., Hinton, G.E., Williams R.J. - Learning representations of back-propagation errors, Londres, Nautre Volume 323, 1986a, 533-536pp;
14 - Seixas, J.M. et al - A Neural Discriminator Capable to Identify Impurities in the Data Sample;
15 - Seixas, J.M. et al - Identifying Relevant Features for Real-Time Neural Discriminating Process Under High-Event Rate Conditions;
16 - Werbos, P.J. - Beyond regression: New tools for prediction and analysis in behavioral sciences, Cambrige MA , Ph.D. Thesis, Harvard University, 1974.

Apêndice A – CD

Dados

Documentação

Artigos

Projeto

Programas

Código Fonte

MATLAB

Executável (Windows)