Gonçalo Vasconcelos Correia, nº 2019216331, pdcorreia@student.dei.uc.pt

João Geirinhas, nº 2019216397, uc2019216397@student.uc.pt

Arquitetura Computacional

OCR – Optical Character Recognition

Trabalho Prático nº 1 - Turma PL 3

Coimbra, 13 de Outubro 2022

Coimbra, 6 de março 2022

Índice

1. Introdução

No âmbito da disciplina de Arquitetura Computacional foi proposto desenvolver e implementar modelos de redes neuronais para reconhecimento de caracteres manuscritos. Para isto, os caracteres que serão utilizados são os dez algarismos arábicos – {1, 2, 3, 4, 5, 6, 7, 8, 9, 0}, sendo que cada número é representado numa matriz binária 16x16.

De facto, para a realização deste trabalho precisámos de percorrer e desenvolver três etapas:

1. Desenvolvimento do Dataset;
2. Training;
3. Classifier;
4. Results.

Assim, no que diz respeito à primeira etapa foram desenhados 50 caracteres de cada vez (através da função *mpaper*. Posteriormente, foi gerada uma matriz de 256x1000 elementos (concatenação de 20 matrizes de 50 dígitos. De seguida, foi aplicado um filtro a esta matriz de modo a testar o seu efeito. Posteriormente foi utilizado um classificador que analisou as imagens e as associou a uma única classe (0, 1, 2, 3, 4, 5, 6, 7, 8, 9). Por último calculámos a *accuracy* para um caso de teste externo. De modo a obtermos os resultados que irão ser abordados ao longo deste trabalho, realizámos cada experiência três vezes, obtendo o valor médio.

Estes passos tiveram como propósito darem-nos a possibilidade de responder a certas questões, que irão ser abordadas posteriormente, relativas às diferentes etapas deste projeto:

Data set:

- How does the data set influence the performance of the classification system?

Neural network architecture:

- Which architecture provides better results:
  - the filter + classifier?
  - only the classifier with one layer?
  - only the classifier with two layers?
  - what is the advantage of the softmax layer?

Results:

- Is the classification system able to achieve the main objectives (classification of digits)? Which is the percentage of well classified digits?
- How is the generalization capacity? Is the classification system robust enough? Which is the percentage of well classified new inputs?

2. Data set

O primeiro objetivo que abordamos neste projeto foi a criação do Data set. De facto, este foi obtido com o apoio da função *mpaper*, onde foram desenhados 50 caracteres um de cada vez e de diferentes, de forma a simular diferentes tipos de letra. Após realizarmos este processo 20 vezes (1000 dígitos) concatenámos tudo numa matriz 256x1000. Por fim, tendo em conta que o Data set era relativamente grande, dividimos a matriz em 2 sets: 85% *training* e 15% *validation*, como recomendado no enunciado.

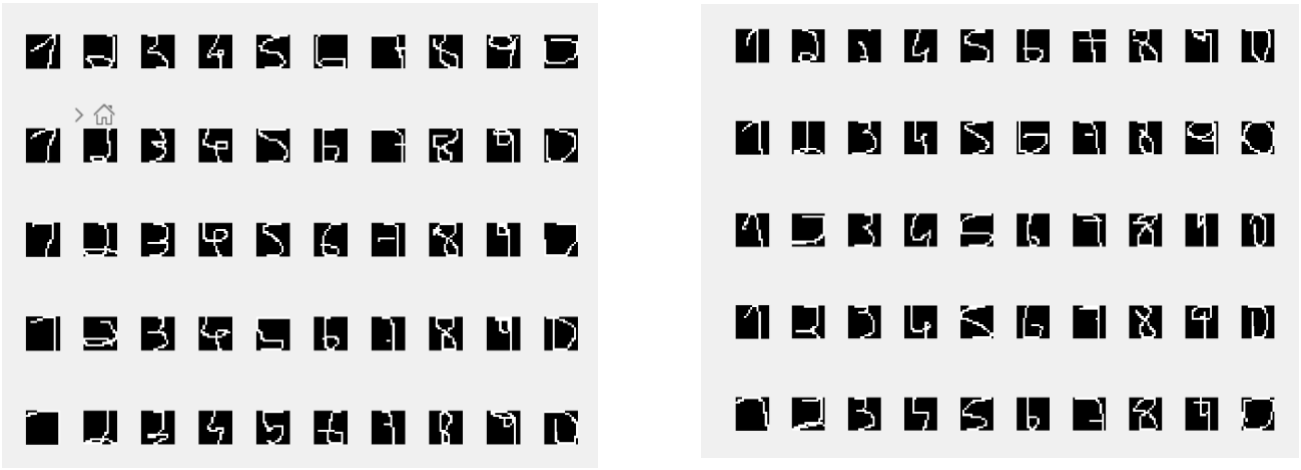

Figura 1. Exemplos de casos de teste

3. Neural network architectures

Primeiramente, de forma a entendermos melhor este ponto necessitamos de perceber o que é uma Neural network. De facto, estas são unidades funcionais de *Deep Learning* que tentam imitar o comportamento do cérebro humano para resolver problemas complexos, onde os dados de input são processados por diferentes camadas de *artificial neurons* de forma a obtermos o output desejado. Para além disso, estas apresentam diferentes arquiteturas que funcionam de diferentes formas, sendo que, neste tópico iremos abordar dois tipos, a primeira será *filter* + *classifier*, enquanto que a segunda será unicamente *classifie*r.

Primeiramente, iremos abordar a arquitetura *filter* + *classifier*. De facto, este tipo de arquitetura é composto por duas redes neuronais que estão conectadas em série, como pode ser visto na Figura 2.

Figura 2 – Filtro + Classifier

A primeira parte, denominada por filtro, recebe como input uma matriz de dimensão 256x1 que define qual o caracter a ser classificado, correspondente à matriz binária 16x16. Este "filtro" tem como principal intuito analisar um input e tenta devolver um output que, preferencialmente, seja um caracter melhor ou mais perfeito. Para além disso, neste tipo de arquitetura podem ser utilizados dois tipos de filtros, *associative memory* ou *binary perceptron*, que serão abordados posteriormente na secção 3.1.

No que diz respeito, à arquitetura só com *classifier* não há pré-filtração dos dados, pelo que o classificador necessita de ter uma considerável capacidade de generalização.

Figura 3- Single Classifier

De facto, para a realização deste trabalho o *classifier* irá receber como input uma matriz 256x1 que contém os dígitos manuscritos e devolve como output uma matriz 10x1 com as diferentes classificações tendo em conta as classes.

De forma a explorarmos este problema da melhor forma fizemos algumas escolhas relativamente à composição da network, tais como o número de layers (*one layer* ou *two layers*) e qual a função de ativação a utilizar (*hard limit*, *sigmoid*, *linear* e ainda *softmax*). Para além disso, com o intuito de percebermos qual função de ativação trazia mais vantagens para este trabalho, desenvolvemos uma função de a*ccuracy*, que nos permite comparar e analisar melhor os resultados. Estes dados podem ser analisados na secção 3.2 deste documento.

3.1 The filter + classifier

Neste ponto iremos abordar os dois tipos de filtros existentes. Primeiramente, o *associative memory* que tem a capacidade de alterar um determinado input para um output que seja um caracter mais perfeito, como pode ser analisado na Figura 4.

Figura 4-Exemplo de matriz input após filtragem com memória associativa

De seguida, temos o *binary perceptron* que utiliza 256 *hardlim neurons* de forma a filtrar um determinado input também para se obter um output mais perfeito, como se pode analisar através da Figura 5.

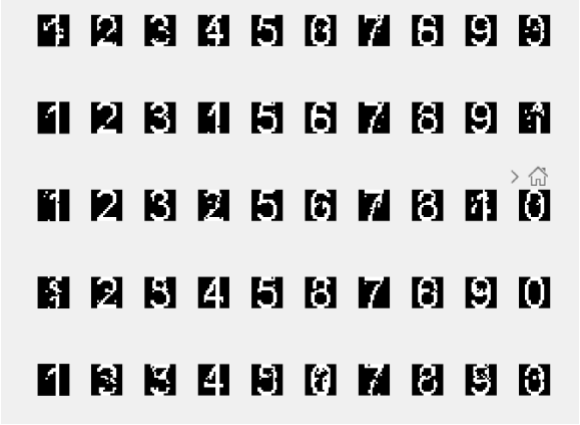

Figura 5-Exemplo de matriz input após filtragem com binary perptron

Por fim, após a aplicação dos filtros podemos utilizar os classificadores. Para esta arquitetura utilizámos três funções de ativação: *Hard Limit*, *Linear* e *Sigmoid*. Para além disso, de forma a podermos comparar resultados devolvemos uma função que nos permite analisar a *accuracy* de cada uma das funções de ativação. Estes valores podem ser consultados na Figura 6.

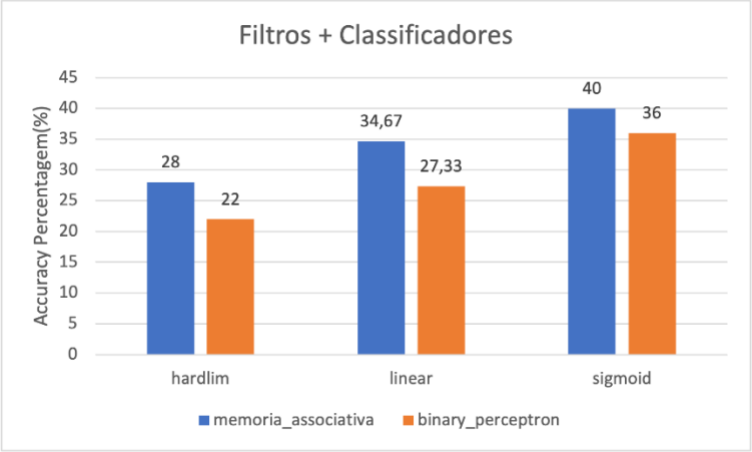

Figura 6 – Accuracy de Filtros + Classificadores

Tal como se pode concluir do gráfico anteriormente mostrado, apesar de os valores serem bastantes baixos, podemos verificar que a função *sigmoid*, usando o filtro *associative memory*, foi a que obteve os melhores resultados. No entanto, o uso do filtro *binary perceptron* também obteve um valor semelhante com a função *sigmoid*. Os valores baixos podem ser explicados pela análise das figuras 4 e 5. Na figura 4 podemos ver que há uma sobreposição e distorção dos dígitos o que torna sua visibilidade e consequentemente classificação difícil. Para além disso, na figura 5 observa-se uma pequena destruição e ruído dos caracteres, embora mais percetíveis do que na figura 4, daí os melhores resultados.

3.2 Classifier with one layer and two layers

- **One layer:**

Ao abordarmos o *classifier* com uma camada, nenhum filtro foi aplicado ao input. Para além disso, para as diferentes funções de ativação usámos distintos *learning algorithms* iterativos. De facto, para o *Hard Limit* recorremos ao *learnp*, enquanto que para a *Linear* e *Sigmoid* utilizámos o *learngd*. Paralelamente para treinar, para também usámos distintas *training functions*. Para o Hard Limit usámos o *trainc* e para as funções de ativação *Sigmoid* e *Linear* usámos uma função de treino *batch*, *traingda*.

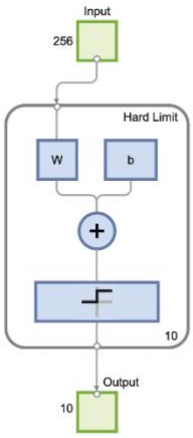

Figura 7- Arquitetura da Rede Neuronal Hardlim para uma camada

Durante a realização deste tópico considerámos inputs com diferentes dimensões – 250, 500, 1000 - de forma a tentarmos entender qual seria a melhor função de ativação e qual a influência do Data Set. De facto, como pode ser visto pela figura 8 conseguimos retirar que quanto maior o número de inputs melhor será a *accuracy*.

Isto pode-se dever ao facto de ao dispormos de mais casos de teste temos, também teremos mais dígitos escritos de diferentes formas, o que permite que se analise e treine melhor o algoritmo de forma a se obter resultados mais positivos e corretos. Para além disso, também podemos analisar que a função *sigmoid* foi a que obteve melhores resultados, independentemente do número de inputs, sendo que também deve ser destacada a função Linear, dado que alcançou uma *accuracy* relativamente boa.

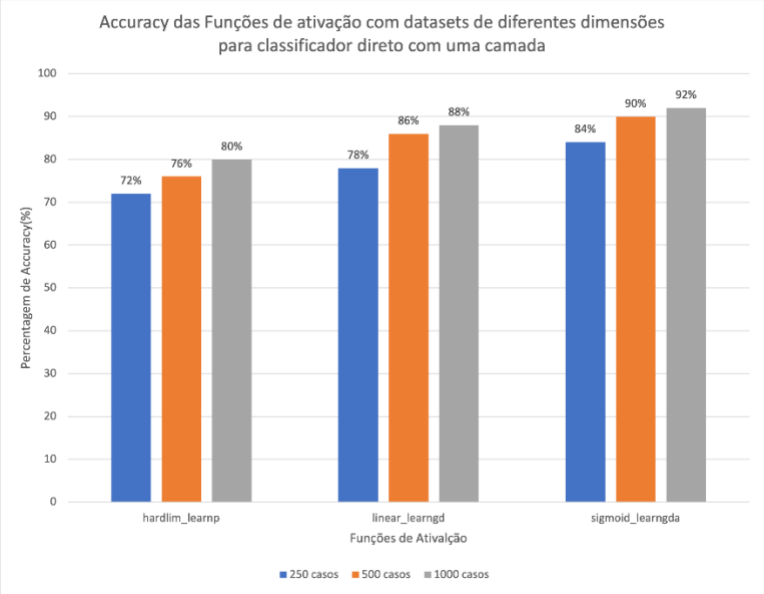

Figura 8 – Accuracy para Classifcadores com uma camada

Paralelamente, para a função de ativação *sigmoid,* podemos ver a percentagem de casos que foram corretamente classificados na validação aquando do treino (Data Set 256 x 1000).

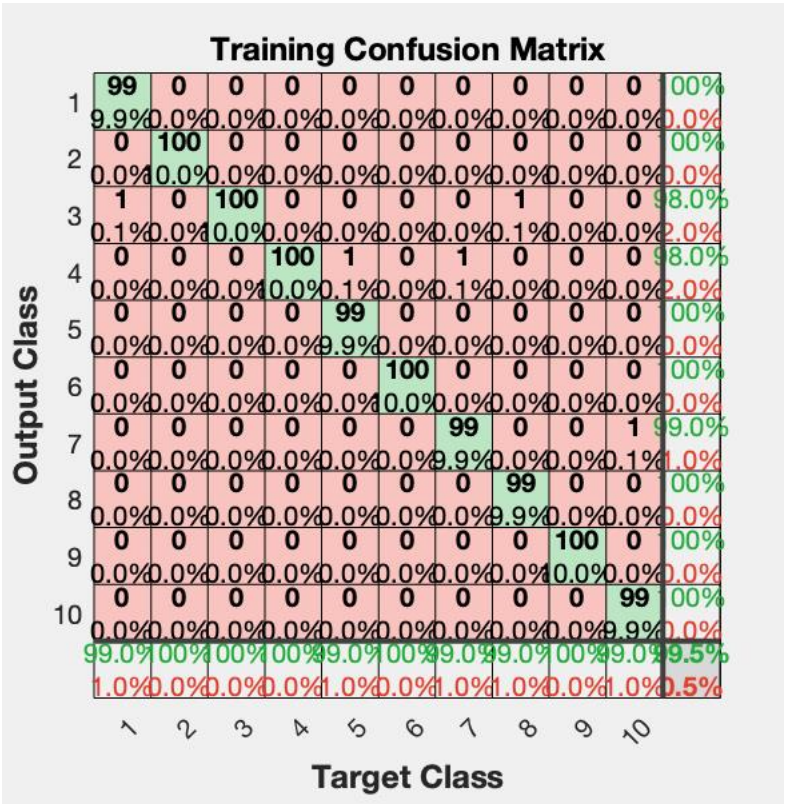

Figura 9- Training Confusion Matrix

- **Two Layers:**

No que diz respeito ao *classifier* com duas camadas, tal como uma camada, não foi utilizado nenhum filtro em

relação ao input. De facto, para este ponto também foram utilizadas as mesmas funções de ativação, bem como o mesmo número de inputs distintos (250, 500, 1000).

Neste tipo de Redes Neronais o input passa por duas camadas. Estas são denominadas por *Hidden layers*, que se encontram entre o input e o output onde *artificial neurons* usam uma gama de inputs com pesos e produzem um output através de uma função de ativação.

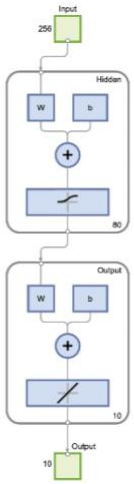

Fig 10- Arquitetura da Rede Neuronal para duas camadas Sigmoid + Linear

Para esta arquitetura com duas camadas adotámos outra função de treino. Nomeadamente optámos por escolher a função *trainscg* visto quue esta é uma função própria para *multilayer network training*. Para além disso, "*The conjugate gradient algorithms, in particular trainscg, seem to perform well over a wide variety of problems, particularly for networks with a large number of weights. The SCG algorithm is almost as fast as the LM algorithm on function approximation problems (faster for large networks) and is almost as fast as trainrp on pattern recognition problems. Its performance does not degrade as quickly as trainrp performance does when the error is reduced. The conjugate gradient algorithms have relatively modest memory requirements*" [1]. Tal como dito anteriormente, foram utilizados casos de teste com dimensões distintas. Para além disso, neste tópico considerámos apenas utilizar as duas melhores funções de ativação, a *Linear* e a *Sigmoi*d, sendo que as combinámos de forma a tentar obter os melhores valores possíveis. Posto isto, como se pode analisar na Figura 11 aquela combinação que atinge o melhor valor para a *accuracy* é a sigmoid-sigmoid. De facto, apesar de ser aquela que apresenta piores valores aquando do início da experiência, à medida que se vai testando/analisando novos casos de teste é a que acaba por alcançar o melhor valor (89.33%). De seguida, podemos também analisar que a combinação *sigmoid-linear* mantém valores bastante constantes ao longo do tempo, alcançando um valor de *accuracy* satisfatório também.

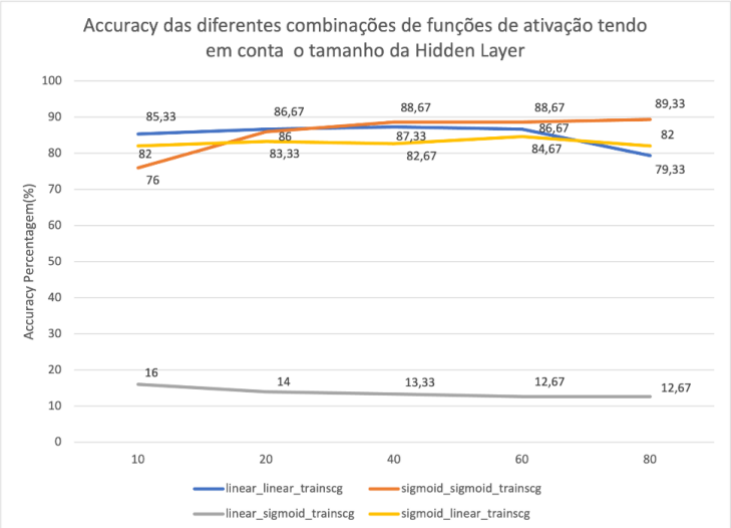

Fig 11 - Accuracy para Classificadores com duas camadas

3.3. Softmax

Neste ponto abordámos dois tipos de combinação de funções de ativação. Primeiramente, focámo-nos na combinação *sigmoid-sigmoid* e, de seguida, na combinação *sigmoid - Softmax Output*. A função *Softmax* é uma *transfer function* que calcula uma camada de output a partir da sua rede de input. De facto, nesta função o *neuron* com maior *net input* obtém um output mais perto de 1, enquanto que os outros *neurons* têm outputs pertos do 0. Para além disso, para a comparação de resultados utilizámos gamas diferentes no tamanho da *Hidden layer*, sendo que estes tamanhos foram 10, 20, 40, 60, 80.

Como pode ser analisado na Figura 12 podemos verificar que a combinação *sigmoid - softmax* foi a que obteve melhores resultados. De facto, analisando os tamanhos um a um, podemos verificar que inicialmente há uma diferença bastante acentuada, que, posteriormente, é eliminada. Por fim, podemos concluir que à medida que o tamanho da *Hidden layer* aumenta, a *accuracy* da *sigmoid - softmax* se vai tornando cada vez melhor, chegando a alcançar os 90%.

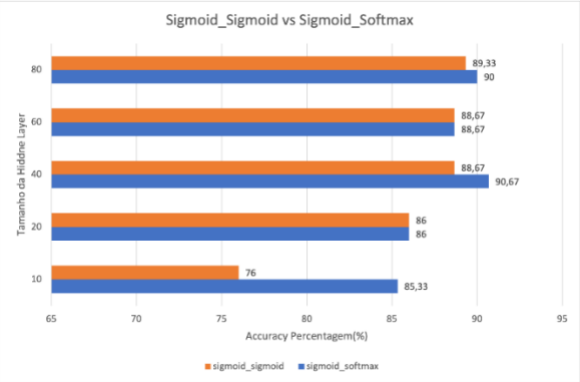

Fig 12 – Sigmoid Output layer vs Softmax Output layer

4. Interface

De forma a manusear e testar as melhores redes neuronais e também aplicar os filtros o utilizador pode usufruir da interface, figura 13.

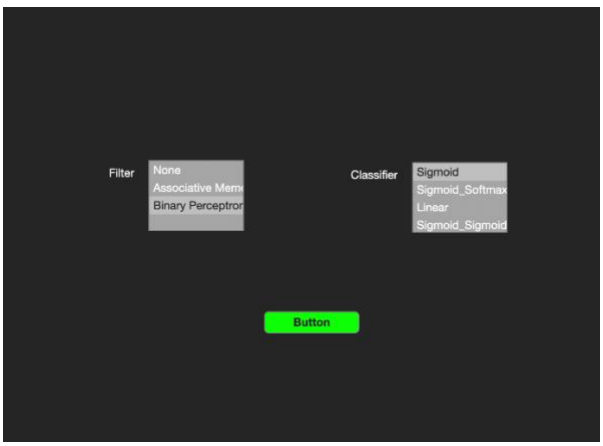

Figura 13 – Interface

Para iniciar a interface é necessária estar dentro da diretoria *source* e colocar o comando *app* na linha de comandos do matlab. Após correr o programa basta selecionar uma opção em cada uma das listas presentes: *Filter* e *Classifier*.

5. Conclusão

Em suma, como podemos analisar pelos pontos anteriormente explicados, no que diz respeito a como é que o data set influência a performance de classificação do sistema, podemos assumir que quanto maior o número

de casos de teste melhores serão os resultados. Isto deve-se ao facto de termos mais dígitos escritos de formas distintas o que permite que se analise e se treine melhor o algoritmo, o que permite alcançar melhores resultados.

Para além disso, a arquitetura que nos permitiu alcançar os melhores resultados foi a *classifier with one layer*, usando como a função *sigmoid* como função de ativação.

Por fim, analisando a *softmax layer*, é possível observar que este tem como vantagem obter uma *accuracy* superior quando comparado com uma arquitetura *sigmoid-sigmoid*, tendo em conta que ambos demoram o mesmo tempo a serem treinadas. Enquanto a função de ativação *sigmoid* é usada para classificação quando há duas classes, como é o caso (0 e 1), a *softmax* é uma versão que permite generalizar mais visto que permite classificar quando existem mais de duas classes.

Paralelamente podemos também inferir que o principal objetivo, classificar corretamente os caracteres foi alcançado. Efetivamente, para exemplo da sigmoid, melhor função de ativação que obtivemos, podemos é possível observar que 99%-100% dos casos de treino foram corretamente validados.

No que diz respeito aos resultados podemos concluir que o sistema (classificação de dígitos) é robusto o suficiente. De facto, como foi abordado ao longo deste trabalho pode ser analisado que a arquitetura *one layer* com a função *sigmoid* foi a que conseguiu alcançar uma melhor classificação de dígitos para um novo input, com uma *accuracy* de 92%. Embora tenha diminuído em relação à percentagem de validação no treino, continua a ser um sistema que possa ser generalizado.

6. Referências

[1] https://www.mathworks.com/help/deeplearning/ug/choose-a-multilayer-neural-network-training-function.html