Aprendizagem Computacional - Trabalho Prático 2

João Tiago Márcia do Nascimento Fernandes - 2011162899
Joaquim Pedro Bento Gonçalves Pratas Leitão - 2011150072

10 de Outubro de 2014

Índice

1 Introdução

Este trabalho foca-se no reconhecimento de caracteres da numeração árabe, ou seja, os caracteres 0 a 9.

Pretende-se que este reconhecimento seja realizado por uma aplicação desenvolvida em *Matlab*, que faz uso de redes neuronais na sua arquitetura interna, disponíveis na *Nerual Networks Toolbox* do próprio *Matlab*.

A aplicação desenvolvida visa o estudo de duas arquiteturas distintas no reconhecimento dos caracteres:

- Na primeira arquitetura a aplicação será constituída por uma *memória associativa* e um *classificador*
- Na segunda arquitetura a aplicação apenas recorre ao *classificador*

As duas arquiteturas apresentadas estão presentes nas figuras que se seguem:

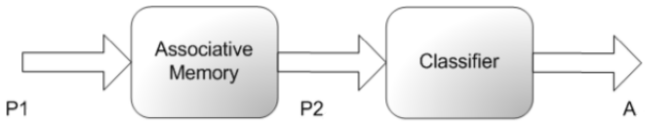

Figura 1: Arquitetura da aplicação com *memória associativa + classificador*

Figura 2: Arquitetura da aplicação apenas com o *classificador*

Através da análise destas figuras podemos determinar um comportamento padrão para a aplicação:

- Numa fase inicial, os caracteres a identificar poderão, ou não, ser fornecidos à *memória associativa*, que está encarregue da sua "filtragem"ou "correção": Se os caracteres fornecidos não forem perfeitos, a memória associativa aproxima-os dos respetivos caracteres perfeitos.
- De seguida os dados, corrigidos ou não, serão fornecidos ao *classificador*, que se encarregará de proceder à identificação dos mesmos.

No presente documento iremos proceder à apresentação em maior detalhe destas duas arquiteturas e das suas implementações, bem como da aplicação *Matlab* desenvolvida, e de como poderá ser utilizada. Pretendemos também fazer uma análise crítica da performance da aplicação, nomeadamente da sua capacidade de classificar corretamente novos caracteres fornecidos.

2 Aplicação Desenvolvida

A aplicação desenvolvida visa identificar corretamente caracteres desenhados pelo utilizador, implementando para isso as duas arquiteturas apresentadas anteriormente: *Memória Associativa + Classificador* e *Classificador*.

Ambas as arquiteturas e os respetivos modos de funcionamento serão apresentados de seguida.

2.1 Memória Associativa + Classificador

Na arquitetura *Memória Associativa + Classificador*, os caracteres desenhados pelo utilizador, que constituem o *input* da nossa aplicação, são fornecidos à *memória associativa*, onde são *"purificados"*.

Com esta operação pretende-se aproximar os caracteres desenhados pelo utilizador, que são naturalmente imperfeitos, dos respetivos caracteres perfeitamente desenhados. Ao purificarmos os dados estaríamos a aumentar a sua precisão, o que, teoricamente, resultará numa melhor classificação por parte do *classificador*.

Por seu turno, o *classificador* recebe como entradas os caracteres purificados e, com recurso a uma rede neuronal previamente treinada, processa cada uma das suas entradas, produzindo uma classificação para cada uma delas, que corresponde à identificação do caracter em questão, classificação essa que será apresentada ao utilizador.

2.2 Classificador

Ao contrário da arquitetura anterior, na arquitetura *Memória Associativa + Classificador*, os caracteres desenhados pelo utilizador não sofrem qualquer tipo de purificação, sendo fornecidos ao *classificador* tal como foram desenhados pelo utilizador.

Tal como o descrito na arquitetura anterior, o classificador recorre a uma rede neuronal previamente treinada para realizar a classificação das entradas. Uma vez classificadas as entradas, o resultado da sua classificação é apresentado ao utilizador.

2.3 Implementação em Matlab

Para o desenvolvimento da aplicação foi-nos fornecido código-fonte, que cria e disponibiliza ao utilizador uma grelha onde este irá desenhar os caracteres a identificar, encarregando-se de toda a lógica interna da aplicação com a exceção da implementação do sistema de classificação dos caracteres e de toda a lógica a ele inerente (implementação das diferentes arquiteturas da aplicação, realização dos treinos das redes neuronais implementadas, etc)

2.3.1 associativeMemory.m

Neste ficheiro encontra-se uma pequena implementação da memória associativa, presente na função *associativeMemory*. Esta implementção visa *"purificar"* os caracteres desenhados pelo utilizador, aproximando-os do respetivo caracter perfeitamente desenhado. Nesta implementação apenas são calculados os pesos da rede neuronal que constituí a *memória associativa*.

O cálculo dos pesos é feito de duas formas diferentes, com dois conjuntos de dado de treino de diferentes dimensões, escolhidos pelo utilizador. Para o efeito, foram utilizados dados de treino com 100 e 500 casos de teste.

Assim, numa situação, os pesos são calculados recorrendo à fórmula: $target \times input^T$. Na segunda situação, os pesos são calculados de acordo com: $target \times pinv(input)$, onde *input* corresponde aos dados de treino fornecidos à *memória associativa*, *target* corresponde aos respetivos valores esperados à saída da *memória associativa* e *pinv* é uma função específica do *Matlab*.

Uma vez calculados os valores desses pesos a função *associativeMemory* retorna, devolvendo os valores calculados. A *"purificação"* dos dados criados pelo utilizador só será realizada na função *myclassify* (presente no ficheiro *myclassify*), após terem sido obtidos os pesos da *memória associativa*.

2.3.2 createNetwork.m

No ficheiro *createNetwork.m* encontramos a função *createNetwork*, responsável pela criação e treino de uma rede neuronal que representará o *classificador*.

Esta rede é criada de acordo com algumas características pré-definidas, e um conjunto de características escolhidas pelo utilizador, como é o caso da função de ativação. Uma vez criada a rede neuronal, esta é treinada com um conjunto de dados previamente obtidos, e que se encontram nos ficheiros *PTreino500.mat* e *Tfinal500.mat*.

2.3.3 myclassify.m

No ficheiro *myclassify.m* encontramos a função *myclassify*, onde se encontra uma boa parte da lógica principal da aplicação.

Esta função é chamada pela função *ocr_func*, que por sua vez é chamada pela função *mpaper*, e recebe como argumento os caracteres desenhados pelo utilizador.

Caso o utilizador tenha indicado a utilização de uma *memória associativa*, e qual o método para o cálculos dos seus pesos, a função verifica se a *memória associativa* pretendida já foi previamente criada. Caso tenha sido criada, os seus pesos são carregados em memória e os caracteres desenhados pelo utilizador são *"purificados"*. Caso nenhuma tenha sido criada, a *memória associativa* é criada, os seus pesos são determinados com base em valores de treino previamente computados, e de seguida os dados fornecidos pelo utilizador são *"purificados"*.

Após esta etapa a função solicita ao utilizador dados relativos a propriedades da rede neuronal implementada pelo *classificador*, como por exemplo a *função de ativação* a utilizar. De seguida é criada e treinada a rede neuronal implementada pelo *classificador*, fazendo posteriormente a classificação dos caracteres desenhados pelo utilizador.

Caso o utilizador tenha optado por utilizar uma *memória associativa*, os dados de treino do *classificador* são também *"purificados"* antes de serem fornecidos à rede neuronal criada como dados de treino.

Após a classificação dos caracteres, esta é retornada pela função, sendo posteriormente apresentada ao utilizador.

2.3.4 run.m

Este ficheiro contém um *script* que permite executar a aplicação. Neste *script* o utilizador poderá definir alguns parâmetros da rede, nomeadamente a sua arquitetura e a função de ativação a utilizar no *classificador*. O funcionamento da aplicação, tal como a estrutura deste ficheiro, serão abordados com maior detalhe na secção que se segue.

2.4 Execução

Para executar a aplicação o utilizador deverá executar o ficheiro *run.m*. Uma vez iniciado, será pedido ao utilizador que selecione uma de duas possíveis arquiteturas para a aplicação: Utilizando *Memória Associativa* em conjunto com o *Classificador*, e utilizando apenas o *Classificador*.

Caso o utilizador selecione a utilização da *Memória Associativa*, então terá de escolher o tipo de treino a realizar na mesma.

Existem dois tipos de treino distintos, utilizando cada um, dados de treino com diferentes dimensões, que o utilizador poderá escolher. Para o efeito, foram utilizados dados de treino com 100 e 500 casos de teste.

Assim, no primeiro tipo de treino, os pesos são calculados de acordo com a seguinte fórmula: $target \times input^T$, enquanto que no segundo caso os pesos são calculados de acordo com a fórmula: $target \times pinv(input)$, onde *input* corresponde aos dados de treino fornecidos à *memória associativa*, *target* corresponde aos respetivos valores esperados à saída da *memória associativa* e *pinv* é uma função específica do *Matlab*.

Após esta seleção a *Memória Associativa* é criada, aparecendo de seguida uma grelha, onde o utilizador desenhará os caracteres a serem classificados. Assim que os caracteres são desenhados, o utilizador tem de carregar no botão do meio do seu rato, de forma a transitar para a fase de execução seguinte.

Nesta fase, o utilizador terá que escolher algumas características da rede neural que o classificador da aplicação implementa. Caso já exista alguma rede previamente criada com as características especificadas pelo utilizador, essa rede é carregada para memória e utilizada na execução. Caso ainda não exista, uma rede neuronal com as características desejadas é criada e treinada, sendo também guardada para posteriores execuções.

Uma vez obtida a rede a utilizar, esta classifica os dados introduzidos pelo utilizador, que lhe serão apresentados numa grelha semelhante à onde inicialmente desenhou os caracteres.

3 Testes Realizados

Descrição de como fizémos os casos de teste, dimensões, etc

3.1 Treino da Aplicação

3.2 Testes com a Rede Treinada

4 Conclusões

Conclusões

FIXME: Testar classificação de dígitos perfeitos e de dígitos não perfeitos (alguns não perfeitos são corretamente classificados, e todos os perfeitos são corretamente classificados)

FIXME: Perguntas relatório:

- How does the data set influence the performance of the classification system?
- Which architecture provides better results: only the classifier or the associative memory+classifier?
- Which is the best activation function: hardlim, linear or logsig?
- Does the Hebb rule perform well?
- 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 to give correct outputs when new inputs are not perfect?
- Which is the percentage of well classified new inputs?