Universidade de São Paulo
Instituto de Matemática e Estatística
Trabalho de Formatura Supervisionado
Monografia

Uma Abordagem Hierárquica para o Reconhecimento de Caracteres em Expressões Matemáticas

Cristiano P. Garcia
nUSP: 3670737
Orientadora:
Nina S. T. Hirata

29 de novembro de 2008

Durante o desenvolvimento deste trabalho, o autor recebeu auxílio financeiro do CNPq

Sumário

1 Introdução

Expressões matemáticas fazem parte do cotidiano de várias pessoas. Um sistema computacional capaz de reconhecer expressões matemáticas seria útil em diversos contextos. Por exemplo, ele poderia ser utilizado para auxiliar portadores de deficiência visual ao sonorizar uma expressão escrita pelo professor ou presente em livros e manuscritos. Outra utilidade de tal sistema seria na simplificação da produção de textos digitais que incluem grande volume de notação matemática pois, mesmo com a tecnologia disponível nos dias atuais, não é uma tarefa trivial inserir uma expressão matemática para ser manipulada pelo computador. A criação de ferramentas como TEX e posteriormente LATEX tornou este trabalho mais fácil. Entretanto, ainda não foi possível igualar estes à facilidade de escrever expressões com lápis e papel.

Esses fatos motivam o estudo e desenvolvimento de sistemas que sejam capazes de reconhecer expressões matemáticas manuscritas. O reconhecimento de expressões matemáticas envolve duas etapas. Uma, de segmentação e reconhecimento de caracteres e símbolos, e outra de reconhecimento estrutural da expressão. Em geral, essas duas etapas são tratadas separadamente [1] [7].

No caso de escrita *online*, na qual os traços são capturados juntamente com a informação temporal associada aos pontos que compõem os traços, o problema de segmentação de caracteres e símbolos é relativamente simples. O processo de segmentação de caracteres pode ser facilitado "educando-se" o usuário a escrever de uma maneira adequada (por exemplo, fazendo uma pausa maior entre traços de caracteres distintos). Deste ponto em diante iremos utilizar o termo **símbolo** para nos referirmos tanto aos caracteres quanto aos símbolos, tais como as letras do alfabeto romano, dígitos, símbolos de operadores matemáticos, etc, que podem aparecer em expressões matemáticas.

O reconhecimento de símbolos, porém, é um problema difícil devido às variações naturais inerentes à escrita manuscrita. Esse problema é agravado mais ainda no caso das expressões matemáticas, pois o número total de possíveis símbolos é bem grande.

Na etapa de reconhecimento estrutural faz-se a análise semântica da expressão. Em geral nesta etapa supõe-se que os símbolos encontram-se classificados. O reconhecimento estrutural envolve fortemente uma análise espacial das posições relativas dos símbolos, pois estes indicarão a semântica do símbolo na expressão.

Esta monografia descreve um trabalho que foi realizado no contexto do projeto **ExpressMath** [2], um projeto que visa estudar e desenvolver técnicas úteis ao reconhecimento de expressões matemáticas manuscritas. Trabalhos desenvolvidos previamente no contexto deste projeto incluem reconhecimento

de símbolos via redes neurais. Neles, porém, o conjunto de símbolos considerados para reconhecimento é bastante restrito devido à dificuldade inerente ao problema, conforme citado anteriormente.

Neste trabalho abordamos o problema de reconhecimento de símbolos em expressões matemáticas. A proposta deste trabalho é uma abordagem hierárquica para a classificação de uma grande quantidade de símbolos distintos.

O texto está organizado da seguinte forma. Na seção 2, descrevemos alguns conceitos e a abordagem proposta. Na seção 3, descrevemos as principais atividades realizadas. Na seção 4, descrevemos os resultados e os produtos (software) obtidos e, na seção 5, apresentamos as conclusões.

2 Conceitos e Tecnologias Estudadas

Nesta seção apresentamos uma descrição sobre os conceitos e as tecnologias estudadas para o desenvolvimento deste trabalho. Inicialmente iremos descrever o processo de classificação. Em seguida iremos fornecer uma breve explicação sobre o funcionamento das redes neurais artificiais e veremos alguns detalhes sobre as características da escrita *offline* e *online*. Por fim descreveremos as possíveis abordagens para o problema do reconhecimento de um grande conjunto de símbolos utilizando redes neurais.

2.1 Classificação / Reconhecimento

O reconhecimento de expressões matemáticas manuscritas envolve uma etapa que consiste do reconhecimento dos símbolos presentes na expressão. Reconhecer um símbolo significa identificar a classe ou categoria a qual ele pertence. Por exemplo, se estivéssemos considerando o problema de reconhecer os dígitos, teríamos um total de dez classes, uma para cada dígito.

Vários sistemas computacionais utilizam classificadores para fazer o reconhecimento ou classificação dos objetos em análise. No caso de expressões matemáticas manuscritas, os objetos a serem reconhecidos são justamente os elementos (símbolos e caracteres) da expressão.

Objetos a serem reconhecidos são usualmente representados por uma tupla com n componentes. Cada componente é denominado característica e tipicamente consiste de alguma medida extraída do objeto.

Um classificador recebe a representação do objeto a ser classificado e devolve o código de uma classe, a classe por ele associada ao objeto. Em geral, os classificadores possuem parâmetros que são ajustados a partir de uma amostra de dados previamente classificados. Essas amostras constituem o

chamado conjunto de treinamento. Naturalmente, um classificador treinado a partir de um determinado conjunto se presta a classificar objetos de natureza similar aos do conjunto.

2.2 Redes Neurais

As redes neurais artificiais são assim denominadas pois a motivação inicial foram as redes neurais biológicas que são redes de neurônios densamente interconectados [4].

Nas redes neurais artificiais, cada unidade que simula o funcionamento de um neurônio recebe o nome de *perceptron*. A entrada de um perceptron é um vetor $\mathbf{x}$ ao qual está associado um vetor de pesos $\mathbf{w}$. Para produzir a saída, o produto escalar $\mathbf{x} \cdot \mathbf{w}$ é calculado e o resultado é submetido a uma função chamada *função de ativação*. Alguns tipos de função de ativação são descritos abaixo ($x \in \mathbb{R}$) [6]:

- Função linear: $f_1(x) = x$.
- Função sigmóide, que comprime a reta real no intervalo [-1,1]: $f_2(x) = \dfrac{1}{1 - e^x}$
- Função binária:

$$f_3(x) = \begin{cases} 0 & \text{se } x < 0 \\ 1 & \text{se } x \geq 0 \end{cases}$$

Desta forma, a saída de um perceptron com função de ativação f, entrada $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ e peso $\mathbf{w} = (w_1, w_2, \ldots, w_n)$, pode ser resumida por:

$$f(\sum_{i=1}^{n} w_i x_i)$$

Uma rede neural pode ser composta de apenas uma camada de perceptrons. Neste caso, cada uma das unidades recebe a mesma entrada e a saída da rede é um vetor composto pelas saídas de cada unidade. A figura 1(a) ilustra uma rede deste tipo, também conhecida como *single-layer perceptron*.

Existem também redes neurais compostas por várias camadas de perceptrons. Este tipo de rede é conhecida como *multi-layer perceptron*. Neste caso, o vetor que é utilizado como entrada por cada uma das unidades é produzido pela camada imediatamente anterior. A primeira camada é chamada de camada de entrada, a última é conhecida como camada de saída e as demais são chamadas camadas ocultas. A figura 1(b) ilustra este tipo de rede.

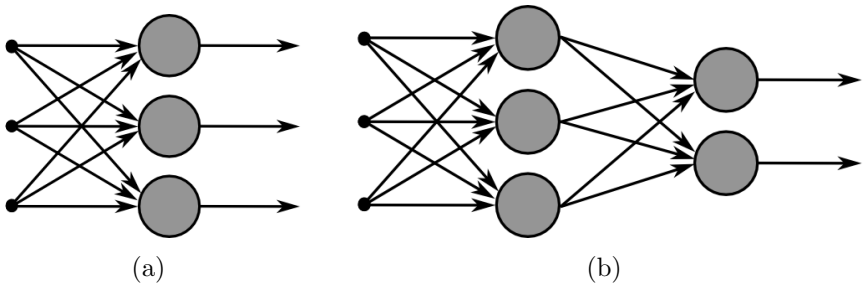

Figura 1: Exempos de redes neurais. (a) Single-layer perceptron. (b) Multi-layer perceptron, com uma camada oculta

O treinamento de uma rede é feito através da mudança no vetor de pesos associada à entrada de cada unidade. O algoritmo de treinamento mais conhecido para as redes do tipo *multi-layer perceptron* é o *backpropagation*. De acordo com a diferença entre a saída dada pela rede e a saída desejada, os pesos dos perceptrons são modificados a cada iteração do processo de treinamento de forma que a saída da rede convirja para a saída desejada. O nome *backpropagation* deriva do fato do ajuste dos pesos ocorrer da camada de saída para as camadas prévias.

A taxa de aprendizagem determina a intensidade dos ajustes a serem aplicados aos pesos de forma que a saída da rede possa convergir de forma mais ou menos rápida para o resultado desejado. Assim, com uma taxa de aprendizagem de 0,1 os pesos irão convergir dez vezes mais rápido para o valor desejado quando comparada com uma taxa de apredizagem de 0,01. Esta convergência dos pesos para um determinado valor pode ser vista como a busca por um mínimo global no espaço $\mathbb{R}^{m \times n}$ onde m e n são o comprimento das camadas de perceptrons que estão sendo interligadas pela camada de pesos. Desta forma, não é possível definir a taxa de aprendizagem como um valor muito alto pois, neste caso, seria impossível atingir o mínimo desejado com deslocamentos cujo comprimento tenha um valor elevado.

A saída da rede neural é composta por um vetor $\mathbf{x} \in [0, 1]^n$. Supondo-se que um símbolo a ser reconhecido pela rede neural pertença à classe i, deseja-se que a saída da rede seja igual ao vetor canônico e_i que contém o número 1 apenas na coordenada i e zeros nas demais $n - 1$ coordenadas. Entretanto, não é viável exigir que a saída da rede seja exatamante igual ao vetor e_i mas sim que ela esteja bem próxima deste vetor. A medida desta distância é chamada de limite de erro ou *error threshold*. Sendo e_i a saída

desejada , o limite de erro é calculado por:

$$\sum_{j=1}^{n}(e_{i_j} - x_j)^2$$

Depois de treinada adequadamente, uma rede neural pode ser utilizada como um classificador, ou seja, para identificar a qual classe pertence um dado objeto representado por uma tupla compatível com o vetor de entrada da rede.

Quando o número de classes é pequeno, as redes neurais geralmente têm uma taxa de acerto bastante elevada e possuem uma boa tolerância a erros que as torna bastante robustas. Entretanto, quando a quantidade de classes é grande, o desempenho tende a degradar.

2.3 Escrita *offline* e *online*

A escrita *off line* refere-se aos casos em que apenas o resultado da escrita, geralmente uma imagem gerada por um *scanner*, está disponível para processamento. A própria imagem de um símbolo, ou quaisquer características que possam ser extraídas da imagem, podem ser consideradas informações *offline*.

A escrita *online*, por outro lado, inclui informações temporais associadas ao processo de escrita. Isto significa que estão disponíveis, além dos traços que compõem um símbolo, também a informação temporal do momento em que foi escrito cada um dos pontos que compõem o traço.

Para coletar os dados da escrita *online*, cada símbolo é visto como um conjunto de um ou mais traços. Cada traço é composto pelo conjunto de pontos que descreve a trajetória da caneta desde o momento em que ela toca o papel até o momento em que ela perde contato com o papel. Por exemplo, o sinal de menos "-" é composto por apenas um traço e o sinal de igual "=" é composto por dois traços.

Durante a escrita do traço, com a utilização de um *tablet* é possível obter as coordenadas de cada um dos pontos pelos quais está passando a caneta. A cada um destes pontos é associada a informação que indica o momento em que ele foi escrito.

De posse destas informações, é possível dizer, por exemplo, se um símbolo parecido com um S maiúsculo foi escrito com um ou dois traços. Desta forma, o classificador poderia mais facilmente identificar se o símbolo em questão é o número 5 ou de fato a letra S.

2.4 Classificação de um grande conjunto de caracteres

Uma característica particular do reconhecimento de expressões matemáticas é a grande quantidade de possíveis símbolos a serem reconhecidos. Nesses casos, em geral, um único classificador não é capaz de fazer um reconhecimento satisfatório. Na literatura encontramos diferentes abordagens para tratar este problema [8]. A seguir consideramos três possíveis abordagens de utilização de redes neurais para esse tipo de classificação.

2.4.1 Uma única rede neural

Esta é a abordagem mais simples, na qual uma única rede neural é criada e o símbolo pode ser classificado apenas analisando-se a saída produzida pela rede. Entretanto, como foi citado anteriormente, este método não apresenta bons resultados. Em um dos testes realizados, uma rede neural treinada para reconhecer 62 símbolos apresentou uma taxa de acerto de cerca de 50%. Ao diminuirmos a quantidade de símbolos para 26, a taxa de acerto subiu para 90%.

2.4.2 Uma rede neural para cada símbolo

Como uma rede neural não apresenta desempenho satisfatório quando a quantidade de classes é muito grande, uma possível solução para este problema seria criar várias redes, cada uma delas especializada no reconhecimento de um determinado símbolo.

Pode-se implementar a rede neural para que a sua saída seja binária, com 0 indicando que o símbolo de entrada não é da classe de especialização daquela rede e 1 indicando o contrário. Esta solução gera um novo problema pois várias redes podem apresentar saída 1, indicando que o símbolo pertence a várias classes ou nenhuma rede pode apresentar saída 1, indicando que o símbolo não foi reconhecido.

Para contornar este problema pode-se implementar a rede de forma que sua saída seja um número real pertencente a um certo intervalo. Esse valor indicaria o grau de pertinência do símbolo de entrada à classe de especialização da rede (por exemplo, quanto maior o valor da saída, mais forte a indicação de que o símbolo de entrada é daquela classe). Porém, desta forma, torna-se necessário que uma função analise as saídas de todas as redes para poder produzir a saída do classificador.

Mesmo resolvido o problema de símbolos "sem classe" ou pertencentes a muitas classes, deve-se ainda considerar o fato de que a solução acima implica na criação de uma quantidade muito grande de redes e de submeter

um símbolo a ser classificado a várias redes durante o processo de reconhecimento.

Claramente esta solução subutiliza a capacidade de redes neurais, que embora não sejam apropriadas para reconhecer muitos símbolos, certamente podem ser treinados para reconhecer uma quantidade maior do que um único símbolo com uma boa taxa de acerto.

2.4.3 Abordagem proposta: decomposição hierárquica

Uma alternativa natural às duas descritas acima seria a utilização de um certo número não muito grande de redes, cada uma capaz de classificar elementos de um grupo pequeno, porém maior que unitário, de símbolos.

A abordagem proposta é subdividir sucessivamente o conjunto de símbolos utilizando como critério de divisão as características extraídas dos próprios símbolos. O processo de divisão deve ser repetido até que cada subgrupo resultante contenha um número de símbolos não superior a um dado limiar. Para cada subgrupo resultante treina-se uma rede neural. A classificação corresponde a um processo hierárquico no qual primeiramente localiza-se o subgrupo ao qual o símbolo a ser classificado pertence (de acordo com os critérios de divisão utilizados), e em seguida aplica-se a rede neural associada àquele subgrupo.

Nossa proposta é utilizar características das escritas *online* e *offline* para realizar as subdivisões. Algumas dessas características estão listadas a seguir [9]:

Características da escrita *offline*:

- **Densidade dos pontos:** baseado na maior ou menor concentração de pontos em determinadas regiões.
- **Formato do símbolo:** baseado na relação entre a largura e a altura do símbolo.

Características da escrita *online*

- **Número de traços**: baseado no número de vezes que a caneta tocou o papel durante a escrita do símbolo.
- **Número de *loops***: de acordo com o número de vezes em que um mesmo traço se intersecta.
- **Número de mudanças bruscas de direção**: o número três escrito pela maioria das pessoas teria uma mudança brusca de direção, já o número zero não teria nenhuma.

- **Posição dos pontos principais:** de acordo com a posição em que são escritos o primeiro e o último traço do símbolo.

- **Relação entre os pontos principais:** de acordo com a direção, em linha reta, a partir do primeiro ponto em direção ao último ponto.

Com todas estas características disponíveis, pode-se escolher mais de uma delas para o processo de divisão. Um conjunto de símbolos seria inicialmente dividido em conjuntos menores de acordo com uma das características e em seguida cada um desses subconjuntos seriam também divididos de acordo com outra característica e assim sucessivamente.

Este processo pode ser repetido enquanto existirem características ainda não utilizadas no processo de divisão. A cada passo, podemos considerar que estamos descendo um nível em uma árvore de decisão. Por isso, a abordagem proposta é chamada de hierárquica.

3 Atividades realizadas

Além do estudo de artigos sobre reconhecimento de expressões matemáticas [1, 7], classificação [5] e redes neurais [4, 6, 13], foram realizadas várias outras atividades, que estão descritas a seguir.

3.1 Implementação da Rede Neural

Com base nos conhecimentos adquiridos na fase anterior, iniciou-se a implementação de uma rede neural, mais precisamente nos moldes descritos por Bishop [6].

A rede neural implementada consiste de um *multi-layer perceptron* contendo duas camadas de pesos. Para facilitar o processamento pela rede neural, a matriz que representa a imagem de um símbolo é transformada em um vetor de bits. A entrada desta rede é composta por um vetor de bits de tamanho 400 que foi obtido a partir de uma matriz 20×20. O número de classes que está contido no conjunto de dados utilizado determina a quantidade de *perceptrons* na camada de saída. Para determinar a quantidade adequada de *perceptrons* na camada intermediária foram realizados os testes abaixo.

3.1.1 Testes na Rede Neural Implementada

Inicialmente foram realizados testes para definição da taxa de aprendizagem e do limite de erro que seriam mais adequados à rede neural. As tabelas 1 e 2 sumarizam os resultados.

taxa de aprendizagem	taxa de acerto(%)
0,1	89,6
0,01	91,3
0,001	91,6
0,0001	91,6

Tabela 1: Relação entre a taxa de aprendizagem e a taxa de acerto

erro (10^{-e})	taxa de acerto(%)
1	89,6
2	88,3
3	88
4	87
5	89,3
7	90
10	89,6
15	89,3

Tabela 2: Relação entre a taxa de aprendizagem e a taxa de acerto

A tabela 1 mostra que mesmo com taxas de aprendizagem muito pequenas a taxa de acerto, após atingir um determinado limite, não melhora.

A tabela 2 mostra que mesmo com um limite de erro da ordem de 10^{-15} não é possível obter uma melhora significativa na taxa de acerto.

A partir dos dados obtidos nestes testes foi possível definir dois dos parâmetros a serem utilizados desde então. A taxa de aprendizagem foi definida como 10^{-2} e o limite de erro como 10^{-1}.

3.1.2 Adaptação dos Dados à Rede

Os dados utilizados nos testes acima foram retirados da base de dados MNIST [12] e de duas bases de dados diponíveis no *Machine Learning Repository* [11] chamadas *Artificial Characters* e *UJI Pen Characters*. Cada um destes conjuntos de dados é fornecido em um formato diferente, portanto foi necessário implementar ou modificar algoritmos já existentes para que os dados pudessem ser fornecidos como entrada à rede neural.

Dos três conjuntos de dados citados acima, o último a ser utilizado foi o *UJI Pen Characters* que é disponibilizado em um formato conhecido como UNIPEN [10]. O fato deste formato ser utilizado em diversos projetos de reconhecimento de caracteres manuscritos fez com que a coleta de dados

descrita abaixo também seguisse este padrão.

3.2 Coleta de Dados

Depois de testar a rede com dados obtidos através da escrita *offline* como foi descrito acima, era necessário a utilização de dados da escrita *online*.

Uma vez que não encontramos uma base de dados de símbolos matemáticos de domínio público, realizou-se uma coleta de dados para montar uma base de símbolos matemáticos. Os dados dessa base foram coletados de dez voluntários, com a utilização de um *tablet*. A cada voluntário foi solicitado que escrevesse duas vezes um conjunto composto por 25 letras gregas e 52 símbolos que aparecem com freqüência em expressões matemáticas, totalizando 77 símbolos.

3.2.1 Modificações no Código do Math-Picasso

O software utilizado para a coleta de dados descrita acima foi obtido a partir de uma modificação do Math-Picasso [3]. Na sua estrutura interna o Math-Picasso já coleta as informações que eram desejadas. Foram realizadas modificações para a geração de um arquivo de saída contendo os dados escritos por cada usuário. Além disso foram feitas pequenas modificações na interface para tornar o uso mais amigável ao usuário uma vez que neste caso a aplicação estava sendo utilizada para um fim diferente daquele para o qual foi planejada.

3.3 Implementação de Algoritmos de Distorção

Para o treinamento adequado da rede neural é necessário que esteja disponível uma grande quantidade de exemplares de um mesmo símbolo escritos pelo mesmo usuário. Uma vez que uma coleta de dados deste tipo seria inviável dentro do escopo deste trabalho, foram desenvolvidos algoritmos que produzem pequenas distorções em um símbolo, permitindo gerar até duzentas variações deste mesmo símbolo que contenham características muito parecidas.

Abaixo temos uma explicação sobre cada um dos tipos de distorção implementados.

Troca de bits na borda da escrita O objetivo deste tipo de distorção é simular a caneta tremendo durante a escrita, algo que pode ocorrer com freqüência durante o uso do sistema. Nos dados obtidos na coleta descrita em 3.2 cada símbolo é representado como uma lista de pontos.

Para realizar a distorção inicialmente é escolhido um número aleatório n de pontos que sofrerão a distorção. Em seguida, são sortedados n pontos p_i, $i \in 1, \ldots, n$. Para cada ponto p_i é removida uma vizinhança sua da lista de pontos. Em seguida é inserido um novo ponto p'_i em uma posição próxima de p_i escolhida ao acaso.

A imagem em baixa resolução resultante deste processo é semelhante à imagem original porém com alguns bits trocados na borda do traço.

Compressão vertical Neste tipo de distorção, cada símbolo é comprimido verticalmente com intensidades variando dentro de determinados limites. Apesar da distorção alterar sensivelmente a imagem, ela não deixa de ser legível para seres humanos.

Compressão horizontal Similar à compressão vertical, mas realizada no eixo horizontal.

Rotação do símbolo Modificaçõs na inclinação da escrita são comuns entre diferentes indivíduos, o objetivo desta distorção é simular este tipo de variação, de forma que o sistema esteja preparado para aceitar símbolos escritos por um número maior de pessoas.

3.3.1 Testes comparando os tipos de distorção

Após a implementação descrita acima, foram realizados alguns testes para comparar os tipos de distorção em relação à taxa de reconhecimento proporcionada. A tabela 3 resume o resultado obtido em um teste no qual um mesmo símbolo sofreu dez distorções de cada tipo. Com base no resultado

Tipo de distorção	taxa de acerto(%)
Troca de bits	81,0
Compressão vertical	63,0
Compressão horizontal	61,5
Rotação	72,6

Tabela 3: Comparação entre os tipos de distorção

obtido neste e em outros testes, foi tomada a decisão de abandonar todos os tipos de distorção, exceto a troca de bits que permanece na versão final do código.

3.4 Implementação da árvore de reconhecimento

Para que as características extraídas dos símbolos pudessem ser utilizadas no seu reconhecimento, foi necessário a realização de modificações na estrutura do *software* desenvolvido. Foram desenvolvidos algoritmos para extração das características descritas em 2.4.3 exceto para a detecção do número de *loops*. A partir do momento em que as características do símbolo estavam disponíveis, foi necessário montar uma estrutura que pudesse utilizar estas características para obter uma melhora na taxa de reconhecimento. A estrutura escolhida foi uma árvore conforme ilustrado na figura 2.

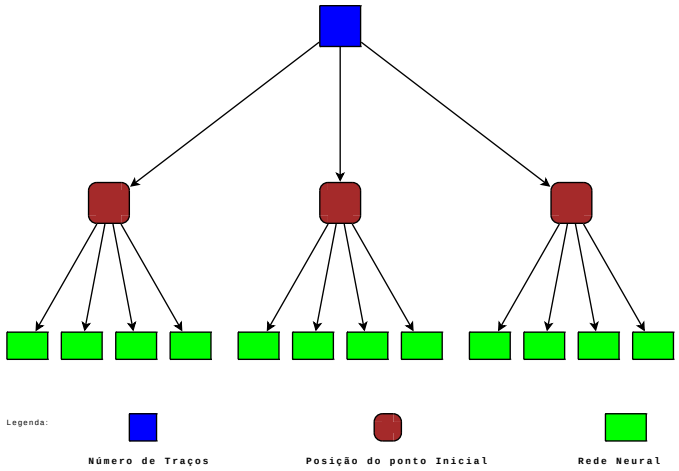

Figura 2: Árvore de reconhecimento.

Nesta estrutura, todos os nós internos que estão no mesmo nível analizam a mesma característica. No exemplo, os nós de cor marrom analizam o símbolo a ser reconhecido para determinar a posição do ponto inicial. De acordo com o resultado este símbolo segue por um dos possíveis caminhos no nível abaixo. As folhas desta árvore estão todas no mesmo nível e cada uma contém uma rede neural. Devido às características analizadas em níveis superiores, uma determinada rede neural irá sempre reconhecer símbolos com as mesmas características.

3.4.1 Testes Realizados com a árvore de reconhecimento

Após a implementação da árvore de reconhecimento, foram realizados testes nos quais apenas uma característica era analisada. Ou seja, a árvore possuía apenas um nó interno que analisava a característica em questão e enviava o símbolo para reconhecimento em uma das redes neurais localizadas nas folhas. Este mesmo tipo de teste foi repetido várias vezes, com mudanças

no número de distorções geradas para avaliar se a geração de distorções realmente melhora a taxa de reconhecimento.

A tabela 4 mostra o resultado dos testes realizados sem a geração de distorções.

Característica analisada	taxa de acerto(%)
Número de traços	70
Posição do primeiro ponto	65
Posição do último ponto	64
Direção do primeiro ao último ponto	46
Relação altura/largura	54
Região de maior densidade	57

Tabela 4: Comparação entre as diferentes características sem a geração de distorções.

A tabela 5 mostra o resultado dos testes nos quais cada símbolo sofria dez distorções.

Característica analisada	taxa de acerto(%)
Número de traços	71,5
Posição do primeiro ponto	71,5
Posição do último ponto	64,5
Direção do primeiro ao último ponto	64,0
Relação altura/largura	63,0
Região de maior densidade	63,0

Tabela 5: Comparação entre as diferentes características com a geração de dez distorções.

Uma vez que a geração de um número maior de distorções proporcionou uma melhora na taxa de reconhecimento, o passo seguinte foi a realização de testes com um número maior de distorções.

A tabela 6 mostra o resultado do teste realizado com a geração de trinta distorções.

Apesar destes resultados confirmarem que a geração de distorções a partir dos símbolos proporciona uma melhora na taxa de reconhecimento, existe um determinado ponto a partir do qual a geração de distorções é incapaz de fornecer resultados melhores, cheganto até a diminuir a taxa de reconhecimento. Nos testes realizados, a partir de trinta distorções já não é possível obter melhoras.

Característica analisada	taxa de acerto(%)
Número de traços	77,0
Posição do primeiro ponto	75,0
Posição do último ponto	70,0
Direção do primeiro ao último ponto	65,0
Relação altura/largura	65,5
Região de maior densidade	67,5

Tabela 6: Comparação entre as diferentes características com a geração de trinta distorções.

3.5 Inconsistências nas Características

Mesmo os melhores resultados obtidos em 6 não chegaram a atingir os níveis desejados. Isso levou a um questionamento sobre a consistência das características utilizadas. Como cada usuário escreveu duas vezes o mesmo símbolo, é desejável que estas apresentem as mesmas características. Então foram realizados testes para verificar quantos símbolos são inconsistentes de acordo com cada característica. A tabela 7 mostra os resultados de um teste

Característica analisada	inconsistências(%)
Número de traços	3,2
Posição do primeiro ponto	13,9
Posição do último ponto	13,6
Direção do primeiro ao último ponto	25,6
Relação altura/largura	19,4
Região de maior densidade	17,9

Tabela 7: Comparação entre as diferentes características para a verificação de inconsistências.

realizado com todos os 770 pares de símbolos escritos pelos dez voluntários. O pequeno número de inconsistências em relação ao número de traços explica o melhor desempenho desta característica nas tabelas 4 e 5.

4 Resultados e produtos obtidos

O objetivo final deste trabalho é comparar a abordagem proposta em 2.4.3 com outras possíveis abordagens. Para realizar esta comparação foram feitos testes comparando a abordagem hierárquica e a abordagem descrita em 2.4.1

na qual uma única rede neural é responsável pelo reconhecimento de todos os símbolos. O gráfico 3 mostra o resultado dos testes realizados. Nele é possível observar que, em todos os casos, a abordagem hierárquica obteve um desempenho superior à abordagem que utiliza apenas uma rede neural.

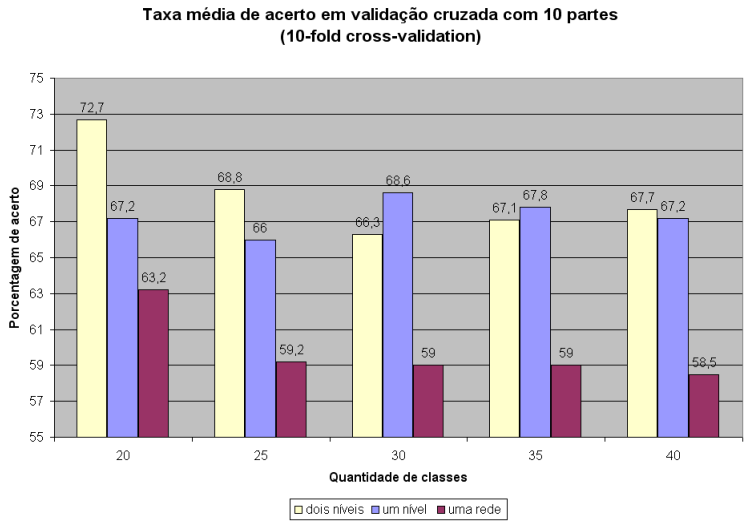

Figura 3: Comparação entre a abordagem hierárquica e a abordagem de rede neural única.

4.1 Software

Para a realização de todos os testes descritos anteriormente, foi desenvolvido um *software* que implementa uma rede neural artificial chamada *Multi-layer Perceptron* que é treinada através do algoritmo *backpropagation*. A estrutura deste programa é composta por uma árvore de decisão ou árvore hierárquica que classifica os símbolos a serem reconhecidos de acordo com características extraídas do próprio símbolo.

O código desenvolvido está sendo entregue juntamente com esta monografia. Trata-se de um projeto desenvolvido no Eclipse e que foi testado para funcionar tanto no sistema operacional *Windows* quanto no *Linux*. Abaixo fornecemos alguns detalhes sobre os pacotes da implementação.

4.1.1 O pacote `src.core`

Este pacote contém a principal classe do programa que é o `MainController`, classe responsável pela execução do *cross-validation* que gerou todos os dados obtidos nos testes. Os principais parâmetros do teste são definidos através

das constantes `TOTAL_CLASSES`, `TOTAL_FOLDS`, e `TOTAL_DISTORTIONS` que definem, respectivamente, o número total de classes a serem reconhecidas pelo sistema, o número total de repetições do mesmo teste para gerar uma média ao final e o número total de distorções a serem executadas sobre cada símbolo.

A classe `ClassifierController` é responsável pela execução cada uma das passagens do *cross-validation*. Inicialmente a árvore de reconhecimento é treinada com todos os símbolos presentes no *array list* `trainingSymbols`. Posteriormente os símbolos presentes no *array list* `recognizingSymbols` são reconhecidos pela árvore já treinada. Durante este processo são medidos os tempos de treinamento e de reconhecimento. Ao término é impressa uma mensagem informando a duração de cada uma destas etapas.

A classe `InputFileReader` é responsável pela leitura dos arquivos de entrada obtidos através da coleta descrita em 3.2. O formato do arquivo de entrada é similar ao formato UNIPEN [10]. Ao término da leitura de cada símbolo, as características deste símbolo são analizadas e ficam armazenadas em uma estrutura de dados chamada `PointList`. Estas listas de pontos armazenam, além das coordenadas de cada ponto que contém o símbolo, também o conjunto de características daquele ponto. Além disso são responsáveis pela análise destas características logo após a leitura do símbolo.

4.1.2 O pacote `src.distort`

Este pacote inicialmente continha todas as distorções descritas em 3.3, entretanto, pelos motivos descritos em 3.3.1 permaneceu no código final apenas a distorção que troca os bits na borda dos traços. A classe `Normalizer` é utilizada para corrigir variações na posição e no tamanho dos símbolos que possam ter ocorrido no momento da escrita. Após o processamento por esta classe, todos os símbolos estão centralizados em uma matriz de tamanho 500 x 500 e ocupando boa parte da matriz. Para que isto ocorra é utilizado um método pertencente à classe `Stretch` que aumenta as dimensões do símbolo conforme necessário. A classe `Shaker` é responsável por realizar a distorção que simula a caneta tremendo durante a escrita. A forma como este resultado é obtido foi descrita em 3.3.

4.1.3 O pacote `src.hashFunctions`

Cada uma das classes deste pacote é responsável por analisar uma determinada característica do símbolo que está sendo reconhecido. Para evitar análises redundantes este processo é feito previamente durante a leitura do arquivo de entrada e o valor é armazenado na estrutura que contém os dados do símbolo chamada de `PatternSymbol`. Sendo assim, estas classes apenas

lêem e retornam o valor armazenado na estrutura de dados.

4.1.4 O pacote `src.mlp`

Neste pacote estão localizadas as classes que irão compor as redes neurais da árvore de reconhecimento. A classe `Network` representa a rede neural em si, por isso armazena as camadas de *perceptrons* e as camadas de pesos. Durante o treinamento para um determinado símbolo, os pesos das duas camadas de pesos são ajustados até que a diferença entre a saída da rede e a saída desejada seja menor do que o *error threshold*.

Para reconhecer um símbolo a rede o processa apenas uma vez e retorna a classe da camada de saída que forneceu o valor mais alto.

A classe `Perceptron` simula o funcionamento de um neurônio biológico. Para produzir a sua saída, são combinados as entradas da camada anterior com pesos definidos pela camada de pesos. O resultado é submetido à função de ativação.

O `PerceptronLayer` contém um vetor de *perceptrons* que corresponde a uma das camadas da rede neural. Por fim, o `WeightLayer` contém uma matriz de números reais que correspondem aos pesos interligando *perceptrons* de duas camadas adjacentes.

4.1.5 O pacote `src.mlp.activationFunctions`

A classe `LinearFunction` contém a função de ativação utilizada pelos perceptrons da camada de saída. Esta função retorna o parâmetro de entrada.

A função de ativação contida na classe `SigmoidFunction` comprime a reta real no intervalo [-1,1] e é utilizada pelos perceptrons da camada oculta.

4.1.6 O pacote `src.mlp.statistics`

Neste pacote estão as classes que registram os dados sobre o reconhecimento dos símbolos que tornam possível saber qual foi a taxa de acerto de um determinado testes.

A classe `ConfusionMatrix` armazena informações detalhadas sobre o reconhecimento de cada um dos símbolos. Isso torna possível a criação de um mapa chamado Matriz de Confusão que mostra quantos e quais foram os erros durante o processo de reconhecimento. Os dados armazenados na matriz de confusão são obtidos pela classe `Statistics` que os resume em um número que revela a taxa de acerto de um determinado teste.

4.1.7 O pacote `src.treeController`

As classes contidas neste pacote são responsáveis pela estrutura da árvore de reconhecimento que é utilizada na abordagem hierárquica.

A classe `InternalNode` é responsável por analisar um determinado símbolo de acordo com a característica contida na função de hash daquele nó. A classe `Leaf` representa uma das folhas da árvore e contém uma rede neural onde o símbolo é, de fato, processado.

5 Conclusão

Neste documento descrevemos um trabalho de formatura supervisionado realizado conforme as exigências da disciplina MAC0499. A proposta deste trabalho é a utilização de uma abordagem hierárquica para o problema de classificação de símbolos em expressões matemáticas.

Nesta abordagem, os símbolos a serem classificados são divididos sucessivamente em subgrupos de acordo com alguma característica facilmente computável, até que cada um dos grupos possua um número de caracteres não superior a um dado limiar.
Em seguida, para cada grupo é treinada uma rede neural para reconhecer os caracteres naquele grupo.

Implementamos e testamos redes neurais do tipo *multilayer perceptrons* com diferentes conjuntos de dados de caracteres e dígitos manuscritos. Esses testes mostraram resultados já conhecidos na literatura da área sobre o desempenho de redes neurais para classificação de dados: o desempenho das redes degrada quando o número de classes torna-se muito grande. Com isso foi possível comprovar que a abordagem hierárquica apresenta um desempenho superior em relação à abordagem na qual uma única rede neural é utilizada para o reconhecimento de todo o conjunto de símbolos

Referências

[1] D. Blostein and A. Grbavec, *Recognition of Mathematical Notation* (H. Bunke and P. Wang, ed.), 1997, Handbook of Character Recognition and Document Image Analysis, pp. 557-582.

[2] Nina S. T Hirata, *ExpressMath - Reconhecimento de expressões matemáticas*, Universidade de São Paulo, Departamento de Ciência da Computação, 2008. `http://www.vision.ime.usp.br/~nina/projetos/expressmath/`.

[3] Ana Paula Santos de Mello and Eduardo Yutaca Komatsu and Fábio Marcos Eiji Okuda and Leonardo Ka Wah Hing, *Math Picasso — Segmentação e reconhecimento de caracteres em expressões matemáticas manuscritas*, 2007. `http://www.linux.ime.usp.br/~eiji/mac499/index.php`.

[4] Tom M. Mitchell, *Machine Learning*, McGraw-Hill, 1997.

[5] Richard O. Duda and Peter E. Hart and David G. Stork, *Pattern Classification*, Wiley-Interscience, 2000.

[6] Christopher M. Bishop, *Neural Networks for Pattern Recognition*, Oxford Universty Press Inc., New York, 1995.

[7] Kam-Fai Chan and Dit-Yan Yeung, *Mathematical Expression Recognition: A Survey*, International Journal on Document Analysis and Recognition **3** (2000), 3-15.

[8] Guobin Ou and Yi Lu Murphey, *Multi-class pattern classification using neural networks*, Pattern Recognition **40** (2007), 4-18.

[9] Stephen M. Watt and Xiaofang Xie, *Recognition for Large Sets of Handwritten Mathematical Symbols*, 2005, Eighth International Conference on Document Analysis and Recognition (ICDAR 2005), pp. 740-744.

[10] I. and Schomaker Guyon L. and Plamondon, *UNIPEN project of on-line data exchange and recognizer benchmarks*, Proceedings of the 12th International Conference on Pattern Recognition (ICPR 94) (1994), 29-33.

[11] A. Asuncion and D.J. Newman, *UCI Machine Learning Repository*, University of California, Irvine, School of Information and Computer Sciences, 2007. `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

[12] Yann LeCun and Corinna Cortes, *The MNIST database of handwritten digits*. `http://yann.lecun.com/exdb/mnist/`.

[13] Richard P. Lippmann, *An Introduction to Computing with Neural Nets*, Signal Processing Magazine, IEEE (1987), 4-22.

6 Parte subjetiva

6.1 Desafios e frustrações

No início deste ano eu pretendia fazer o trabalho de formatura mas não tinha idéia de nenhum tema e também não tinha conversado com nenhum orientador. Em janeiro recebi um e-mail que a Nina havia mandado para os alunos do BCC para a divulgação de uma pesquisa na qual havia vaga para um aluno de Iniciação Científica. Analisei a página do projeto ExpressMath e achei bastante interessante. Quando comecei a desenvolver a rede neural, tive um pouco de receio de não conseguir compreender o código do projeto Math-Picasso ou acabar estragando um programa que me parecia ter sido muito bem feito. Depoi de um tempo percebi que poderia desenvolver uma rede neural em um projeto independente, conhecendo todo o seu conteúdo e sem correr o risco de danificar o que não me pertence.

Ao terminar a implementação da rede neural tive um certo contratempo ao tentar fazer com que ela funcionasse corretamente. A idéia de um sistema que aprende a reconhecer objetos é bastante interessante, mas no princípio foi difícil fazer com que os pesos convergissem para os valores corretos. Talvez essa dificuldade tenha aumentado o sentimento de realização quando descobri o erro em um sinal trocado e tudo finalmente funcionou.

Outra experiência frustrante ocorreu quando implementei os algoritmos de distorções. Foram quatro algoritmos diferentes que consumiram as férias de julho para ficarem prontos e que no final dos testes se mostraram pouco eficientes, sendo que apenas um deles foi mantido na versão final do código.

6.2 Disciplinas relevantes

MAC110 - Introdução à Computação Ao ingressar no IME eu não tinha experiência em programação, então esta disciplina foi importante para mim pois nela eu aprendi como dar ordens a um computador.

MAC122 - Princípios de Desenvolvimento de Algoritmos No segundo semestre tivemos contato com a linguagem **C** e com alguns problemas um pouco mais complicados, o que me fez perceber que escrever um algoritmo é, essencialmente, encontrar uma forma de resolver um problema.

MAC323 - Estruturas de dados e MAC338 Análise de Algoritmos A meu ver a disciplina Análise de Algoritmos foi um aprofundamento do que foi aprendido em Estruturas de Dados. Ambas foram importantes por proporcionar uma maior agilidade em programação devido

aos Exercícios-Programa e por ensinar e analisar a importância de diferentes formas de armazenamento e manipulação de dados.

MAC211 - Laboratório de Programação I Nesta disciplina tivemos a oportunidade de desenvolver um projeto com a duração de um semestre, o que foi uma experiência nova em comparação com os EPs das disciplinas anteriores que tinham um escopo menor.

MAC332 - Engenharia de Software Foi muito importante pois executamos um projeto em grupo durante um semestre, além de aprender os fundamentos para o desenvolvimento de código de qualidade em grandes projetos.

6.3 Aplicações de conceitos

Dentre as disciplinas cursadas, não houve nenhuma que tivesse aplicação direta no trabalho de formatura. Entretanto os conceitos aprendidos em quase todas elas tiveram aplicação pois para a realização do projeto foi necessário o conhecimento de algoritmos e a experiência de programação adquirida no decorrer destes quatro anos. Mesmo disciplinas do Departamento de Matemática como Cálculo II e III ajudaram a compreender porque o processo de ajuste dos pesos que ocorre durante o treinamento da rede neural fazem com que a saída da rede convirja para a saída desejada.

6.4 Aprimoração dos conhecimentos

Estou terminando o trabalho de formatura este ano mas ainda não vou me formar devido a matérias optativas livres ainda não cursadas. Pretendo adquirir maiores conhecimentos na área cursando disciplinas do Departamento de Visão Computacional.

6.5 Agradecimentos

C. P. Garcia é bolsista ITI do Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq. N. S. T. Hirata é bolsista Produtividade em Pesquisa do CNPq. Este trabalho está sendo desenvolvido no âmbito de projetos apoiados pelo CNPq.

É necessário agradecer à orientadora Nina S. T. Hirata pela sua paciência e disposição em dirimir todas as minhas dúvidas, assim como guiar-me para que o trabalho pudesse prosseguir na direção certa.

Aos colegas Breno F. Franco, Caue H. P. Guerra, Cecilia Fernandes, João P. K. Catunda, Luiz F. O. C. Real,Peter N. Nyumu, Rafael C. S. Shouery,

Rodrigo M. Flores e Stephanie Blum também agradeço por terem fornecido parte do seu tempo para me ajudar a coletar os dados que foram utilizados nos testes.

Por fim, mas não menos importante, gostaria de agradecer à minha esposa Maisa por compreender as minhas dificuldades, ajudar no que estava ao seu alcance e por me apoiar sempre.