André Camargo
Chrystian Guth
Gabriel Gava
Lucas Pereira

Redes neurais

O primeiro passo para a criação da rede neural foi normalizar os dados de entrada. Para realizar essa tarefa é importante saber antes quais funções de saída serão utilizadas. No nosso caso, utilizamos a função **tangente hiperbólica**. A função tangente hiperbólica gera valores no intervalo aberto `(-1, 1)`, ou seja, ela nunca irá gerar -1 e nem 1. Devido a isso, normalizamos nossos dados de entrada no intervalo `[-0.9, 0.9]`. Para realizar essa normalização, utilizamos a função `mapminmax` do Matlab. Essa função recebe três parâmetros como entrada: a matriz que se deseja normalizar, o valor mínimo e o valor máximo.

```
x; %entradas%
t; %saídas&
xn = mapminmax(x, -0.9, 0.9); %normalização das entradas%
```

Com os dados de entrada normalizados, partimos para a definição dos parâmetros de configuração da nossa rede. Decidimos utilizar 10 neurônios na camada intermediária, utilizar a função de saída **tangente hiperbólica** (`tansig`) e utilizar o algoritmo de treinamento ***Gradient descent backpropagation*** **com** ***momentum*** (`traingdm`). Uma vez tendo estipulado os parâmetros de configuração, criamos a nossa rede:

```
neuronios = 10;
funcaoDeSaida = {'tansig'};
```

```
algoritmoDeTreinamento = 'traingdm';
rede = newff(
     xn,
     t,
     neuronios,
     funcaoDeSaida,
     algoritmoDeTreinamento);
```

Após criar a rede foi necessário definir as configurações do treinamento. Definimos um treinamento com 10000 **épocas**, **erro** de 0.01, **taxa de aprendizado** de 0.05 e **taxa de momento** de 0.85.

```
rede.trainParam.epochs = 10000; %número de épocas%
rede.trainParam.goal = 0.01; %erro%
rede.trainParam.lr = 0.05; %taxa de aprendizado%
rede.trainParam.mc = 0.85; %taxa de momento%
```

Para realizar a divisão do conjunto de padrões foi utilizada a proporção de 0.7, 0.15 e 0.15. Ou seja, dividimos o nosso conjunto de padrões como sendo 70% para treinamento, 15% para validação e 15% para testes.

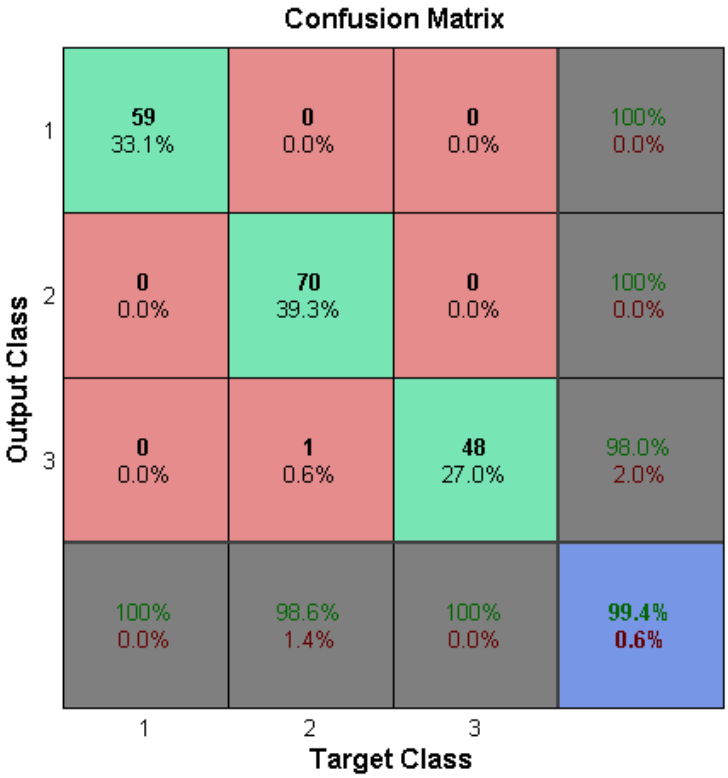
```
rede.divideParam.trainRatio = 0.7; %treinamento: 70%%
rede.divideParam.valRatio = 0.15; %validação: 15%%
rede.divideParam.testRatio = 0.15; %teste: 15%%
```

Após realizar todas as configurações, fizemos o treinamento da rede. O treinamento com 10000 épocas foi relativamente rápido e durou cerca de 13 segundos. Uma vez treinada, simulamos a rede passando o conjunto de padrões de entrada como parâmetro. O resultado dessa simulação é a saída gerada pela rede. Para verificar a eficácia da rede no reconhecimento dos padrões, plotamos em um gráfico a saída

gerada pela rede e a saída esperada. O resultado foi bastante positivo e obtivemos em algumas das simulações uma taxa de acerto de 100% na classificação dos padrões aprendidos. Realizamos a simulação diversas vezes e a taxa de acerto ficou entre 98% e 100%.

```
redeTreinada = train(rede, xn, t);
simulacao = sim(redeTreinada, xn);
plotconfusion(t, simulacao);
```

O gráfico mostrando a relação das saídas obtidas pela rede e as saídas desejadas se encontra abaixo. Com o gráfico é possível ver como os vinhos foram classificados pela rede e qual é a real classificação deles. Por exemplo, na coluna 2 da linha 3 é mostrada a quantidade de vinhos classificados como sendo da vinícola 3, mas que na verdade são da vinícola 2. Já a coluna 2 da linha 2 mostra os vinhos classificados como sendo da vinícola 2 e que efetivamente são da vinícola 2.



Por fim, o último passo foi normalizar as saídas da rede neural. A rede possui três saídas e cada uma delas tem associada a si um valor que indica a sua pertinência em relação a entrada. Por exemplo, quanto mais o padrão de entrada for reconhecido como sendo da vinícola 3, mais o valor da saída 3 se aproximará de 1. Assim, para normalizar a saída o que fizemos foi pegar a saída máxima dentre as três. Caso a saída máxima for a primeira, então a nossa saída normalizada será o texto **Vinícola 1**. Já caso a saída máxima for a segunda, então a saída normalizada será o texto **Vinícola 2** e assim por diante.

```
simulacaoNormalizada = {}
for i=1:size(simulacao, 2)
     maximo = max(simulacao(:, i));
     if maximo == simulacao(1, i)
          simulacaoNormalizada{i} = 'Vinícola 1';
     elseif maximo == simulacao(2, i)
          simulacaoNormalizada{i} = 'Vinícola 2';
     elseif maximo == simulacao(3, i);
          simulacaoNormalizada{i} = 'Vinícola 3';
     end
end
simulacaoNormalizada
```

Código

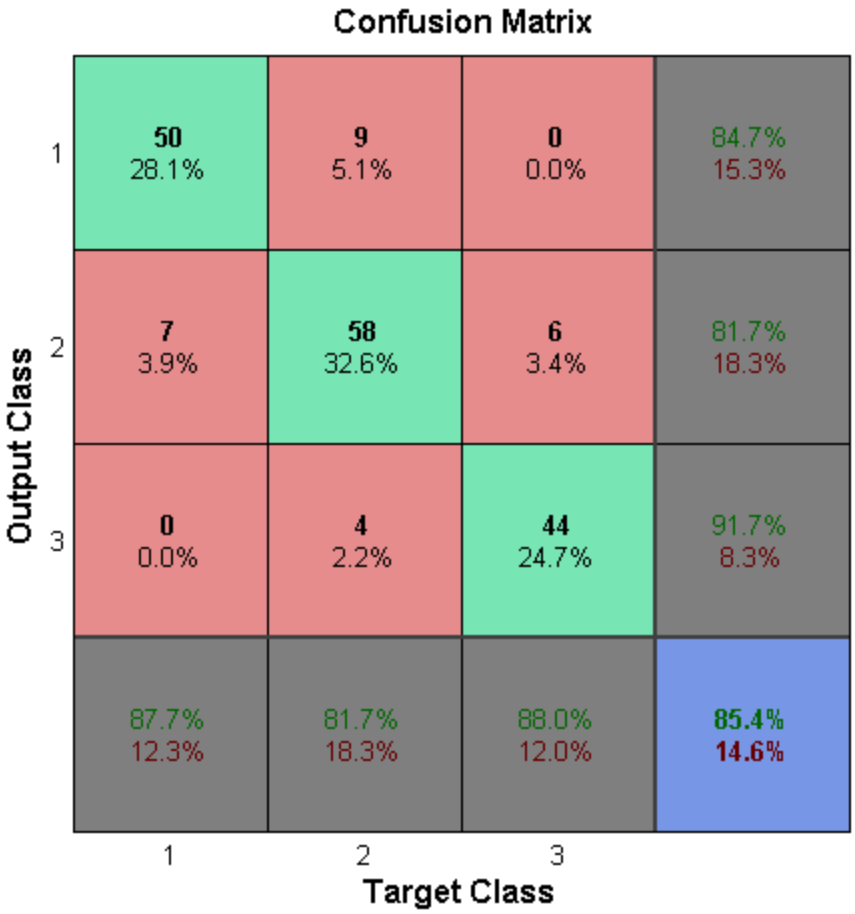
O códio completo com a criação e execução da rede neural é o que segue:

```
x; %entradas%
t; %saídas&
xn = mapminmax(x, -0.9, 0.9); %normalização das entradas%
neuronios = 10;
funcaoDeSaida = {'tansig'};
algoritmoDeTreinamento = 'traingdm';
rede = newff(
     xn,
     t,
     neuronios,
     funcaoDeSaida,
     algoritmoDeTreinamento);
rede.trainParam.epochs = 10000; %número de épocas%
rede.trainParam.goal = 0.01; %erro%
rede.trainParam.lr = 0.05; %taxa de aprendizado%
rede.trainParam.mc = 0.85; %taxa de momento%
rede.divideParam.trainRatio = 0.7; %treinamento: 70%%
rede.divideParam.valRatio = 0.15; %validação: 15%%
rede.divideParam.testRatio = 0.15; %teste: 15%%
redeTreinada = train(rede, xn, t);
simulacao = sim(redeTreinada, xn);
plotconfusion(simulacao, t);
simulacaoNormalizada = {}
for i=1:size(simulacao, 2)
     maximo = max(simulacao(:, i));
     if maximo == simulacao(1, i)
          simulacaoNormalizada{i} = 'Vinícola 1';
```

```
    elseif maximo == simulacao(2, i)
        simulacaoNormalizada{i} = 'Vinícola 2';
    elseif maximo == simulacao(3, i);
        simulacaoNormalizada{i} = 'Vinícola 3';
    end
end
simulacaoNormalizada
```

Outros testes

Decidimos então, realizar outros testes e diminuir o número de épocas e o valor do erro. Diminuímos o número de épocas para 1000 e o valor do erro para 0.1. Após isso, treinamos novamente a rede e simulamos ela várias vezes. Nessas simulações a taxa de acerto ficou entre 81% e 93%.

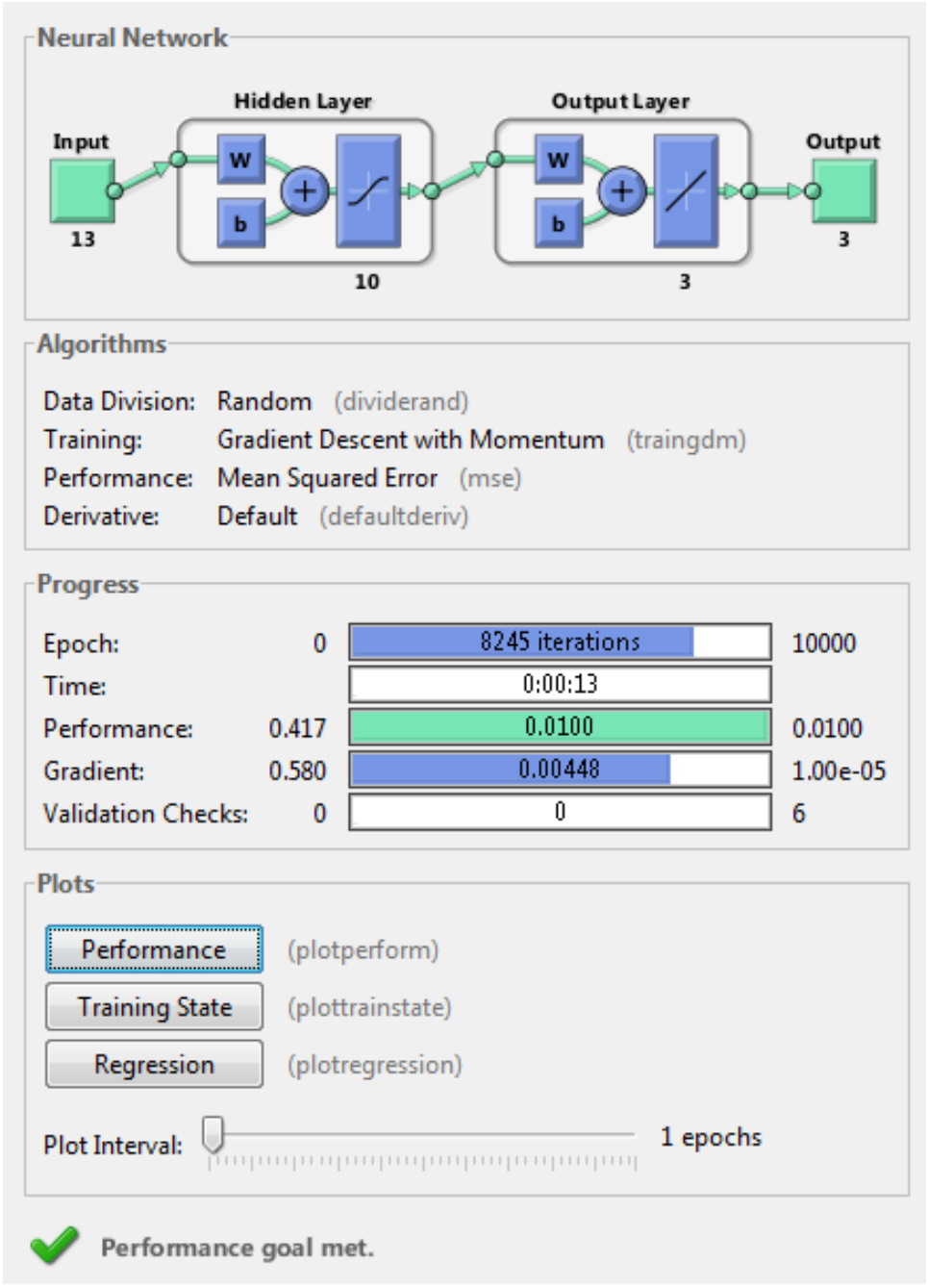


O próximo passo foi avaliar o impacto que o número de neurônios produz na taxa de acertos. Em nosso treinamento inicial utilizamos 10 neurônios na camada intermediaria. Ao alterar esse valor para 5 e 15 e não percebemos diferenças significativas. Decidimos então, testar casos extremos e utilizamos apenas 1 neurônio na camada intermediaria. O resultado nos surpreendeu já que com apenas 1 neurônio a taxa

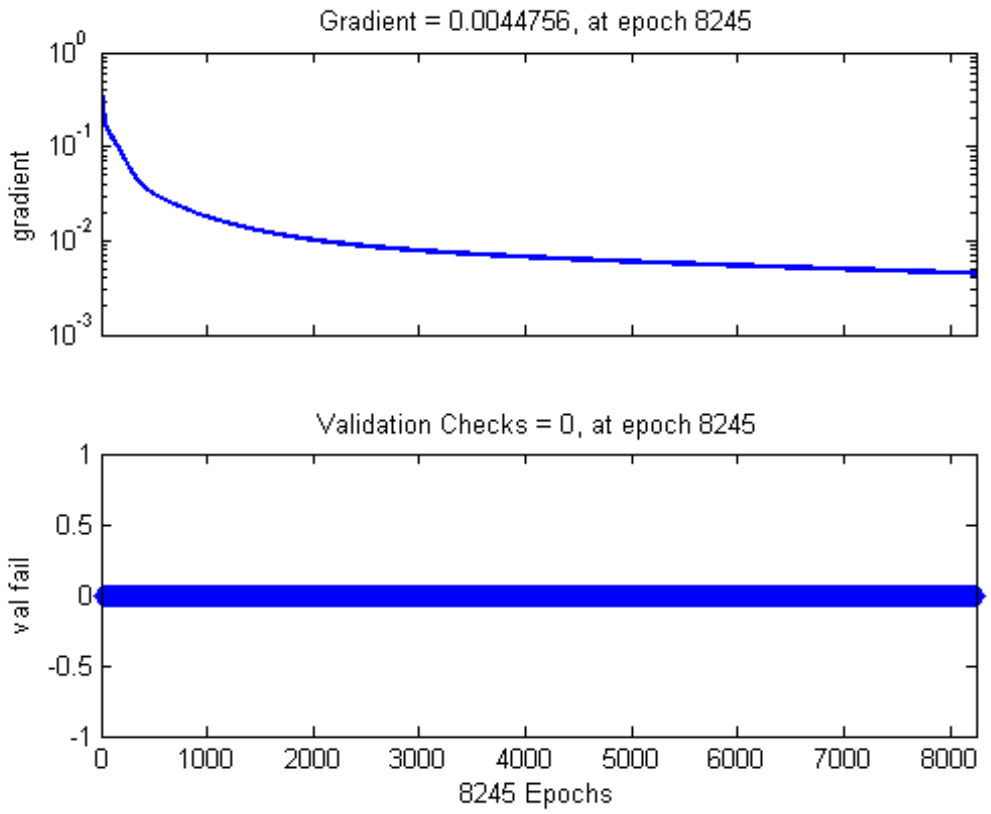
de acerto ficou acima de 60% em todas as vezes que realizamos a simulação. Tentamos também utilizar uma grande quantidade de neurônios na cada intermediaria (acima de 100 neurônios) e nesses testes percebemos o que já era esperado: uma queda na taxa de acerto que caiu para valores em torno de 30%. Com esses testes percebemos que no caso específico desse problema o número de neurônios não causa tanto impacto para uma boa performance da rede a não ser que esse número seja muito grande.

Também realizamos um teste substituindo a função de saída tangente hiperbólica por uma função sigmóide. Para realizar esse teste foi necessário alterar o intervalo de normalização já que a função sigmóide tem imagem no intervalo `(0, 1)`. Nesse caso, normalizamos a nossa entrada no intervalo `[0.1, 0.9]`. A substituição da função de saída nessa caso pouco interferiu no processo de aprendizado da rede. A taxa de acerto da rede com a utilização da função sigmóide foi semelhante a taxa de acerto com a utilização da função tangente hiperbólica.

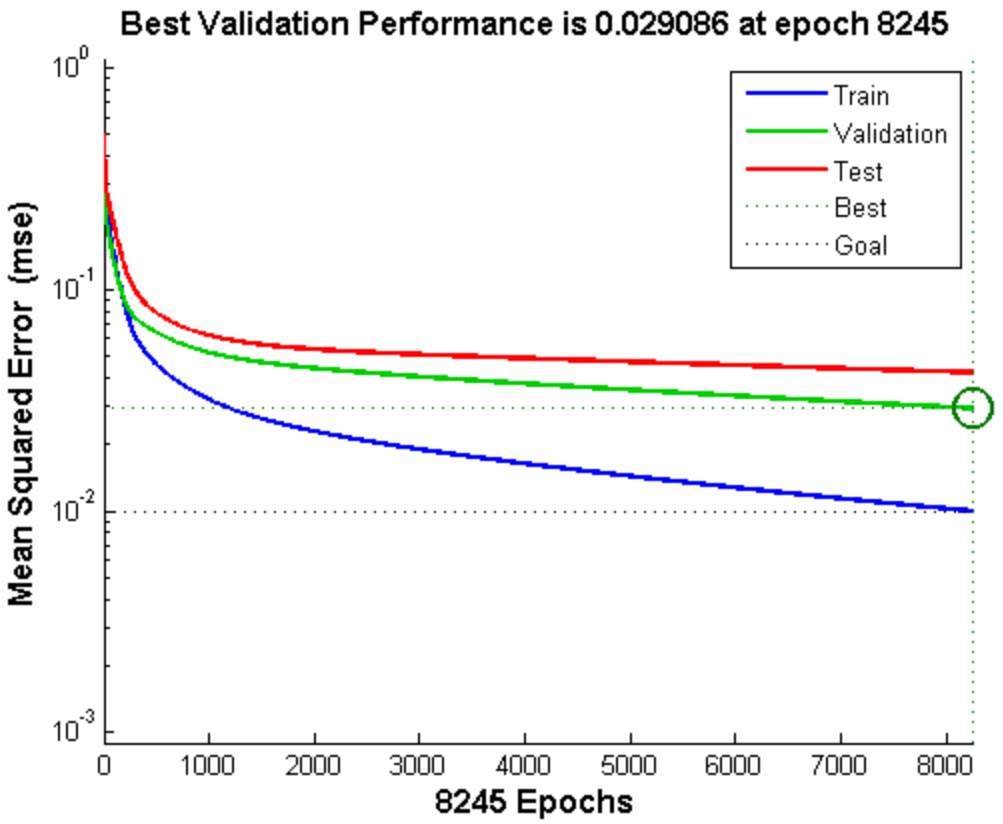
Anexo I


Neural Network
Hidden Layer
Output Layer
Input
W
b
W
b
Output
13
10
3
3
Algorithms
Data Division: Random (dividerand)
Training: Gradient Descent with Momentum (traingdm)
Performance: Mean Squared Error (mse)
Derivative: Default (defaultderiv)
Progress
Epoch: 0 8245 iterations 10000
Time: 0:00:13
Performance: 0.417 0.0100 0.0100
Gradient: 0.580 0.00448 1.00e-05
Validation Checks: 0 0 6
Plots
Performance (plotperform)
Training State (plottrainstate)
Regression (plotregression)
Plot Interval: 1 epochs
Performance goal met.

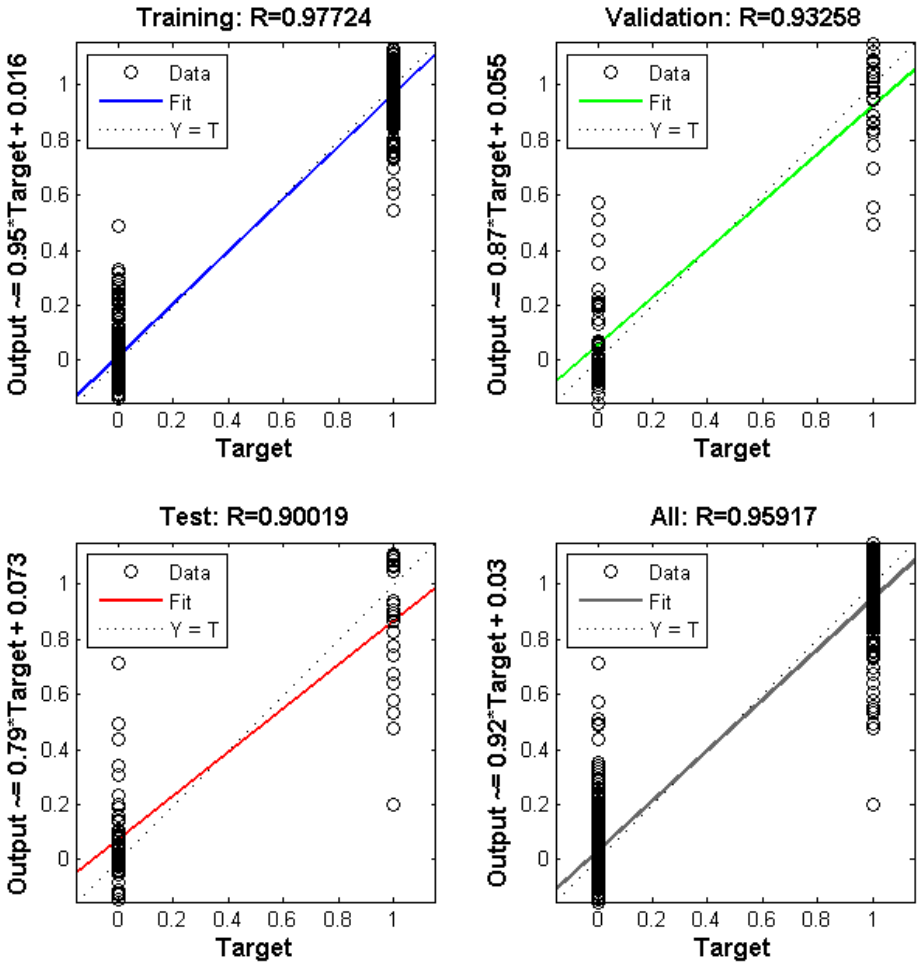
Anexo II


Gradient = 0.0044756, at epoch 8245
gradient
10^{0}
10^{-1}
10^{-2}
10^{-3}
Validation Checks = 0, at epoch 8245
val fail
1
0.5
0
-0.5
-1
0
1000
2000
3000
4000
5000
6000
7000
8000
8245 Epochs

Anexo III


Best Validation Performance is 0.029086 at epoch 8245
Mean Squared Error (mse)
8245 Epochs
Train
Validation
Test
Best
Goal
10^0
10^-1
10^-2
10^-3
0
1000
2000
3000
4000
5000
6000
7000
8000

Anexo IV


Training: R=0.97724
Output ~= 0.95*Target + 0.016
Data
Fit
Y = T
Target
Validation: R=0.93258
Output ~= 0.87*Target + 0.055
Data
Fit
Y = T
Target
Test: R=0.90019
Output ~= 0.79*Target + 0.073
Data
Fit
Y = T
Target
All: R=0.95917
Output ~= 0.92*Target + 0.03
Data
Fit
Y = T
Target