Aprendizagem Computacional

Relatório do Trabalho 2

"OCR – Optical Character Recognition"

André Gouveia – 2014216306
Joel Pires - 2014195242

Universidade de Coimbra

Departamento de Engenharia Informática

Mestrado em Engenharia Informática

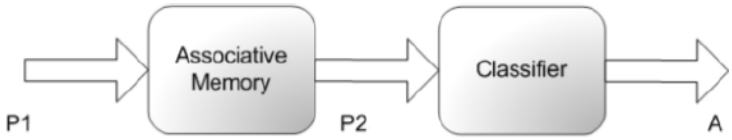
Índice

Introdução

Este trabalho é feito no âmbito da cadeira de Aprendizagem Computacional. Tem como finalidade fazer um programa reconhecimento ótico de caracteres (OCR), nomeadamente dos caracteres de 0 a 9. Para tal vai se recorrer ao conhecimento adquirido nas aulas teóricas sobre redes neuronais e ao Matlab (com o Neural Networks Toolbox) para implementar o OCR.

Vão ser usadas duas arquiteturas distintas :

1) Uso de uma memória associativa + um classificador



2) Uso de apenas um classificador

Assim sendo, poderemos fornecer ou não à memória associativa os caracteres a identificar. Esta encarrega-se de filtrar ou corrigir os caracteres que não são perfeitos, de modo a aproximá-los dos caracteres perfeitos. Caso se faça essa correção ou não prévia, os dados passarão pelo classificador que tratará de identificar os caracteres.

Implementação

Requer-se então construir um programa que classifique corretamente caracteres que o utilizador desenhe. Para esse efeito atribui-se a cada elemento do conjunto de dados de entrada uma classe de 1 a 10 que corresponde intuitivamente ao número 1, 2, 3… 0(classe 10).

Se usarmos a memória associativa, então esses caracteres desenhados passarão por uma purificação. Independentemente de este passo ser feito ou não, o classificador recebe os caracteres purificados ou não e, recorrendo a uma rede neuronal previamente treinada, cada uma das suas entradas é processada de forma a produzir uma classificação para cada uma delas e respetiva identificação do caracter em questão.

O treino da rede neuronal consiste em passar à função *train* casos que nós fizemos onde consta repetidas vezes a sequência de 1 a 0 e um determinado target.

Conforme é dito no enunciado, a rede neuronal a usar tem apenas uma camada de 10 neurónios com uma certa função de ativação e um certo bias. Usamos uma aprendizagem e treino cíclico (*learnp & trainc*) para o caso de a função ser *hardlim* e gradiente descendente (*leargd & traingd*) no caso do *purelin* e *logisg.*

A teoria do funcionamento do nosso programa está resumidamente explicada, passemos a uma análise mais detalhada da implementação do mesmo no Matlab:

Código Previamente Fornecido

- *mpaper.m*: através desta função o utilizador pode desenhar os dígitos a classificar e chamar a função *ocr_fun*.

- *ocr_fun.m*: chama a função *myclassify* e mostra o resultado da classificação.

- *grafica.m*: mostra até 3 dígitos desenhados pelo utilizador.

- *showim.m*: mostra os dígitos desenhados

Código Implementado

- *myclassify.m*: é através desta função que o utilizador pode escolher a arquitetura e função de ativação da rede neuronal quer para classificar os dígitos desenhados.

- *main.m*: funcão principal que recebe dois inteiros que significam: o tipo de arquitetura escolhida e a função de ativação. É nesta função que o utilizador pode treinar a sua rede neuronal com os casos de treino que definiu.

Correr Programa:

- Para treinar uma rede neuronal com a arquitetura e função de ativação que se queira basta correr por exemplo: *main(x,y), onde:*

    - *Se **x** é igual a:*
        - ***1***: é escolhida a arquitetura classificador + memória associativa
        - ***2***: é escolhida a arquitetura com classificador só.

    - *Se **y** é igual a:*

        - ***1***: é escolhida a função de ativação *harlim*
        - ***2***: é escolhida a função de ativação *purelin*
        - ***3***: é escolhida a função de ativação *logsig*

- *Para classificar dígitos desenhados, basta correr mpaper* -> desenhar dígitos -> *alt + botao direiro do* rato -> escolher opção para classificar dígitos dentre o menu apresentado:

```
[Tipo de classificador]
 [1]- Associative Memory + Hardlim
 [2]- Associative Memory + Purelin
 [3]- Associative Memory + Logsig
 [4]- Hardlim
 [5]- Purelin
 [6]- Logsig
```

Testes e Observações

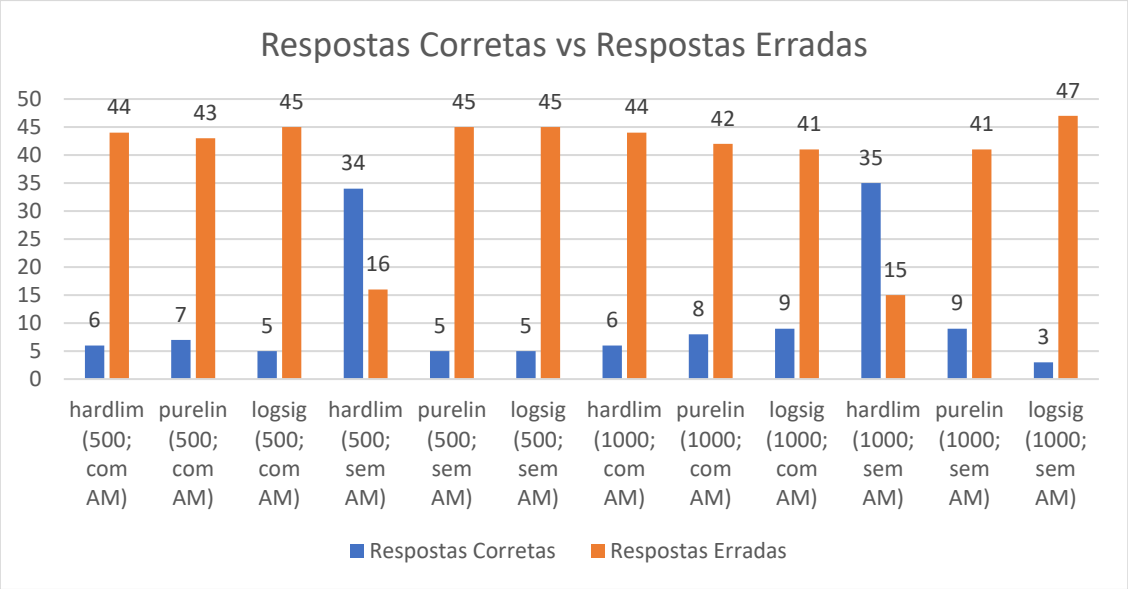
Foi feita uma bateria de testes em que vários parâmetros foram testados, a saber:

- Nº de casos de Treino Diferentes: 500 e 1000
- Arquiteturas Diferentes: classificador ou classificador + AM
- Funções de Ativação Diferentes: hardlim, purelin, logsig

Todos os resultados dos testes encontram-se expostos na seguinte tabela. As imagens dos testes realizados de onde se retiraram estes dados encontram-se no **Anexo A.**

nº casos de treino	Arquitetura	Função de Ativação	Nº de dígitos corretos	Nº de dígitos errados	Percentagem de dígitos corretos	Percentagem de dígitos errados	Épocas
500	Com AM	hardlim	6	44	12%	88%	1000
500	Com AM	purelin	7	43	14%	86%	10000
500	Com AM	logsig	5	45	10%	90%	10000
500	Sem AM	hardlim	34	16	68%	32%	1000
500	Sem AM	purelin	5	45	10%	90%	10000
500	Sem AM	logsig	5	45	10%	90%	10000
1000	Com AM	hardlim	6	44	12%	88%	1000
1000	Com AM	purelin	8	42	16%	84%	10000
1000	Com AM	logsig	9	41	18%	82%	10000
1000	Sem AM	hardlim	35	15	70%	30%	1000
1000	Sem AM	purelin	9	41	18%	82%	10000
1000	Sem AM	logsig	3	47	6%	94%	10000

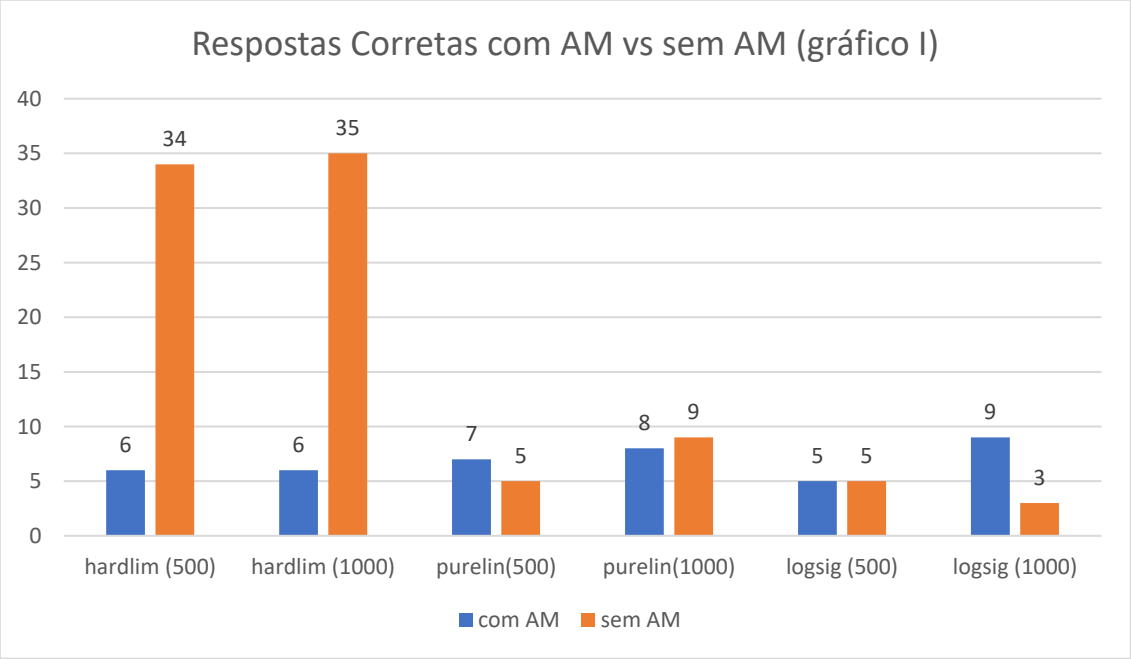
Recorramos a gráficos que nos permitam visualizar melhor os resultados obtidos. Uma mera representação dos dígitos corretos e errados de cada entrada da tabela em cima permite-nos criar o seguinte gráfico de colunas:

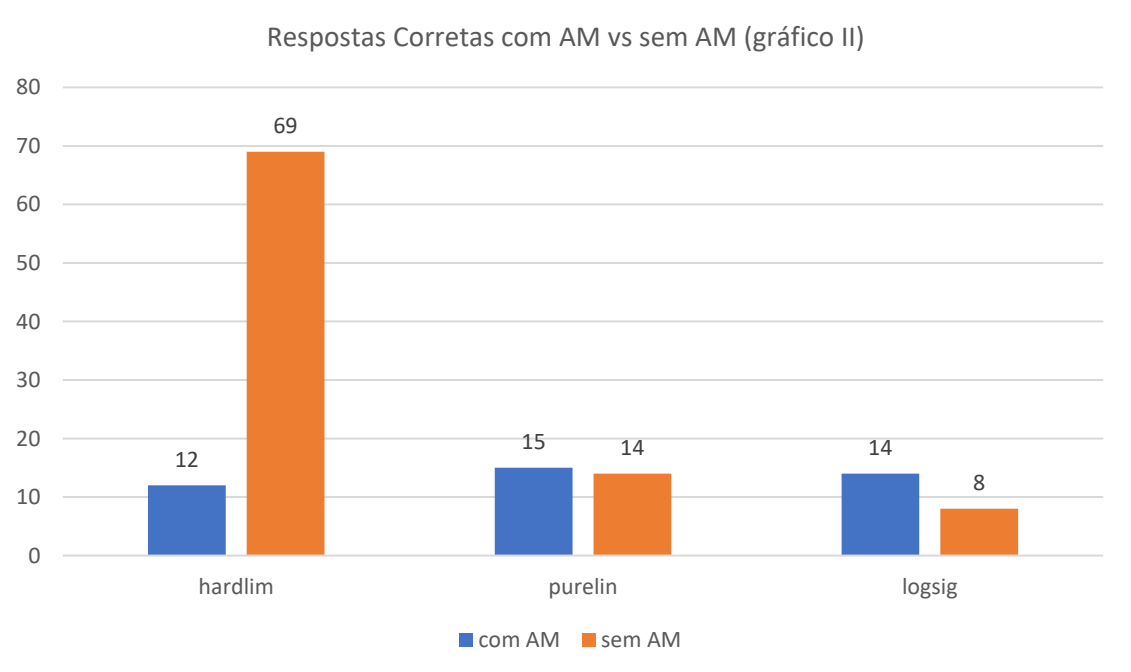


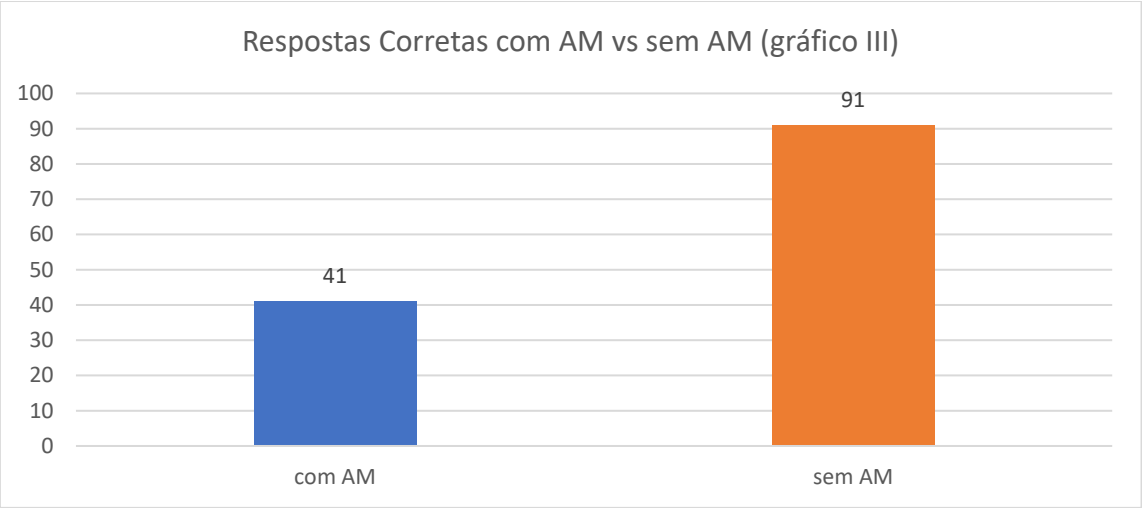
Daqui podemos concluir que: **obtivemos a nossa melhor performance usando a função de ativação *hardlim* com 1000 casos de treino e sem uso de memória associativa. Também, a nossa pior performance verificou-se quando usamos a função de ativação *logsig* com 1000 casos de treino e sem uso de memória associativa.**

Mas, qual é a influência que a memória associativa tem na performance?

Exploremos a influência que a memória associativa tem na performance da nossa rede neuronal com auxílio dos seguintes 3 gráficos:





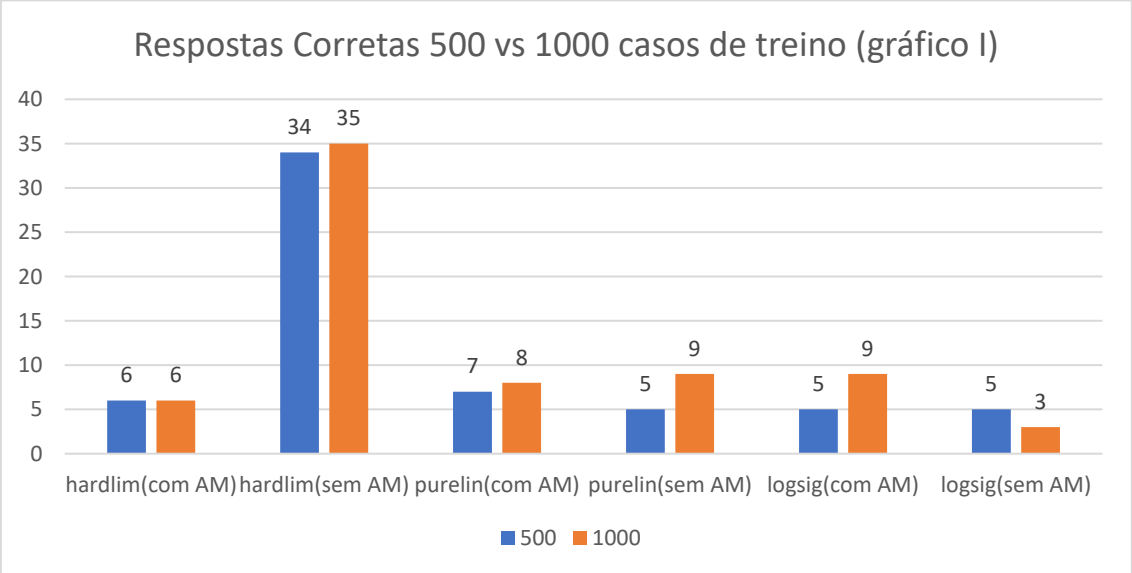


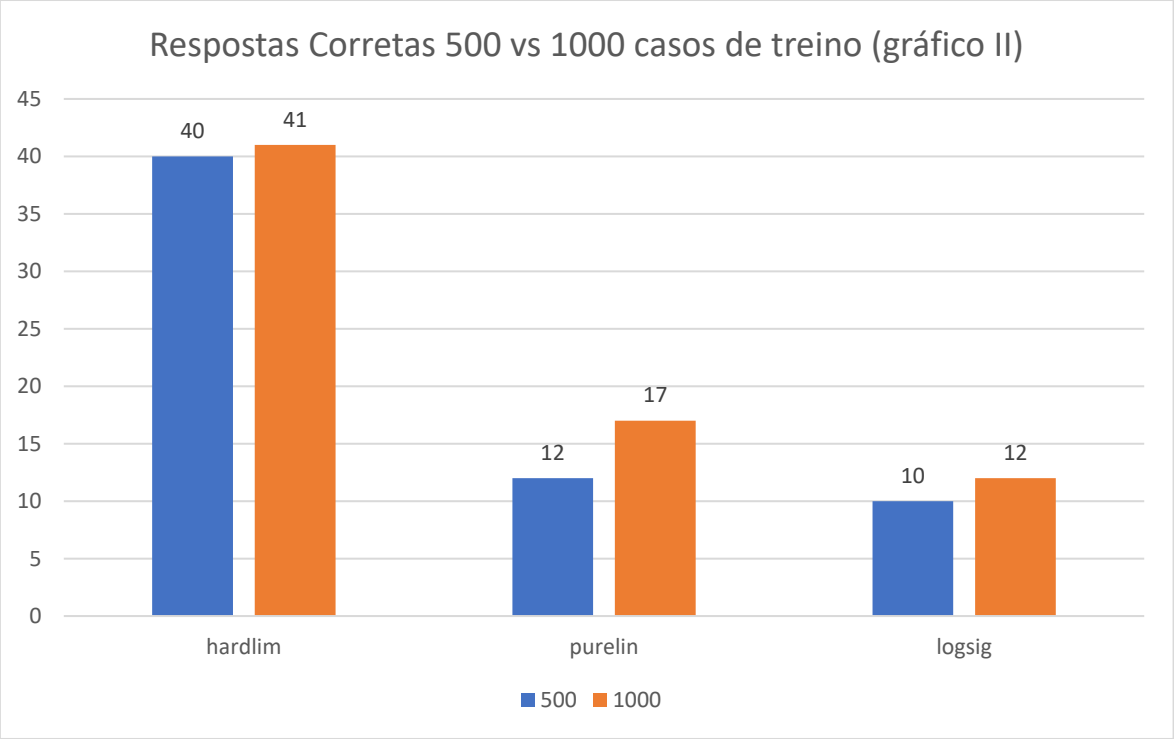
Da análise dos gráficos depreende-se que de uma forma geral, o uso de memória associativa faz decrescer a performance da nossa rede neuronal. No entanto, não se verificou isso para todas as funções de ativação, notemos que apenas a função *hardlim* recebeu um avanço considerável de performance quando se deixou de usar a memória associativa. De qualquer forma, no geral a utilização de apenas um classificador aumentou a taxa de sucesso.

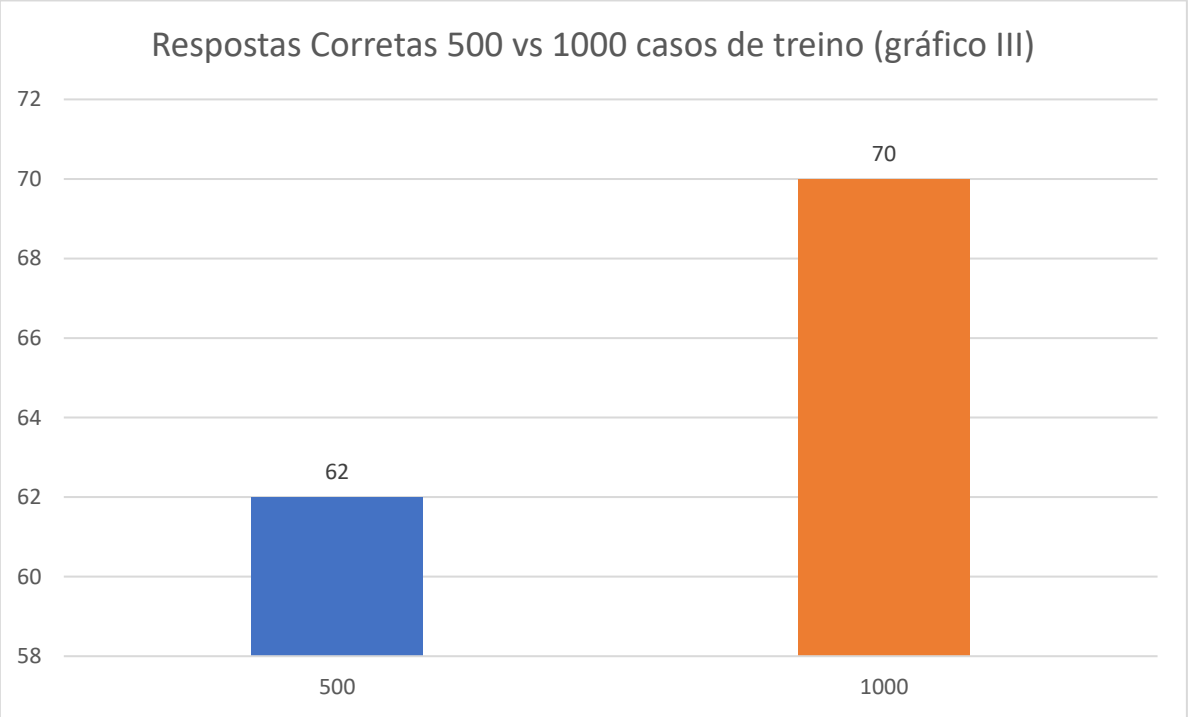
Teoricamente parecia-nos que a memória associativa deveria gerar melhores resultados dado o "aperfeiçoamento" que faz. No entanto vejamos que esta simula um conjunto de neurónios onde se um deles é ativado, então muito provavelmente os neurónios à sua volta também serão ativados; por isso a probabilidade de uma dada entrada ser classificada em várias classes simultaneamente é muito maior.

Que dizer da Influência do nº de casos de treino com que treinamos a nossa rede neuronal?

Vejamos os seguintes 3 gráficos que põem em confronto os resultados obtidos do uso de 500 casos de treino e 1000 casos de treino:



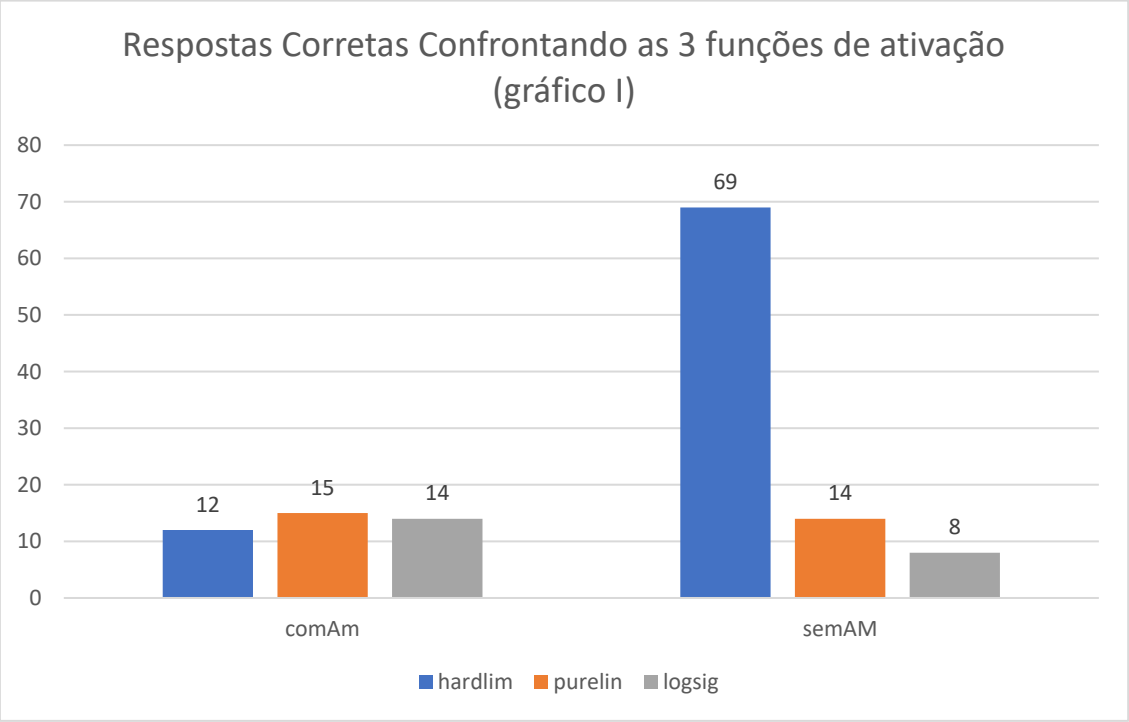


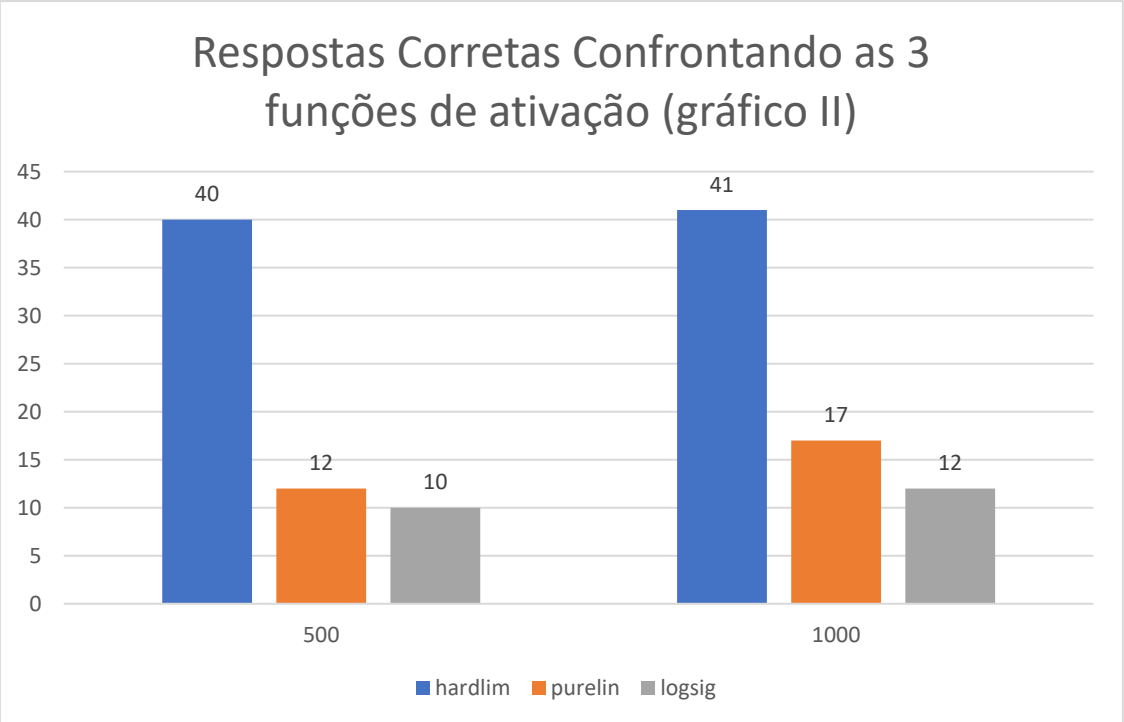


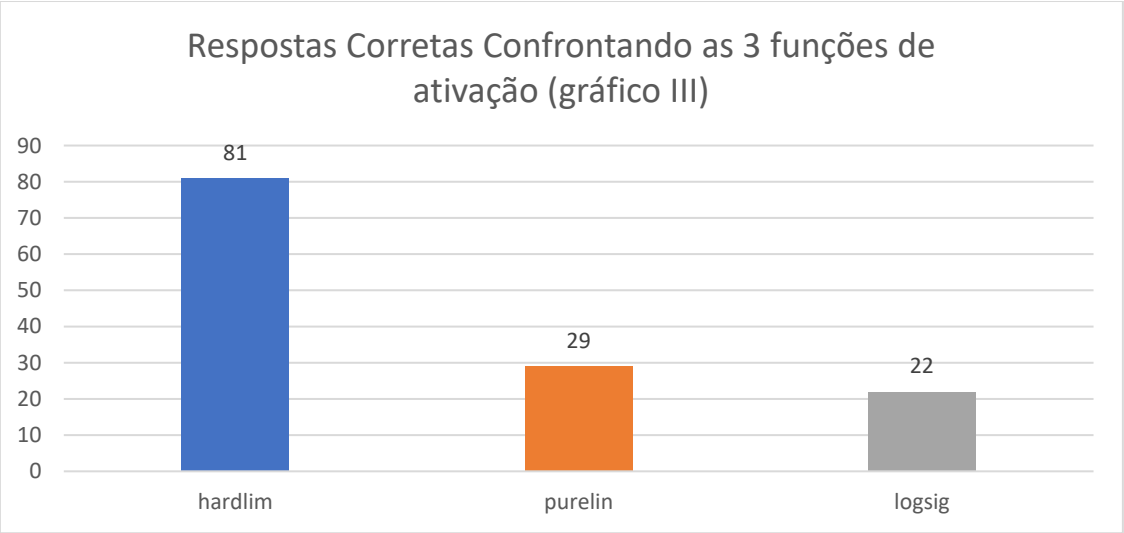
A conclusão da análise dos gráfico é óbvia e podemos dizer com certeza que o aumento dos casos de treino aumenta de facto a performance da nossa rede neuronal. De destacar que essa subida de performance se verifica em qualquer função de ativação que se use.

Qual a função de ativação que nos permite obter melhores resultados?

Mais 3 gráficos foram elaborados com o propósito de comparar lado a lado a performance das 3 funções de ativação utilizadas consoante se fazia variar a arquitetura e o nº de casos de treino.







Usando memória associativa, a função *purelin* é a que verifica maior vantagem; mas sem essa memória é a função *hardlim* que manifesta melhor performance. A variação do nº de casos de treino afeta da mesma forma as funções de ativação. No fim, conclui-se que foi com a função *hardlim* que se obteve a melhor performance de todas.

Depois de toda esta análise, há espaço ainda para responder a outras perguntas importantes:

Como é que o data set influencia a performance do sistema classificador?

Tem que haver um grande número de casos de treino e estes devem ser diversificados de forma a que o sistema classificador tenha uma boa performance. Convém então variar o tipo de escrita e o tamanho dos caracteres. Só assim é que podemos ter uma aprendizagem mais generalizada. Se a caligrafia for sempre muito semelhante (tipo e tamanho de caracteres muito semelhantes) a aprendizagem será indesejavelmente mais específica.

O sistema classificador é capaz de cumprir o objetivo de classificar os dígitos? Que percentagem de dígitos foi bem classificada?

Podemos responder positivamente à primeira pergunta, já que, conforme se observa na tabela inicial desta secção, conseguimos chegar a uma percentagem de sucesso de 70%. Se aumentássemos o número de casos de treino e a diversidade destes estamos certos que o nosso classificador teria taxas de sucesso perto dos 100%.

Como é a capacidade de generalização? O sistema classificador é suficientemente robusto? Qual a percentagem de novos dígitos bem classificados?

Sim, o classificador tem a capacidade de generalizar, mas os desenhos têm que ser razoavelmente legíveis. Novos tipos de letra e representações estranhas originaram taxas de sucesso baixas. É natural que a performance seja inferior para novos exemplos não contemplados nos exemplos de teste - overfitting. Agora, quando isso não acontece, poderemos dizer que conseguimos resultados bem razoáveis.

Conclusão

Chegamos ao fim do trabalho e podemos dizer que conseguimos ter contacto com diversos métodos de otimização e construção de redes neuronais. Percebemos também melhor o funcionamento de memórias associativas.

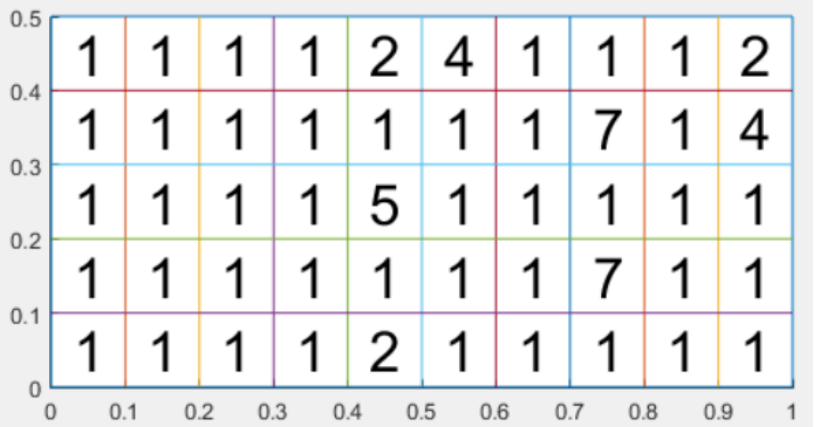
Inicialmente julgamos que auxiliar o classificador com uma memória associativa produzisse maior taxa de sucesso. No entanto, no fim deste trabalho concluímos que o contrário se verifica, apenas o classificador produz melhores resultados. Este facto dá-se muito provavelmente porque os casos de treino efetuados não foram muito bons para a rede associativa.

Ainda com respeito ao treino gostaríamos de salientar que tentámos diversificar os casos de treino nos 500 e 1000 desenhados, de modo a que a rede neuronal fosse capaz de generalizar o máximo possível no seu reconhecimento de caracteres minimamente diferentes dos padrões de treino. De facto, concluímos que o classificador é capaz de classificar corretamente caracteres ligeiramente diferentes dos casos de treino.

Gostaríamos ainda de referir que o enunciado menciona ainda a regra de Hebb mas só faz sentido aplicar essa regra para matrizes ortonormadas. Ora, o utilizador desenha qualquer conjunto de caracteres e nem sempre é possível é possível produzir uma matriz ortonormada.

Anexo A

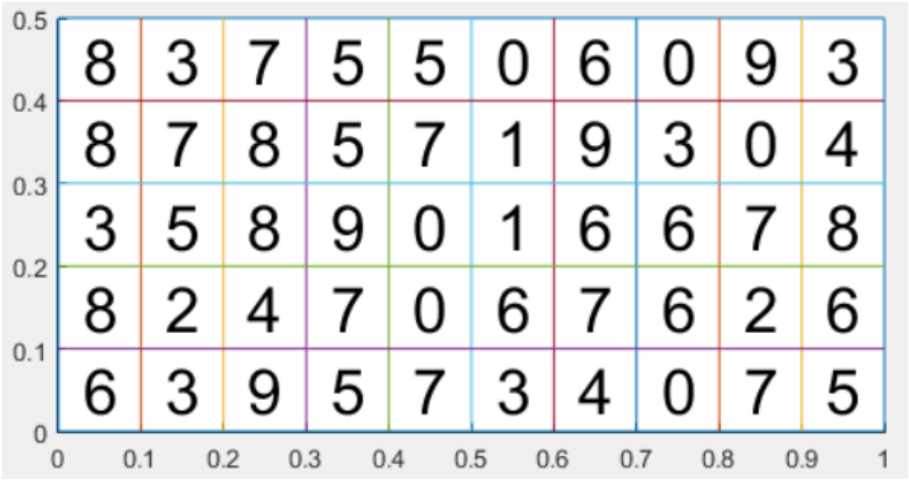
Nº de Casos de Treino – 500, Função de Ativação – hardlim, com AM? - Sim:



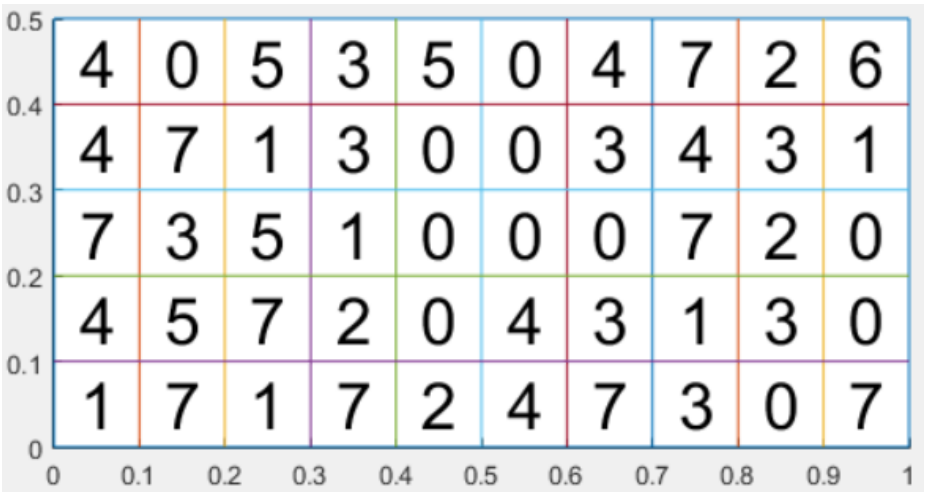
Nº de Casos de Treino – 500, Função de Ativação – hardlim, com AM? - Não:

Nº de Casos de Treino – 500, Função de Ativação – purelin, com AM? - Sim:

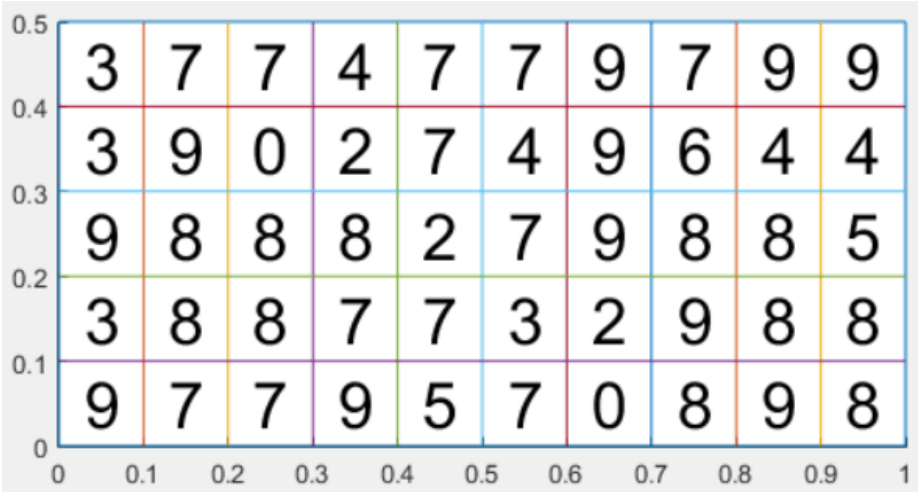
Nº de Casos de Treino – 500, Função de Ativação – purelin, com AM? - Não:



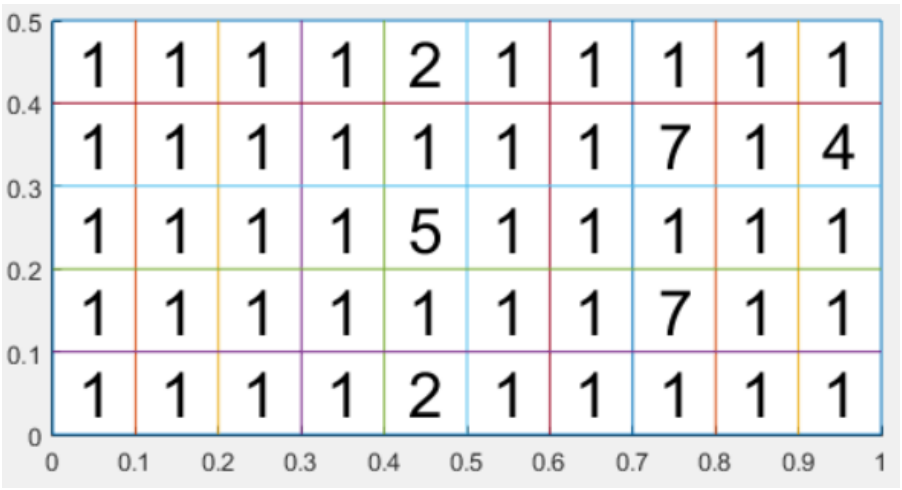
Nº de Casos de Treino – 500, Função de Ativação – logsig, com AM? - Sim:



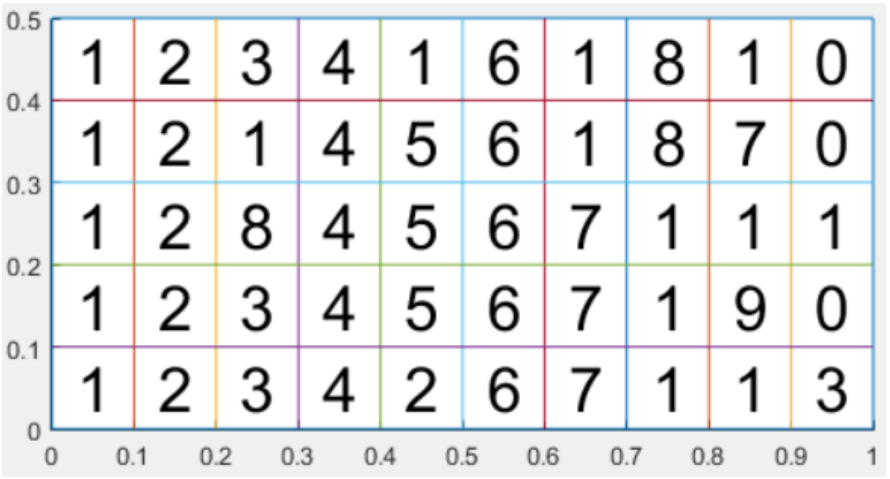
Nº de Casos de Treino – 500, Função de Ativação – logsig, com AM? - Não:



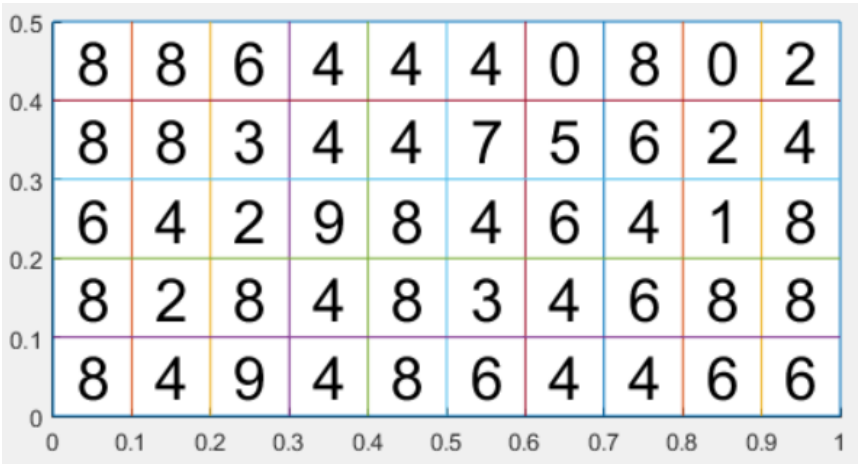
Nº de Casos de Treino – 100, Função de Ativação – hardlim, com AM? - Não:



Nº de Casos de Treino – 1000, Função de Ativação – hardlim, com AM? - Não:



Nº de Casos de Treino – 1000, Função de Ativação – purelin, com AM? - Sim:



Nº de Casos de Treino – 1000, Função de Ativação – purelin, com AM? - Não:

Nº de Casos de Treino – 1000, Função de Ativação – logsig, com AM? - Sim:

Nº de Casos de Treino – 1000, Função de Ativação – logsig, com AM? - Não: