



IME - Instituto de
Matemática e Estatística

Universidade de São Paulo
Instituto de Matemática e Estatística
Departamento de Ciência da Computação

MAC 499 Trabalho de Formatura Supervisionado

Math-Picasso

Ana Paula Santos de Mello
Eduardo Yutaca Komatsu
Fábio Marcos Eiji Okuda
Leonardo Ka Wah Hing

Universidade de São Paulo
Instituto de Matemática e Estatística
Departamento de Ciência da Computação

Ana Paula Santos de Mello
Eduardo Yutaca Komatsu
Fábio Marcos Eiji Okuda
Leonardo Ka Wah Hing

Math-Picasso

Segmentação e Reconhecimento de Caracteres
e Expressões Matemáticas Manuscritas

Orientadora: *profª. drª. Nina S. T. Hirata*

São Paulo
3 de dezembro de 2007

Agradecimentos

Agradecemos a profª. drª. Nina S. T. Hirata pela sua paciência e dedicação durante o desenvolvimento deste projeto, e também aos alunos Ricky, Eduardo e Pedro Henrique, por terem cedido o código desenvolvido por eles no ano passado.

Resumo

A escrita é uma maneira natural de se expressar idéias e o uso de computadores torna-se uma ferramenta essencial, uma vez que permitem lidar com grande quantidade de dados de forma prática. Isso é verdade quando a tarefa envolve apenas a criação de textos que utilizem alfabetos ocidentais. Porém, quando se trata de caracteres especiais, como ideogramas ou mesmo símbolos matemáticos, surgem alguns problemas.

Este trabalho está relacionado ao desenvolvimento de um sistema para reconhecimento de expressões matemáticas manuscritas, que geralmente consiste de duas etapas principais:

- a segmentação e reconhecimento dos caracteres e símbolos
- a análise estrutural da expressão

O projeto Math-Picasso foi concebido tendo em vista o desenvolvimento de um sistema computacional para reconhecimento de expressões matemáticas. Por enquanto, suas funcionalidades estão restritas à primeira etapa (segmentação e reconhecimento), porém sua estrutura foi planejada de forma a permitir futuras melhorias e expansões.

Palavras-chave: Redes neurais, neocognitron, reconhecimento de escrita online, segmentação de caracteres, reconhecimento de caracteres

Sumário

I Monografia **1**

1 Introdução **2**

2 O reconhecimento de escrita **4**

2.1 Segmentação 5

2.2 Reconhecimento 6

2.2.1 Redes Neurais 7

2.2.1.1 MLP (*Multi-Layer Perceptron*) 9

2.2.1.2 Neocognitron 11

3 Atividades realizadas **14**

3.1 Captação de requisitos do *software* 14

3.2 Estudo de redes neurais 14

3.3 Estudo de técnicas de segmentação 15

3.4 Projeto do *software* 15

4 O projeto **16**

4.1 Projeto do sistema computacional 16

4.2 Sistema do ponto de vista estrutural 16

4.2.1 Camada de negócio 17

4.2.2 Camada de visualização 17

4.2.3 Camada de controle 18

4.3 Implementações 18

5 Resultados e conclusão **20**

5.1 Resultados 20

5.2 Conclusões 20

6 Apêndice **22**

6.1 *Math-Picasso* do ponto de vista operacional 22

6.1.1 Área de treinamento 22

6.1.2 Área de reconhecimento 23

II Parte subjetiva **25**

7 Parte Subjetiva **26**

PARTE I

Monografia

CAPÍTULO 1

Introdução

O aumento do poder de processamento dos computadores permite tratar grande volume de dados com velocidades muito superiores às que o ser humano conseguiria.

Um problema que surge é como inserir os dados que serão processados. Quando se trata apenas de texto, considerando-se alfabetos ocidentais, tal tarefa é simples, utilizando-se de teclados. Porém, quando se trata de caracteres especiais, como ideogramas orientais ou símbolos matemáticos, a questão é mais complexa. Uma possível solução seria proporcionar uma maneira mais confortável para o usuário inserir os dados no computador, por exemplo, por meio da escrita. Para tanto, pode-se usar o mouse, por estar presente em todos os computadores pessoais e até mesmo em laptops, ou utilizar tecnologias como *tablets* ou painéis eletrônicos.

Surge então a necessidade de criar um sistema capaz de reconhecer a escrita à mão livre. Essa tarefa, no entanto, não é trivial, dado que mesmo uma pessoa, ao ler um texto manuscrito, pode cometer erros de reconhecimento devido a tremores na escrita ou mesmo inclinações que fujam dos padrões para determinado caractere.

As aplicações para tal sistema são diversas, como utilização em salas de aula, ferramentas para sistemas de ensino à distância [3], sistemas de leitura para auxiliar deficientes auditivos, *software* auxiliar para produção de texto científico (que exija inserção de fórmulas) ou mesmo *games* e *softwares* para auxílio no aprendizado de idiomas orientais (como dicionários ou treinadores de *kanji*).

Este projeto teve como inspiração o sistema apresentado no trabalho de formatura no ano de 2006 apresentado pelos alunos Ricky Ye Lun Chow, Pedro Henrique Simões de Oliveira e Eduardo Gusmão Caceres Pires, entitulado SisTREO [24]. O objetivo do SisTREO foi construir um *framework* que pudesse segmentar e reconhecer símbolos escritos à mão e então reconhecer expressões matemáticas possibilitando a geração de código LaTeX.

Decidimos, junto com nossa orientadora, a professora e doutora Nina Hirata, construir um novo *software*. Porém, desta vez, o escopo seria outro, visando às partes de segmentação e reconhecimento de símbolos de maneira a tornar o sistema mais robusto, além de criar uma

ferramenta com interface amigável ao usuário. Além disso, pretende-se que o sistema seja bastante interativo, permitindo assim aumentar a capacidade de reconhecimento, utilizando-se das informações passadas pelo usuário. Como a intenção é de que o projeto seja extendido, outra grande preocupação foi com relação à modularização do software, facilitando adição de novas funcionalidades ou mesmo implementação de algoritmos diferentes dos utilizados nesta etapa.

Este documento está organizado da seguinte forma. No *capítulo 2* é descrito o problema de reconhecimento *online* de escrita, em particular os problemas de segmentação e reconhecimento de caracteres e expressões matemáticas manuscritas, bem como algumas das principais técnicas para resolvê-los. No *capítulo 3* são descritas as atividades realizadas ao longo do desenvolvimento deste projeto. No *capítulo 4* é descrito o *software* desenvolvido, ao qual denominamos *Math-Picasso*. Por último, no *capítulo 5*, são descritos os resultados e as conclusões. O apêndice apresenta o *software* do ponto de vista operacional.

CAPÍTULO 2

O reconhecimento de escrita

Ao tratar o problema do reconhecimento de escrita, fala-se em reconhecimento *offline* (apenas as coordenadas de cada ponto são conhecidas) e reconhecimento *online* (onde, além das coordenadas, também existe a informação do instante de tempo em que cada ponto foi originado).

Quando se pretende reconhecer uma expressão escrita como, por exemplo, uma expressão matemática, divide-se o problema em duas etapas:

- segmentação e reconhecimento de símbolos
- reconhecimento da expressão (ou seja, o reconhecimento da estrutura, com as devidas relações entre os símbolos)

Geralmente essas etapas são realizadas de maneira independente, porém existem abordagens em que elas são executadas simultaneamente a fim de obter informações que tornem o reconhecimento mais preciso [22]. Neste texto não será tratado o problema de reconhecimento da expressão, mas sim a segmentação e reconhecimento de símbolos matemáticos (aqui estão inclusos também caracteres dos alfabetos grego e romano, bem como os numerais).

Para a descrição dos problemas de segmentação e reconhecimento de caracteres, apresentado nas próximas subseções, convém considerar as seguintes definições:

- *Stroke Point*: estrutura básica. Um ponto que tem coordenadas (*x*,*y*) e o tempo (no caso de escrita *online*)
- *Stroke* (traço): composto por um conjunto de Stroke Points
- *Symbol* (símbolo): um conjunto de *Strokes*
- *Bounding box*: menor retângulo que envolve um objeto. No nosso caso, um símbolo
- *Character*: um caractere

2.1 Segmentação

A segmentação consiste em, dado um conjunto de traços, descobrir a qual símbolo cada traço pertence. Algumas técnicas de segmentação são listadas a seguir:

- intersecção de *Strokes* [10]
  Ao observar traços que se cruzam ou se tocam, é razoável considerar que façam parte do mesmo símbolo, apesar de que, ao se falar de letra cursiva, isto não seja uma regra. Por exemplo, símbolos como i, j e $=$, por não possuírem traços que se interceptam (figuras 2.1(a), 2.1(b)), fogem a essa regra e teriam que ser tratados como casos especiais nessa abordagem (da mesma forma, para cada símbolo sem traços conexos, seria necessário criar uma nova regra).



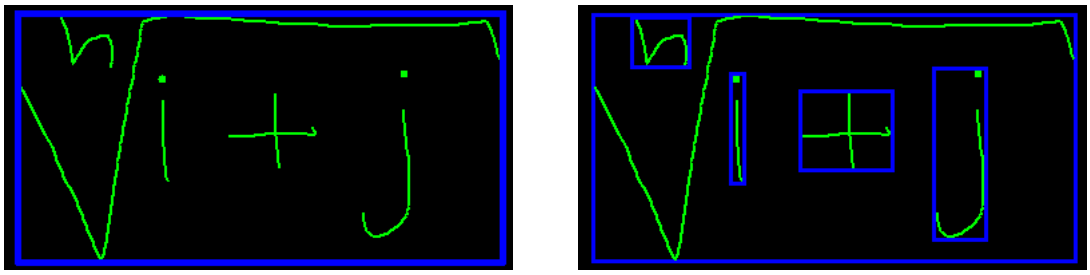
(a) Segmentação correta (b) Segmentação incorreta

Figura 2.1 Segmentação usando traços conexos

- segmentação por *bounding box* [10]
  Essa abordagem segmenta símbolos através da delimitação por *bounding box*, ou seja, tudo que estiver dentro da área delimitada por um *bounding box* é um símbolo. Porém, essa segmentação é um tanto imprecisa, dado que a área ocupada por um símbolo pode englobar outros símbolos. Como exemplo disso, podemos considerar a expressão $\sqrt{x+2}$, onde o símbolo $\sqrt{}$ contém os símbolos x, $+$ e 2 dentro de seu *bounding box*, como visto na figura 2.2(a). No entanto, por se tratarem de símbolos diferentes, cada um deve ter seu próprio *bounding box* (figura 2.2(b)).

- tempo entre *Strokes* [26]
  Uma outra forma de segmentar símbolos é usar o tempo como um parâmetro de distinção entre os símbolos. Quando um usuário de telefone celular deseja digitar uma palavra que contenha duas letras representadas por uma mesma tecla, este deve esperar um pequeno intervalo de tempo. Da mesma forma, o usuário do *Math-Picasso* também poderá ser

educado a esperar um certo tempo entre a escrita de um símbolo e outro. Com essa abordagem, os caracteres i, j, $=$ e $\sqrt{}$, por exemplo, não correm o risco de serem segmentados erroneamente (e, em conseqüência disso, não serem reconhecidos).



(a) Segmentação errada (b) Segmentação correta

Figura 2.2 Segmentação por *bounding box*

Outras abordagens para tratar o problema são possíveis, assim como é possível combinar técnicas a fim de aumentar a eficiência do segmentador.

2.2 Reconhecimento

Após segmentar um símbolo, o próximo passo é reconhecê-lo. Tipicamente, o caractere a ser reconhecido é representado por um conjunto de características ou pela própria matriz correspondente ao conjunto de traços. Os objetos assim representados são comumente denominados **padrões**. Dentre as possíveis técnicas para abordagem do problema, é possível citar [10]:

- extração de características estruturais e classificação por árvore de decisão;
- *Hidden Markov Model*;
- redes neurais;
- extração de características e classificação pelo vizinho mais próximo.

2.2.1 Redes Neurais

As redes neurais artificiais (RNA) surgiram na década de 1950 baseadas em modelos matemáticos do funcionamento do cérebro humano [11], apesar de quase não possuírem semelhanças do ponto de vista biológico. Como grande semelhança, podemos citar o fato de que a estrutura de uma rede neural, assim como a do cérebro, ser composta por células, denominadas **neurônios**, capazes de detectar características que se comunicam através de conexões (sinapses) [9].

Cada célula é composta por um ou mais **sinais de entrada**, uma **função de ativação** e uma **saída**, que é determinada pela função de ativação calculada sobre as entradas correntes. A rede neural é formada por camadas de células conectadas umas às outras, sendo que a cada conexão é atribuído um peso. Para que a rede torne-se capaz de classificar padrões, deve-se utilizar um algoritmo de treinamento que possibilite o ajuste dos pesos das conexões entre as células.

As redes neurais artificiais são largamente utilizadas por serem capazes de tratar problemas complexos que seriam de difícil solução utilizando métodos convencionais de computação. Além disso, apresentam capacidade de generalização, extraindo características a partir de um conjunto inicial de exemplos para então exibir uma resposta correta para padrões diferentes dos já apresentados [9]. Redes neurais possuem alto grau de adaptabilidade, pois são capazes de se ajustarem às mudanças no ambiente.

A capacidade adaptativa das RNAs permite que elas sejam utilizadas na resolução de diversos problemas, como análise de pesquisa de mercado, controle de processos industriais, aplicações climáticas, diagnóstico médico e também, é claro, reconhecimento de padrões (imagens, voz, texto, entre outros).

Como citado anteriormente, para que a rede consiga classificar padrões, deve antes ser treinada. Dessa forma, existem diversos tipos de algoritmos de treinamento que são divididos em duas categorias: supervisionados e não-supervisionados.

Algoritmos supervisionados. Um supervisor (professor) fornece uma entrada e a resposta desejada, permitindo assim dividir o espaço de busca em classes. Uma desvantagem desse tipo de aprendizado é que, na ausência do professor, a rede não é capaz de aprender novas situações não cobertas pelos exemplos. Como exemplo de algoritmo supervisionado, podemos citar o algoritmo ***backpropagation*** para treinar redes MLP.

Algoritmos não-supervisionados. Somente os padrões de entrada estão disponíveis para

a rede. Neste caso, é necessário haver redundância nos exemplos, para que a rede possa estabilizar-se, e então extrair características dos padrões, que permitirão realizar a separação em classes. Ao final desse processo, é necessário estabelecer correspondências entre os padrões classificados e o objeto que representam. Como exemplo deste tipo de algoritmo, podemos citar a versão do Neocognitron [18] utilizada neste trabalho.

Ao projetar uma rede neural, é necessário que haja flexibilidade suficiente para que a rede seja capaz de se adaptar às variações presentes no conjunto de dados a serem classificados, porém sem modelar o ruído contido neles. Isso torna necessário realizar um pré-processamento nos dados de entrada. Como parte do pré-processamento, é possível realizar a extração de características dos padrões e então utilizá-las na classificação dos mesmo [25]. Algumas características que podem ser consideradas são:

- coordenadas
- ângulo de curvatura
- variações no ângulo de curvatura
- ângulo acumulado
- centro de gravidade
- comprimento
- comprimento relativo

Antes de discorrer sobre os tipos de redes neurais utilizadas neste trabalho, segue um breve histórico, baseado em [9] e [11]:

- 1943 - Warren McCulloch e Walter Pitts apresentam o primeiro modelo artificial de um neurônio biológico, chamado de nó MCP.
- 1949 - Donald Hebb apresenta o primeiro modelo de aprendizado, baseado no ajuste dos pesos entre as conexões de dois neurônios ativos simultaneamente. Ele conseguiu mostrar que o aprendizado é possível através da variação dos pesos das entradas dos nós, propondo uma regra para tal fim, conhecida como *Regra de Hebb*. Mais tarde, uma nova regra, baseada “no método do gradiente para minimização de erro” (a regra delta), foi proposta por Widrow e Hoff.

- 1958 - Frank Rosenblatt apresentou um novo modelo, o *perceptron*, que alia conexões ajustáveis aos nós MCP, permitindo que a rede seja treinada, e assim, classifique padrões. No entanto, a estrutura apresentada, constando de três camadas, só é capaz de diferenciar classes linearmente separáveis.

- 1969 - Minsky e Papert criticam o modelo perceptron, por não ser capaz de detectar paridade, conectividade e simetria que não são linearmente separáveis. Além disso, ressaltaram o fato de não existir um modelo de treinamento para redes perceptron multi-camadas. Essas críticas acabam causando diminuição na pesquisa.

- 1983 - Kunihiko Fukushima cria o neocognitron, rede neural auto-organizável, especializada no reconhecimento de caracteres. Este tipo de rede, objeto de estudo no presente trabalho, será abordado logo mais.

- 1986 - Surge o algoritmo *backpropagation*, que derrubou as críticas de Minsky e Papert, uma vez que permite o treinamento de redes multi-camadas, sendo, portanto, responsável pela retomada das pesquisas nessa área.

2.2.1.1 MLP (*Multi-Layer Perceptron*)

A rede neural perceptron em multi-camadas é composta por uma camada de entrada, uma ou mais camadas intermediárias (*hidden layers*) e uma camada de saída, como podemos ver na figura 2.3. A saída de cada uma das células de uma camada é ligada às entradas de todas as células da próxima camada.

Segundo Minsky e Papert, “uma rede neural com somente uma camada é capaz de aprender apenas problemas linearmente separáveis” [11]. Mesmo quando se trata de uma rede neural multi-camadas, o problema se repete, caso a função de ativação seja linear. Uma RNA que tenha ao menos duas camadas intermediárias (e células cujas funções de ativação sejam não lineares) é capaz de aprender problemas que não são linearmente separáveis, uma vez que são capazes de aproximar qualquer função matemática [9].

O algoritmo mais utilizado para treinamento de MLP é o *backpropagation*. Como já foi dito, este é um algoritmo supervisionado, ou seja, recebe como entrada o padrão a ser classificado, e a resposta desejada. O treinamento é feito por meio do ajuste de pesos, visando-se à correção de erros [9]. Utiliza-se o método do gradiente descendente para minimizar uma função

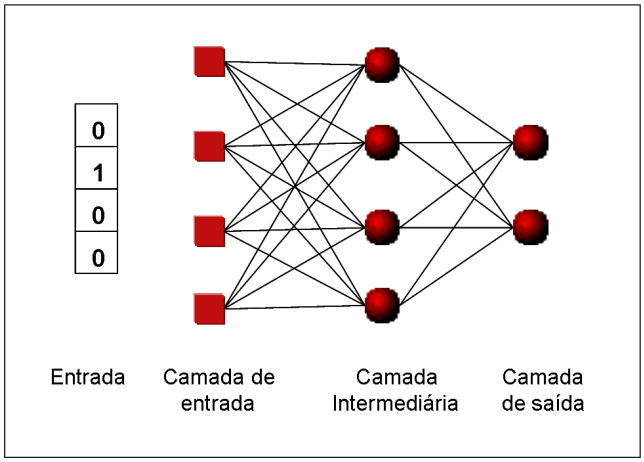
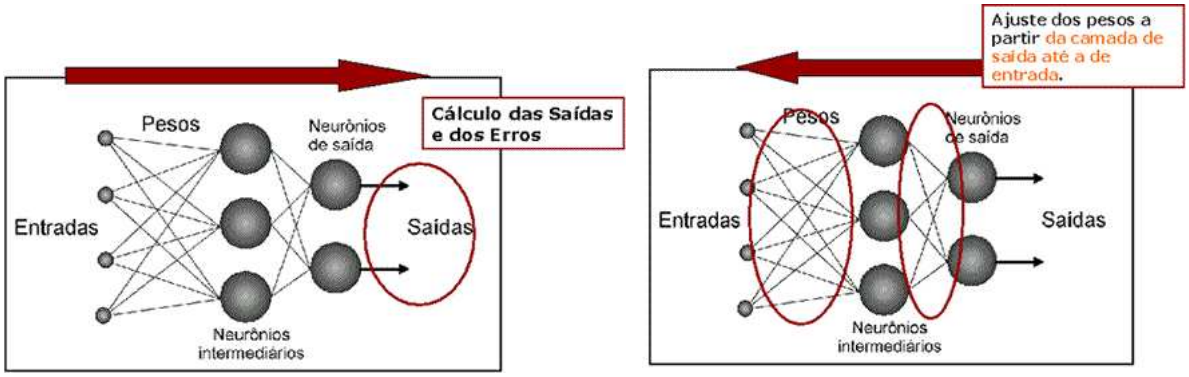


Figura 2.3 MLP

de erros. Para problemas complexos, porém, o algoritmo pode convergir para mínimos locais (que representam uma "solução estável que, porém, não é correta" [9]), uma vez que fornece aproximações para a solução. O treinamento é feito em duas fases, propagação (*forward*) e retro-propagação (*backward*), como pode ser visto nas figuras 2.4(a) e 2.4(b), respectivamente.



(a) Propagação (b) Retro-propagação

Figura 2.4 Tipos de propagação da rede neural (Figuras extraídas de [6])

Existem algumas dificuldades inerentes ao treinamento de RNA do tipo MLP, entre eles, a definição dos parâmetros de treinamento, por exemplo, a taxa de aprendizado, pois a escolha dos parâmetros, geralmente, é empírica [9], [11].

2.2.1.2 Neocognitron

O neocognitron é um algoritmo baseado na estrutura do sistema nervoso visual. Consiste de uma rede neural multi-camadas, que utiliza um algoritmo de aprendizado não-supervisionado e foi criado por Kunihiko Fukushima na década de 80. A grande vantagem do neocognitron em relação ao MLP é sua tolerância a pequenos deslocamentos e deformações, reduzindo assim o trabalho no pré-processamento e aumentando o poder de reconhecimento. O neocognitron é uma evolução do algoritmo Cognitron [13], [12], que realiza o reconhecimento de padrões através de *template matching*, mas assim como as redes do tipo MLP, são sensíveis a deslocamentos ou inclinações nos dados de entrada [18]. Além disso, existem diversas versões do neocognitron, sempre apresentando melhorias com relação às anteriores [14], [15], [16] e [17].

Estrutura e Comportamento

As definições, fórmulas e algoritmos apresentados nesta seção foram extraídas de [18]. O neocognitron é composto por módulos e um foto-receptor. O foto-receptor é um plano responsável por capturar sinais de entrada do padrão. Um módulo é uma estrutura que possui um par de camadas: uma *camada S* seguida de uma *camada C*. A camada *S*, por sua vez, contém *planos S* e estes contêm *células S*, assim como a camada *C* contém *planos C* que contêm *células C*.

A camada *S* é responsável por selecionar as características do padrão e a camada *C* é responsável por reforçá-las. Cada camada *C* reforça apenas as caraterísticas da camada *S* precedente. Desse modo, as caraterísticas são passadas de módulo para módulo e cada módulo vai, ao poucos, tendo uma visão mais geral de todas as características extraídas e, ao final, o algoritmo é capaz de reconhecer e classificar o padrão apresentado juntando todas as características.

As ligações entre as camadas do neocognitron são ilustrada na figura 2.5, onde U_0 é o foto-receptor, U_{Sl} e U_{Cl}, $l = 1,2,3$, são as camadas *S* e *C*, respectivamente e k_l são os planos celulares. Nessa figura, temos 3 módulos.

Num módulo, cada célula *C* tem como entrada as saídas de um conjunto de células *S* de um plano *S* e cada célula *S* tem como entrada as saídas de um conjunto de células *C* de todos os planos *C* precedentes. A esse conjunto de células denomina-se *campo receptivo*.

Considere que a saída de uma célula *S* seja representada por $u_{Sl}(k_l, n)$, onde k_l é o plano que contém a célula e n é sua posição. Considere também $u_{Cl}(k_l, n)$ a saída de uma célula *C* definida de forma análoga.

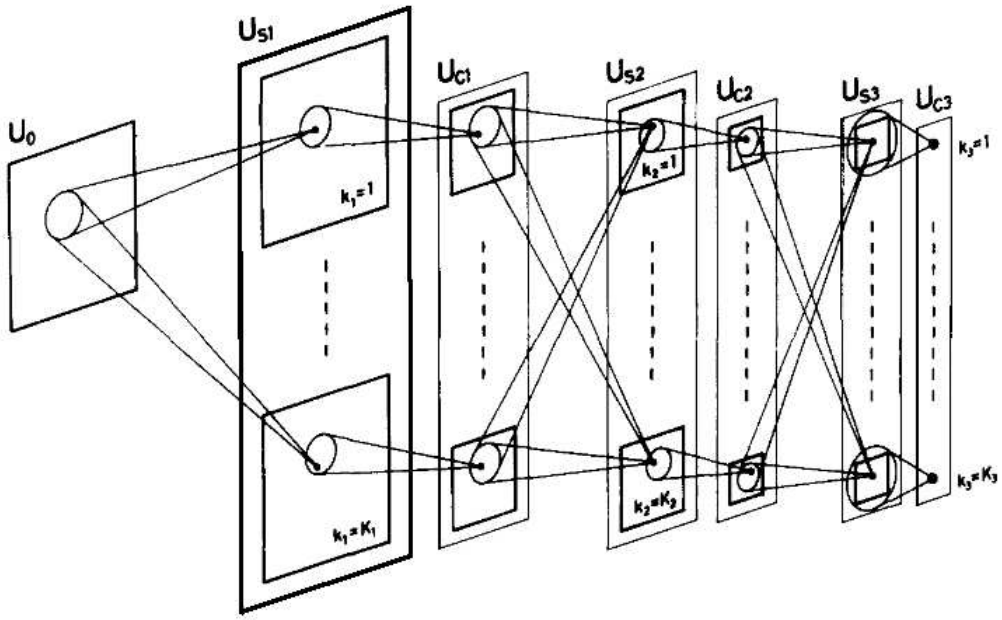


Figura 2.5 Interconexões entre camadas no neocognitron (Figura adaptada de [18])

As células C respondem às saídas das células S com a seguinte função de propagação:

$$u_{Cl}(k_l,n) = \psi\left[\frac{1+\sum_{v\in D_l} d_l(v)\cdot u_{Sl}(k_l,n+v)}{1+v_{Sl}(n)} - 1\right], \tag{2.1}$$

onde $v_{Sl}(n)$ é o sinal enviado pela célula inibitória que é proporcional à média aritmética ponderada de suas entradas como consta em [18]; $d_l(v)$ são as conexões fixas cujo valor decresce monotonicamente a respeito de $|v|$ e D_l é o campo receptivo. $\psi[\,]$ é uma função definida por

$$\psi[x] = \begin{cases} \frac{x}{\alpha+x} & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{2.2}$$

e α é uma constante positiva que determina o grau de saturação da saída.

A saída das células S obedece à seguinte fórmula:

$$w = \varphi\left[\frac{1+\sum_{v=1}^{N} a(v)\cdot u(v)}{1+b\cdot v} - 1\right], \tag{2.3}$$

onde $\varphi[\,]$ é definida por

$$\varphi[x] = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{2.4}$$

e $u(1), \cdots, u(N)$ são sinais excitatórios de entrada e v é o sinal da entrada de inibição, $a(v)$ e b são, respectivamente, os coeficientes de excitação e inibição das ligações entre as células.

Auto-Organização

Originalmente, o neocognitron foi desenvolvido utilizando um algoritmo não-supervisionado, ou seja, sem a entrada de padrões conhecidos a priori e sem uma saída esperada no final de cada módulo. Ao entrar com uma característica na entrada do módulo é escolhido, segundo o critério *winner takes all*, o plano que contém a célula mais representativa, ou seja, aquela célula que tem a maior saída em resposta à característica considerada. Após a escolha do plano e da célula representativa, os pesos de ligação entre as células do plano escolhido e seu campo receptivo são ajustados segundo o ajuste de pesos da célula representativa.

Sejam $a(k_{l-1}, v, \hat{k_l})$ e $b_l(\hat{k_l})$ os coeficientes de excitação e inibição das ligações da célula representativa. As equações seguintes representam os valores de reforço das ligações:

$$\Delta a(k_{l-1}, v, \hat{k_l}) = q_l \cdot c_{l-1}(v) \cdot u_{Cl-1}(k_{l-1}, \hat{n} + v) \tag{2.5}$$

$$\Delta b_l(\hat{k_l}) = q_l \cdot v_{Cl-1}(\hat{n}) \tag{2.6}$$

onde q_l é uma constante positiva que determina a velocidade de aprendizado.

A cada etapa na qual os pesos são reforçados, a rede neural torna-se cada vez mais especializada em reconhecer diferentes padrões formados por diferentes características. Somente os pesos das ligações da camada C para a camada S são reforçadas, uma vez que as ligações entre células da camada S para a camada C são fixas.

CAPÍTULO 3

Atividades realizadas

3.1 Captação de requisitos do *software*

Primeiramente foi verificado juntamente com orientadora quais necessidades que o *software* deveria atender: quais as funcionalidades desejadas e como o *software* deveria se comportar. O objetivo seria desenvolver um sistema capaz de reconhecer expressões matemáticas de maneira eficaz, que fosse de fácil utilização pelo usuário, além de ser modularizado, adaptável e expansível, facilitando também o trabalho dos programadores. Sendo assim, optamos por desenvolver o projeto utilizando a linguagem Java, devido à portabilidade e outras vantagens da orientação a objetos e também por ser a linguagem mais familiar para os membros da equipe.

Fizemos um comparativo entre o que seria necessário e o que o SisTREO [24] oferecia. Com isso, foi possível refinar ainda mais os requisitos do sistema. Após estudar o código do SisTREO, decidiu-se aproveitar o conhecimento adquirido para desenvolver um novo *software* mais robusto e expansível. Assim foi criado o projeto *Math-Picasso* que tem como objetivo reconhecer a escrita do usuário deixando o reconhecimento da expressão matemática para uma segunda etapa. O escopo menor, se comparado com o Projeto SisTREO, permitiu atentar ainda mais à estrutura do projeto e à interação do usuário com o *software*.

Pesquisamos métodos de segmentação e reconhecimento. Dentre tantos algoritmos de reconhecimento nos interessamos por aquelas que utilizam redes neurais.

3.2 Estudo de redes neurais

Para entender redes neurais, estudamos um pouco sua história [7] bem como suas principais aplicações [4]. Os algoritmos de redes neurais estudados foram: MLP com *back-propagation* e neocognitron.

A rede MLP foi estudada pelo fato de ser mais conhecida e utilizada. Após algumas pes-

quisas, foi encontrado o algoritmo Neocognitron e concluímos que seria interessante implementá-lo. A motivação para isso foi o fato de que o neocognitron é tolerante a pequenas deformações e deslocamentos, dispensando, assim, a fase do pré-processamento.

3.3 Estudo de técnicas de segmentação

Inicialmente estudamos a implementação do segmentador utilizado no SisTREO [24]. Para a segmentação, esse sistema utilizava-se de traços conexos [10].

Decidimos utilizar a segmentação por tempo entre *strokes*, pelo fato da escrita ser *online*, pode-se usar o tempo entre traços para determinar a segmentação.

3.4 Projeto do *software*

Decidimos projetar o *software* utilizando a plataforma Java [2], pelo fato de ser portátil, robusto, escalável entre outros benefícios.

Para o desenvolvimento da interface gráfica, utilizamos a biblioteca SWT [5], um *framework* desenvolvido pela IBM usado para a criação de interfaces gráficas em Java. Com isso foi possível criar a interface gráfica do *Math-Picasso* e entender como funcionaria a captura de dados do usuário para o reconhecimento. Com a captura de dados e a interface em mãos, o próximo passo seria estudar e modelar as estruturas básicas para representação dos símbolos a serem reconhecidos.

Para estruturar o software, utilizamos o padrão de projeto MVC (*model-view-controller*), que permite uma separação entre o modelo, a visualização e o controle da aplicação, facilitando adaptações e melhorias do código.

Maiores detalhes sobre a implementação do *Math-Picasso* podem ser vistos no próximo capítulo.

CAPÍTULO 4

O projeto

4.1 Projeto do sistema computacional

Para este projeto, escolhemos implementar redes neurais para realizar o reconhecimento de símbolos. Mais especificamente, as redes MLP (também utilizada no SisTREO [24]) e o neocognitron [18]. Para a segmentação de símbolos, utilizamos a técnica de segmentação por tempo entre *Strokes*.

Em todo o projeto, procuramos manter as classes com alta coesão e baixo acoplamento, para isso utilizamos alguns padrões de projetos [19].

O padrão MVC foi utilizado para desvincular as camadas de negócio (*model*) e de visualização (*view*). Isso torna o projeto mais claro e possíveis modificações da interface gráfica podem ser realizadas sem a necessidade de alterar o modelo.

O padrão *State* foi utilizado para controlar os estados da parte de treinamento. Além disso, podemos citar outros padrões utilizados como o *Singleton* e o *Builder*.

Para a interface gráfica, utilizamos a biblioteca SWT.

Procuramos utilizar boas práticas de programação orientada a objetos além de periódicas refatorações.

4.2 Sistema do ponto de vista estrutural

Como dito anteriormente, a estrutura do *Math-Picasso* foi projetada baseada no padrão MVC.

Separamos o projeto em pacotes segundo sua funcionalidade.

4.2.1 Camada de negócio

Esta camada contém todos os estados do programa, a lógica de fluxos e algoritmos de segmentação. Mais precisamente, contém os pacotes:

- ***model***

  Este pacote contém a lógica do programa, o estado do programa e controle de interação do usuário. Construímos classes para controlar cada ambiente: ambiente de treinamento (`TrainingModel`) e ambiente de entrada de dados (`InputModel`). Estas classes delegam funções para outros objetos de acordo com a entrada de dados do usuário.

  Ao receber dados de escrita de símbolo, composto por pontos (`StrokePoint`), essas classes delegam para o objeto `StrokeBuilder` a construção dos traços. O `StrokeBuilder` por sua vez passa o `Stroke` para o objeto `SymbolBuilder` que faz a segmentação e a construção do símbolo (`Symbol`). Ao enviar de volta, o modelo constrói um novo objeto `CharSymbol` que associa o símbolo ao caractere. Este objeto (`CharSymbol`) é armazenado numa coleção que é mostrado de volta na tela.

- ***structure***

  Objetos que compõem a representação de pontos, traços, símbolos, caracteres e *bounding box*.

- ***recognizer***

  Contém os reconhecedores de símbolos. Contém duas implementações de reconhecedores contidos nos subpacotes: o `neocognitron` e o `mlpRecognizer`. Desse modo, um novo reconhecedor pode ser facilmente implementado desde que contenha um objeto de fachada [19] que obedeça a interface `SymbolRecognizer`.

- ***misc***

  Contém as miscelâneas como *perfil*, *constantes* e *seletor de linguagem*.

4.2.2 Camada de visualização

Representada pelo pacote `gui` (interface gráfica do usuário), contém todas as interfaces de interação com o usuário. Possui duas classes principais, `InputTab` e `TrainningTab`, relacionadas, respectivamente, aos ambientes de Entrada de Dados e Treinamento. As classes

da GUI correspondem à camada *View* do padrão MVC. Essas classes informam dados de entrada do usuário para o controle (`InputControl` ou `TrainningControl`) e reproduzem o estado do Modelo.

4.2.3 Camada de controle

Esta camada está representada pelo pacote ***core***. Contém as classes que fazem o papel da camada *Controller* do MVC. Recebem eventos das GUI e alteram o modelo conforme a lógica do programa e não a lógica de negócio que faz parte do *Model*.

4.3 Implementações

Inicialmente, estruturamos o projeto e separamos em camadas como explicado acima. Assim criamos grande parte da interface do usuário.

A próxima etapa foi criar o editor de escrita e os objetos que compõem a estrutura dos símbolos (classes contidas no pacote `structure`).

Com a possibilidade de escrever os símbolos, criamos o segmentador de símbolos e acessórios como o demarcador/visualizador de tempo para a segmentação.

A próxima etapa foi criar funcionalidades da área de treinamento como: seleção de caracteres para treinamento e possibilidade de salvar/carregar símbolos (escrito pelo usuário) associados com o caractere para o treinamento.

Criamos adaptadores para utilizar o reconhecedor do SisTREO [24] em nosso sistema. Como a taxa de reconhecimento era muito baixo, decidimos melhorar o pré-processamento. Mesmo assim, o reconhecedor do SisTREO não era satisfatório, visto que para caracteres como "*I*", "1", "/", "*l*" e símbolos parecidos como "2" e "*Z*" falhava. Por isso, decidimos construir um novo reconhecedor, o neocognitron.

Durante a execução do projeto, mantivemos a implementação do MLP enquanto desenvolvíamos o neocognitron. Corrigimos alguns problemas da rede MLP e adicionamos o retreinamento para que o sistema pudesse cada vez mais se adaptar à escrita do usuário. Implementamos a RNA neocognitron, porém encontramos algumas dificuldades na parametrização de variáveis de valores empíricos e determinação de características.

O projeto *Math-Picasso* atualmente funciona utilizando o reconhecedor MLP com *back-*

propagation. O reconhecedor com o neocognitron ainda está em fase de ajustes e os testes de reconhecimento realizados não apresentam resultados adequados.

CAPÍTULO 5

Resultados e conclusão

5.1 Resultados

Descrevemos aqui alguns resultados obtidos da implementação do projeto.

O segmentador do *Math-Picasso* funciona de maneira correta uma vez que o usuário se adequa ao sistema.

A idéia inicial do reconhecimento de símbolos no *Math-Picasso* era realizá-lo em tempo de execução, ou seja, no momento em que o usuário estivesse escrevendo (no Modo Reconhecimento), o *Math-Picasso* já reconheceria o símbolo. Porém, essa idéia foi logo descartada, pois o desempenho do segmentador foi severamente afetado para um alfabeto de aproximadamente 20 caracteres.

Dado o problema do reconhecimento em tempo de execução, a solução foi, então, deixar que o usuário decida quando reconhecer os símbolos. Desse modo, além de o usuário poder reconhecer os símbolos individualmente ou todos ao mesmo tempo, o desempenho do segmentador não é afetado.

Obtivemos uma interface gráfica intuitiva e amigável ao usuário, visto que o *Math-Picasso* pode ser utilizado sem apresentar um comportamento inesperado.

O projeto atingiu um grau de desenvolvimento satisfatório e estável.

5.2 Conclusões

A RNA MLP é mais simples de ser implementada, porém exige uma fase de pré-processamento. Foi constatado que, mesmo depois de muito estudar o neocognitron, existem algumas dificuldades na sua implementação, como extração de características, ajuste de parâmetros para o aprendizado da rede e o modo como os módulos são treinados.

À medida que o conjunto de caracteres aumenta, a RNA MLP é capaz de adaptar-se fa-

cilmente à nova configuração, pois é criada uma RNA para cada caractere adicionado. Porém, o reconhecimento nessa rede exige um maior processamento, pois o sistema compara as saídas de todas as RNA para cada um dos símbolos. Além disso, com o aumento da quantidade de caracteres treinados, aumenta a probabilidade desta RNA reconhecer um símbolo erroneamente. Esta probabilidade só se mantém se, a medida que aumenta a quantidade de caracteres, aumentar também a quantidade de exemplos de treinamento de todos os caracteres já treinados.

No neocognitron é necessário configurar o número máximo de padrões (caracteres) que esta RNA irá reconhecer e, caso necessite reconhecer mais caracteres além deste limite, deve-se modificar a rede para suportar a nova configuração. O processamento no reconhecimento não é afetado, como no MLP, pois existe uma única rede para todos os caracteres. Porém, é necessário treinar muitas vezes todos os caracteres até que a rede torne-se estável antes de realizar o reconhecimento. Este comportamento não é interessante neste projeto, visto que muitas vezes não sabemos a priori o conjunto de símbolos que queremos reconhecer. O re-treinamento também é uma tarefa difícil no neocognitron, pois, uma vez estabilizada as ligações da rede, a adição de um novo símbolo na rede pode não ser significativa o bastante para o reconhecimento correto do mesmo.

A segmentação com base no intervalo de tempo entre traços é uma técnica muito prática e eficaz, visto que basta o usuário adaptar-se ao sistema para a correta segmentação dos símbolos. Dessa forma, obtém-se, também, um aumento na qualidade do reconhecimento.

CAPÍTULO 6

Apêndice

6.1 *Math-Picasso* do ponto de vista operacional

Ao usar/utilizar o *Math-Picasso*, o usuário logo percebe que pode trabalhar com duas áreas independentemente: a área de reconhecimento (figura 6.3) e a de treinamento (figura 6.1).

6.1.1 Área de treinamento

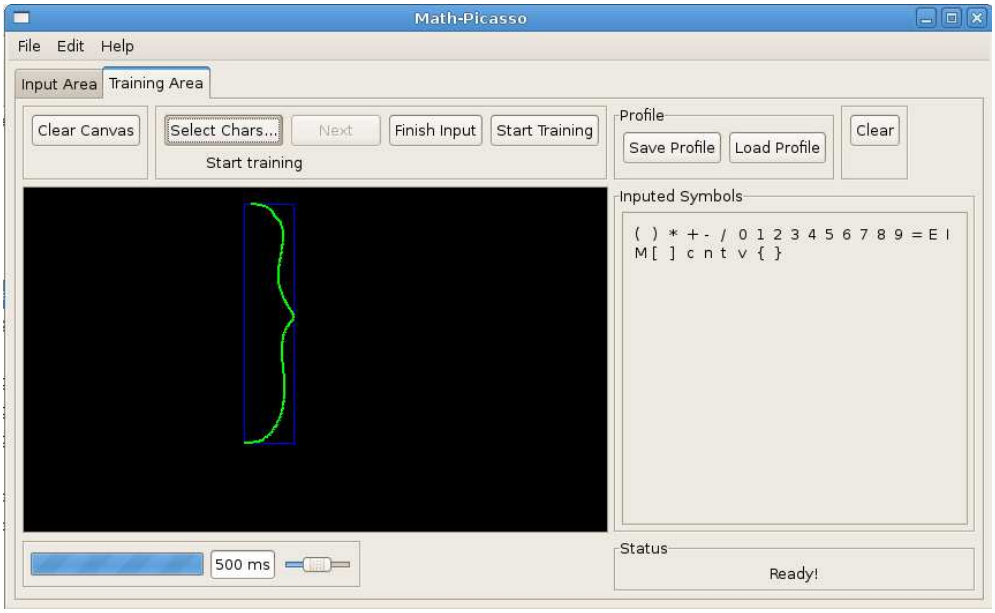


Figura 6.1 Área de treinamento do *Math-Picasso*

É onde o usuário ensina o software a reconhecer a sua escrita. Tal ensinamento é feito da seguinte forma:

- o usuário escolhe os caracteres que deseja ensinar ao *software* (figura 6.2)

- em seguida, escreve os caracteres escolhidos (figura 6.1)
- então a rede é treinada e o *Math-Picasso* está apto a reconhecer os símbolos

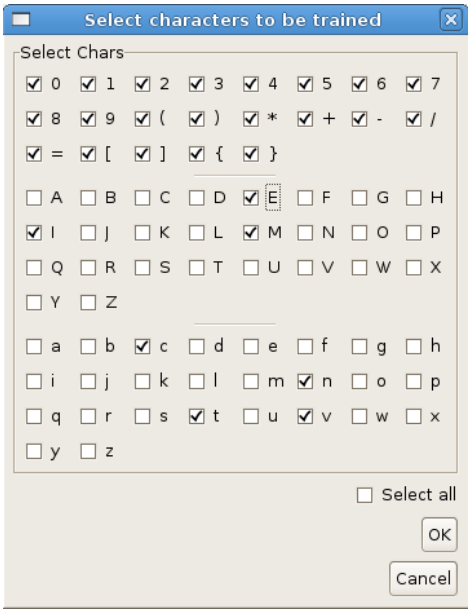


Figura 6.2 Selecionando caracteres

6.1.2 Área de reconhecimento

É onde a escrita é segmentada e em seguida reconhecida.

Para reconhecer os símbolos, basta escrever sobre a área de reconhecimento (figura 6.3) e dar o comando "reconhecer todos"para reconhecer todos os símbolos escritos ou, se desejar, reconhecer cada símbolo individualmente. No caso de erro no reconhecimento, basta retreinar a rede com o caractere desejado.

Cada símbolo contém um menu de contexto onde as seguintes opções são disponíveis (figura 6.4):

- remover o símbolo - remove o símbolo da tela
- retreinar como - treina o *Math-Picasso* para reconhecer o símbolo corretamente
- reconhecer - reconhece apenas o símbolo deste menu de contexto

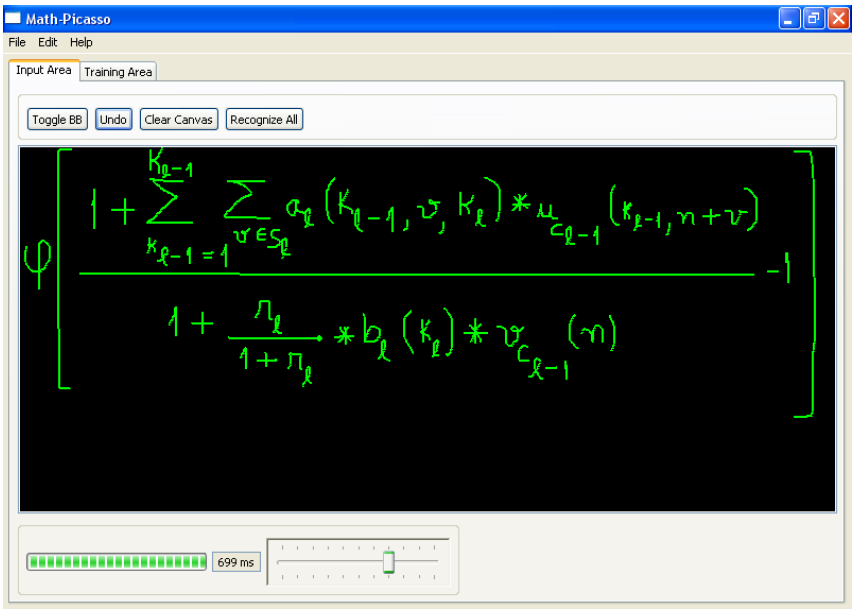


Figura 6.3 Área de reconhecimento do *Math-Picasso*

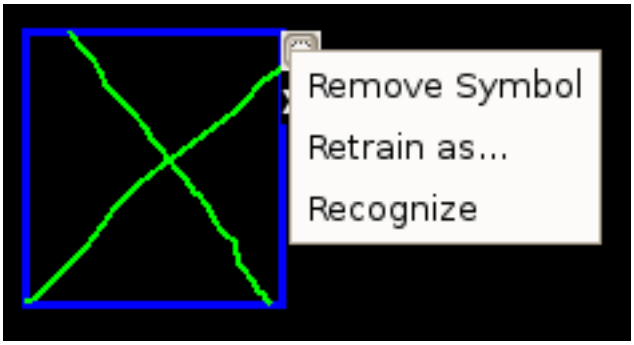


Figura 6.4 Menu de contexto

PARTE II

Parte subjetiva

CAPÍTULO 7

Parte Subjetiva

Desafios e Frustrações

Deparar com um problema que eu nunca tinha visto antes e pesquisar maneiras de solucionar o mesmo foi uma experiência muito interessante e desafiadora. Porém, o maior desafio foi conciliar o tempo entre as disciplinas, o projeto de formatura e o estágio. Apesar de o estágio não ser obrigatório no curso eu acho importante uma experiência profissional durante a graduação, seja para facilitar a entrada do aluno no mercado de trabalho, seja para aprimorar e aplicar os conhecimentos adquiridos durante o curso.

Paciência, comprometimento e determinação são imprescindíveis para qualquer trabalho em equipe. Um bom relacionamento com os integrantes do grupo garante um resultado melhor no projeto. Por isso, considero o desenvolvimento do *Math-Picasso* um sucesso.

Disciplinas mais relevantes

- *MAC*0122 – Princípios de Desenvolvimento de Algoritmos
  Disciplina fundamental que é utilizada no decorrer do curso inteiro.

- *MAC*0332 – Engenharia de Software
  Disciplina importante para se ter idéia de como projetar software de maneira pragmática, desde o rascunho inicial até a sua concretização.

- *MAC*0441 – Programação Orientada a Objetos
  Esta disciplina foi muito importante para conhecer as técnicas de modelagem e estruturação de projetos, bem como boas práticas de programação. Diversos padrões apresentados nessa disciplina foram aplicados neste projeto.

Conhecimentos adquiridos em outras disciplinas (*MAC*0211 – Laboratório de Programação I; *MAC*0242 – Laboratório de Programação II; *MAC*0438 – Programação Concorrente; *MAC*0338 – Análise de Algoritmos) também contribuíram para do desenvolvimento deste projeto.

Aplicação de conceitos

Aplicar o aprendizado de cada disciplina independentemente não é uma tarefa muito fácil e prática, dado que a carga teórica é muito grande. Porém, ao cursar *MAC*0499, foi possível juntar o todo conhecimento adquirido nas disciplinas e aplicá-lo de forma interdisciplinar, o que amadurece bastante a formação acadêmica.

O Futuro

Achei o desenvolvimento de projeto muito divertido e construtivo para minha formação acadêmica. Tenho intenções de dar suporte e continuar melhorando o projeto como um todo.

Como esse projeto trata de processamento de padrões, fica claro a existência de uma ligação com processamento de imagens, uma área que muito me interessa. Assim, pretendo fazer um estudo maior nesse campo.

Referências Bibliográficas

[1] http://fbim.fh-regensburg.de/∼saj39122/jfroehl/diplom/e-index.html.

[2] http://java.sun.com.

[3] http://kazan.inf.fu-berlin.de/echalk/en/index.php.

[4] http://www.din.uem.br/ia/neurais.

[5] http://www.eclipse.org/swt/.

[6] http://www.lncc.br/∼labinfo/tutorialrn/frm4_backpropagation.htm.

[7] http://www.neurocomputing.org/nnhistoryto1990.aspx.

[8] D. Blostein and Ann Grbavec. Recognition of mathematical notation. *Handbook on Optical Character Recognition and Document Image Analysis*, 1996.

[9] Antonio Braga, André Carvalho, and Teresa Ludermir. *Fundamentos de Redes Neurais Artificiais*. 11 ª Escola de Computação, 1998.

[10] K. Chan and D. Yeung. Mathematical expression recognition: a survey. 2000.

[11] Laurene Fausett. *Fundamentals of Neural Networks Architectures, Algorithms and Applications*. Prentice Hall International, Inc, 1994.

[12] Kunihiko Fukushima. Cognitron a self-organizing multi-layered neural network. *Biol Cybernet*, 20:121–136, 1975.

[13] Kunihiko Fukushima. Cognitron a self-organizing multi-layered neural network model. *NHK Technical Mongraph*, 30, 1981.

[14] Kunihiko Fukushima. Analysis of the process of visual pattern recognition by the neocognitron. *Neural Networks*, 2:413–420, 1989.

[15] Kunihiko Fukushima. Self-organization of shift-invariant receptive fields. *Neural Networks*, 12:791–801, July 1999.

[16] Kunihiko Fukushima. Neocognitron for handwritten digit recognition. *Neurocomputing*, 51:161–180, April 2003.

[17] Kunihiko Fukushima. Neocognitron capable of incremental learning. *Neural Networks*, 17:37–46, January 2004.

[18] Kunihiko Fukushima and Sei Miyake. Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position. 1983.

[19] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[20] K. Ishigashi, H. Tanaka, and N. Iwayama. Interactive character recognition technology for pen-based computers. 1999.

[21] Daniel Karrer, Renato Cameira, André Vasques, and Marcos Benzecry. Redes neurais artificiais: Conceitos e aplicações.

[22] A. Kosmala and G. Rigoll. Recognition of on-line handwritten formulas. *6th Int. Workshop on Frontiers in Handwriting Recognition (IWFHR)*, 1998.

[23] Yann LeCun, Bottou Léon, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition.

[24] Eduardo Pires, Ricky Chow, and Pedro Henrique Oliveira. Sistema titanium de reconhecimento de escrita online - sistreo. 2006.

[25] Ernesto Tapia and Raul Rojas. Recognition of on-line handwritten mathematical formulas in the e-chalk system.

[26] C. Tappert, C. Suen, and T. Wakahara. The state of the art in on-line handwriting recognition. 1990.

Este volume foi tipografado em LaTeX na classe UFPEThesis (www.cin.ufpe.br/~paguso/ufpethesis).