Redes Neurais no Reconhecimento de Padrões: Estudo de Caso

Márcia Cristina Cera

Trabalho da disciplina CMP135 -
Arquiteturas Especiais de Computadores
Professor: Philippe Olivier Alexandre Navaux
Semestre: 2005/2

1 Introdução

É de conhecimento geral o emprego de redes neurais como ferramenta no reconhecimento de padrões, como por exemplo fala, caracteres, imagens, entre outros. Uma rede neural é caracterizada por uma coleção massivamente paralela de unidades de processamento pequenas e simples, onde as interligações formam a maior parte da inteligência da rede. Essas redes geram uma grande quantidade de processamento que pode ser empregada na identificação de padrões em uma analogia ao cérebro humano.

Para que uma rede neural seja capaz de fazer o reconhecimento de padrões, inicialmente a estrutura da rede é treinada, ou seja adquire conhecimento, para poder identificar os padrões almejados. Um algoritmo bastante conhecido na literatura é o chamado *backpropagation*. Com esse algoritmo, os dados de entrada são repetidamente apresentados para a rede neural. A casa apresentação a saída da rede neural é comparada com a saída desejada e um valor de erro é calculado. Esse erro é o propagado de volta (*backpropagated*) a rede neural e usado para ajustar os pesos buscando reduzir o erro a cada iteração para que o resultado aproxime-se cada vez mais da saída desejada.

O objetivo deste trabalho foi ter uma experiência prática com uma rede neural para a solução de um problema, que no caso foi o reconhecimento de padrões. Para atender ao objetivo, inicialmente buscou-se um simulador de redes neurais capaz de oferecer as condições necessárias para a realização do trabalho. O simulador escolhido será descrito brevemente na seção 2. A seguir, este texto apresenta o experimento realizado e os resultados obtidos. Por fim tem-se a conclusão desse trabalho.

2 Simulador de Rede Neural

Dentre as opções de simuladores de rede neural encontradas na internet optou-se pelo simulador de Rede Neural SNNS (*Stuttgart Neural Network Simulator*) [5] que foi desenvolvido no *Institute for Parallel and Distributed High Performance Systems* (IPVR) da Universidade de Stuttgart. O objetivo do SNNS foi criar um ambiente de simulação eficiente e flexÃvel para a pesquisa de aplicações de rede neural. O SNNS apresenta uma interface de utilização bastante amigável incluindo uma série de funcionalidades que possibilitam a solução de uma grande quantidade de problemas através desse software. O SNNS apresenta uma versão em Java - JavaNNS [4] - que foi instalada e utilizada nos experimentos nesse trabalho.

Primeiramente será apresentado a estrutura de uma rede neural que pode ser implementada através do simulador. A figura 1 mostra uma rede neural composta de 4 unidades de processamento ou neurônios. Como pode ser visto, a rede é composta por três tipos de unidades de processamento: as de entrada (*input units*), as intermediárias ou escondidas (*hidden inuts*) e as unidades de saída (*output units*). Ligando as unidades processadoras existem conexões (*lins*) que levam os dados de uma unidade a outra.

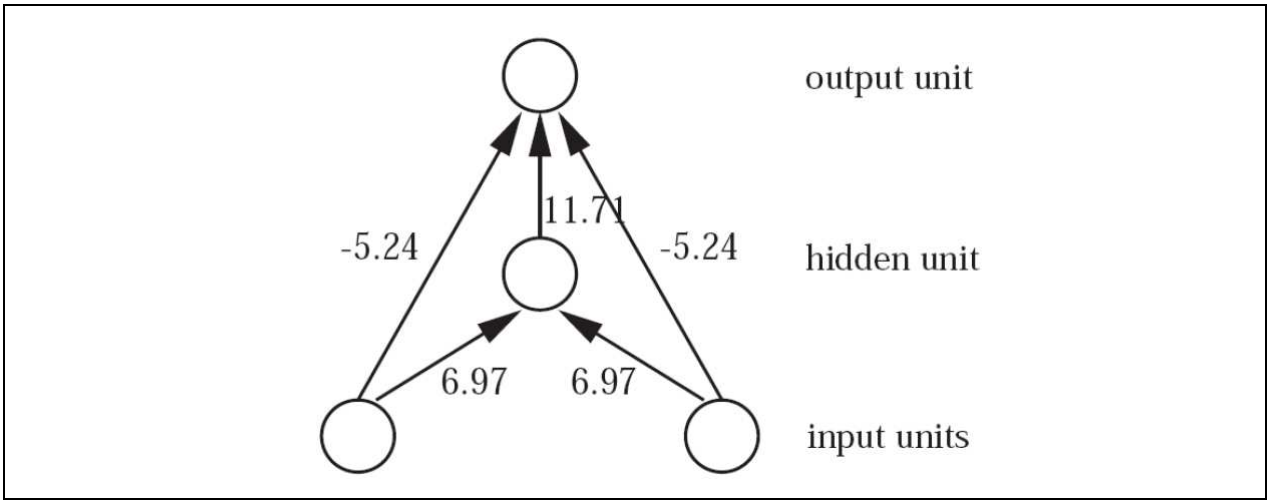

Figure 1: Estrutura simples de uma rede neural [1]

As informações processadas dentro das unidades (neurônios) no SNNS é modelada através de funções de ativação (*activations functions*) e funções de saída (*output functions*). A função de ativação de uma unidade primeiro computa os dados de entrada que foram originados de dados de saída de unidades antecessoras. Então ele computa um novo valor de ativação gerado a partir das entradas e possivelmente a partir das ativações anteriores. A função de saída toma o resultado gerado pela função de ativação para gerar a saída da unidade. Essas funções podem ser funções arbitrárias ligadas ao kernel do simulador e podem ser diferente para cada unidade. A figura 2 possui uma representação das funções de ativação e saída em uma unidade processadora. A função de ativação esta representada pelo somatório e a função de saída pelo f.

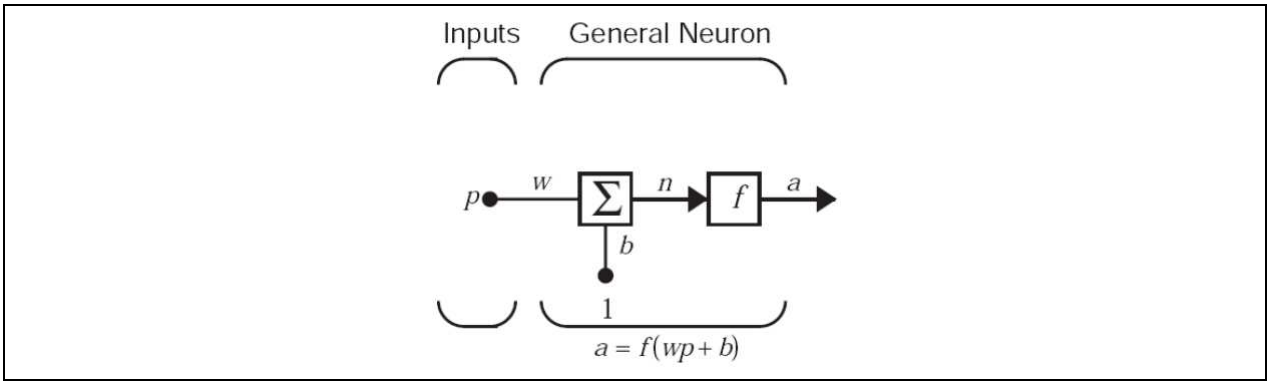

Figure 2: Representação das funções da unidade processadora [3]

A fim de demostrar a interface do simulador JavaNNS será apresentado um exemplo de rede neural oferecido pelo próprio sistema. A figura 3 apresenta uma visão do simulador. A rede apresentada é treinada para identificar, a partir da ativação do primeiro nível da rede, caracteres definidos em um padrão. A figura apresenta uma rede neural composta por 35 unidades de entrada (na forma de uma matriz 5×7), um nível de unidades intermediárioas com 10 neurônios e, por fim, 26 unidades de saída que representam as letras do alfabeto a serem reconhecidas. A rede neural é armazenada pelo simulador em um arquivo do tipo ".net".

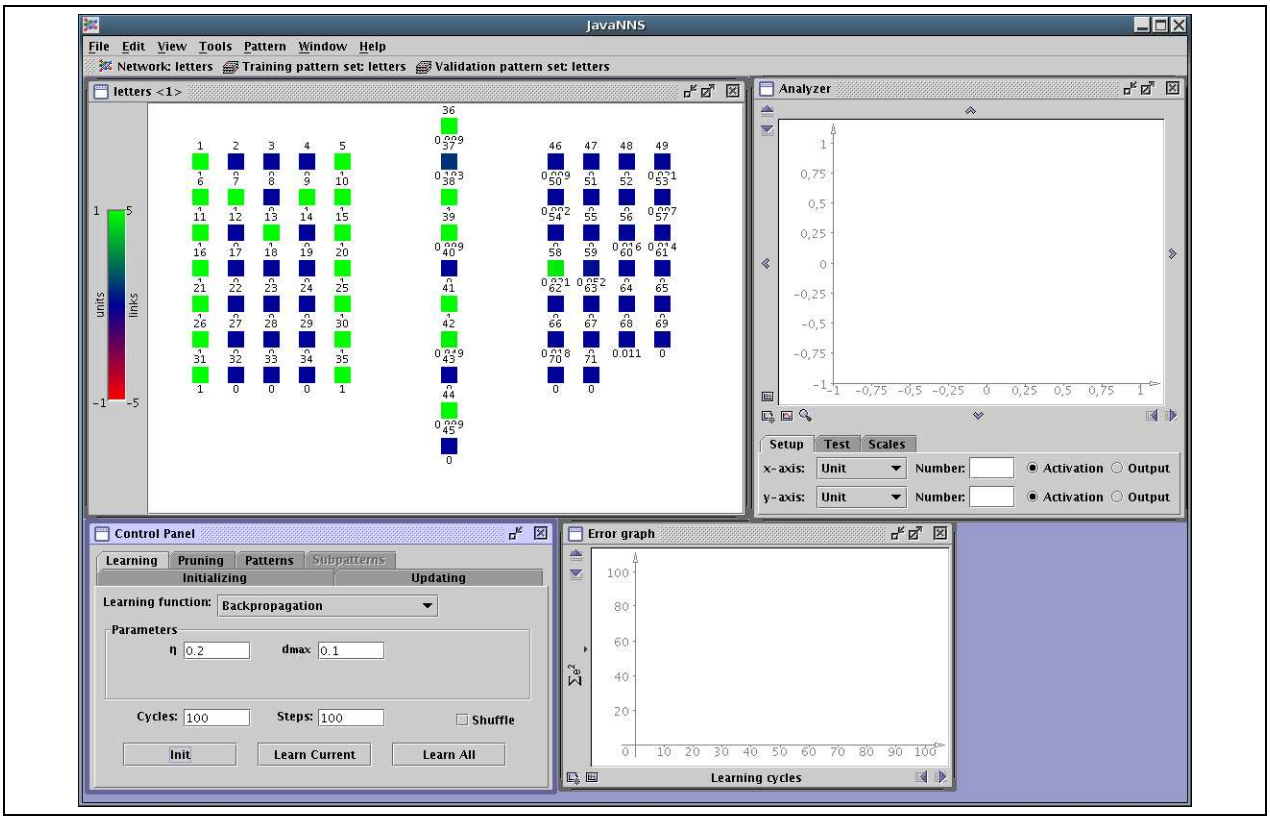

Figure 3: JavaNNS com uma rede para reconhecimento de caracteres

Os demais itens apresentados na figura 3 são oferecidos pelo simulador para controlar a execução da rede e verificar seu funcionamento. A caixa mais a esquerda é o painel de controle onde pode-se escolher o algoritmo de treinamento conforme a rede empregada (maiores informações em [1, 2]) e pode-se alterar os parâmetros da rede. O algoritmo mais popular é o *Backpropagation* que está sendo mostrado na figura. A caixa ao lado apresenta o gráfico de erro que é gerado durante a execução do treinamento da rede onde pode-se verificar o ajuste dos valores no decorrer da execução do treino, o resultado após a execução do treino pode ser visto na figura 4. Por fim, tem-se a caixa para a análise e teste da rede neural onde podem ser especificador parâmetros de teste e visualizado gráficos com diferentes argumentos nos eixos x e y.

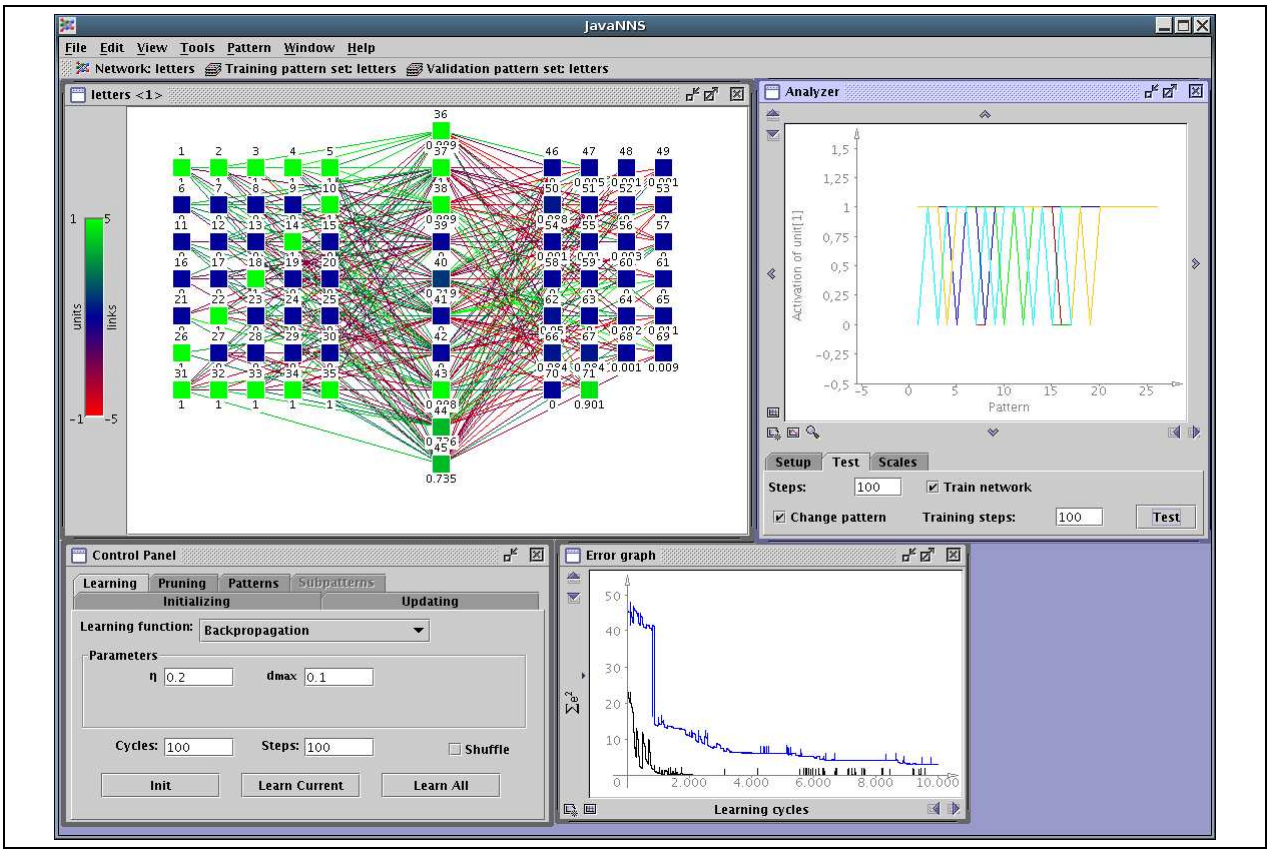

Figure 4: JavaNNS com uma rede para reconhecimento de caracteres

Para o treinamento de uma rede neural é inserido na rede um padrão, o qual especifica que tipo de informação busca-se reconhecer com a rede neural. Todo padrão para o JavaNNS é armazenado em um arquivo com a extensão ".pat". Um fragmento do padrão utilizado na rede apresentada anteriormente para o reconhecimento de caracteres está apresentado na figura 5. Nele têm-se a definição da configuração das entradas que especificam as letras do alfabeto e, para cada entrada, a saída obtida ou seja a letra correspondente. Por questões de espaço foram omitidos dados do cabeçalho do arquivo e o restante do padrão. Vale lembrar que esta rede neural é um exemplo apenas ilustratório que não pode ser empregado para o reconhecimento de caracteres em um ambiente real.

```
...
# Input pattern 1:
0 1 1 1 0
1 0 0 0 1
1 0 0 0 1
1 1 1 1 1
1 0 0 0 1
1 0 0 0 1
0 0 0 0 1
# Output pattern 1:
1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0
# Input pattern 2:
1 1 1 1 0
1 0 0 0 1
1 0 0 0 1
1 1 0 1 0
1 0 0 0 1
1 0 0 0 1
1 1 1 1 0
# Output pattern 2:
0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0
...
```

Figure 5: Padrão para reconhecimento de caracteres

3 Experimento Realizado

Tendo por base os conhecimentos adquiridos através do estudo do exemplo apresentado na seção anterior, buscou-se solucionar um novo problema empregando o simulador. Decidiu-se construir uma rede neural capaz de fazer a identificação de números de 0 a 9 nos mesmos moldes da rede que faz o reconhecimento de caracteres. Para isso identificou-se algumas mudanças na estrutura da rede neural e construiu-se uma nova rede.

A seguir, na figura 6, pode-se visualizar as ferramentas oferecidas pelo simulador para a construção de uma nova rede neural. A caixa mais a esquerda destina-se a criação dos níveis de unidades processadoras que integram a rede neural. Através dessa caixa define-se, principalmente, quantas unidades exitirão em cada nível, qual o tipo de unidade e as funções de ativação e saída. A caixa ao lado possibilita a criação de conexões entre as unidades processadoras o que permitirá o funcionamento da mesma. Esta caixa possilita a escolha da forma de conexão entre as unidades. Por fim, a caixa restante possibilita a edição dos parâmetros para cada uma das unidades processadoras pertencentes a rede. Assim, o simulador oferece flexibilidade para que ocorram ajustes finos na rede neural.

A figura 7 mostra a rede neural construída para o reconhecimento de números. Essa rede é composta por 35 unidades processadoras no primeiro nível ou nível de entrada, onde a entrada terá a mesma função que na rede para reconhecimento de caracteres. O segundo nível ou nível intermediário possui 10 unidades processadoras e no último nível têm-se 10 unidades que representam a saída da rede, ou seja, os números de 0 a 9. Embora tenha sido omitida na figura 7, por questões de simplicidade de visualização, as unidades de processamento foram conectadas de forma *feed forward*.

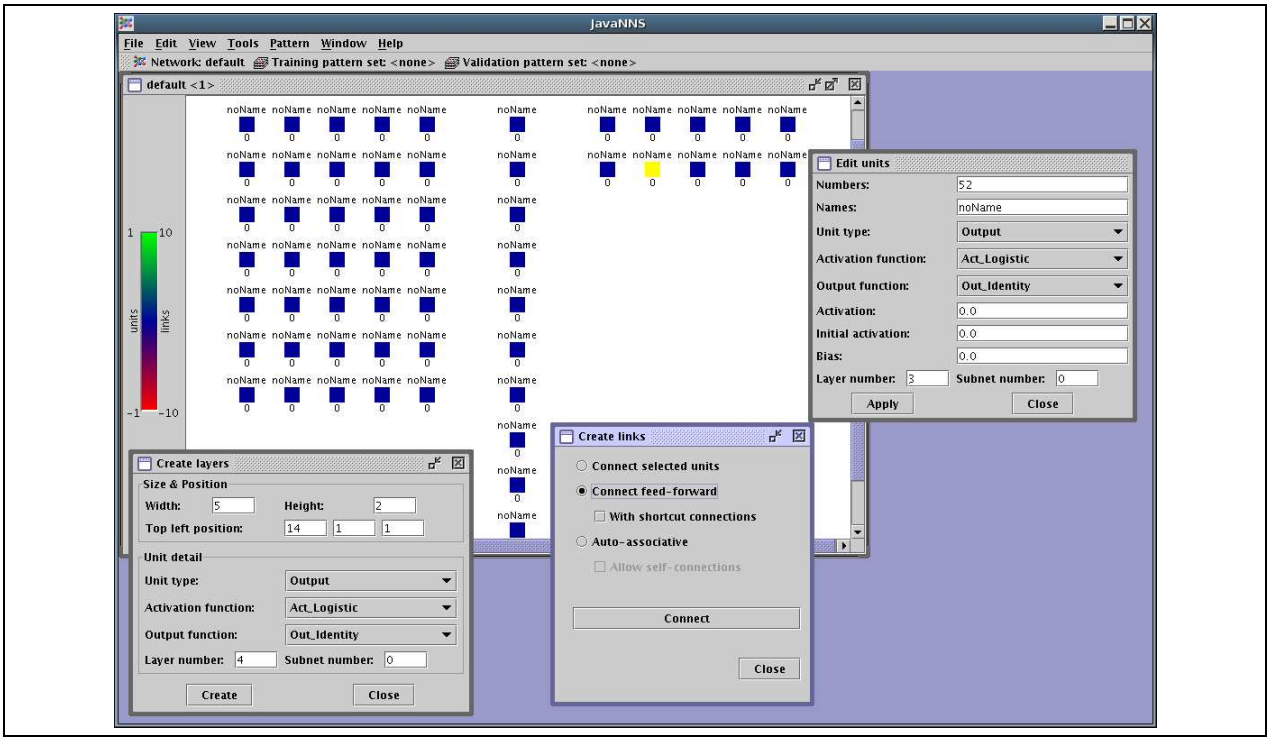

Figure 6: Ferramentas para a criação de uma nova rede neural

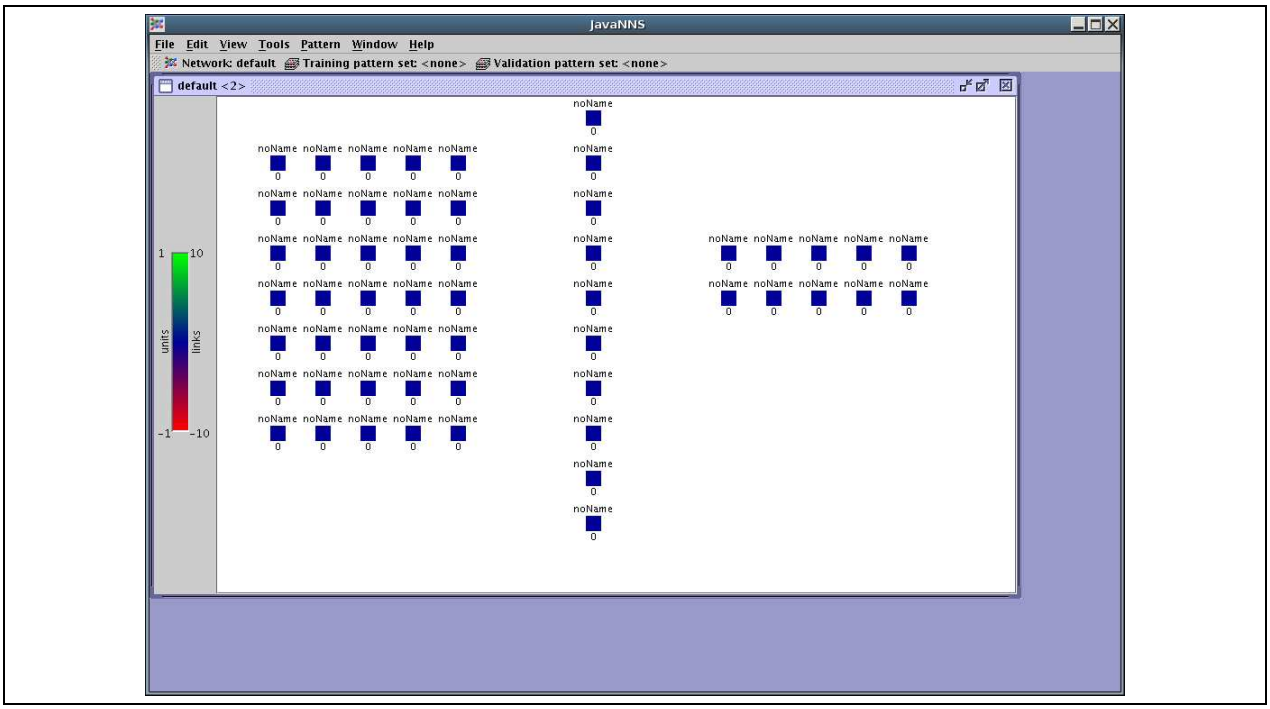

Figure 7: Rede neural para o reconhecimento de números

Para que fosse possível fazer o reconhecimento de números a rede neural precisa ser alimentada com um padrão, a figura 8 apresenta o padrão construído para este fim. O padrão para o reconhec-

imento de números foi baseado no padrão apresentado na figura 5 onde é fornecida a configuração da entrada e a saída correspondente.

```
...
# Input pattern 1:
0 1 1 1 0
1 0 0 0 1
1 0 0 0 1
1 0 0 0 1
1 0 0 0 1
1 0 0 0 1
0 1 1 1 0
# Output pattern 1:
1 0 0 0 0
0 0 0 0 0
# Input pattern 2:
0 0 1 0 0
0 1 1 0 0
1 0 1 0 0
0 0 1 0 0
0 0 1 0 0
0 0 1 0 0
0 1 1 1 0
# Output pattern 2:
0 1 0 0 0
0 0 0 0 0
...
```

Figure 8: Padrão para reconhecimento de números

Com a construção da estrutura da rede neural e o padrão para reconhecimento fez-se a execução do treinamento da rede. O resultado após a execução está mostrado na figura 9. Nela vê-se que com o treinamento, tanto as unidades processadoras quanto as conexões entre elas apresentam um valor específico que resultou das várias iterações entre o padrão e a rede. A quantidade de iterações e demais parâmetros incluindo o algoritmo de treinamento podem ser facilmente escolhidos através do painel de controle. Para este caso usou-se o algoritmo de *Backpropagation* com 100 ciclos de treinamento em 100 passos.

No gráfico de erro identificam-se 3 linhas. Cada uma dessas linhas representa um valor de inicialização diferente da rede, sendo que a linha preta representa o intervalo de inicialização [-1,1], a linha azul o intervalo [-4,4] e a linha vermelha o intervalo [-8,8]. Nesse gráfico identifica-se o comportamento do algoritmo *Bachpropagation* onde o erro é propagado de volta na rede a fim de reduzi-lo até o ponto em que este normaliza-se.

Por fim, tem-se a caixa de análise na figura 9. Ela disponibiliza a execução de testes sobre a rede podendo-se fixar os parâmetros desejados. Pode-se também escolher que variáveis serão apresentada nos eixos do gráfico. O gráfico apresentado na figura possui como parâmetro de eixo x os padrões oferecidos a rede e como eixo y o valor da função de ativação da unidade 1.

4 Conclusão

Este trabalho objetivou empregar na prática uma rede neural para solucionar um problema de reconhecimento de padrões. Para isso buscou-se um simulador de rede neural optando-se pelo

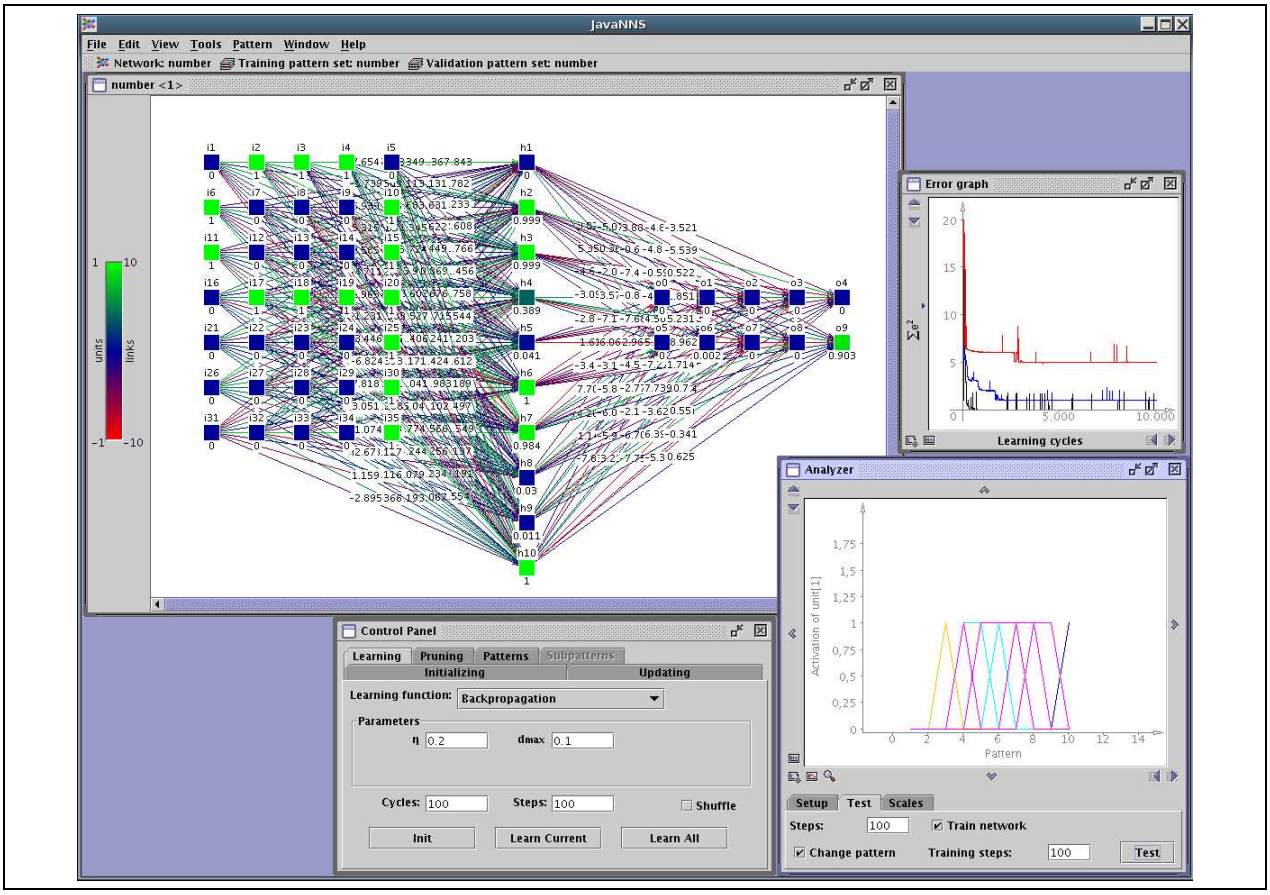

Figure 9: Rede para o reconhecimento de números após a execução

simulador SNNS na versão em Java (JavaNNS). Tendo por base um exemplo de rede neural que faz a identificação de padrões de caracteres construiu-se uma rede capaz de fazer o reconhecimento de padrões de números.

Para que a rede pudesse reconhecer números foi construída um rede neural com 35 unidades de entrada, 10 unidades intermediárias e 10 unidades de saída (uma para cada número reconhecido). A rede foi conectada de forma *feed forward* e foi criado um padrão específico para o reconhecimento de números. Com esta estrutura treinou-se a rede e com as ferramentas oferecidas pelo JavaNNS pode-se verificar que foi obtido o objetivo esperado que é o reconhecimento de números.

Importante salientar que o simulador de rede neural JavaNNS pode ser facilmente adquirido no site [4] e que sua instalação é bastante simples. A interface de utilização do simulador é amigável e é possivel fazer ajustes nos parâmetros da rede facilmente. Também é possivel implementar redes neurais para fins específicos já que o sistema apresenta grande flexibilidade e diversidade de parâmetros a serem utilizados deixando o usuário livre para personalizar sua rede.

References

[1] Andreas Zell at al. Snns - stuttgart neural network simulator - user manual, version 4.2, 2005. http://www-ra.informatik.uni-tuebingen.de/SNNS/ - acessado em dezembro 2005.

[2] Igor Fischer, Fabian Hennecke, Christian Bannes, and Andreas Ze. Javanns - java neural network simulator - user manual, version 1.1, 2005. http://www-ra.informatik.uni-tuebingen.de/downloads/JavaNNS/ - acessado em dezembro 2005.

[3] M. Hagan, H. Demuth, and M. Beale. *Neural Network Design*. PWS Publishing Company, Boston, MA, 1996.

[4] Andreas Zell. Javanns, 2005. http://www-ra.informatik.uni-tuebingen.de/downloads/JavaNNS/ - acessado em dezembro 2005.

[5] Andreas Zell. Stuttgart neural network simulator, 2005. http://www-ra.informatik.uni-tuebingen.de/SNNS/ - acessado em dezembro 2005.