INEB- PSI Technical Report 2007-1

Ferramenta computacional para implementação de redes neuronais: a primeira versão.

Alexandra Oliveira
aao@fe.up.pt
Professor Joaquim Marques de Sá

June 2007
INEB - Instituto de Engenharia Biomédica
FEUP/DEEC, Rua Dr. Roberto Frias, 4200-645 PORTO

Conteúdo

Capítulo 1

Motivação para o desenvolvimento do software

O cérebro sempre se apresentou como um grande mistério para o homem. A capacidade de processamento das informações recebidas pelo exterior reveste-se de grande complexidade. Por isto, surgiram inúmeros trabalhos relacionados com as capacidades cerebrais. Estes estudos do comportamento cerebral desenvolveu no homem o interesse na criação de sistemas artificiais que mimetizem algumas das suas potencialidades. Assim, e entre outras ferramentas, apareceram as redes neuronais artificiais.

O sistema nervoso dos seres vivos é composto por uma intrincada redes de unidades básicas de processamento da informação. Estas unidades básicas são os neurónios; uma representação destas estruturas encontra-se na figura 1.1. Um neurónio biológico recebe impulsos eléctricos através das dendrites de outros neurónios a ele ligados. Após processar esse impulso transmite-o a outros neurónios através dos axônios. "O espaço entre as dendrites de um neurónio e o axônio de outro é o que chama uma sinapse: os sinais são transportados através das sinapses por uma variedade de substâncias químicas chamadas neurotransmissores"(em [1]). Quando um neurónio recebe um impulso nervoso ocorrem alterações na polaridade dentro destas estruturas desencadeando-se um processo de "alterações bruscas e intensas do potencial da sua membrana"(ver [2]) fazendo com que outros neurónios sejam estimulados.

De forma análoga, uma rede neuronal artificial é uma rede composta por pequena unidades de processamento - os neurónios artificiais - ligadas entre si. Cada neurónio pode receber estímulos quer do exterior - os inputs - quer de neurónios que a ele estão ligados. Estes estímulos são processados dentro do neurónio através de uma função matemática que o caracteriza. A cada ligação entre neuró-

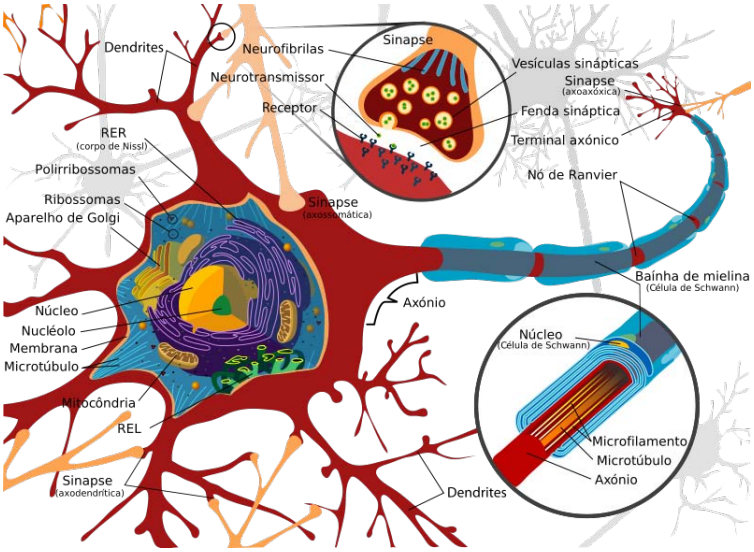

Figura 1.1: Esquema de um neurónio. Esquema elaborado por Mariana Ruiz e permissão de uso livre sem restrições.

nios está associado um peso que, à semelhança dos neurotransmissores, irá influenciar a estimação do neurónio receptor por parte dos neurónios transmissores a ele ligado. Assim, a cada neurónio em fase de recepção chega uma soma pesada do processamento feito pelos neurónios responsáveis pela sua estimulação.

Tal como no caso do sistema nervoso central de um ser vivo, as estruturas das redes neuronais artificiais podem ser deveras complexas. Uma das mais simples pode ser encontrada representada na figura 1.2 em que os neurónios estão estruturadas por camadas e em que o fluxo de informação se processa da camada de entrada (I) até à saída (O) sem nunca existir um retrocesso - rede MultiLayer Perceptron (MLP) feedfoward.

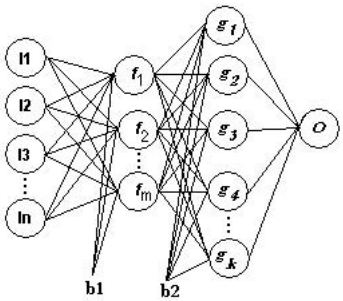

Figura 1.2: Rede Neuronal por camadas do tipo feedfoward.

Tal como os animais, as redes neuronais artificiais podem aprender. Esta aprendizagem é feita

por diversos algoritmos que, de forma simplificada, consistem na alteração dos pesos associados às ligações entre os neurónios de acordo com exemplos apresentados à rede neuronal.

A forma como os neurónios artificiais se podem organizar e o algoritmo utilizado para a aprendizagem é, neste momento, a área científica a que se dedica o grupo no qual estou inserida - o NNIG - e que fomentou o aparecimento deste software.

Capítulo 2

Porquê mais um software?

Como é do conhecimento da comunidade científica da área da inteligência artificial, existe no mercado várias ferramentas de software, algumas delas de livre utilização, que implementam algoritmos de redes neuronais artificiais. Sendo assim, surge a questão sobre a utilidade da implementação de mais uma?

Como o grupo NNIG - Neural Network Interest Group da Divisão de Sinal e Imagem do INEB - Instituto de Engenharia Biomédica, ao longo da sua existência desenvolveu algoritmos inovadores na área das redes neuronais artificiais e que, portanto, não se encontram nas ferramentas de redes neuronais existentes, a criação de uma ferramenta computacional nova tornou-se uma necessidade. Assim nasceu este projecto: a criação de uma ferramenta computacional para implementação de algoritmos de redes neuronais com uma interface gráfica de fácil utilização e que integre os algoritmos inovadores desenvolvidos pelo grupo.

Esta ferramenta está a ser implementa em Microsoft Visual C# e tem uma estrutura modular. Estes módulos, re-utilizáveis, representam características diferentes das redes neuronais e que podem ser combinados formando situações de desenvolvimento de redes neuronais distintos.

Capítulo 3

Acknowledgments

O desenvolvimento deste software é suportado pela FCT (Fundação para a Ciência e Tecnologia) e FEDER sob o projecto $POSCEIA569182004$.

Este software foi inspirado no projecto *Neural Network Library Version 0.1 (April 2002)* de *Franck Fleurey*. Este projecto foi distribuído sob a licença de uso: GNU general public license. Este projecto é constituído por duas livrarias: *NeuralNetworkLibrary* e *NeuralNetworkGUI*. Se no caso da livraria *Neural Network Library* apenas foram acrescentadas classes e foram feitas pequenas alterações, o mesmo não aconteceu com a interface que o autor desenvolveu. Esta interface serviu de inspiração de alguns componentes gráficos que estão a ser desenvolvidos.

A livraria ZedGraph desenvolvida por John Champion é a livraria usada no nosso software para desenhar gráficos 2D. Esta livraria é formada por um conjunto de classes, escritas em C#, disponibilizando várias opções de gráficos e várias funcionalidades que se revestem de grande importância para o desenvolvimento deste projecto.

Uma caixa de texto que faz a validação de diferentes tipos de representações numérica é também usado neste software e o seu desenvolvimento deve-se a Oscar Bowyer.

O projecto do NNIG - Neural Network Interest Group conta ainda com uma livraria da autoria Chester Ragel que consiste num texto que pode ser colocado na posição que se desejar.

Uma outra contribuição de relevo para o nosso software é da autoria de Yossi Rozenberg e consiste no repositórios de diferentes algoritmos que calculam diferentes estatísticas para um dado conjunto de valores.

Dentro da área da estatística falta ainda realçar o trabalho do CenterSpace Software cuja sua contribuição se encontra num conjunto de rotinas que implementam um histograma.

Até este momento, está incluído no software do grupo NNIG um projecto chamado ChartLibrary

do qual não se conhece autoria e que em breve será retirado.

O desenvolvimento deste software contou ainda com a preciosa contribuição de Daniel Carrilho e Paulo Ricca ao nível da programação de algumas funcionalidades assim como com a contribuição de todos os membros do NNIG - Neural Networks Interest Group que testaram e deram sugestões.

Capítulo 4

A solução NNIG_ NN_SOFTWARE

Até ao momento a solução NNIG_NN_SOFTWARE, que implementa o software descrito neste relatório, é constituída por 5 projectos diferentes:

1. o ChartLibrary,
2. o NeuralNetwork,
3. o NeuralNetworkGUI,
4. o NNIG_software e o
5. RedesNeuronais_NNIG.

No diagrama (4.1) encontra-se como se organizam os cinco projectos, sendo que, no diagrama, o símbolo → significa "é um componente de".

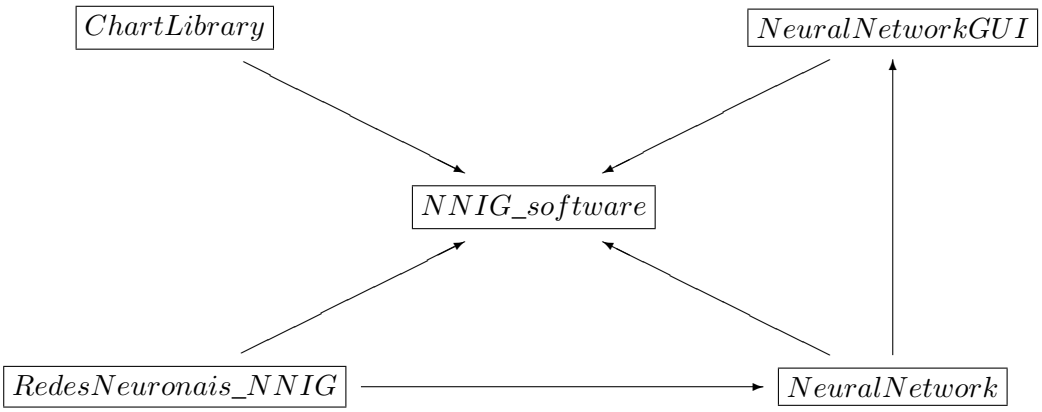

Figura 4.1: Dependências entre os projectos da solução NNIG_NN_SOFTWARE

Este software conta ainda com o auxílio de cinco classes desenvolvidas por investigadores externos que disponibilizaram os seus códigos. Este três projectos são:

- Numeric Edit Control;
- Oriented Text Label;
- ZedGraph;
- statistics e
- Histo.

O Numeric Edit control e o Oriented Text Label encontram-se no projecto NeuralNetworkGUI e são da autoria de Oscar Bowyer e de Chester Ragel, respectivamente. O primeiro control é constituído por uma caixa de texto que faz a validação dos dados introduzidos apenas permitindo a escrita de diferentes representações numéricas e o segundo coloca uma string com uma orientação pré-definida.

O ZedGraph foi incluído no nosso projecto de forma estática, ou seja, usando a referência à sua dll. Assim, este projecto é independente do nosso sendo-nos apenas possível usar as suas funcionalidades.

A classe statistics, como foi referido anteriormente, é um repositório de algoritmos que calculam estatísticas relevantes a partir um conjunto de dados. Esta classe encontra-se no projecto RedesNeuronais_NNIG e a sua autoria deve-se a Yossi Rozenberg.

A classes Histo, da autoria de CenterSpace Software, constrói e mantém um histograma de dados introduzidos.

Como estas três primeiras classes estão adicionadas de forma estática não permitindo a alteração das suas características não serão descrita em pormenor neste relatório.

4.1 ChartLibrary

O projecto ChartLibrary é uma livraria de livre acesso que produz gráfico de 2D. Contudo, como é de difícil utilização e com uma interface que não satisfaz os nossos objectivos, em breve será eliminada da nossa solução.

4.2 NeuralNetwork Library

Esta livraria constitui a base matemática para a implementação de uma rede neuronal. Assim, na secção seguinte, serão descritos conceitos básicos de redes neuronais que suportam os algoritmos que compõem esta livraria. Terminada esta breve introdução serão descritos aspectos técnicos relativos a estes algoritmos.

4.2.1 Conceitos básicos

Cada rede neuronal artificial é composta por camadas (**layers**) de neurónios (**neurons**) ligadas entre si. Dependendo do modo como estas camadas se encontram ligada podemos ter diferentes tipos de redes neuronais (feedforward NN, recurrent NN, etc). Até ao momento, o software que está em desenvolvimento no INEB apenas inclui o caso de redes feedfoward.

Numa rede neuronal feedfoward cada camada de neurónios está ligada à camada seguinte por sinapses permitindo que o fluxo de dados passe da primeira para a seguinte sem retrocessos. Cada camada é composta por um número arbitrário de neurónios apresentando todos as mesmas características (como por exemplo função de activação, número de dados de entrada, etc). No software que se está a descrever neste relatório considera-se que, numa rede feedfoward, cada neurónio de uma determinada camada está ligado a todos os neurónios da camada seguinte e que a estas ligações se encontra associado um peso.

Com esta ferramenta pode-se, então, construir uma rede neuronal feedfoward contendo um número arbitrário de camadas e tendo cada uma um número arbitrário de neurónios (com a excepção da camada de entrada e de saída cujo o número de neurónios depende do número de características

escolhidas para input e do número de classes descritas nos dados, respectivamente). Cada camada tem uma função de activação específica. Assim, a rede neuronal pode processar o sinal de input transferindo-o desde a camada de entrada até à camada de saída ([3]).

Após o processamento de um sinal de input pela rede neuronal, o valor de saída é comparado com um valor alvo. Com base neste confronto a rede neuronal é sujeita a uma acção correctiva com o objectivo de tornar a sua resposta tão próxima do alvo quanto possível. Às instruções que regem esta acção de aprendizagem chamamos algoritmo de treino.

4.2.2 Aspectos Técnicos

A livraria NeuralNetwork, da autoria de Franck Fleury, foi alterada no decurso do desenvolvimento do software nomeadamente acrescentando funcionalidades: funções de activação e um algoritmo de treino. Houve ainda necessidade de alterar a expressão de uma função de activação e dotar a classe que implementa a rede neuronal com a funcionalidade de retirar e acrescentar uma determinada camada de neurónios.

O objectivo deste projecto é implementar todos os algoritmos necessário para a criação de uma rede neuronal. Estes algoritmos estão organizados em classes e numa interface que se encontram descritos nas tabelas 4.1 e 4.2.

Classe	**Descrição**
Layer	Camada de uma rede neuronal
LearningAlgorithm	Classe abstracta que implementa o algoritmo de aprendizagem de uma rede neuronal
NeuralNetwork	Implementação de uma rede neuronal
Neuron	Implementação de um neurónio artificial

Tabela 4.1: Classes que compõem o projecto NeuralNetwork

Interface	**Descrição**
ActivationFunction	É uma interface de uma função de activação

Tabela 4.2: Interface incluída no projecto NeuralNetwork

A classe LearningAlgorithm é a classe base de um algoritmo de treino da qual derivam mais duas classes. Estas classes implementam um algoritmo de treino específico e encontram-se descritas na tabela 4.3.

Classe	Descrição
BackPropagationLearningAlgorithm	Algoritmo de aprendizagem supervisionado por correcção do erro
GeneticLearningAlgorithm	Algoritmo genético de aprendizagem

Tabela 4.3: Classes que derivam da classe LearningAlgorithm

Por seu lado, a classe BackPropagation é a classe base de mais duas classes que se encontram descritas na tabela 4.4.

Classe	Descrição
BatchBackPropagationLearningAlgorithm	Implementa o algoritmo de BackPropagation cuja actualização dos pesos ocorre após realizadas todas as iterações do algoritmo de treino
OnlineBackPropagationLearningAlgorithm	Implementa o algoritmo de backpropagation sequencial cuja actualização dos pesos é feita após cada iteração

Tabela 4.4: Classes que derivam da classe BackPropagation

Como a classe GeneticLearningAlgorithm ainda não se encontra testada não vai ser descrita no decurso deste relatório.

ActivationFunction.cs

A interface ActivationFunction contem dois métodos e uma propriedade. Um método permite avaliar a função dado um certo argumento e o outro permite determinar o valor da derivada da função de activação. A propriedade permite o acesso ao nome da função. Estes métodos e propriedade são implementados pelas classes descritas na tabela 4.5 .

Nome da Classe	Função
GaussianActivationFunction	Função de activação Gaussiana
HeavisideActivationFunction	Função de activação Step
HyperbolicTangentActivationFunction	Função de activação Tangente Hiperbólica
LinearActivationFunction	Função de activação Linear
SigmoidActivationFunction	Função de activação Logística

Tabela 4.5: Classes que implementam a interface ActivationFunction

As expressões matemáticas das funções de activação implementadas neste software encontram-se representadas na tabela 4.6.

Função	Expressão da Função
Gaussiana	$f(x) = \frac{e^{-\frac{(x-\mu)^2}{2\times\sigma^2}}}{\sqrt{2\pi}\times\sigma}$
Heaviside	$f(x) = \begin{cases} 0 & se \quad 0 > x \\ 1 & \text{se} \quad 0 < x \end{cases}$
Tangente hiperbólica	$f(x) = \frac{e^{\beta\times x} - e^{-\beta\times x}}{e^{\beta\times x} + e^{-\beta\times x}}$
Função Linear	$\begin{cases} 1 & se \quad x > \frac{1}{2A} \\ A\times x + \frac{1}{2} & se \quad \frac{1}{2A} > x > -\frac{1}{2A} \\ 0 & se \quad -\frac{1}{2A} > x \end{cases}$
Logística	$f(x) = \frac{1}{1+e^{-\beta\times x}}$

Tabela 4.6: Expressões das funções de activação dos neurónios

Neuron.cs

Um neurónio artificial é implementado na classe Neuron com as seguintes características por defeito:

1. os pesos são gerados aleatoriamente no intervalo $[-0.3, 0.3]$, contudo são todos inicializados com 0;

2. a função de activação é a Logística;

É de notar que estas opções por defeito podem ser alteradas de acordo com as necessidades do utilizador.

Esta classe permite o acesso:

1. ao número de inputs em cada neurónio;
2. aos pesos e bias associados ao neurónio;
3. ao valor da função de activação dado um argumento;
4. ao valor da derivada da função de activação dado um argumento;
5. ao argumento da função de activação;
6. à função de activação;
7. ao valor de pesos e bias obtidos pelo algoritmo de treino na iteração anterior.

Layer.cs

Uma camada de neurónios é implementada na classe Layer que apresenta as seguintes características por defeito:

1. os neurónios que constituem a camada têm como função de activação a função Logística assim como todos pesos inicializados com o valor 0;
2. todos os pesos associados a esta classe podem ser gerados aleatoriamente. O intervalo de onde podem ser seleccionados estes valores é igual ao intervalo definido na classe anterior;

Esta classe disponibiliza o acesso:

1. ao número de neurónios que a constituem;
2. ao número de inputs que recebe;
3. a um neurónio em particular que a constitui com todas as suas características;
4. às saídas de todos os neurónios que a constituem;
5. à função de activação que caracteriza a funcionalidade dos seus neurónios.

NeuralNetwork.cs

Esta classe implementa uma rede neuronal artificial quer no que diz respeito à sua arquitectura quer às suas funcionalidade. A classe tem por defeito as seguintes características:

1. todos os construtores desta classe consideram a primeira camada de neurónios como uma camada de processamento. A função de activação associada a cada neurónio é a função logística com parâmetro $\beta = 1$;

2. os construtores que inicializam o algoritmo de treino fazem-no com o algoritmo backpropagation;

Uma funcionalidade nova desta classe é a possibilidade de remover uma dada camada da arquitectura de rede neuronal.

Nesta classe dá-se acesso a todas as características estruturais e funcionais de uma rede neuronal do tipo Multilayer Perceptron. Esta flexibilidade permite construir uma rede neuronal com uma arquitectura arbitrária associando-lhe o algoritmo de treino que melhor se ajuste às necessidades do problema em estudo.

LearningAlgorithms.cs

Esta classe é uma classe abstracta que descreve um algoritmo de treino para uma rede neuronal.

Esta classe assenta nas seguintes características:

1. necessita de um objecto do tipo NeuralNetwork;

2. assume 0.001 como valor de erro mínimo;

3. assume 1000 como valor de número máximo de iterações;

4. guarda o valor do erro obtido na comparação entre a resposta da rede neuronal e o valor alvo.

Esta classe disponibiliza o acesso:

1. à arquitectura da rede neuronal;

2. ao erro obtido na iteração anterior;

3. ao valor mínimo estabelecido para erro;

4. ao número de iterações efectuadas pelo algoritmo;

Desta classe deriva uma outra com o nome **BackPropagationLearningAlgorithm** que tem como objectivo implementar as funções comuns às classes que implementam o Sequencial Backpropagation e o Batch Backpropagation.

Esta classe tem as seguintes características :

1. o valor da learning rate é, por defeito, 0.5 ;

2. o momentum é, por defeito, 0.2;

3. tal como a classe da qual deriva, necessita de um objecto do tipo NeuralNetwork;

4. necessita dos valores de entrada da rede e dos valores de alvo em formato *Jagged Arrays*;

5. estabelece como critério de paragem do algoritmo de treino a ocorrência de um valor de erro menor do que erro mínimo estabelecido ou o número de iterações atingir o número máximo escolhido.

Todos os valores estabelecidos por defeito podem ser alterados.
Esta classe disponibiliza o acesso aos parâmetros do algoritmo de treino:

1. learning rate e

2. momentum.

A principal diferença entre as duas classes que derivam da classe BackPropagationLearningAlgorithm é a forma como processam a actualização dos pesos associados às ligações entre os neurónios da rede neuronal: o BatchBackPropagationLearningAlgorithm (que implementa o algoritmo de Batch Backpropagation) actualiza os pesos após o processamento de todos os dados disponíveis para treino enquanto que a OnlineBackPropagationLearningAlgorithm (que implementa o algoritmo de Sequencial Backpropagation) o faz após cada exemplo analisado.

4.3 NeuralNetworkGUI

Este projecto contem as componentes gráficas desenvolvidas por Fleury que apenas servem de controlo e de inspiração para o nosso trabalho e, por isso mesmo, não será descrita.

4.4 NNIG_software

Este projecto contem classes desenvolvidas com o objectivo de implementar uma interface gráfica "user-frendly" para uma utilização agradável de todas as ferramentas do projecto NeuralNetwork.

Muitas das componentes aqui desenvolvidas foram inspiradas nas componentes que constam do projecto NeuralNetworkGUI.

4.4.1 Aspectos Técnicos

Este projecto é composto pelas classes referidas na tabela 4.7.

Classe	Descrição
allDataPreview	Dispõe, numa tabela, todos os dados de entrada disponíveis para a rede neuronal
BorderPoints	Guarda a informação sobre os pontos que compõem a fronteira entre as classes representadas nos dados (determinada pela rede neuronal)
ErrorChart	Representação gráfica do erro de treino
ErrorGraph	Representação gráfica do erro de treino
NNIGActivationFunctionChooser	Control que permite a escolha de uma função de activação para uma determinada estrutura de processamento
NNIGActivationFunctionGUI	Janela de suporte do control NNIGActivationFunctionChooser
NNIGBackPropagationGUI	Permite ao utilizador escolher o algoritmo de treino assim como os seus parâmetros. Permite ainda visualizar o erro obtido durante o treino e o número de iterações realizadas pelo algoritmo
NNIGClassificationMatrix	Permite ver como se distribuem as classificações realizadas pela rede neuronal após o treino
NNIGDecisionBorder	Permite visualizar a curva que separa pontos da mesma classe
NNIGFunctionPreview	Representação gráfica das funções de activação
NNIGInputDataStructureGUI	Control onde se pode carregar os dados para a rede neuronal assim como fazer um pré-processamento dos mesmos
NNIGLayerGUI	Control que permite visualizar e alterar todas as características de uma determinada camada de neurónios da rede neuronal
NNIGLayerProperties	Janela que suporta o control NNIGLayerGUI

NNIGNeuralNetworkEditor	Control para editar todas as características estruturais de uma rede neuronal
NNIGNeuronGUI	Representação esquemática de um neurónio
NNIGSoftware	Control principal que suporta todos os outros e que contem a janela principal do programa
viewoutput	Apresenta uma matriz com os valores de saída da rede neuronal

Tabela 4.7: Classes do projecto NNIG_Software

Control NNIGSoftware

Este control é o control principal da ferramenta computacional que se está a desenvolver no INEB - Instituto de Engenharia Biomédica. Esta janela encontra-se representada na figura 4.2.

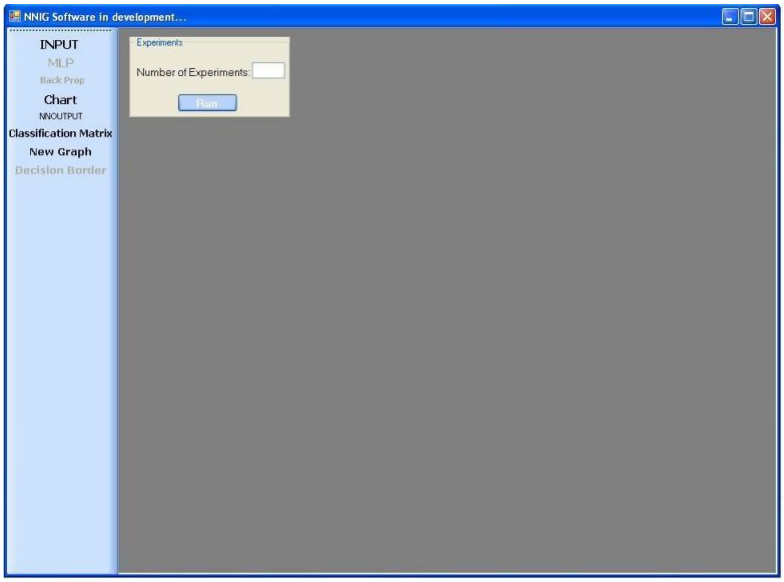

Figura 4.2: Janela principal do software

Esta janela é constituída por uma barra de ferramentas que cria todas as componentes gráficas necessárias para a implementação de uma rede neuronal e para a visualização dos resultados obtidos com o treino da rede; por um painel onde se pode introduzir o número de experiência que se deseja fazer e por um botão que inicia o processo de treino.

A barra de ferramentas contem 8 botões estando 3 deles inicialmente inactivos. Esta restrição foi implementada devido à existência de uma grande dependência entre os controlos. Esta característica irá ser alterada em breve. Estes botões serão descritos com pormenor nas secções seguintes.

NNIGInputDataStructure

O primeiro botão da barra de ferramentas da janela principal do software é o botão Input. Este botão cria na área de trabalho desta janela um controlo do tipo NNIGInputDataStructure que se encontra representado na figura 4.3.

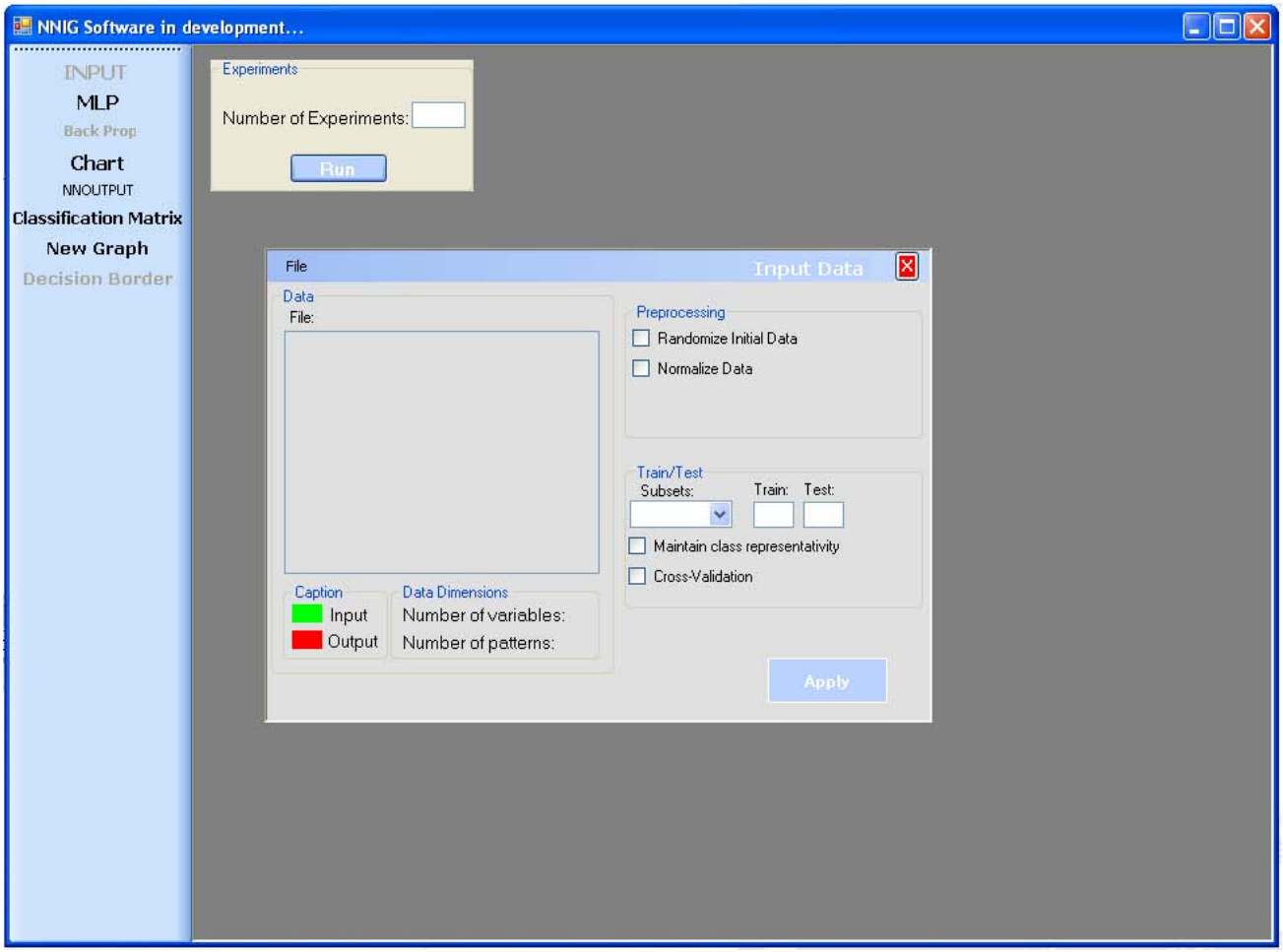

Figura 4.3: Control NNIGInputDataStructure

Este controlo proporciona a visualização dos dados assim como permite ao utilizador fazer um pré - processamento a estes valores de entrada. No caso dos dados possuírem muitos padrões serão representados apenas 7.

No menu File pode-se encontrar uma opção - Open - que permite abrir os ficheiros que contêm os dados a serem usados pela rede neuronal.

Estes ficheiros devem obedecer às seguintes especificações para que a ferramenta consiga lê-los:

1. deve ser do tipo txt;

2. em cada linha representar um padrão;

3. as características de cada padrão devem ser separadas por ponto e vírgula (representado pelo símbolo ;) ;

4. os ficheiros não podem conter nenhuma linha em branco.

Após a leitura do ficheiro o programa automaticamente estabelece a última coluna como a coluna contendo a referência às classes de cada exemplo e as outras colunas como sendo as colunas contendo as características que se pretende estudar. Esta pré-especificação pode ser alterada pelo utilizador seleccionando a coluna que pretende alterar e clicando com o botão direito do rato. Após estas acções irá aparecer um menu com as opções disponíveis, como se ilustra na figura 4.4.

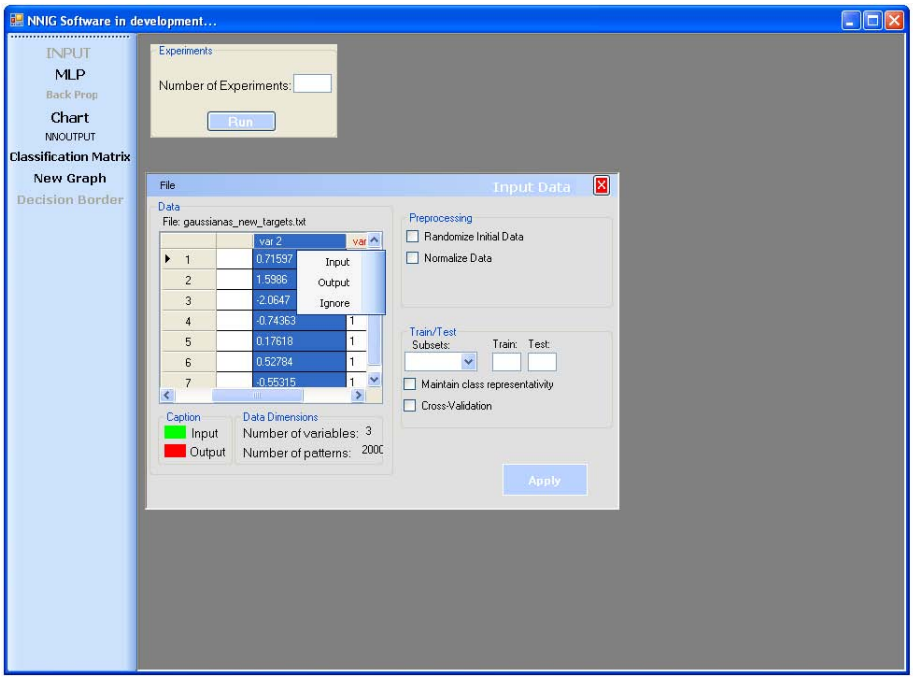

Figura 4.4: Alteração das assunções feitas pelo programa para as variáveis

No caso em que os ficheiros de dados contêm muitos exemplos optou-se por representar apenas sete linhas de exemplos. Esta opção foi tomada numa tentativa de evitar um uso desnecessário e excessivo dos recursos computacionais. Caso o utilizador deseje visualizar todos os dados basta clicar duas vezes na tabela que prontamente abre um controlo com todos os dados. Este controlo encontra-se representado na figura 4.5 .

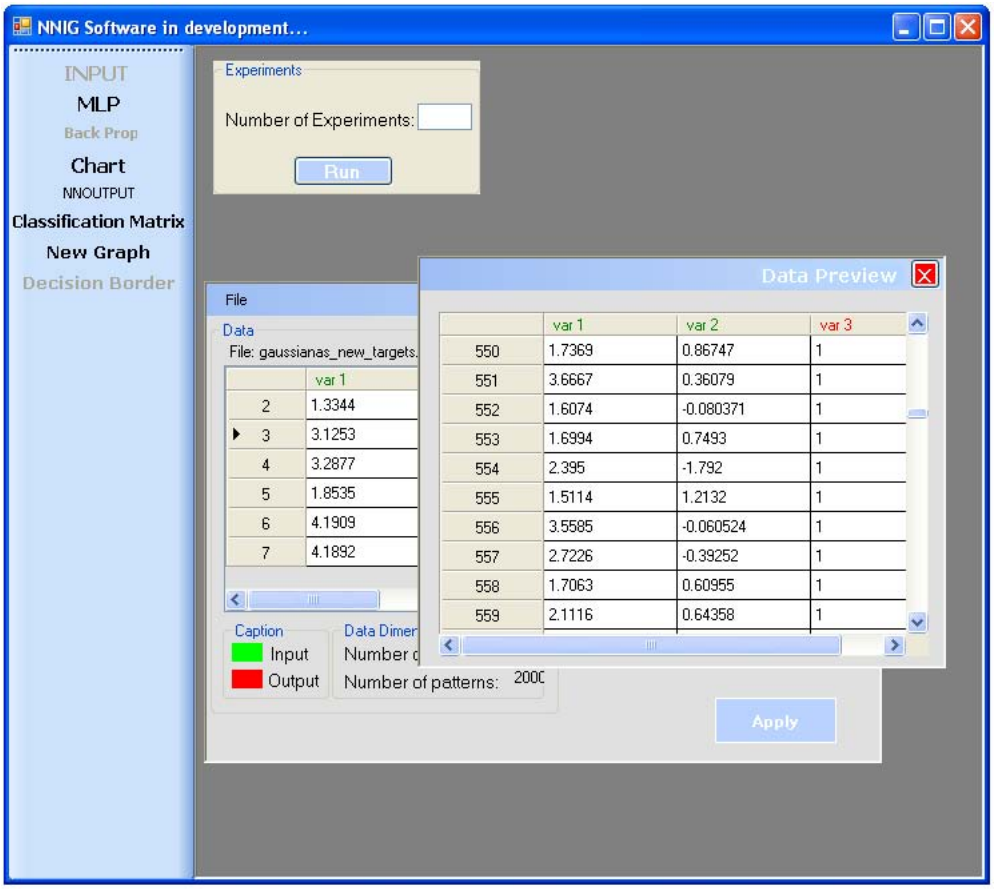

Figura 4.5: Representação de todos os dados de entrada

No controlo Input Data pode-se ainda fazer um pré-processamento dos dados de duas formas distintas:

1. re-ordenar as linhas de exemplos de forma aleatória e
2. normalizar as colunas. Esta normalização, por seu lado, pode ser feita de três formas distintas:
   (a) projectar os valores dados para um intervalo pré-definido pelo utilizador;
   (b) centrar os dados à volta de média e,
   (c) standardizar (ver figura 4.6).

Na secção Train/Test deste controlo pode-se escolher o número de conjuntos em que se quer dividir os dados originais e distribuir esses conjuntos por dois novos conjuntos: um para treino e outro para teste. Nesta distribuição dos dados originais pode-se optar por manter ou não, em cada conjunto, a representatividade de cada classe e optar por correr ou não o algoritmo de cross-validation.

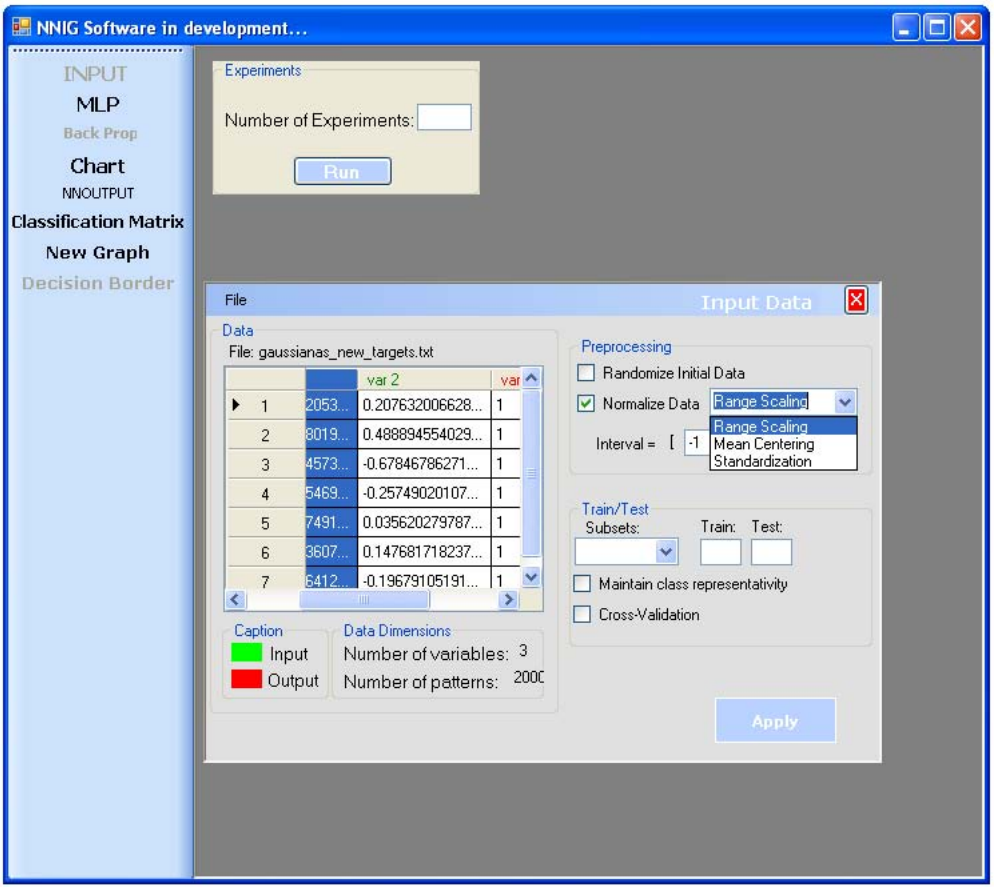

Figura 4.6: Opções disponíveis para normalização dos dados

NNIGNeuralNetworkEditor

Quando o controlo Input Data é criado, o botão MLP da barra de ferramentas da janela principal é imediatamente activado. Ao clicar neste botão aparece na área de trabalho um controlo relativo à estrutura de uma rede neuronal artificial do tipo MLP - Multilayer Perceptron Feedfoward. Este controlo está representado na figura 4.7.

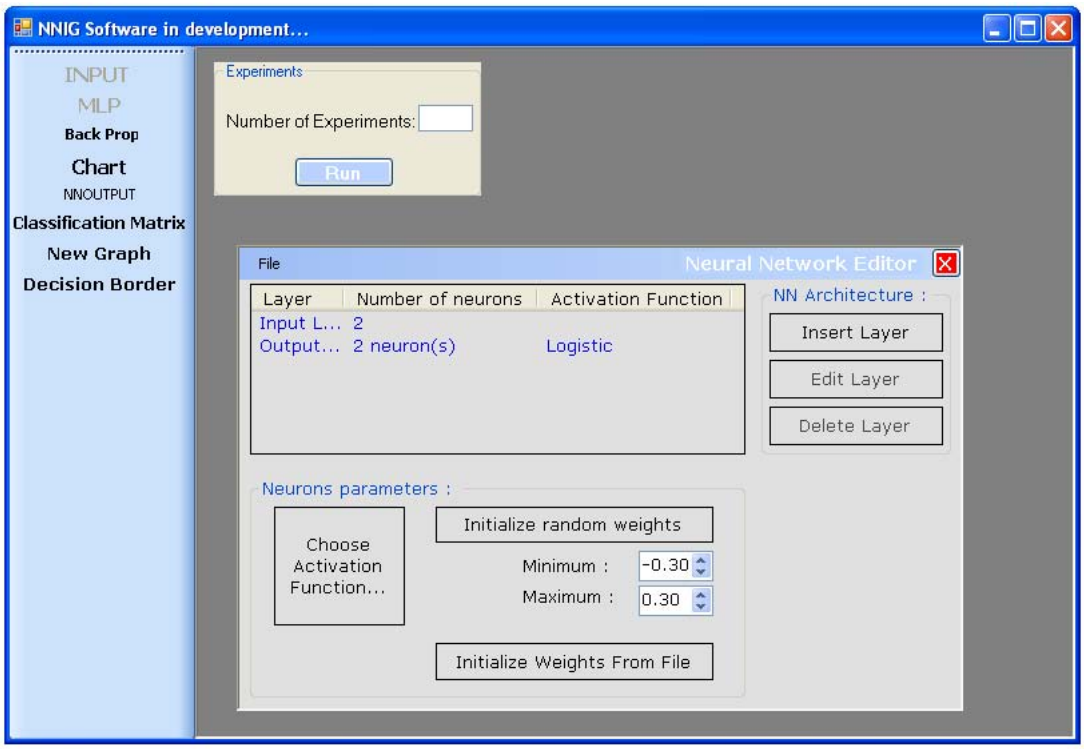

Figura 4.7: controlo Neural Network Editor

Por defeito é criada uma rede neuronal com apenas duas camadas: a camada de input e a camada de output. O número de neurónios em cada uma destas camadas é determinada de acordo com o número de colunas consideradas para entrada (colunas marcadas como Input) no controlo Input Data e com o número de classes contidas no ficheiro de dados e representadas na coluna Output, respectivamente. Nesta arquitectura, a única camada de processamento - a camada de output - tem como função de activação a função logística.

Esta arquitectura pode ser alterada introduzindo novas camadas de neurónios com número arbitrários de neurónios, como sugere a figura 4.8. Cada nova camada é criada com função de activação logística.

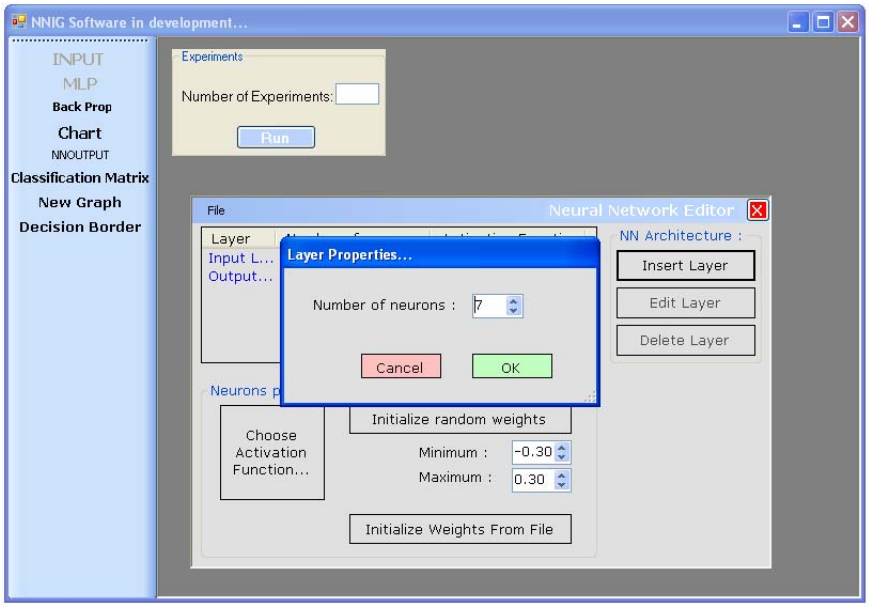

Figura 4.8: Criação de uma nova camada de neurónios

Ao seleccionar uma camada escondida da rede neuronal ficam activas duas opções: Edit Layer e Delete Layer. Tal como as designações sugerem, a primeira permite alterar o número de neurónios dessa camada enquanto que a segunda permite eliminar esta camada da arquitectura da rede neuronal.

Uma outra funcionalidade deste software é permitir aceder às propriedades de uma determinada camada de neurónios com capacidades de processamento. Para aceder a estas funcionalidades basta seleccionar uma camada que satisfaça os requisitos anteriores e clicar duas vezes. Aparecerá o controlo representado na figura 4.9.

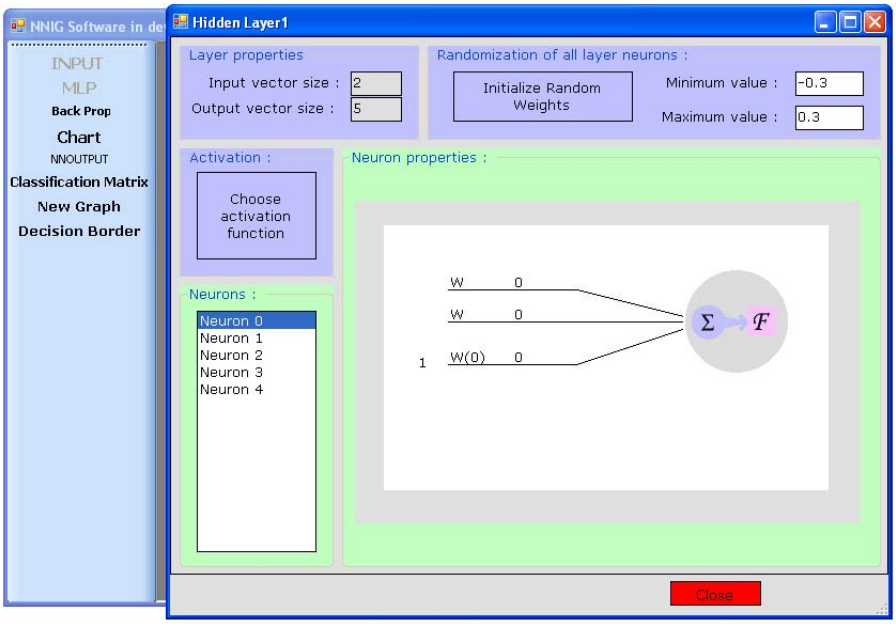

Figura 4.9: Propriedades de uma camada de neurónios

No grupo *Layer Properties* encontra-se manifesto o número de neurónios da camada anterior e o número de neurónios da própria camada em "Input Vector Size" e em "Output Vector Size", respectivamente. No grupo *Randomization of all layer neurons* encontra-se a opção de gerar pesos aleatórios para esta camada no intervalo especificado. No grupo *Activation* encontra-se um botão que permite escolher uma função de activação para todos os neurónios da camada. No grupo *Neuron properties* está representado esquematicamente o neurónio que está seleccionado na lista *Neurons*, assim como os pesos (W) e bias ($W\,(0)$) que lhe estão associados. Estes pesos são, por defeito e antes do treino, nulos mas podem ser alterados, por exemplo e para apenas esta camada, através do botão *Initialize Random Weights*.

No controlo Neural Network Editor (representado na figura 4.7) pode-se ainda alterar as funções de activação associadas a todos os neurónios de processamento da rede neuronal. Para isso basta clicar no botão *Choose Activation Function* que de imediato abre a janela representada na figura 4.10.

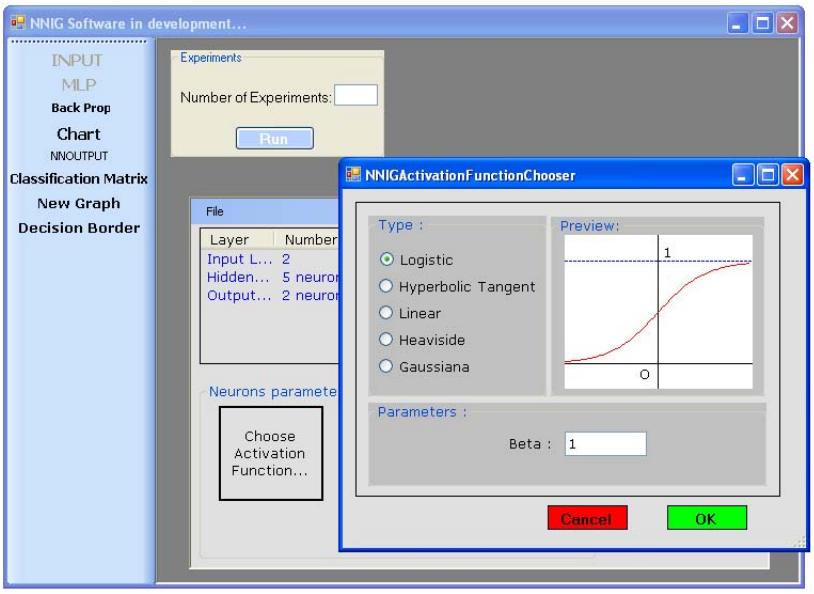

Figura 4.10: controlo Activation Function Chooser

Neste controlo estão representadas cinco famílias de funções mais usadas no desenvolvimento de trabalhos usando redes neuronais. Aqui, pode-se ver a representação gráfica de cada uma das funções assim como alterar os parâmetros específicos de cada função.

Todos os pesos iniciais da rede neuronal, que por defeito são nulos, podem ser alterados de duas formas diferentes:

1. ou são inicializados de forma aleatórias dentro de um intervalo especificado ou
2. são inicializados a partir da leitura de um ficheiro.

Neste último caso o ficheiro deve obedecer às mesmas recomendações feitas para um ficheiro contendo dados de entrada. Além disso, deve ser preenchido de forma a que as matrizes de pesos associadas às sinapses entre camadas de neurónios, se sucedam desde a camada de input até à camada de output. Para tornar mais claro como se deve construir este ficheiro considere-se a seguinte notação:

Seja $M^i_{n,m}$ a matriz de pesos associada às sinapses entre a camada $i-1$ e a camada i, com $i = 1, \ldots, k$ (o valor $i - 1 = 0$ corresponde à camada de input e $i = k$ a camada de output). O valor n corresponde ao número de neurónios da camada i e o valor m é o número de neurónios na camada $i-1$. Assim, no ficheiro terão de aparecer estas matrizes ordenadas da matriz entre a camada de input

e a primeira camada escondida, ou seja para $i = 1$, até à matriz entre a última escondida e a camada de output.

Estas matrizes estão em linhas distintas e sem nenhuma linha vazia entre elas. As colunas de cada matriz é separada por ponto e vírgula (representada pelo símbolo ;) e as linhas da matriz encontram-se em linhas distintas do ficheiro.

NNIGBackPropagationGUI

Com a criação do controlo Neural Network Editor, o botão Back Prop fica activo de imediato na barra de ferramentas da janela principal do programa. Este botão permite criar uma interface gráfica onde é possível a escolha de um algoritmo de treino. Os algoritmos de treino disponíveis são o Batch Backpropagation e Sequential Backpropagation. Pode-se ainda adaptar os parâmetros destes algoritmos como ilustra a figura 4.11.

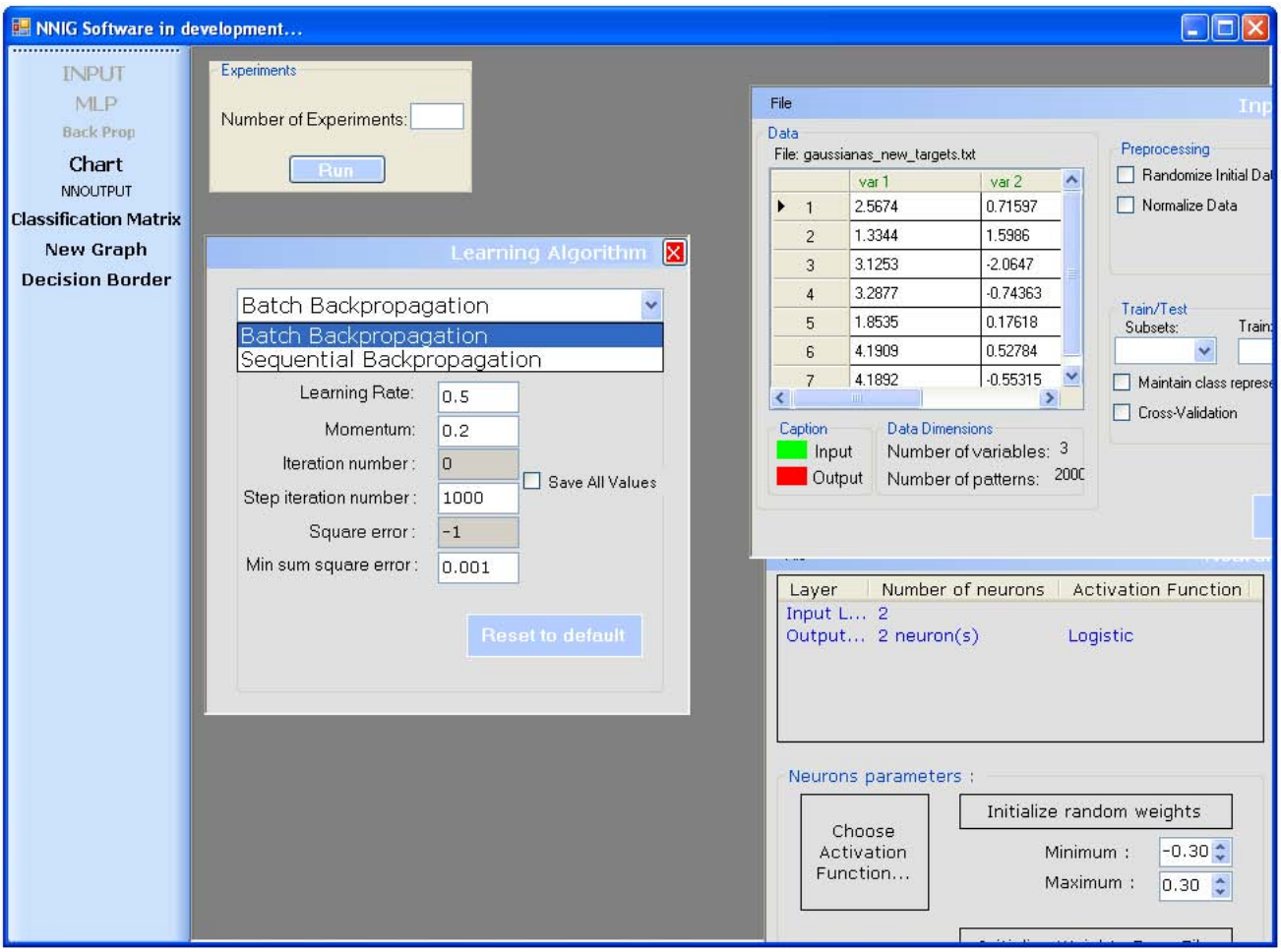

Figura 4.11: controlo NNIGBackPropagationGUI

ErrorChart

O botão Chart da barra de ferramentas da janela principal cria um controlo que permite visualizar a evolução do erro durante as iterações do algoritmo de treino. Este controlo será eliminado do projecto uma vez que não satisfaz as necessidades do grupo. Este controlo está representado na figura 4.12.

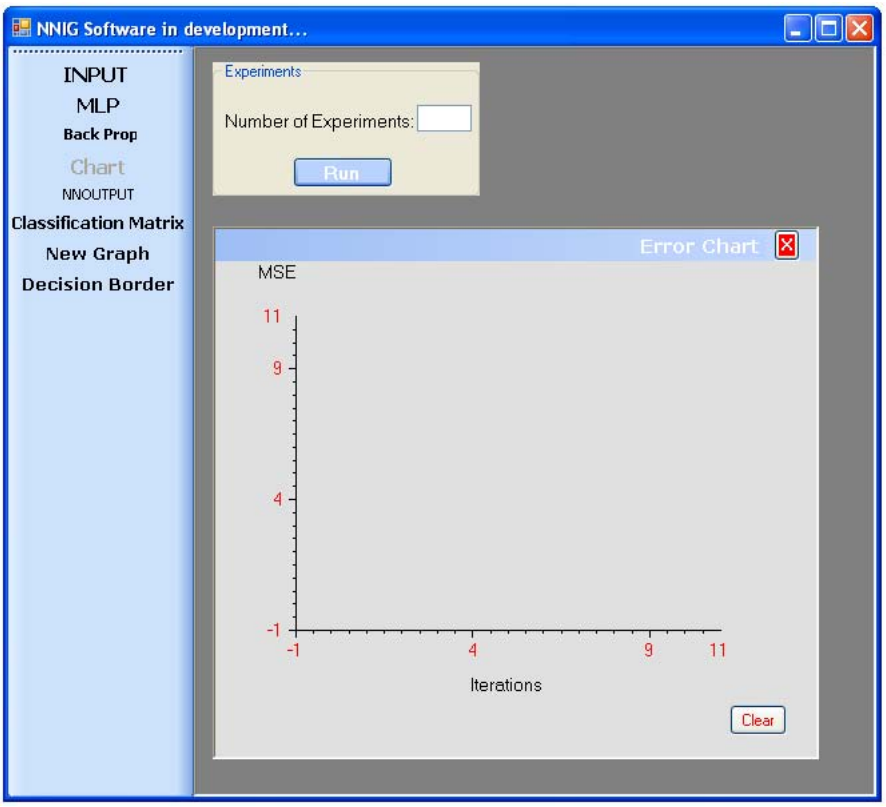

Figura 4.12: controlo ErrorChart

NNOUTPUT

O botão NNOUTPUT cria, na janela principal do software, um controlo que permite visualizar as respostas da rede neuronal após a última iteração realizada pelo algoritmo de treino. Este controlo está representado na figura 4.13.

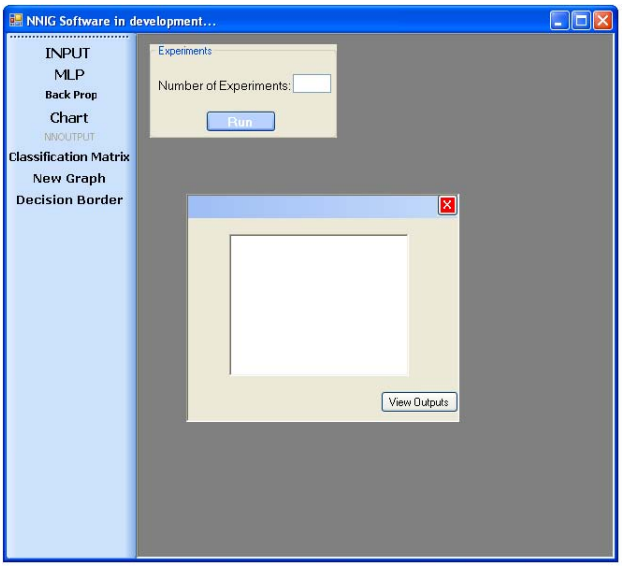

Figura 4.13: controlo NNOUTPUT

Classification Matrix

O botão Classification Matrix cria, na janela principal, um controlo que se encontra representado na figura 4.14.

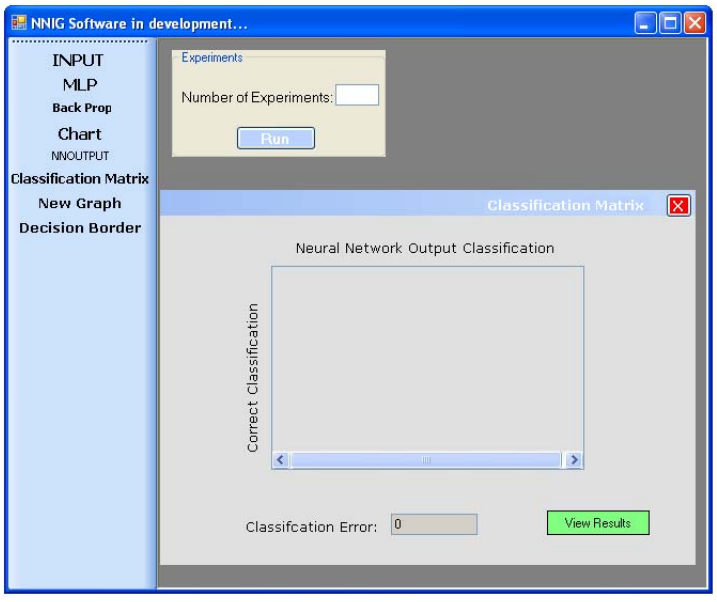

Figura 4.14: controlo Classification Matrix

Este controlo permite ao utilizador verificar com decorreu o processo de classificação.

ErrorGraph

O botão New Graph cria, na janela principal, um controlo que permite ao utilizador visualizar a evolução do erro de treino durante as diversas iterações. Este controlo é mais eficiente do que o criado pelo botão Chart e encontra-se representado na figura 4.15.

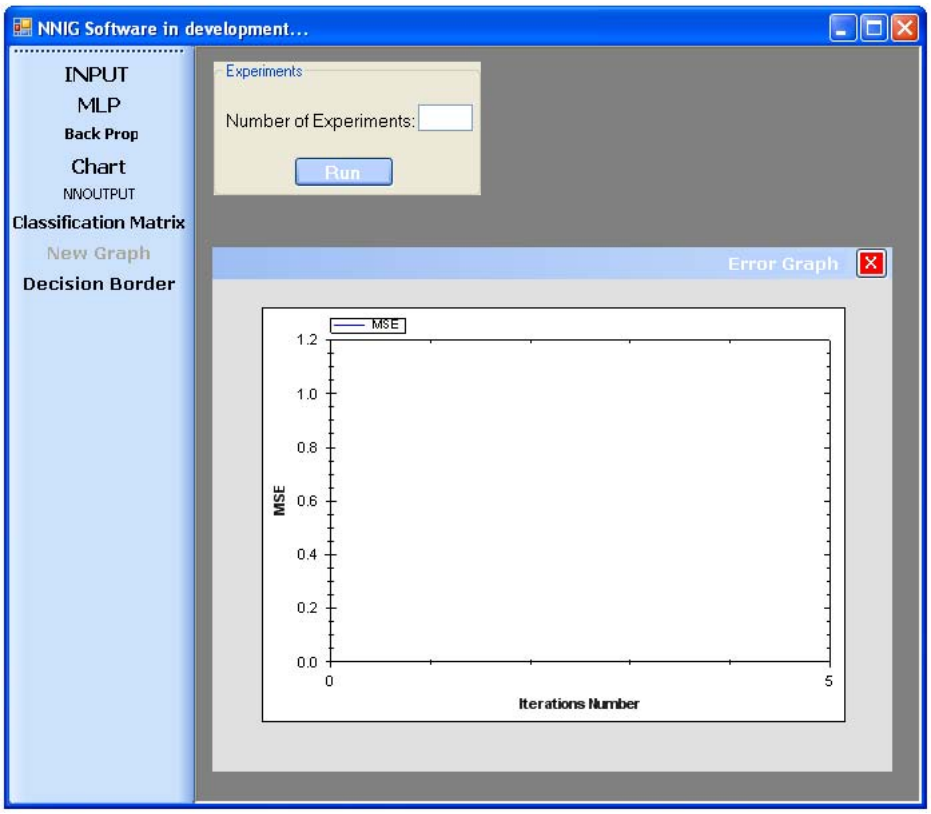

Figura 4.15: controlo ErrorGraph

Este controlo apresenta uma função que permite fazer um zoom numa determinada zona do gráfico representado.

NNIGDecisionBorder

O botão Decision Border cria, na janela principal, um controlo igual ao que se encontra na figura 4.16.

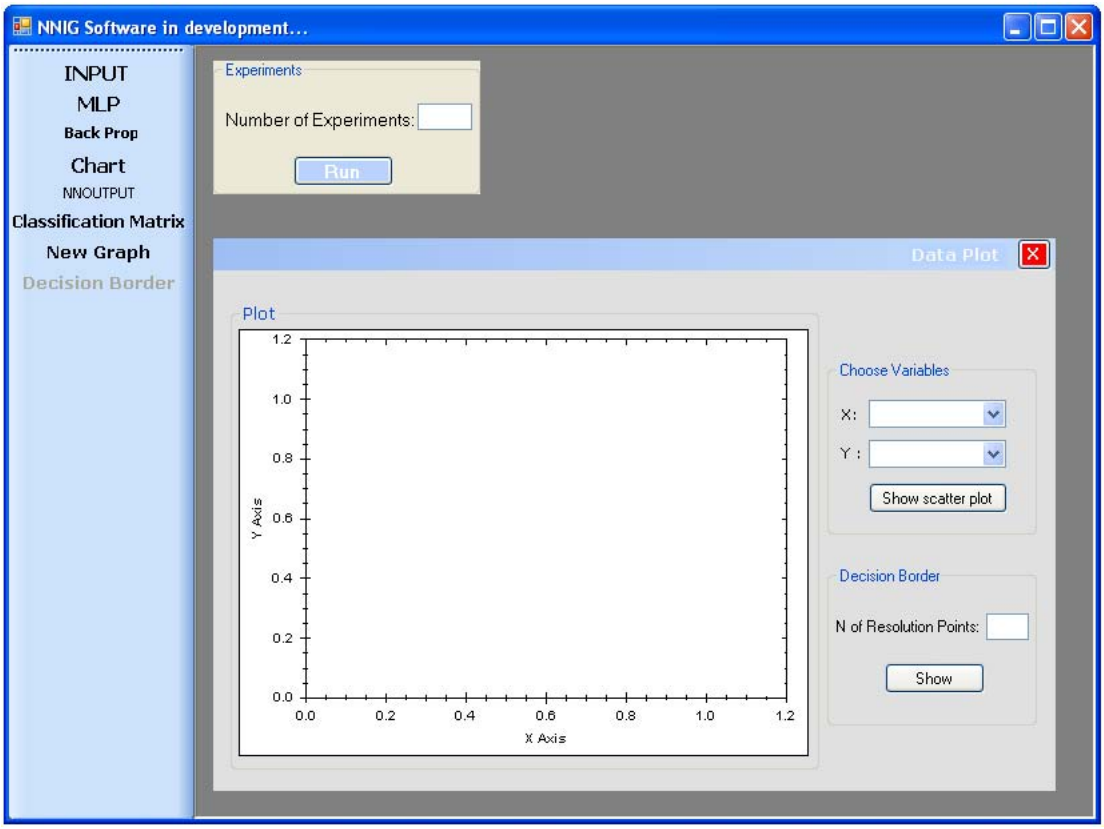

Figura 4.16: controlo Decision Border

Uma das principais funções deste controlo é representar num plano os dados de entrada tendo em consideração a classe da qual provêm. Assim, o utilizador deve escolher duas das características que os dados. Cada classe será representada com um cor diferente. Após o processo de treino da rede neuronal poderá ser representado neste controlo a curva de decisão que separa as classes.

4.5 RedesNeuronais_NNIG

Este projecto tem como principal função servir de repositório para diversas funções matemáticas. Estas funções apoiam o desenvolvimento de um rede neuronal permitindo a analise os resultados obtidos no treino e apoiando a preparação dos dados.

4.5.1 Aspectos Técnicos

Este projecto é composto pelas classes referenciada na tabela 4.8.

Nome da Classe	Função
AlgoritmoBacthBackpropagationUmaCamadaEscondida	Ensaio de implementação de um algoritmo de treino para uma rede neuronal com arquitectura fixa
ClassificationMatrix	Implementação do algoritmo para verificação de como se processou a aprendizagem da rede neuronal
LibAlg	Contem algoritmos básicos de matemática
LinearScaling	Permite projectar os dados introduzidos para um intervalo especificado
MeanCenteringScaling	Retira a cada elemento de um vector a média de todos os seus elementos
StandarizationScaling	Permite standardizar os dados
statistics	Algoritmos básicos de estatística
VectorPoint	Guarda a posição e o valor de um vector

Tabela 4.8: Classes que compõem o projecto RedesNeuronais_NNIG

AlgoritmoBacthBackpropagationUmaCamadaEscondida

A classe AlgoritmoBacthBackpropagationUmaCamadaEscondida é um ensaio que implementa o algoritmo de treino Bacth Backpropagation aplicado a uma rede neuronal do tipo MLP com uma camada escondida. A implementação deste algoritmo tão particular deveu-se a um exercício de aprendizagem e como tal já não tem interesse para o futuro desta ferramenta.

ClassificationMatrix

A classe ClassificationMatrix permite a visualização do processo de aprendizagem da rede neuronal, ou seja, contabiliza e expõe os casos bem e mal classificados para cada classe. Os casos mal classificados são expostos de forma a contabilizar o número de casos que pertence a uma classe mas aprendidos como pertencentes a outra (sendo estas classes especificadas).

LibAlg

A classe LibAlg é um repositório de algoritmos matemáticos básico e, apesar de ainda serem utilizadas algumas funções desta livraria, a principal função que esta classe desempenhou foi a de treino e aprendizagem de questões ligadas à linguagem de programação em que está a ser desenvolvido este

software. Nesta classe estão incluídas as seguintes funções associadas:

1. Concatenar uma matriz com um vector cujos elementos são todos iguais a 1.
2. Implementar uma função logística bipolar;
3. Implementar uma função tangente hiperbólica;
4. Implementar uma função linear;
5. Implementar uma função sigmoidal;
6. Implementar uma função step;
7. Implementar uma função logística unipolar;
8. Implementar um algoritmo Bacth Backpropagation para um perceptrão simples com função de activação logística bipolar;
9. Determinar a matriz de classificação;
10. Dadas duas matrizes determinar a diferença entre elas;
11. Dados dois vectores determinar a diferença entre eles;
12. Dada uma matriz determinar uma matriz cujos os elementos de cada coluna resultam de uma projecção dos elementos originais para o intervalos $[0, 1]$;
13. Dado um vector determinar o elemento de maior valor;
14. Dado um vector determinar o elemento de menor valor;
15. Dado um escalar e uma matriz determinar o seu produto;
16. Dado um escalar e um vector determinar o seu produto;
17. Dado um vector determinar a posição do elemento de maior valor. Caso existam dois valores máximos opta-se pela posição de maior valor.
18. Dado um vector determinar a posição do elemento de menor valor. Caso existam dois valores mínimos considera-se o último desses dois.
19. Dados dois vector determinar o seu produto escalar;

20. Dadas duas matrizes, tal que o número de colunas da primeira é igual ao número de linhas da segunda, determinar o produto matricial das duas;

21. Determinar o produto matricial entre duas matrizes usando o produto escalar entre dois vectores;

22. Dada uma matriz somar todos os elementos que a constituem;

23. Dado um vector determinar a soma de todos os seus elementos;

24. Dados dois vectores determinar a sua soma;

25. Determinar o erro quadrático médio entre dois vectores;

26. Determinar o erro quadrático médio total entre dois vectores;

27. Dada uma matriz determinar a matriz transposta;

28. Dados dois vectores determinar o vector cujos elementos são o produto dos elementos correspondentes dos vectores dados;

29. Dado um vector projecta todos os elementos para o intervalo $[0, 1]$;

30. Dado um vector cujos elementos são números naturais constrói uma matriz com número de linhas igual ao número de elementos do vector e colunas igual ao valor máximo do vector. Cada linha da matriz toma o valor 0 ou 1, estando estes últimos localizados na coluna indicada pelo valor do elemento do vector que se encontra na mesma linha.

31. Dada uma matriz copiar os seus elementos para uma nova matriz com a mesma dimensão;

32. Criar uma matriz de qualquer dimensão e cujos elementos estão num intervalo da forma $[-l, l]$.

33. Dada a dimensão de uma matriz constrói uma matriz cujos elementos são todos iguais a 1;

34. Dada a dimensão de uma matriz constrói uma matriz cujos elementos são todos iguais a 0;

35. Implementar a função derivada da função logística bipolar;

36. Implementar a função derivada da função tangente hiperbólica;

37. Implementar a função derivada da função linear;

38. Implementar a função derivada da função sigmoidal;

39. Implementar a função derivada da função step;

40. Implementar a função derivada da função logística unipolar;

41. Implementar a saída de uma camada de neurónios em que a função de activação é uma função logística bipolar ;

42. Implementar a saída de uma camada de neurónios em que a função de activação é a função derivada da função logística bipolar;

43. Dados dois vectores determinar a distancia entre eles;

44. Dada uma matriz qualquer preenchê-la com zeros;

45. Dado um vector preenchê-lo com um valor dado;

46. Dado um vector e um escalar determinar as posições em que o escalar ocorre no vector;

47. Determina todas as saídas de uma camada de neurónios para uma dada matriz de entrada e uma determinada matriz de pesos;

48. Dadas duas matrizes determina a matriz cujos elementos são o produto dos elementos correspondentes nas duas matrizes;

49. Determinar a saída de um neurónio com função de activação logística bipolar;

50. Determinar a saída de um neurónio com função de activação logística bipolar e com matriz inicial de pesos aleatória;

51. Dada uma matriz fazer uma partição das suas linhas, de acordo com a percentagem especificada pelo utilizador, construindo duas matrizes distintas. Nota: é usado um cast para inteiro resultando no menor inteiro contido no número. Portanto, por vezes (principalmente no caso de 50%) o conjunto teste pode ficar com mais elementos do que o conjunto treino;

52. Dada uma matriz fazer uma partição das suas linhas de acordo com percentagens especificadas pelo utilizador construindo assim três matrizes distintas.

53. Dada uma matriz retirar-lhe a última coluna;

54. Dada uma matriz retirar uma coluna especificada;

55. Dada uma matriz construiu um vector igual a um determinado vector linha da matriz;

56. Dada uma ArrayList dispõe os elementos de forma aleatória;

57. Dado um vector, cria uma ArrayList cujos elementos são do tipo ArrayList. Os elementos destas últimas Arraylist são grupos de posições de elementos do vector inicial, mantendo a proporção de elementos de cada classe;

58. Trocar de forma aleatória as linhas uma matriz.

59. Dado um vector criar um novo cujos elementos são iguais aos do vector inicial mas sem repetições.

LinearScaling

A classe LinearScaling permite projectar um vector ou matriz para num dado intervalo, guardando todos os coeficientes para se poder voltar ao dados originais.

MeanCentering

A classe MeanCentering permite retirar a cada elemento de um vector a média de todos os valores que o formam.

Standardization

A classe Standardization permite retirar a cada elemento de um vector a média e dividir o resultado pelo valor do desvio padrão da totalidade dos valores do vector original.

statistics

A classe statistics deve a sua criação a Yossi Rozenberg e permite determinar, entre outras, as seguintes estatísticas:

1. covariância;

2. intervalo entre quartis;

3. o valor máximo dos dados;

4. o valor mínimo dos dados;

5. a média;
6. a mediana;
7. a moda;
8. o primeiro quartil;
9. o segundo quartil;
10. o terceiro quartil;
11. o coeficiente de correlação;
12. o desvio padrão;
13. a variância amostral.

Capítulo 5

Aplicação do software num problema de separação de duas classes

Neste capítulo pretende-se criar uma rede neuronal simples com o objectivo de separar dados pertencentes a duas classes distintas. Estes dados foram gerados a partir de duas funções gaussianas distintas tendo cada classe 1000 exemplares. Cada exemplar é descrito por duas características distintas.

1. Usando o *Input Data*, abrir o ficheiro contendo estes dados. O controlo ficará com uma tabela semelhante à da figura 5.2.

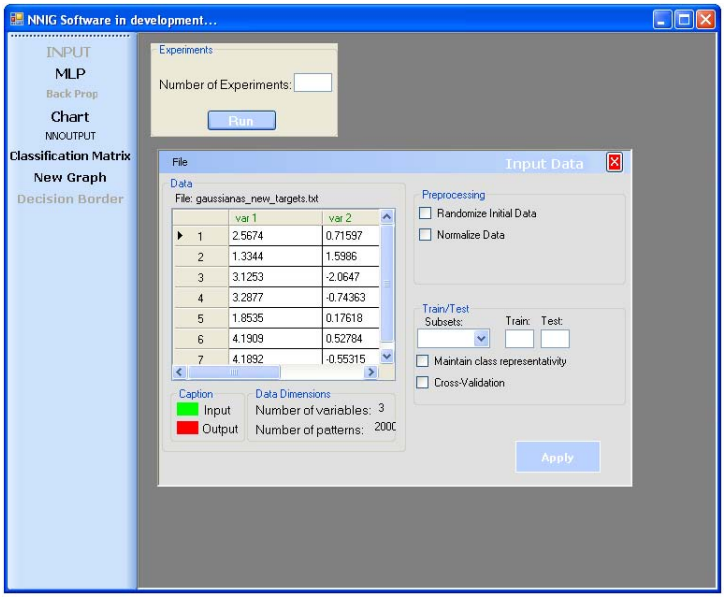

Figura 5.1: Dados das funções Gaussianas

2. Aceitando as opções feitas pelo software, clicar no botão *MLP* para escolher a arquitectura da rede neuronal. Aqui, adiciona-se uma camada de neurónios (clicando em *Insert Layer*) e escolhe-se 2 como número de neurónios para esta camada. Aceita-se a função de activação pré-definida (que é a função logística).

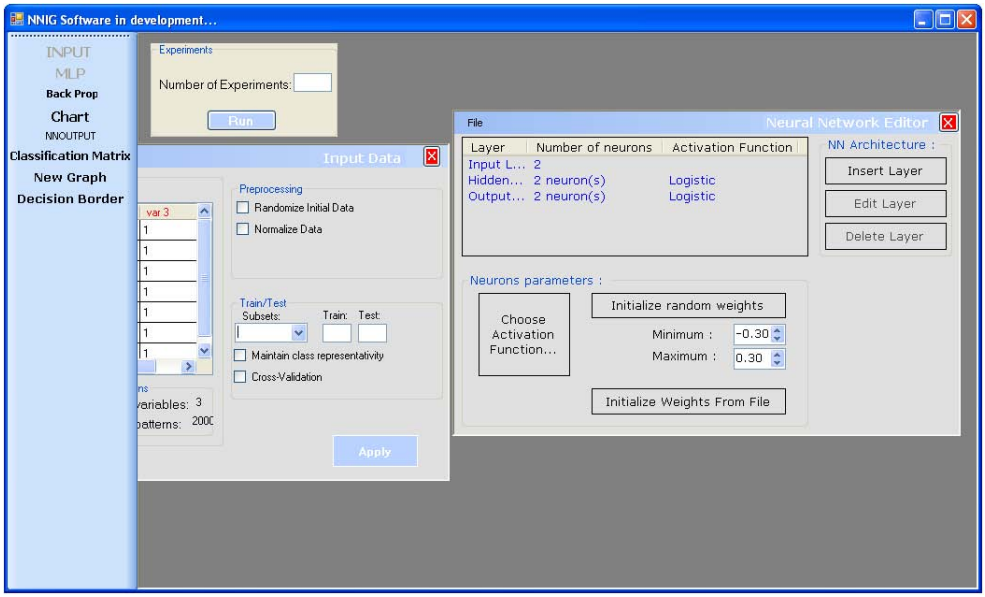

Figura 5.2: Arquitectura da rede neuronal

3. Clicar no botão *Initialize Weights From File* e abrir o ficheiro com os pesos iniciais da rede (não esquecer que por defeito estes pesos iniciais são nulos).

4. Escolher o *Batch Backpropagation* no controlo criado após clicar no botão *Back Prop*. Colocar *Learning Rate* $= 0.01$, *Momentum* $= 0$ e *Step iteration number* $= 100$.

5. Criar um controlo do tipo *Classification Matrix*, um do tipo *New Graph* e um do tipo *Decision Border*.

6. Clicar no botão *Run*. O programa inicia o processo de aprendizagem.

   Com o fim das iterações o controlo *Back Prop* é actualizado, a matriz de classificação é preenchida e o gráfico representa a evolução do erro ao longo das iterações. Este quadro está representado na figura 5.3.

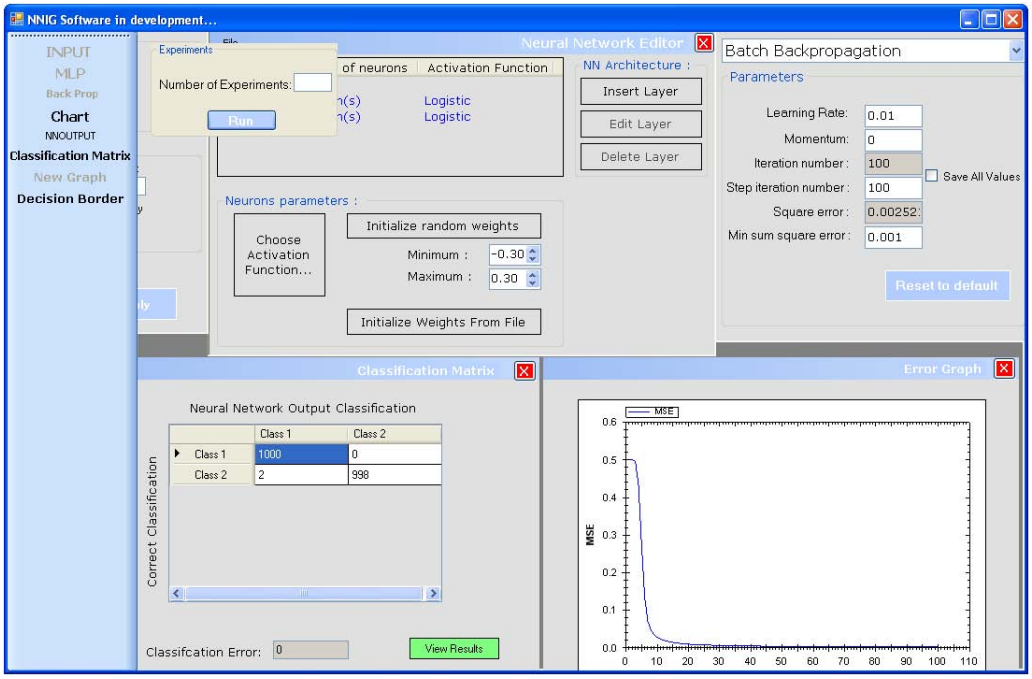

Figura 5.3: Fim do processo de aprendizagem

7. Criar um controlo do tipo *Decision Boder*.

8. Escolher para variável X a opção *var1* e para variável Y a opção *var2*.

9. Clicar em *Show scatter plot*. O resultado é a representação de dois conjuntos distintos: um azul e outro vermelho.

10. Escolher para *"N of Resolution Points"* o valor 100.

11. Clicar em *Show*. De imediato aparece representada a fronteira de decisão determinada pela rede neuronal como ilustra a figura 5.4.

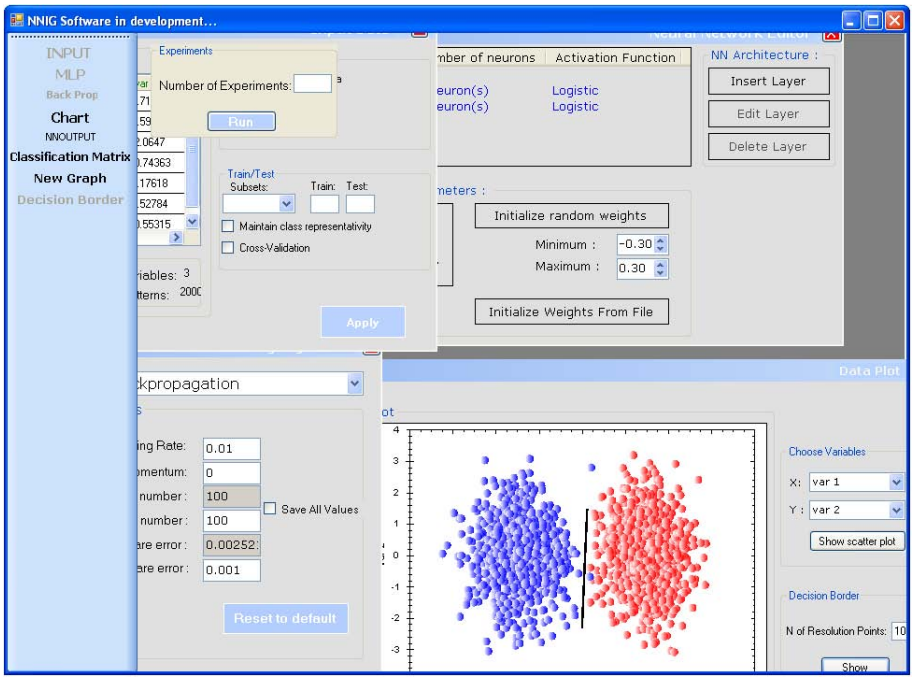

Figura 5.4: Decision Border

Bibliografia

[1] http:// pt.wikipedia.org/wiki/Neur%C3%B3nio

[2] http://www.simbiotica.org/impulsonervoso.htm

[3] Marrone, Paolo; *Joone - Java Object Oriented Neural Engine; The Complete Guide - All you need to know about Joone*; February, 2005