Universidade de São Paulo
Instituto de Matemática e Estatística
MAC 499 - Trabalho de Formatura Supervisionado
Arcabouço para Reconhecimento de Escrita Online
SisTREO

Sistema Titanium de Reconhecimento de Escrita Online

Ricky Ye Lun Chow
rylchow@gmail.com
Pedro Henrique Simões de Oliveira
pedrohenriquesimoesdeoliveira@gmail.com
Eduardo Gusmão Caceres Pires
eduardocaceresp ires@gmail.com

Orientadora: Prof. Dra. Nina S. T. Hirata
nina@ime.usp.br
04/12/2006

1 Introdução

Esta monografia faz parte do projeto desenvolvido, durante um ano, como Trabalho de Formatura Supervisionado. A idéia inicial foi sugestão de nossa orientadora, a professora doutora Nina Hirata. A sugestão foi desenvolver um arcabouço para algoritmos de reconhecimento de escrita, mais especificamente para reconhecer a escrita de fórmulas matemáticas. Uma possível aplicação para este reconhecimento é transformá-lo num código em LaTeX para inserir tais fórmulas em relatórios e artigos, sem a necessidade do usuário conhecer a fundo como utilizar LaTeX para escrever artigos com fórmulas matemáticas.

Como nosso projeto tem como assunto o reconhecimento de escrita *online*, demos o nome SisTREO (Sistema Titanium de Reconhecimento de Escrita Online) ao nosso arcabouço.

1.1 Motivação

Quando digitamos um texto no teclado, o computador consegue muito bem interpretar o que está querendo ser passado como entrada. Mesmo que haja erros de digitação, esses erros

dependem somente do usuário, mas mesmo que a obrigação do computador não seja corrigí-los, muitos editores de texto conseguem corrigir certos tipos de erros. Já na escrita humana é totalmente diferente. Não há certeza do que o usuário quer passar como entrada, e erros não podem ser tratados como exceção, mas sim como regra.

O problema de reconhecimento de escrita humana envolve inúmeras variáveis que dependem não só da capacidade do algoritmo de prever o que o usuário realmente quer dizer, mas também do próprio usuário, que pode, por diversos motivos, escrever de diversas maneiras. Existem muitas formas de se resolver, em partes, este problema. A nossa proposta é desenvolver um arcabouço capaz de padronizar o tratamento dos sinais e o comportamento dos algoritmos com relação a uma estrutura de dados, para que a interação entre tais algoritmos seja possível, possibilitando uma combinação de técnicas que possa melhorar a capacidade de reconhecimento.

Antes, porém, é necessário conhecer como o reconhecimento é feito em grande parte dos casos. Para isso, recorremos a diversos artigos focados em diferentes problemas de reconhecimento:

- Texto (letra de forma e à mão) [Jaeger . 2000]
- Ideogramas orientais
- Fórmulas matemáticas
- Diagramas de blocos [Kara 2004]

Cada problema possui requisitos diferentes, mas muitas vezes técnicas utilizadas em uma abordagem podem servir para melhorar o reconhecimento de um outro problema. Estes requisitos serão explorados na próxima seção.

2 Conceitos

Atualmente surgem cada vez mais maneiras e idéias de interação entre o homem e o computador. Porém, para o caso aqui tratado podem ser utilizados acessórios simples, desde que atendam a alguns requisitos. Se a partir desse dispositivo conseguimos distingüir a ação de segurar (*pen-down*), a ação de largar (*pen-up*) e a posição em que o dispositivo se encontra, podemos, a partir dele, simular a escrita humana. Este é o caso do *tablet*, de uma caneta do tipo *Stylus* ou até mesmo de um simples mouse. Definidos os dispositivos de entrada, podemos então introduzir os conceitos usualmente envolvidos no reconhecimento de escrita.

2.1 Definições

2.1.1 Escrita *Online*/*Offline*

Escrita *online* é aquela onde a ordem dos traços é dada. Reconhecimento *online* é o reconhecimento de um objeto onde há a captura dos traços em relação ao tempo (x, y, t). Já a

escrita *offline* é aquela onde não há indicação temporal dos traços, que já estão definidos.

2.1.2 Traço (*Stroke*)

Conjunto de pontos gerados entre a ativação (ex.: segurar o botão do mouse) do dispositivo de entrada e a sua desativação (ex.: soltar o botão do mouse).

2.1.3 Símbolo (*Symbol*)

Conjunto de Strokes que pode ser associado a zero ou mais Caracteres. Explicaremos como decidimos se dois Strokes pertencem ao mesmo símbolo posteriormente.

2.1.4 Caractere (*Character*)

Elemento do alfabeto reconhecível. É associado a um Símbolo a uma certa probabilidade, dependendo dos casos, para atribuir um valor que possa ser utilizável computacionalmente.

2.1.5 Expressão (*Expression*)

Conjunto de todos os Símbolos escritos pelo usuário dentro de uma área específica. Também pode ser reconhecida através de uma gramática posicional.

2.1.6 Caixa Envoltória (*Bounding Box*)

Menor retângulo que envolve todos os pontos que formam um Símbolo, ou um Stroke.

2.2 Reconhecimento de Texto

Em problemas de reconhecimento de texto, temos dois casos completamente diferentes, tanto na abordagem quanto na sua dificuldade. Isso fica claro quando separamos os sub-problemas que devem ser resolvidos em cada caso.

2.2.1 Agrupamento de traços em símbolos

No reconhecimento de textos em letra de forma, podemos identificar claramente onde começa e onde termina cada letra, facilitando assim o agrupamento de traços em símbolos. Um modo simples de se delimitar símbolos é pela intersecção, o que em alguns casos pode não funcionar, como é o caso do pingo no i, ou de um acento, mas são casos simples de se resolver. Juntando grosseiramente traços em símbolos, caso haja intersecção entre traços, teremos dois símbolos para o i. Por exemplo: um símbolo seria o corpo do i, e o outro seria formado pelo pingo do i. Pegando os pontos centrais dos *bounding boxes*, podemos gerar um grafo conexo completo(não há pontos isolados), para através das arestas, extrairmos alguma correlação entre símbolos. Em outras palavras, bastaria achar uma árvore geradora mínima para verificar que o pingo do i muito provavelmente se ligará ao corpo do i, neste grafo. Esta técnica será ainda mais útil em fórmulas matemáticas, pois, para textos, pode-se considerar, ainda, que símbolos acima de um outro símbolo em uma mesma linha não existem.

Com letra de mão, não temos divisão clara entre as letras, somente entre as palavras. Um traço não necessariamente compõe um símbolo. Muitas vezes um traço é que representa vários símbolos, por isso devemos quebrar os traços em símbolos. A indicação do tempo nestes casos é muito importante, principalmente em técnicas que tentam achar um padrão conforme o texto vai sendo escrito. O tempo é a única indicação relevante para a separação em símbolos, considerando que o usuário escreverá um símbolo após o outro. As exceções ficam por conta de traços atrasados, como o corte no t ou novamente, o pingo no i.

2.2.2 Pré-Processamento

Com os símbolos separados, ainda há etapas de pré-processamento necessárias, dependendo do algoritmo de reconhecimento utilizado. Em geral, há a necessidade de se normalizar o tamanho do símbolo para que ele possa ser comparado com o modelo. Pode-se utilizar amostragem, onde pegamos algumas informações de um símbolo e transformamos num formato comparável com o modelo. Também pode-se simplesmente redimensionar o símbolo, achando linhas que delimitam a base, também chamada de *baseline*, e a linha superior, depois que achamos o *baseline*. Para encontrar o *baseline* usa-se regressão linear, achando uma reta que mais se aproxima de todos os pontos de uma expressão. Como a reta de regressão estará cortando as letras, desloca-se essa reta até acharmos um limite, usando a intersecção. Se a reta passar a cruzar poucos pontos, estamos abaixo do *baseline*, pois casos onde há pontos abaixo da linha são poucos (como a letra q, ou p). Esta normalização serve tanto para letras de forma quanto para letras de mão.

Outros problemas de pré-processamento envolvem:

- **Interpolação dos pontos**: pontos consecutivos podem estar longe um do outro, caso a velocidade da escrita seja muito alta. Se houver um outro traço cruzando este espaço vazio, poderá ocorrer um caso onde a intersecção será ignorada. Para tratar este problema estima-se os pontos intermediários. Essa técnica pode ser especialmente útil se a taxa em que o sistema reconhece os pontos de entrada for baixa. Há várias formas de se fazer isso, em especial destacamos Sp-lines e curvas de Bezier.

- **Normalização da inclinação**: muitas vezes uma pessoa escreve de forma inclinada, ou mais inclinada do que de costume. Isso pode causar erros de reconhecimento e, portanto, é um problema tratado em muitas técnicas. Mais uma vez a regressão linear pode ser útil, mas desta vez considerando um símbolo, e não a expressão toda. Dentro de um símbolo podemos encontrar o grau de inclinação da letra e, caso esteja longe da inclinação média das letras que temos como modelo, podemos deslocar os pontos de acordo, até encontrarmos uma inclinação próxima.

- **Smoothing**: a escrita humana é, acima de tudo, suscetivel a erros. Tremidas durante a escrita não só podem como de fato acontecem. Para evitar que isso afete o reconhecimento, podemos deslocar certos pontos observando localmente, dentro do traço.

2.2.3 Reconhecimento

O reconhecimento em letra de forma utiliza técnicas muitas vezes estatísticas. Elas tentam comparar o que foi escrito com o modelo que ele tem do alfabeto. Técnicas como redes neurais precisam de uma fase de treinamento onde o usuário entra com a sua escrita para que depois o algoritmo tente achar semelhança entre o que foi escrito e o que foi ensinado a ele. Também há a possibilidade de se ensinar durante o uso, ou seja, o algoritmo se adapta ao usuário. A utilização de *baseline* também auxilia no reconhecimento distingüindo formas que podem das que não podem cruzar o *baseline*.

No caso de escrita em letra de mão, pode-se ainda considerar pontos de interesse. São pontos onde a velocidade muda bruscamente, indicando uma mudança na direção, ou limites. Observando a velocidade durante a escrita, podemos identificar máximos e mínimos locais. Os mínimos podem indicar um ponto de interesse, onde a partir dali tomamos uma nova direção. Comparando os pontos de interesse de um símbolo com os pontos definidos num caractere, pode-se extrair semelhanças ou diferenças.

2.3 Reconhecimento de Fórmulas Matemáticas

Duas principais diferenças notadas entre fórmulas matemáticas e textos são:

- **Relacionamento entre símbolos/caracteres**: se num texto temos um caractere seguido do outro, sempre posicionados numa linha, isso não ocorre em fórmulas matemáticas. Não podemos assumir que teremos simplesmente um caractere seguido do outro, pois temos expoentes e índices que podem expandir-se até quando for necessário. Isso significa que se reconhecermos uma seqüência de símbolos e simplesmente os devolvermos nessa mesma ordem teríamos 22222 como resultado do reconhecimento de $2^{2^{2^{2^2}}}$. Uma gramática que leva em consideração a posição dos símbolos é extremamente útil para uma generalização onde o reconhecimento não se limita a linhas como num texto.

- **Ambigüidade**: textos só são ambígüos se reconhecemos além de simplesmente a escrita. Ambigüidade só existe no sentido das frases, mas numa expressão matemática muitas vezes pelo fato de levar em consideração a posição dos símbolos, podemos ter interpretações diferentes para uma mesma expressão. Dependendo da escrita do usuário, podemos tratar o último 2 da expressão "$2_{2^2}2$" como expoente do terceiro 2, como se formasse 22 com o segundo 2, ou ainda como se estivesse simplesmente multiplicando o primeiro 2. Este problema é uma conseqüência do uso de uma gramática posicional e na maioria das vezes só conseguimos solucionar com a intervenção do usuário indicando o erro manualmente ou até mesmo como corrigir.

2.4 Outros tipos de reconhecimento

No reconhecimento de diagramas de blocos, associamos símbolos a formas como retângulos ou flechas. Técnicas de agrupar traços em símbolos usados em textos podem não funcionar em diagramas. Considerar que símbolos são separados uns dos outros por um espaçamento

não ocorre com freqüência. Conectores interceptam blocos, ou seja, a separação em símbolos depende de informações sobre o que cada parte pode representar. Não são poucos os casos em que o reconhecimento não é linear. Muitas vezes é necessário voltar à etapa anterior e realizar mais alguma função de pré-processamento para poder prosseguir. O reconhecimento pode requerer informações de pré-processamento, e o pré-processamento pode precisar de informações de reconhecimento. Saber quando parar e voltar/avançar varia muito de abordagem para abordagem, e isso pesou muito no planejamento da arquitetura e das estruturas de dados.

Em reconhecimento de ideogramas, podemos levar em consideração a ordem dos traços, por exemplo. Além disso, as formas dos traços também são importantes, e ideogramas podem ser formados por radicais, ou seja, símbolos dentro de símbolos.

3 Atividades Realizadas

Nosso primeiro passo foi estudar as técnicas utilizadas atualmente. Não só estudamos artigos relacionados a fórmulas matemáticas, mas também artigos que tratavam casos como ideogramas e diagramas de blocos. Levantados os requisitos e definições, partimos para o planejamento da arquitetura do arcabouço. Decidimos que o desenvolvimento seria em Java pela sua portabilidade. A portabilidade é muito importante para o nosso projeto porque potenciais dispositivos que poderiam se utilizar do reconhecimento de escrita não se limitam somente a computadores pessoais convencionais. Atualmente PDAs e até videogames apresentam dispositivos capazes de aproveitarem técnicas de reconhecimento de escrita, e futuramente, com a forte tendência de convergência de dispositivos, até celulares poderão se aproveitar, numa escala muito superior da que vemos atualmente.

3.1 Isolamento de símbolos

Uma das principais vantagens do reconhecimento de escrita *online* sobre o *offline* é a presença de informações adicionais como o tempo e os traços. Um traço é composto por todos os pontos (posições) gerados desde o tempo em que o dispositivo foi acionado até onde ele foi desativado. Essas informações são muito importantes, já que isolar um símbolo pode ser uma atividade bem complexa. Isolar símbolos de uma imagem obtida através de um scaner, por exemplo, envolve vários algoritmos e grande parte dos erros em sistemas de reconhecimento de escrita *offline* ocorrem por erro de separação dos símbolos, e não em erros de reconhecimento. Por isso, para o isolamento dos símbolos, o fato do sistema ser *online* é muito importante. Vamos explicar a seguir como desenvolvemos o sistema de isolamento de símbolos.

3.1.1 Estudo de Técnicas e Implementação dos Algoritmos

No inicio, quando ainda discutíamos como iríamos isolar um símbolo na expressão escrita pelo usuário, pensamos em uma solução aparentemente simples, que achávamos que funcionaria verificando se os *bounding boxes* (caixas envoltórias) dos símbolos se interceptavam. Essa solução foi rapidamente descartada, já que ela facilmente reconheceria dois símbolos como sendo um só, como mostra a figura 1

Figura 1: Estes dois símbolos não são o mesmo, mas se a técnica utilizada fosse intersecção de *bounding box*, eles seriam identificados como sendo um símbolo só.

Logo, precisamos desenvolver uma nova técnica, que de certa forma foi baseada nesta:

- Verificávamos se o *bounding box* dos dois traços se interceptavam, em caso negativo, eles pertenciam a símbolos diferentes.
- Se eles se interceptavam, dividíamos os traços em conjuntos de 10 pontos. Estes conjuntos são obtidos pegando os pontos na ordem em que eles foram obtidos (isso só é possível em sistemas de reconhecimento *online*), ou seja, se temos n pontos, os primeiros $n/10$ pontos obtidos pertencem ao primeiro conjunto, e assim por diante.
- Depois, criávamos os *bounding boxes* destes conjuntos. Se qualquer um dos *bounding boxes* dos subconjuntos de um dos traços se interceptassem com o *bounding box* de algum subconjunto do outro traço, eles seriam considerados traços do mesmo símbolo.

É importante deixar claro que a técnica de intersecção utilizada entre os *bounding boxes* leva em conta um Δ, para que traços que não se interceptam, mas que estejam suficientemente próximos sejam considerados parte do mesmo símbolo. Isso é necessário porque o usuário pode escrever, por exemplo, uma somatória em duas partes, como mostrado nas figuras 2 e 3.

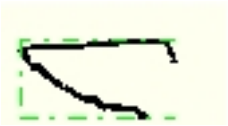

Figura 2: Parte de cima da somatória

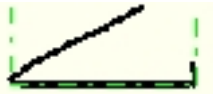

Figura 3: Parte de baixo da somatória

No exemplo, as duas partes se encostariam, mas poderiam não se interceptar, causando problemas se este Δ não fosse considerado.

Os resultados obtidos por esta técnica foram muito satistatórios, os símbolos são isolados da maneira correta, para que possam ser usados pelos algoritmos de reconhecimento de símbolos e expressão, que serão explicados nas próximas seções.

3.2 Reconhecimento de símbolos

Um símbolo pode ser entendido como um conjunto de propriedades de um sinal gráfico. Propriedades essas que são únicas no conjunto bem determinado ao qual ele pertence. A um conjunto de símbolos, chamamos alfabeto. Reconhecer um símbolo é na verdade, associar um sinal gráfico dado (a representação do símbolo) à categoria a que ele pertence. Para tanto, é necessário reconhecer as propriedades do sinal. Reconhecimento de padrões é uma área que já foi bastante estudada em computação e existem diversas técnicas bem estabelecidas para abordar este problema. Abaixo listamos algumas:

- Classificadores Bayesianos
- Redes Neurais
- Máquinas de Suporte Vetorial
- Transformadas de Fourier

3.2.1 *Online* vs *Offline*

A forma como esse sinal é lido tem relevância fundamental no processo de reconhecimento. O conhecimento do momento e da ordem em que os pontos são inseridos possibilita abordagens diferentes no reconhecimento de símbolos. Por exemplo, ideogramas chineses, por construção, obedecem a uma determinada seqüência que só pode ser aproveitada no reconhecimento quando temos informação sobre a ordem em que os elementos constituintes do símbolo foram inseridos. Quando não temos informações temporais, a única coisa que nos resta é a imagem do símbolo, que precisa ser tratada de forma a conseguirmos extrair o máximo de informações para o posterior reconhecimento.

3.2.2 Estudo de Técnicas e Implementação dos Algoritmos

- **O Arcabouço - Reconhecimento de Símbolos**

  O nosso arcabouço tinha como premissa ser flexível o bastante para suportar implementações de diversas técnicas de reconhecimento. Para tanto, todas as classes que reconheçam símbolos, devem implementar uma interface que recebe um "Symbol" (de acordo com a nossa definição) e devolve o "Character" reconhecido. Na aplicação desenvolvida, optamos por utilizar redes neurais para implementar o nosso reconhecedor de símbolos. A escolha levou em consideração apenas dois fatos: boa parte dos artigos consultados que utilizaram essa técnica em seus experimentos obtiveram bons resultados e o

fato dessa técnica se adaptar à escrita de cada pessoa, devido ao processo de treinamento (também comum a outras técnicas citadas anteriormente).

- **Redes Neurais Artificiais**

  Redes neurais artificiais podem ser entendidas como uma variação da idéia de processamento distribuído paralelo. Embora tenham se inspirado fortemente no comportamento das redes neurais biológicas, as semelhanças e diferenças entre ambas não serão abordadas aqui.

- **Um "arcabouço" para representação distribuída**

  Uma rede artificial consiste de um conjunto de unidades de processamento que enviam sinais umas as outras através de uma grande quantidade de conexões que atribuímos diferentes pesos. Dessa representação, podemos destacar os seguintes aspectos:

  - Um conjunto de unidades de processamento ("neurônios", "células").
  - Um estado de ativação y_k para cada unidade, que é equivalente a saída de cada unidade.
  - Conexões entre as unidades. Usualmente cada conexão é definida por um peso w_{jk} que determina o efeito que o sinal da unidade j tem na unidade k.
  - Uma regra de propagação, que determina a entrada efetiva s_k de uma unidade a partir de suas entradas externas.
  - Uma função de ativação F_k, que determina o novo nível de ativação baseado na entrada efetiva $s_k(t)$ e no valor atual de ativação $y_k(t)$ (i.e., a atualização).
  - Uma entrada externa (bias, offset) θ_k para cada unidade.
  - Um método para o recolhimento de informações (a regra de aprendizagem).
  - Um ambiente dentro do qual o sistema deve operar, fornecendo os sinais de entrada e, se necessário, os sinais de erro.

- **Unidades de Processamento**

  Cada unidade executa um trabalho relativamente simples: recebe entrada de seus vizinhos ou fontes externas e os usa para computar um sinal de saída que será propagado para outras unidades. Além do processamento, uma atividade secundária é o ajuste dos pesos. O sistema é inerentemente paralelo, no sentido que muitas unidades podem realizar seus cálculos ao mesmo tempo. Dentro de uma rede neural, costuma ser útil separar as unidades em 3 tipos básicos: unidades de entrada, unidades intermediárias e unidades de saída.

  - Unidades de Entrada: recebem a entrada de fora da rede.
  - Unidades Intermediárias: recebem os sinais de entrada de outras unidades e mandam os sinais de saída para outras unidades da rede.
  - Unidades de Saída: enviam dados para fora da rede.

Durante a operação de uma rede neural, as unidades podem ser atualizadas de maneira síncrona (todas são atualizadas uma vez) ou assíncrona (uma unidade atualizada por vez).

- **Conexões entre as Unidades**

  Usualmente, é assumido que a entrada de uma unidade é a média ponderada pelos pesos das saídas das unidades conectadas a ela. Ou, a entrada $s_k(t)$ de uma dada unidade k pode ser definida da seguinte forma:

$$s_k(t) = \sum_j w_{jk}(t) y_j(t) + \theta_k(t)$$

  Embora seja a mais usual, esta não é a única forma de se computar a entrada de uma unidade. Feldman e Ballard (Feldman & Ballard, 1982) propuseram a seguinte regra de propagação:

$$s_k(t) = \sum_j w_{jk}(t) pi_m y_{jm}(t) + \theta_k(t)$$

- **Ativação e Regras de Saída**

  Uma função de ativação de uma unidade k, F_k, deve levar em consideração a entrada s_k e o valor atual da saída y_k, produzindo um novo valor de ativação para a unidade k:

$$y_k(t+1) = F_k(y_k(t), s_k(t))$$

  Freqüentemente são usadas funções não decrescentes da entrada total da unidade s_k:

$$y_k(t+1) = F_k(s_k(t)) = F_k(\sum_j w_{jk}(t) y_j(t) + \theta_k(t))$$

  Geralmente algum tipo de função de limiar é utilizada. Em especial, destacaremos a função sigmóide (em forma de S):

$$F_k(s_k) = 1/1 + e^{(-s_k)}$$

- **Topologias de Rede**

  Podemos dividir a topologia da rede, isto é, a forma como as unidades se conectam, de duas formas básicas:

  - "Feed-forward networks": onde o fluxo vai da entrada para a saída de maneira direta. Ex: Perceptron e Adaline.
  - Redes recorrentes que contêm conexões de "feedback", cujo comportamento dinâmico tem importância relevante. Ex: Kohonen e Hopfield.

- **Treinamento de Redes Neurais Artificiais**

  Uma rede neural tem que ser configurada de forma tal que, a aplicação de um conjunto de entradas produz (de maneira direta ou através de um processo de relaxação) o conjunto de saídas desejado. Há várias maneiras de reforçar o peso das conexões. Uma maneira, é modificar o valor dos pesos explicitamente, usando conhecimentos a priori. Outro modo é "treinar" a rede neural, alimentando-a com "padrões de ensino", deixando-a mudar os pesos de acordo com alguma regra de aprendizagem.

- **Paradigmas de Aprendizado**

  Nós podemos categorizar as situações de aprendizado em dois tipos distintos:

  - Aprendizado Supervisionado ou Aprendizado Associativo - a rede é treinada tendo como informação uma entrada e a respectiva saída desejada.
  - Aprendizado não supervisionado ou Auto-organização - as unidades de saída são treinadas para responder a "clusters" de padrões dentro da entrada. Neste paradigma o sistema supostamente descobre estatísticamente características da população da entrada.

- **Modificando o padrão de conectividade**

  Ambas as técnicas discutidas acima resultam no ajuste dos pesos das conexões entre as unidades, de acordo com uma regra de modificação. Virtualmente todas as regras de aprendizado para modelos desse tipo podem ser considerados uma variante da regra sugerida por Hebb. A idéia básica é que se duas unidades j e k estão ativas simultâneamente, então, a conexão entre elas deve ser reforçada. Em sua versão mais simples, o ajuste de peso será:

  $$\Delta w_{jk} = \gamma y_j y_k$$

  onde γ é uma constante positiva de proporcionalidade representando a taxa de aprendizado.

  Uma outra regra comum não usa o verdadeiro valor de ativação da unidade k, mas sim a diferença entre o valor verdadeiro e o valor desejado para ajustar os pesos:

  $$\Delta w_{jk} = \gamma y_j (d_k - y_k)$$

  onde d_k é o valor desejado para a saida k. Esta é freqüentemente chamada de regra de Widrow-Hoff ou regra do delta (*delta rule*).

- **Implementação das Redes Neurais**

  Embora existam arcabouços já desenvolvidos, disponíveis para a utilização que implementem redes neurais, optamos por desenvolver a nosssa própria implementação. Inicialmente, desenvolvemos os elementos básicos constituintes de um arcabouço de redes neurais descritos acima. O tipo de rede neural escolhido para fazer o reconhecimento foi a Perceptron multi-camadas.

- **Rede Perceptron multi-camadas**

  Uma rede perceptron multi-camadas tem as seguintes características:

  - Contém uma camada de unidades de entrada
  - Contém uma camada de unidades de saída
  - Contém uma ou mais unidades intermediárias: Uma dada unidade de uma camada se conecta com todas as unidades da camada seguinte, e somente com essas unidades
  - A unidade de entrada não realiza cálculos, apenas passa para as outras camadas os valores das entradas.

  A figura 4 exemplifica a topologia de uma rede perceptron multi-camadas.

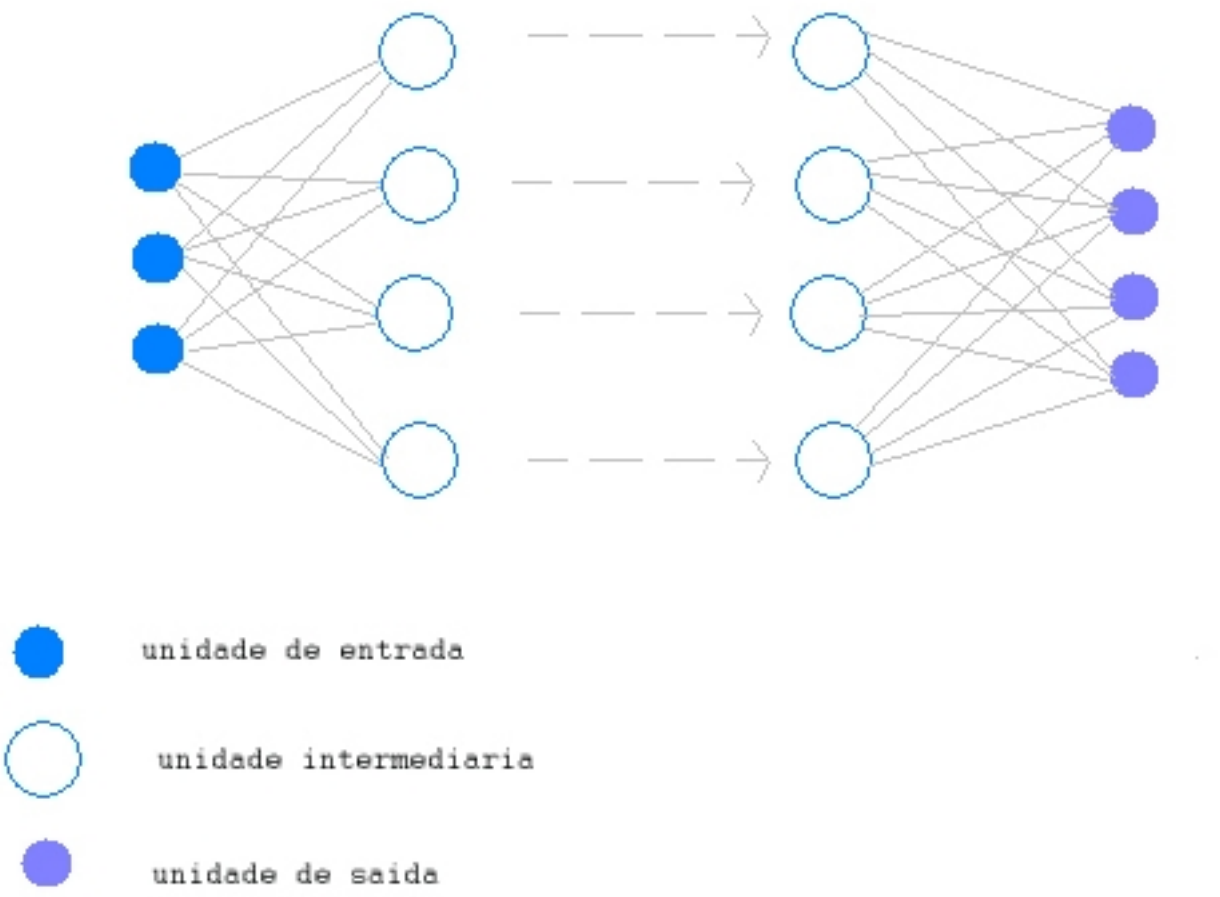

Figura 4: Rede Perceptron multi-camadas

- **Ajuste dos Pesos e poder de expressão**

  De acordo com o Teorema da Aproximação Universal, uma rede perceptron multi - camadas com funções de ativação não - lineares e pelo menos uma camada intermediária pode representar qualquer função não linear (ou classificador, conforme o desejado no nosso caso). Por isso, utilizamos uma rede com uma camada de entrada, uma camada intermediária e uma camada de saída. Sendo que as unidades da camada intermediária utilizam a função sigmoide descrita anteriormente. Já as unidades da camada de saída utilizam uma função de ativação linear da forma $y = ax + b$.

  Para ajustarmos os pesos, utilizamos a regra do gradiente, que é mais uma generalização da regra do delta descrita anteriormente.

  - ajuste:

    $$\Delta w_{jk} = \gamma \delta k y_j$$

  - para as unidades de saida:

    $$\delta o = (d_o - y_o) F'(s_o)$$

  - para as intermediárias ligadas com intermediárias :

    $$\delta_h = F'(s_h) \sum_{o=1}^{N_o} \delta_o w_{hh+1}$$

  - para as intermediárias ligadas com a camada de saída:

    $$\delta_h = F'(s_h) \sum_{o=1}^{N_o} \delta_o w_{ho}$$

  - Na implementação, adicionamos um termo de "momentum" à taxa de atualização dos pesos:

    $$\Delta w_{jk}(t+1) = \gamma \delta_k y_j + \alpha \Delta w_{jk}(t)$$

  A razão para a utilização de um termo de "momentum" é que embora a regra do gradiente tenha convergência linear, resultados empíricos mostram que a adição desse termo aumenta a eficiência desse método, pois de certa forma o passo anterior acaba aproveitando informações sobre a curvatura da função.

  Existem outros métodos como o do gradiente conjugado, que têm convergência super-linear teoricamente comprovadas. Não o implementamos por simplicidade.

Seja E_i o erro cometido entre o ponto atual e o ponto de ótimo Um método converge linearmente quando $E_{i+1} = cE_i$, com $c < 1$. Dizemos que um método tem convergência superlinear se $E_{i+1} = c(E_i)^m$, para $m > 1$.

- **Utilizando redes neurais no reconhecimento dos símbolos**

  Como visto, uma rede neural artificial é um classificador de padrões. E um símbolo pode ser entendido como um padrão a ser reconhecido. Para tanto, precisamos tornar o sinal gráfico inteligível para a rede neural, além de ser necessário associar a esses símbolos, uma saída que os identifiquem univocamente. Abaixo mostraremos como fizemos as etapas que acabamos de descrever.

  Para que pudessemos tornar a entrada da rede neural consistente tivemos que fazer as seguintes considerações: Símbolos podem ter diferentes tamanhos, portanto de alguma forma deveriam ser "moldados" em um determinado formato; Sinais gráficos são, usualmente, sensíveis à escala; Como a rede tem uma quantidade de unidade de entradas fixa, deveríamos extrair de alguma forma a mesma quantidade de informação de cada símbolo.

  A solução que encontramos foi particionar a caixa envolvente dos símbolos em uma grade conforme mostra as figuras 5 e 6.

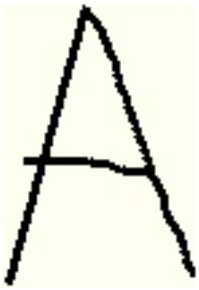

Figura 5: Símbolo inicial

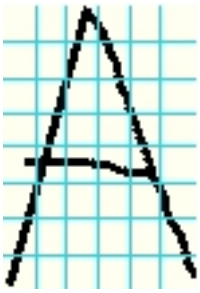

Figura 6: Símbolo com a grade

A partir daí ordenamos a grade da esquerda para a direita, de cima para baixo, de forma que se uma célula contém um ponto ela representará uma entrada 1 na rede, -1 caso contrário. Destacamos que esta solução resolve os problemas que acabamos de expor.

Durante o período de treinamento, passamos o valor de saída que obtemos a partir de índices associados aos caracteres. O vetor de saída, basicamente consiste da representação binária do valor desse determinado índice. Então a partir dos exemplos de um mesmo símbolo, passados durante o período de treinamento, associamos essa rede já treinada com um caractere. Na posterior fase de reconhecimento, apos a leitura do sinal, verificamos de qual caractere esse sinal mais se aproxima (isto é, de qual rede ele se aproxima mais da saída obtida no treinamento) e o identificamos como aquele símbolo.

- **Resultados**
  Os resultados obtidos na aplicação se mostraram bastante interessantes, embora não tenhamos feito uma análise quantitativa da capacidade de reconhecimento. Destacamos aqui, que as redes neurais são apenas uma implementação para utilizar essa funcionalidade específica do arcabouço. Considerações teóricas ainda podem ser feitas sobre a suscetibilidade das redes neurais a mínimos locais. Máquinas de Suporte vetorial, provavelmente, seria a opção mais imediata, utilizando um método de treinamento, pela menor suscetibilidade da última com relação a primeira.

3.3 Reconhecimento de Expressões

Como a intenção inicial do projeto era criar um framework que pudesse ser utilizado para auxiliar na criação de programas de reconhecimento de símbolos e expressões, não era apenas necessário reconhecer um símbolo isolado, além disso, precisávamos ter alguma informação sobre como estes símbolos se relacionavam, pois dessa maneira teríamos uma expressão inteira reconhecida. Para que fosse possível reconhecer de forma correta e simples as expressões, vários textos foram estudados, verificando-se quais algoritmos seriam mais apropriados para este fim.

Para a resolução do problema, várias premissas foram assumidas sobre a estrutura das expressões que seriam reconhecidas, explicadas abaixo:

Um símbolo estará relacionado diretamente com no máximo cinco símbolos, sendo que estes símbolos podem estar relacionados com mais cinco, e assim por diante. Estes cinco símbolos são: A cima, abaixo, a frente, a frente e a cima (como um expoente), a frente e abaixo (como um índice). As figuras 7 e 8 ilustram estas idéias:

Para saber em que posição um símbolo se encontra em relação ao outro simplesmente faz-se uma análise das posições entre os *bounding boxes* de ambos. Assim definíamos facilmente se um símbolo está acima, abaixo, a frente, etc. Assumindo isso, agora era necessário implementar um algoritmo que pudesse reconhecer uma expressão.

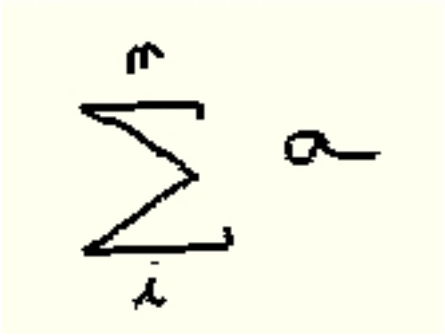

Figura 7: Uma somatória, com um i abaixo, um n acima e um a a frente.

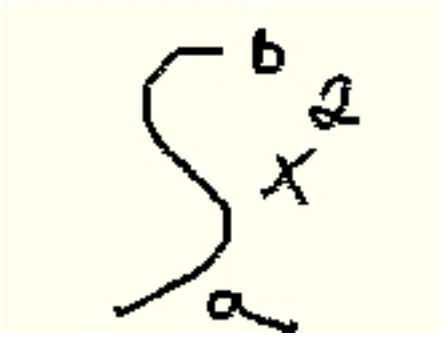

Figura 8: Uma integral com um a a frente e abaixo, um b a frente e acima, e com um x a frente. O x possui um 2 acima.

3.3.1 Estudo de Técnicas e Implementação dos Algoritmos

Foram estudadas algumas técnicas, e devido aos benefícios de ambas, foi utilizada uma mistura que serão explicadas a seguir:

- **Primeira técnica utilizada: Reta de regressão linear**

  Obviamente, antes de se achar índices, expoentes, intervalos de uma integral ou de uma somatória é necessário se encontrar os símbolos centrais da expressão, como mostrado na figura 9.

  Para isso, foi necessário calcular a reta de regressão linear, utilizando a técnica de mínimos quadrados, de todos os pontos dos símbolos que formam a expressão, porque como a reta de regressão linear é a reta que melhor aproxima todos os pontos, esta tende a cruzar os *bounding boxes* dos símbolos centrais. Para achar esta reta, primeiramente montamos algumas matrizes da seguinte maneira: montamos uma matriz A, $m \times 2$, onde m é o número de pontos da expressão, e a primeira coluna é preenchida com 1, enquanto a segunda é preenchida com todos os x dos pontos, e um vetor, $m \times 1$, com os y. Logo, chamando o vetor de coeficientes da reta de b, onde $b = (c, d)^T$ temos que, se $|Ab - Y|$ é o

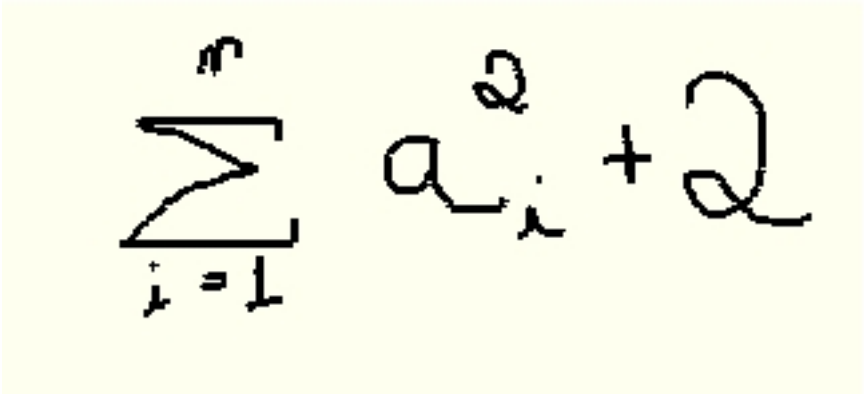

Figura 9: Expressão cujos símbolos principais são a somatória, o a e o último 2.

menor possível, b são os coeficientes da reta que melhor aproxima os pontos da expressão. Logo fazemos o seguinte.

$$A^T Ab = A^T Y \Rightarrow (A^T A)^{-1}(A^T A)b = (A^T A)^{-1}A^T Y \Rightarrow b = (A^T A)^{-1}A^T Y$$

Deste modo, a reta $dx + c$ é a reta que melhor aproxima os pontos da expressão. A partir deste momento, é necessário determinar os símbolos centrais desta expressão utilizando a reta obtida, e isso é feito da seguinte maneira:

1) Achar o símbolo mais a esquerda que tem seu *bounding box* se interceptando com a reta. Este símbolo será considerado o primeiro símbolo da expressão, e será incluído na lista de símbolos centrais da expressão.

2) Caso haja outros símbolos interceptando a reta, verificamos se ele se encontra a frente do último símbolo adicionado a lista de símbolos centrais a expressão, caso isso ocorra este também será adicionado a lista de símbolos centrais da expressão, e será também adicionado no link *front* do símbolo anterior.

Ao final deste processo todos os símbolos centrais da expressão estão interligados, mas ainda era necessário obter os símbolos não centrais, como índices e expoentes, deste modo uma outra técnica foi utilizada.

- **Segunda técnica utilizada: Grafos e árvores geradoras mínimas.**

Observa-se que em uma expressão, símbolos que estão diretamente relacionados normalmente estão próximos, como pode ser observado na figura 11:

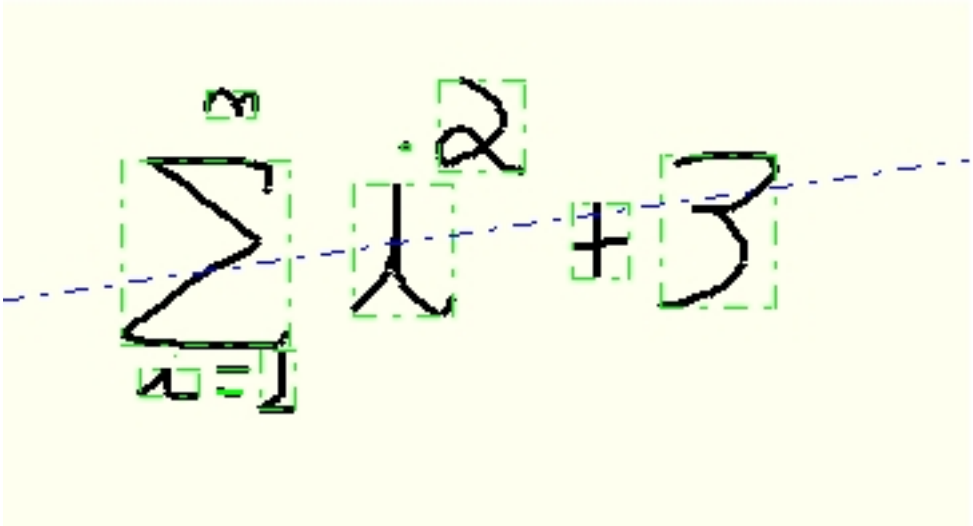

Figura 10: Expressão com sua reta de regressão linear em azul. Vemos que os símbolos centrais como a somatória, o i e o 3 cruzam a reta e encontram-se um a frente do outro. Logo, neste caso, a somatória seria o primeiro símbolo, com o link *front* sendo o i. O i teria como *front* o $+$, e o $+$ teria como *front* o 3.

Observado isso, era preciso se escolher uma técnica que ajudasse a escolher de melhor forma símbolos próximos entre si. Para isso foi utilizado um grafo, e sua árvore geradora mínima, como será descrito a seguir:

Foi utilizado um grafo completo, onde os centros dos *bounding boxes* de cada símbolo seriam considerados os vértices destes grafos, e as arestas do grafo teriam peso igual a distância entre seus vértices.

Como a intenção era usar este grafo para obter os símbolos mais próximos entre si, apenas um grafo completo não ajuda muito. Ainda era preciso processar este grafo para obter novas informações. Estas informações foram retiradas de uma árvore geradora mínima do grafo. Uma árvore geradora consiste em um conjunto de $n - 1$ arestas que ligam todos os vértices de um grafo, e uma árvore geradora mínima e uma árvore geradora onde as somas dos pesos das arestas é a menor possível entre todas as árvores geradoras possíveis. Logo, com uma árvore geradora mínima temos que os símbolos mais próximos entre si estão ligados através de arestas desta árvore, com isso podemos obter expoentes, índice, etc.

Para achar a árvore geradora mínima foi utilizado o algoritmo de Kruskal, utilizando uma estrutura de UNION- FIND. No inicio, houve dúvida se iríamos utilizar o algoritmos de Kruskal ou o algoritmo de Prim, já que em tese ambos têm $O(E \log V)$, onde E é o número de arestas, e V o número de vértices. Mas, dependendo da implementação

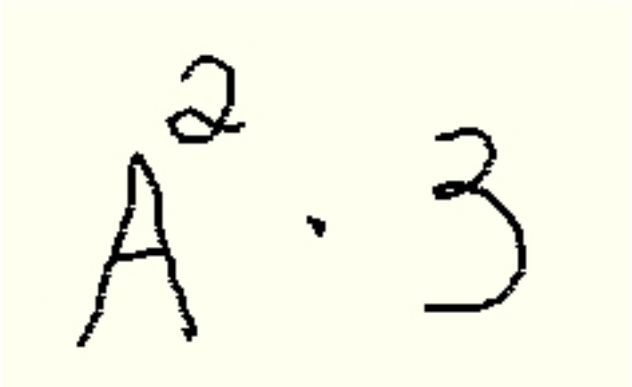

Figura 11: Expressão com um expoente 2 que se relaciona diretamente com o símbolo A, por estar mais próximo deste símbolo do que do símbolo '.', ou do símbolo 3.

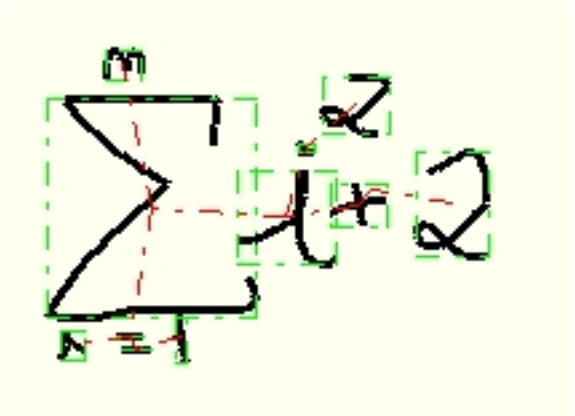

Figura 12: Expressão com as arestas da árvore geradora mínima do grafo em vermelho.

de Prim podemos chegar a $O(E + V \log V)$, o que o torna mais eficiente para grafos completos, que é o nosso caso. Apesar disso, o número de vértices do nosso grafo tende a ser pequeno, o que torna esta diferença irrelevante. Por isso optamos por utilizar Kruskal, pela sua simplicidade.

- **Fusão das duas técnicas: Por quê?**

  As duas técnicas descritas acima são usadas em conjunto. Aparentemente, apenas o grafo já seria o suficiente para obter as informações necessárias. A reta de regressão linear poderia ser utilizada somente para achar o primeiro símbolo, usando o grafo para solucionar o resto do problema. No começo foi exatamente assim que implementamos, mas não demorou muito para percebermos que esta técnica não era completamente correta, e que apresentava alguns erros. Símbolos que a princípio não têm nenhuma relação entre si são ligadas por arestas, como mostrado nas figuras 13 e 14.

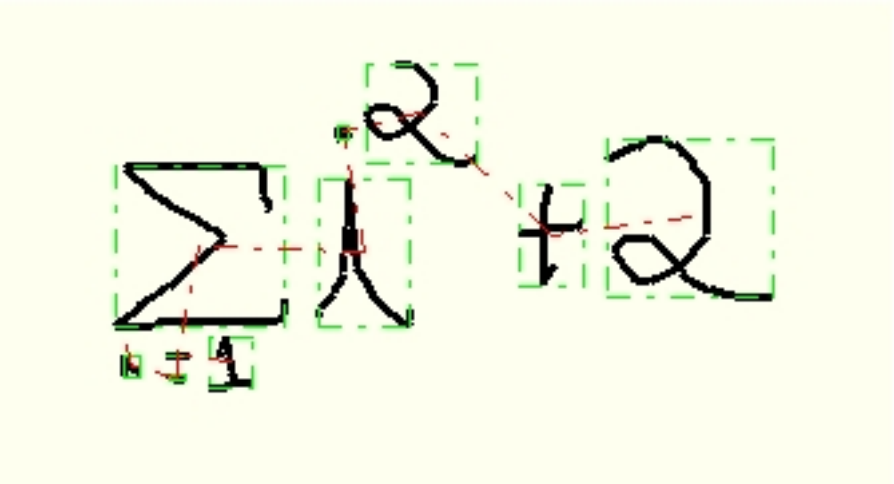

Figura 13: Expressão onde o expoente 2 se liga com o + ao invés do i.

É por isso que a reta de regressão linear não é utilizada somente para achar os primeiros símbolos, mas também para ligar os símbolos que a cruzam. Deste modo, na figura acima, o i estará ligado com o +, e as duas somatórias estarão ligadas. Mas ainda é necessário saber como podemos excluir as ligações adicionais, como a do expoente 2 com o +, e a do 1 com o j.

Para eliminar estes links, utilizamos a seguinte técnica:

1) Ordene os símbolos pelo ponto mais à direita do *bounding box*.
2) Consideramos dois símbolos A e B, tal que A $<$ B de acordo com o critério de ordenação, que sejam cruzados pela reta de regressão linear
3) Se B estiver à frente de A , então adicionamos B como link à frente de A, caso já não o seja.
4) Depois, removemos todos os links entre símbolos em que o ponto mais à direita de seu *bounding box* está à esquerda do ponto mais à esquerda do *bounding box* de B e estão ligadas a símbolos em que o ponto mais à esquerda do *bounding box* está à direita do ponto mais à esquerda do *bounding box* de B.

- **Fusão de símbolos**

Da maneira como isolamos os símbolos, nunca um "igual" seria considerado um símbolo só. Ele seria formado por dois símbolos, que provavelmente seriam reconhecidos como dois sinais de menos (−), um acima do outro. O mesmo ocorre com o i e o j, por exemplo, com os seus pingos. Para resolver este problema, mais uma vez utilizamos a árvore geradora mínima do grafo descrito acima. Apenas procuramos na lista de símbolos da expressão um sinal de menos ou um pingo. Se acharmos um menos, verificamos se há uma aresta ligando este menos com um outro menos abaixo dele. Em caso positivo,

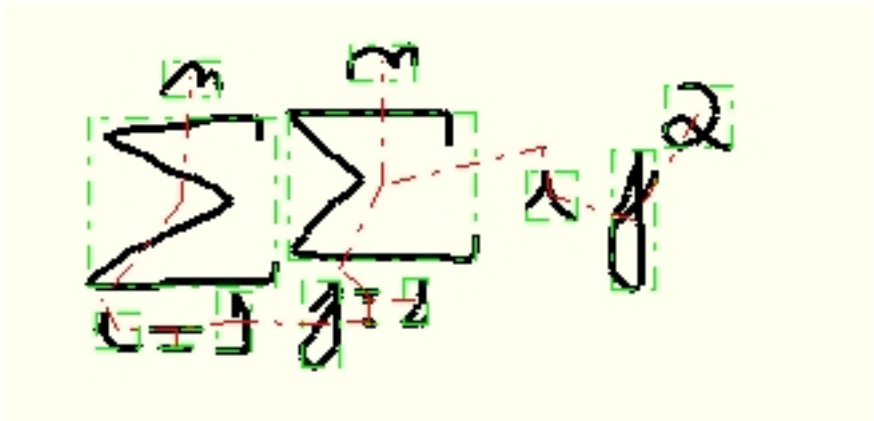

Figura 14: Expressão com duas somatórias que não estão ligadas e o 1 está ligado com o j nas expressões abaixo da somatória.

fundimos os dois símbolos em um só. O mesmo é feito com o pingo. Se acharmos o corpo do i, ou o corpo do j abaixo dele, ligados por uma aresta, eles são fundidos num mesmo símbolo. Um exemplo está na figura 15.

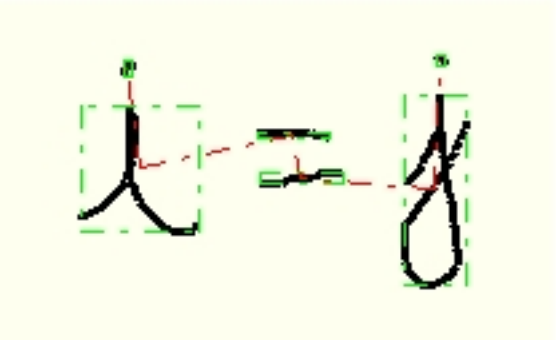

Figura 15: Expressão onde o pingo do i está ligado com o corpo do i, assim como o j. Dois sinais $-$ são ligados formando na verdade um $=$.

- **Resultados**

  No inicio obtemos resultados não tão satisfatórios, mas com a melhoria e adição de novas técnicas para revolver os problemas que foram verificados no processo de implementação chegamos a um resultado satisfatório, que ainda erra em algumas situações, mas que na maioria das vezes consegue acertar. Casos específicos podem sempre ser resolvidos pontualmente como nos casos do sinal de igual e nos pingos do i e do j, através de uma técnica específica para eles.

4 O Arcabouço desenvolvido

Optamos por desenvolver o arcabouço em Java pela portabilidade, que era uma premissa básica do nosso projeto. Os elementos constituintes da estrutura da expressão foram implementados, basicamente, seguindo a descrição de requisitos da sessão 2, e estão dispostos da seguinte forma:

- **StrokePoint**: implementação de um JavaBean da definição
- **Stroke**: classe que contém uma lista de StrokePoints
- **Character**: classe que representa uma letra do alfabeto de símbolos reconhecíveis
- **CharacterSet**: abstração que representa o alfabeto de caracteres reconhecíveis
- **Symbol**: conjunto de Strokes que pode ser associado a um Character
- **SymbolRecognizer**: interface que contém o método recognize, que recebe um Symbol como parâmetro e devolve um Character
- **Graph**: componente que contém a representação estrutural dos elementos da expressão. Implementa grande parte das técnicas descritas acima, para obtenção de informação estrutural.
- **Expression**: representa a expressão propriamente dita. De certa maneira, encapsula todos os elementos anteriores.

Além disso, o arcabouço disponibiliza ferramentas básicas para a construção de uma interface gráfica para softwares de reconhecimento de escrita *online*. Também incluímos a nossa implementação de reconhecimento de símbolos que utiliza redes neurais no arcabouço.

Inicialmente tínhamos em mente o desenvolvimento de um arcabouço abrangente o bastante para ser útil, em sua integridade, para aplicações de diversos domínios (ex: reconhecimento de expressões matemáticas e de diagramas de blocos). O sistema desenvolvido conseguiu utilizar a arquitetura planejada para solucionar um problema de reconhecimento de expressões matemáticas de forma simples, usando técnicas comuns.

4.1 O SisTREO

Para testarmos o arcabouço desenvolvido e atestarmos a sua utilidade, resolvemos desenvolver um aplicativo de reconhecimento de expressões matemáticas capaz de gerar o LaTeX da expressão correspondente.

O aplicativo é dividido em 3 modos:

- **Modo Treinamento**
  Modo de treinamento das redes neurais, onde dados iniciais de um usuário podem ser gravados para serem utilizados no reconhecimento.

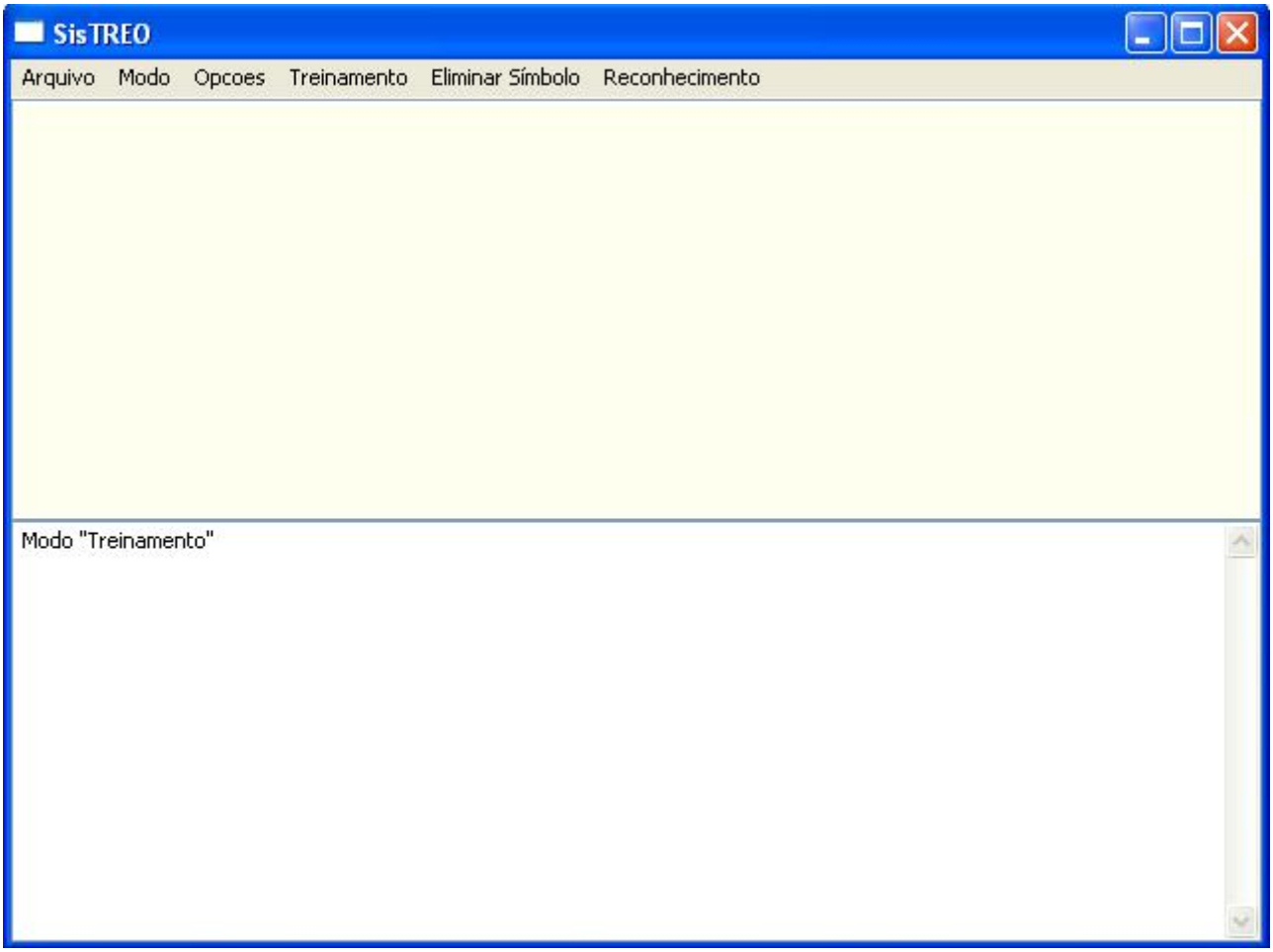

Figura 16: Tela do SisTREO em Modo Treinamento.

- **Modo Reconhecimento**
  Modo onde o usuário escreve a expressão e o programa reconhece e exibe a expressão LaTeX reconhecida.

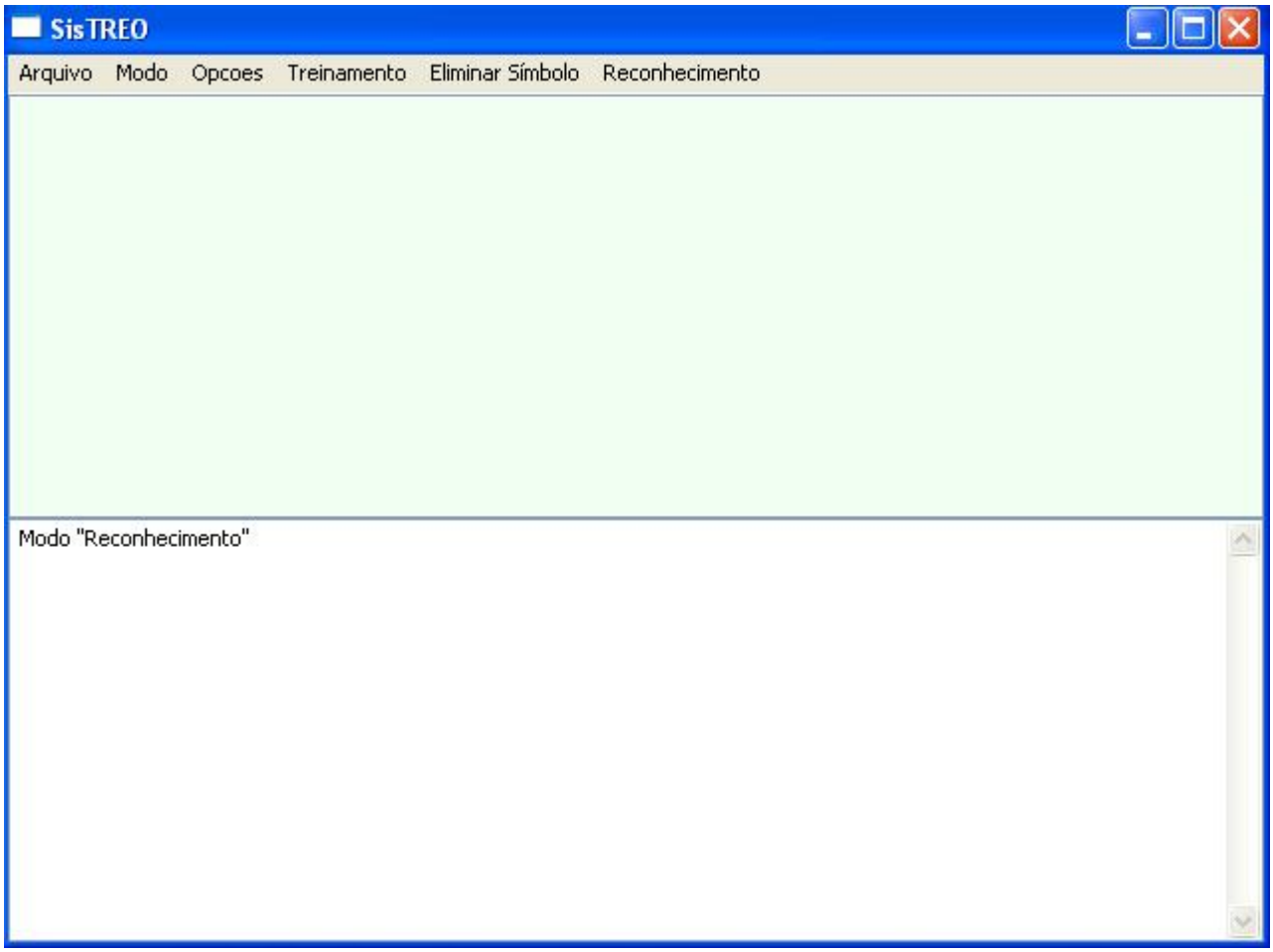

Figura 17: Tela do SisTREO em Modo Reconhecimento.

- **Modo Escrita**
  Modo posterior ao reconhecimento, onde o usuário insere a expressão LaTeX gerada num documento.

No sistema, ainda é possível a correção de erros. Com a eliminação de símbolos ativada, pode-se remover símbolos ao traçar uma reta sobre eles. Com isso permite-se que o usuário possa escrever somente aquele símbolo novamente, de uma forma mais clara que possa ser reconhecida.

5 Resultados Obtidos

Com o SisTREO (Sistema Titanium de Reconhecimento de Escrita Online), nossa intenção de implementar um sistema baseado numa arquitetura de um framework genérico para reconhecimento de escrita *online* foi bem sucedida. A aplicação reconhece expressões matemáticas

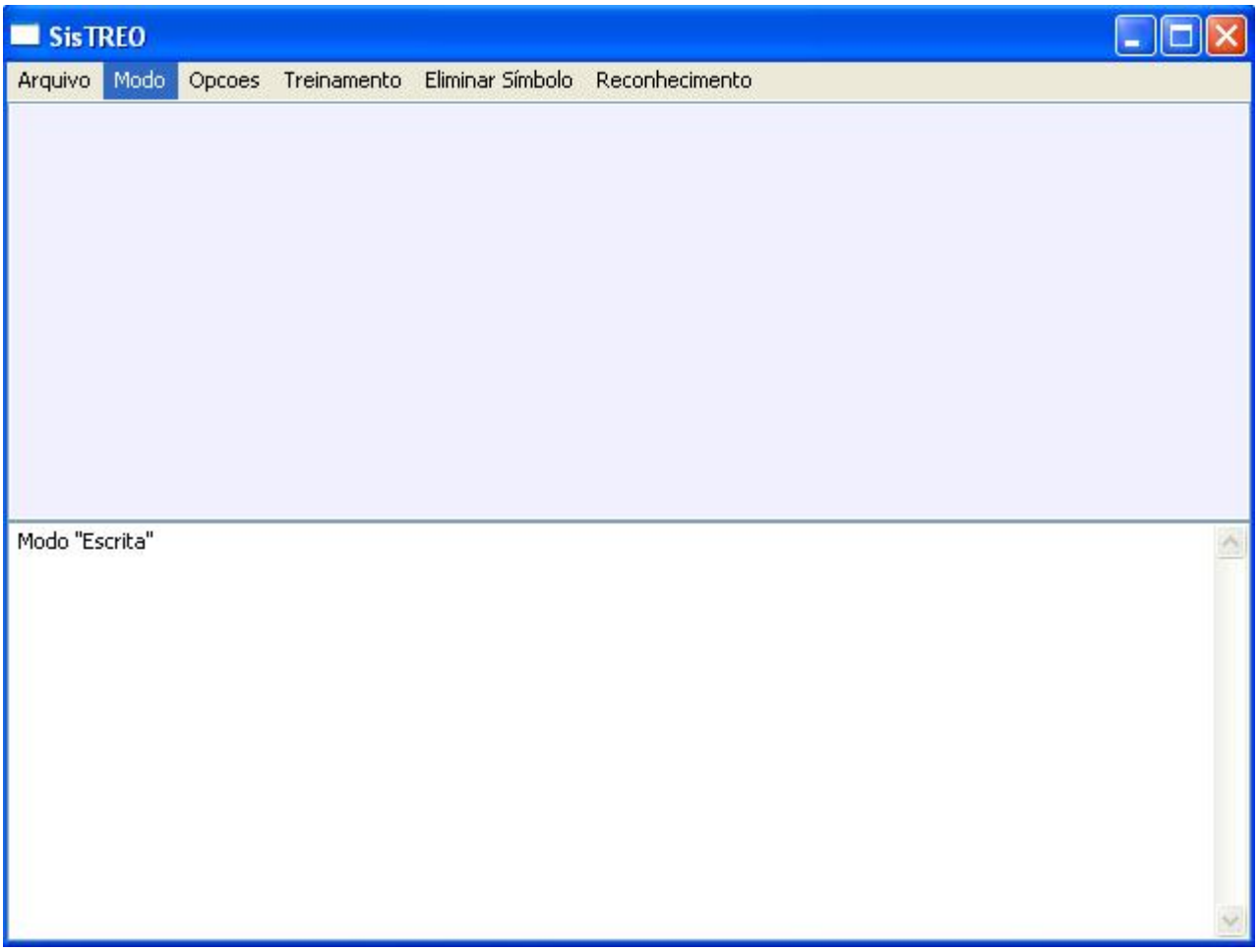

Figura 18: Tela do SisTREO em Modo Escrita.

escritas à mão e cria um código em LaTeX. Com ele, cada usuário pode treinar o programa com sua própria letra. Depois de treinar o programa, o usuário escreve a expressão e o sistema é capaz de gerar o código em LaTeX da expressão.

As figuras 19 e 20 ilustram a utilização do sistema:

Utilizando o código gerado na figura 19, temos a seguinte expressão:

$$\sum_{2}^{y} 3X^{6}$$

E o código gerado a partir da figura 20 gera a seguinte expressão:

$$\int_{3}^{7} X + 2$$

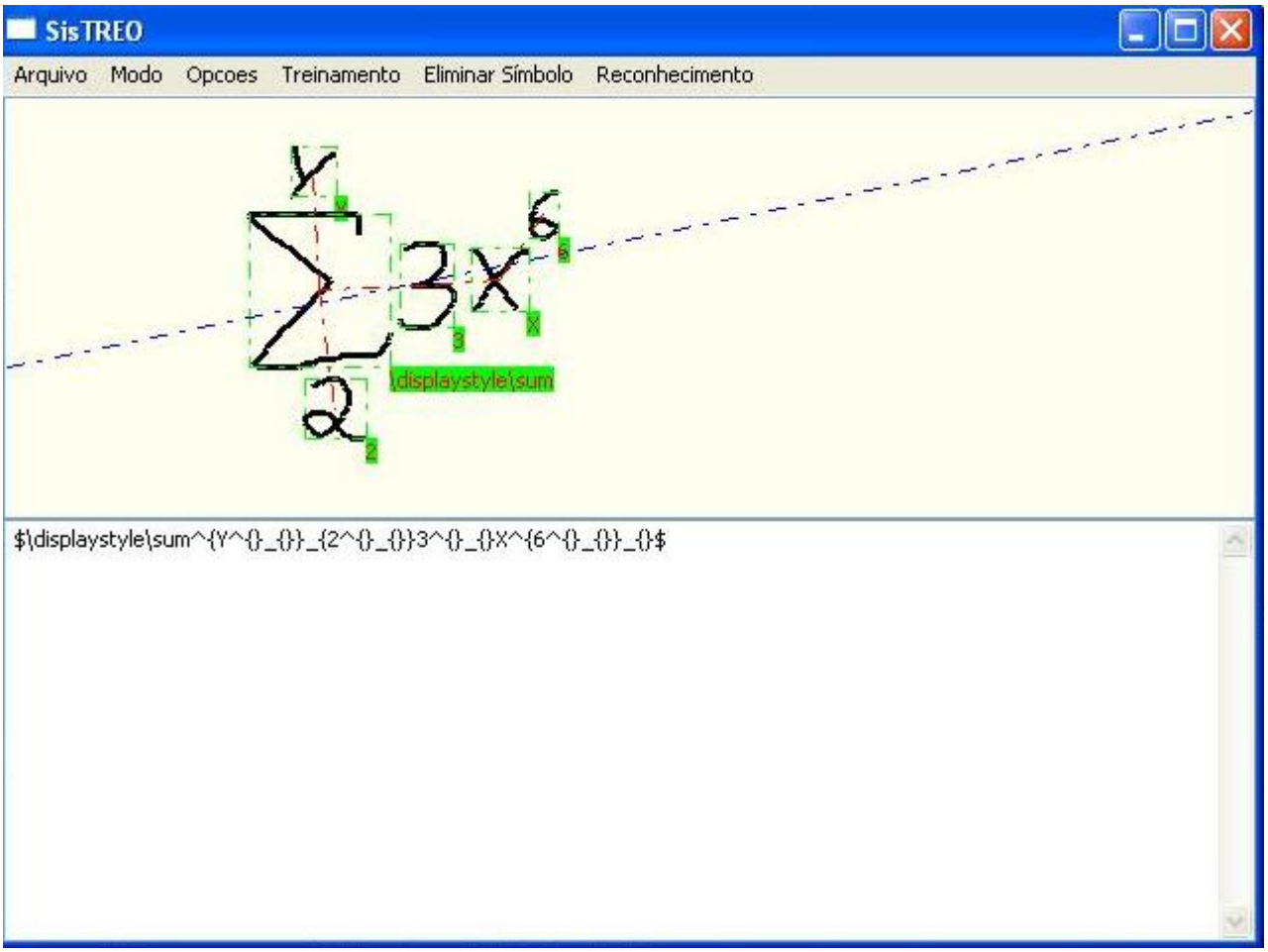

Figura 19: Tela do SisTREO com uma expressão reconhecida e o seu código em LaTeX.

6 Conclusões

O índice de acerto do reconhecimento, tanto da expressão como dos símbolos, não foi medida por não ser nosso objeto de estudo. Apesar disso, a partir do que foi observado em outros artigos e teses, podemos afirmar que se aumentarmos a resolução da rede neural já teríamos uma grande melhora. No caso da expressão, já estamos desenvolvendo um algoritmo que não se baseia somente na árvore geradora mínima do grafo.

A arquitetura do arcabouço foi testada com sucesso, utilizando-a como base para desenvolver o SisTREO. Muitas técnicas vistas nos artigos estudados poderiam se utilizar da estrutura elaborada, com a vantagem de se tratar de uma especificação abrangente, que possa trazer compatibilidade entre técnicas de diversas fontes. A nossa intenção futura é disponibilizar o arcabouço como base para próximos projetos, desta vez focada nos algoritmos e na resolução de problemas mais difíceis. E depois, a combinação desses projetos seria uma etapa que finalizaria os testes da estrutura.

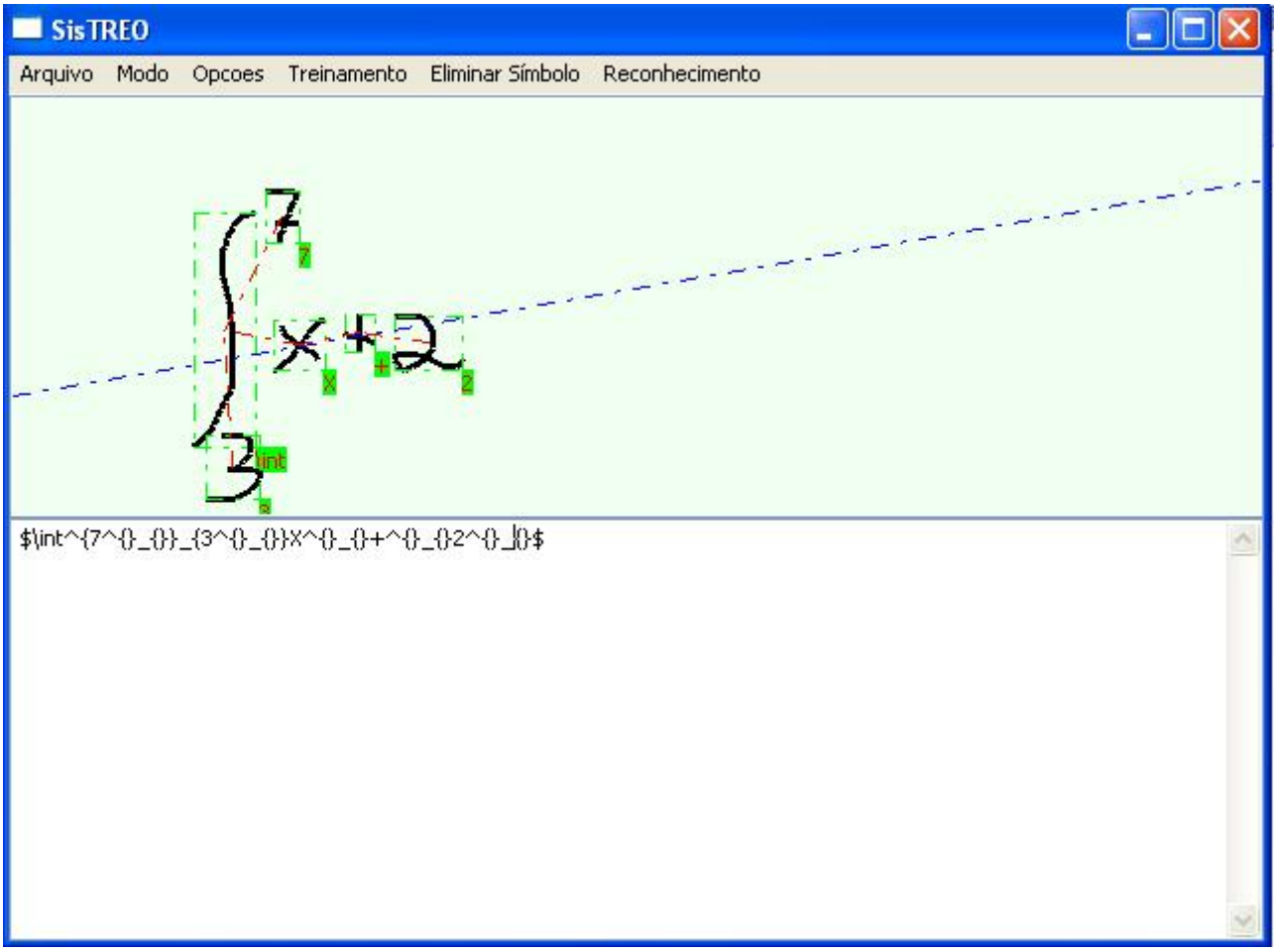

Figura 20: Tela do SisTREO com uma expressão reconhecida e o seu código em LaTeX.

Além de termos como resultado a arquitetura e o sistema, com o projeto conseguimos ter uma visão bem ampla de como o reconhecimento de escrita se situa atualmente. Todas as técnicas vistas possuem pontos fortes e fracos. Uns possuem restrições mas conseguem altas taxas de acerto. Outros são bem genéricos, mas têm problemas com erros de reconhecimento e/ou com desempenho. O futuro dessa área depende muito da interação e integração de técnicas para que o uso do computador se torne cada vez mais natural, usando a escrita humana como forma de entrada de dados. Soluções específicas para certas plataformas e aplicações existem em vários lugares, mas ainda estamos longe de uma solução final para reconhecermos plenamente o que escrevemos.

Referências

[Chan . Yeung 2000] Chan, K.-F. und Yeung, D.-Y. (2000). *Mathematical expression recognition: a survey*.

[Fitzgerald .] Fitzgerald, J. A., Geiselbrechtinger, F., und Kechadi, T. *Structural Analysis of Handwritten Mathematical Expressions Through Fuzzy Parsing*.

[Garain . Chaudhuri 2003] Garain, U. und Chaudhuri, B. B. (2003). *Recognition of Online Handwritten Mathematical Expressions*.

[Ishigaki . 1999] Ishigaki, K., Tanaka, H., und Iwayama, N. (1999). *Interactive Character Recognition Technology for Pen-based Computers*.

[Jaeger . 2000] Jaeger, S., Manke, S., Reichert, J., und Waibel, A. (2000). *On-Line Handwriting Recognition: The NPen++ Recognizer*.

[Kara 2004] Kara, L. B. (2004). *Automatic Parsing and Recognition of Hand-drawn Sketches for Pen-based Computer Interfaces*. ., Carnegie Mellon University, Pittsburgh, PA.

[Kosmala . Rigoll 1998] Kosmala, A. und Rigoll, G. (1998). *Recognition of On-Line Handwritten Formulas*. . *6th Int. Workshop on Frontiers in Handwriting Recognition (IWFHR)*, Taejon, Korea.

[Kröse . van der Smagt 1996] Kröse, B. und van der Smagt, P. (1996). *An Introduction to Neural Networks*. Universidade de Amsterdam: 8th Edition .

[Matsakis 1999] Matsakis, N. E. (1999). *Recognition of Handwritten Mathematical Expressions*.

[Miller . Viola 1998] Miller, E. G. und Viola, P. A. (1998). *Ambiguity and Constraint in Mathematical Expression Recognition*.

[Oetiker . 2006] Oetiker, T., Partl, H., Hyna, I., und Schlegl, E. (2006). *The Not So Short Introduction to LaTeX2e*.

[Plamondon . Srihari 2000] Plamondon, R. und Srihari, S. N. (2000). *On-Line and Off-Line Handwriting Recognition: A Comprehensive Survey*.

[Smirnova . Watt] Smirnova, E. und Watt, S. M. *A Context for Pen-Based Mathematical Computing*.

[Smithies .] Smithies, S., Novins, K., und Arvo, J. *A Handwriting-Based Equation Editor*.

[Tapia . Rojas] Tapia, E. und Rojas, R. *Recognition of On-line Handwritten Mathematical Formulas in the E-Chalk System*.

[Tappert . 1990] Tappert, C. C., Suen, C. Y., und Wakahara, T. (1990). *The State of the Art in On-Line Handwriting Recognition*.

[Zanibbi 2000] Zanibbi, R. (2000). *Recognition of Mathematical Notation via Computer Using Baseline Structure*.

[Zanibbi . 2002] Zanibbi, R., Blostein, D., und Cordy, J. R. (2002). *Recognizing Mathematical Expressions Using Tree Transformation*.

7 Desafios e Frustrações

Meu projeto foi desenvolvido em grupo, com meus colegas Pedro Henrique Simões de Oliveira e Eduardo Gusmão Caceres Pires. A escolha do nosso grupo deve-se ao fato de já termos trabalhado juntos durante os três anos anteriores do curso. Com o Pedro e mais um outro colega nosso, o Luís, fizemos nossos projetos em Laboratório de Programação I e II, durante um ano inteiro. Já em Engenharia de Software, meu grupo de seis pessoas continham não só eu e o Pedro, mas também o Eduardo.

Durante o início do projeto, mantivemos reuniões a cada duas semanas com nossa orientadora a fim de esclarecer dúvidas relacionadas ao problema, que era uma novidade para todos nós, pois não tivemos quase nenhum contato com estudos desta área. A falta de conhecimento da área foi a principal dificuldade no início.

Na medida em que a gente foi se familiarizando com o assunto, lendo artigos e discutindo com a professora, começamos a pensar em como queríamos que o nosso projeto final ficasse. Dois pontos importantes a serem atacados foram:

- Como a arquitetura do arcabouço poderia ser utilizável em diversas situações?
- Como resolver o problema do reconhecimento para gerar LaTeX?

O Pedro tinha conhecimentos sobre redes neurais, e isso facilitou na parte de desenvolvimento, que resolvia grande parte do segundo ponto. Devido às circunstâncias (estávamos no fim do primeiro semestre, e, por causa de provas de recuperação, acabei tendo mais tempo do que o Pedro e o Eduardo), eu comecei estudando e lendo grande parte dos artigos selecionados pela nossa orientadora, a professora Nina Hirata. Com isso, conseguimos levantar os requisitos necessários em grande parte das abordagens empregadas. A idéia de generalidade do arcabouço a princípio parecia algo muito vago, pois não deixava quase nada definido.

Apesar de termos demorado a começar a nossa implementação, o tempo empregado em planejamento ajudou e facilitou a segunda parte do trabalho. Mesmo durante a implementação foram surgindo novas idéias que poderiam ajudar a resolver o problema de reconhecimento. Isso de certa forma mostrou que o arcabouço servia para suportar mudanças e novas idéias de uma forma flexível. Acho que uma das frustações que tivemos foi que imaginamos algo muito elaborado, que realmente tivesse cara de um software comercial. No final apenas tivemos tempo de fazer uma implementação simples, mas que pelo menos conseguiu mostrar a principal finalidade do projeto.

Outro grande desafio encontrado por nós foi que eu comecei a estagiar no início do ano e, ao longo do ano, acabamos todos nós em algum estágio. Além disso, cada um de nós tínhamos um cronograma para terminar o curso, com matérias totalmente diferentes, ou seja, não nos encontrávamos nem durante o curso. Era um desafio enorme conciliar um horário que fosse viável para todos nós, e até para a professora, já que as nossas reuniões eram importantes. Essa falta de encontros entre nós acabou impossibilitando que pudéssemos programar pareadamente.

Assim, o projeto foi dividido e muitas vezes um não ficou sabendo da parte que o outro membro fez. Apesar destes problemas, o projeto final teve um resultado que nos agradou como um todo, bem como a nossa orientadora e vários outros alunos.

8 O Trabalho de Formatura e o BCC

O curso influenciou bastante o andamento do nosso projeto em diversos aspectos.

- MAC 110 - Introdução à Computação: Como matéria introdutória, serviu como base para todos os conceitos aprendidos e que foram utilizados no projeto.
- MAC 122 - Princípios de Desenvolvimento de Algoritmos: Nessa disciplina vimos muitos algoritmos e técnicas que indiretamente influenciaram o modo como planejamos a estrutura do arcabouço.
- MAC 211 - Laboratório de Programação I: Foi a primeira vez no curso que tivemos um projeto grande, em grupo, onde pudemos desenvolver idéias conjuntas e desenvolver técnicas de trabalho em grupo.
- MAC 323 - Estruturas de Dados: Essa foi uma das disciplinas mais importantes para o nosso projeto. Nosso arcabouço baseia-se em uma estrutura de dados, e sem ele não teríamos como desenvolver uma arquitetura compatível com os nossos objetivos.
- MAC 328 - Algoritmos em Grafos: Muitos problemas podem ser mapeados como um problema de grafos, e, a partir daí, serem resolvidos usando uma técnica conhecida. Esse foi o nosso caso, já que tínhamos a utilização de grafos nas árvores geradoras mínimas para auxiliar no reconhecimento da expressão e no agrupamento de símbolos.
- MAC 441 - Programação Orientada a Objeto: Nosso projeto teve muita influência de orientação a objetos. Isso se reflete na própria motivação do trabalho. A disciplina ajudou a desenvolver e estruturar nosso sistema de forma fácil e simples através do Java.
- MAC 332 - Engenharia de Software: Nessa disciplina, tivemos mais uma vez um grande projeto a ser desenvolvido, mas desta vez com um grupo bem maior, de 6 pessoas. Não só ajudou o fato de termos trabalhado em grupo anteriormente nesta disciplina, mas também usamos conceitos de engenharia de software, como análise de requisitos e identificação de casos de uso.
- MAC 340 - Laboratório de Engenharia de Software: Sua influência no projeto é bem semelhante à da Engenharia de Software, mas com um enfoque mais detalhado na especificação do projeto, uma vez que o projeto de Engenharia de Software esteve muito mais focado na implementação.
- MAC 414 - Linguagens Formais e Autômatos: Utilizamos conceitos como gramáticas, e extendemos o conceito para gramáticas posicionais, onde leva-se em consideração a posição dos símbolos. A disciplina foi importante para introduzir conceitos e técnicas na produção de uma gramática.

- MAC 417 - Visão e Processamento de Imagens/MAC 447 - Análise e Reconhecimento de Formas: Teoria e Prática: As matérias foram importantes para ajudar nos conceitos envolvidos no projeto. Há uma sutil diferença entre a área do nosso projeto e as disciplinas, que é o fato do nosso projeto ser baseado em escrita *online*, enquanto que nestas disciplinas tratamos casos *offline*.

9 O Grupo

Seria mentira dizer que nosso grupo não teve divergências. Até o fim do projeto houveram discussões. No início foi mais tranqüilo, todos queriam trabalhar juntos, a gente se encontrava durante as aulas e marcávamos reuniões a cada duas semanas com a Nina para discussão. Depois que o Eduardo começou no estágio, por volta do meio do ano, não tínhamos muitas formas de nos encontrar, então era essencial utilizar algum meio como MSN ou e-mails. O nosso Moodle, que foi feito especialmente para projetos como o nosso, também servia como uma forma de comunicação entre nós. O único problema era que no início não usávamos com freqüência o Moodle, por exemplo. Comecei a encontrar formas de podermos avançar com o projeto mesmo distantes um do outro, como no seviço ou em casa. No final, agora também com o Pedro estagiando e até pela necessidade, acabamos usando bastante o MSN para nos comunicar, além de usar uma espécie de chat que existe no Moodle.

Problemas de horário e encontro não foram as únicas discussões que tivemos. Também tínhamos pensamentos um pouco diferentes quanto à estrutura do arcabouço. Por parecer algo muito abstrato, o Eduardo sentia muito a falta de algo concreto, mas às vezes eu não concordava com algumas coisas sugeridas por ele por achar que isso comprometeria o intuito principal do arcabouço que era a portabilidade, a generalidade. Talvez se iniciássemos mais cedo a implementação poderíamos ter menos discussões quanto ao arcabouço, pois desta forma teríamos algo visível para mostrar, e não somente diagramas e estruturas.

Não me arrependo do grupo que formamos. Com certeza trabalharia com eles novamente num projeto tão grande como esse, pois apesar de tudo, um acabava ajudando (ou tentando ajudar) sempre o outro. Trabalhar em grupo sempre é um desafio, mas é muito mais interessante do que um trabalho individual, sem discussões e interação.

10 Próximos Passos

Acho que se eu continuasse trabalhando neste projeto, começaria revisando a estrutura para iniciar a implementação de algum algoritmo de reconhecimento de ideogramas que seja bem completo, podendo levar em consideração as ordens dos traços, além do ideograma inteiro. Também tentaria divulgar o arcabouço para que ele possa ser usado em algum outro projeto, independente, para que posteriormente possamos interagir um projeto com o outro. Isso levaria em conta a capacidade do arcabouço de ser reutilizável em diversos problemas sem a necessidade de se preocupar com implementações e estruturas de dados distintas.