RECONHECIMENTO OFFLINE DE ESCRITA CURSIVA USANDO REDES NEURAIS RECORRENTES MULTIDIMENSIONAIS

Trabalho de Conclusão de Curso

Engenharia de Computação

Saulo Cadete Santos Machado
Orientador: Prof. Dr. Byron Leite Dantas Bezerra

Universidade de Pernambuco
Escola Politécnica de Pernambuco
Graduação em Engenharia de Computação

SAULO CADETE SANTOS MACHADO

RECONHECIMENTO OFFLINE DE ESCRITA CURSIVA USANDO REDES NEURAIS RECORRENTES MULTIDIMENSIONAIS

Monografia apresentada como requisito parcial para obtenção do diploma de Bacharel em Engenharia de Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

Recife, Maio de 2011.

De acordo

Recife

____/_____/_____

_____

Prof. Dr. Byron Leite Dantas Bezerra

Dedico este trabalho a meu avô materno, Joel José dos Santos, que muito se orgulharia ao testemunhar primeira graduação de seus netos.

Agradecimentos

Agradeço a Deus por permitir que eu chegasse a este momento marcante de minha vida e a meus pais que tanto investiram em minha educação.

Também registro meus agradecimentos para:

- Meus amigos que sempre me motivaram;
- Professores da graduação de Engenharia de Computação da UPE que se esforçam para criar um curso cada vez melhor para seus alunos;
- Meus colegas de trabalho que colaboraram para a compreensão técnica e teórica necessária para a realização deste trabalho;
- A empresa *Stefanini Document Solutions* por ter permitido usar a base de imagens de meses em cheques bancários nos experimentos.

A todos que acreditaram em meu esforço, meu muito obrigado.

Resumo

O reconhecimento *offline* de escrita cursiva é um desafio motivado pela grande quantidade de informações que existe e que ainda é produzida em papel. A capacidade de memorizar informações contextuais dinamicamente faz das redes neurais recorrentes excelentes soluções para esse problema. Algumas melhorias nessa arquitetura deram origem à considerada estado da arte para reconhecimento de escrita cursiva, a arquitetura *Multidimensional Long Short-Term Memory*. Neste trabalho foi criada uma biblioteca que reúne as funcionalidades de treinamento e classificação para redes dessa arquitetura. Seu principal objetivo é facilitar o uso e a integração. Também foi analisado o impacto usar imagens com resoluções diferentes para treinamento e classificação. Os resultados comprovaram a eficiência do biblioteca criada e permitiram identificar a que parâmetros do treinamento afetam sua robustez. É possível dar continuidade a este trabalho focando tanto na aplicação criada quanto na análise.

Abstract

Offline handwriting recognition is a challenge motivated by the huge amount of information that exists and still being produced in paper. The ability to dynamically store context information make the recurrent neural networks excellent solutions for this problem. Some enhancements in the architecture of these networks created the one considered state of art for handwriting recognition, the Multidimensional Long Short-Term Memory architecture. A library was created in this work that includes the training and classification functionality for networks of that architecture. It's mail objective is to make the use and integration of easier. The impact of using different image resolutions for training and classification was also analyzed. The results proved the efficiency of the created library and allowed to identify that training parameters affects its robustness. It's possible to continue this work focusing both in the application and in the analysis.

Sumário

Índice de Figuras

Índice de Tabelas

Tabela de Símbolos e Siglas

BPTT - *Backpropagation Through Time*

FN – *False negative*

FP – *False positive*

LSTM – *Long Short-Term Memory*

HMM – *Hidden Markov Model*

MDRNN – *Multi-dimensional Recurrent Neural Network*

MLP – *Multilayer Perceptron*

RNN – *Recurrent Neural Network*

ROC - *Receiver Operating Characteristics*

TN – *True negative*

TP – *True positive*

Capítulo 1

Introdução

Apesar da facilidade do acesso a informações em mídia eletrônica é impossível ignorar a quantidade de informações em papel. São formulários, memorandos, cartas, requerimentos, cheques e muitos outros documentos. Para que eles sejam transcritos manualmente em conteúdo eletrônico é necessário muito tempo e esforço em tarefas pouco produtivas. Para solucionar este problema são utilizados sistemas computacionais inteligentes capazes de reconhecer o conteúdo dos documentos.

O reconhecimento de manuscritos é uma tarefa que envolve visão computacional e o aprendizado de sequências. A visão computacional é responsável por extrair as principais características das imagens que são apresentadas em série para compor o aprendizado sequencial. Na maioria dos sistemas essas etapas são tratadas separadamente. A Rede Neural Recorrente Multidimensional, MDRNN (*Multi-dimensional Recurrent Neural Network*) [4], solução utilizada neste trabalho, atua nesses dois momentos.

Este trabalho teve duas principais abordagens. A primeira foi a criação de uma biblioteca que inclui funcionalidades de treinamento e classificação de MDRNNs. A segunda abordagem foi avaliar, através de resultados obtidos com o uso desta biblioteca, a influência de alguns fatores sobre a qualidade de classificação. Para isso foram realizados testes em um cenário de uso real.

A biblioteca criada atua tornando transparente ao usuário as operações realizadas para treinamento e classificação. Ela facilita tanto a demonstração quanto a integração da solução para reconhecimento de escrita cursiva através de MDRNNs.

Para os experimentos foram classificadas imagens de meses em cheques bancários brasileiros. Eles permitiram identificar importantes características do algoritmo estudado. Redes treinadas com parâmetros diferentes sofreram os impactos também diferentes.

Foi possível identificar que um tipo de treinamento é mais adequado quando há mudanças na resolução das imagens para treinamento e classificação. Em contrapartida, este não classifica tão bem quando comparado ao outro tipo de treinamento quando a resolução é mantida a mesma.

1.1 Motivação

Apesar da constante evolução de tecnologias voltadas para o gerenciamento eletrônico de documentos, a quantidade de informações que ainda existe e que ainda é produzida "em papel" é incomensurável. Em muitos casos a informação que se quer obter destes documentos está escrita cursivamente[1]. Transpor essa informação para um meio eletrônico manualmente é um trabalho pouco produtivo e custoso que envolve pessoas, muitas vezes pouco preparadas para o uso de ferramentas especializadas, como editores de texto. [1]

A busca por soluções para o reconhecimento automático de escrita cursiva não é recente. A principal dificuldade é a variedade de escritas possíveis, já que a maneira de escrever difere entre as pessoas e sofre influência de fatores como pressão e velocidade usados na caneta, tipo de caneta, tipo de papel, e até fatores emocionais. O problema é ainda mais complexo quando a única informação que o sistema tem disponível é a imagem. Este caso é chamado de reconhecimento *offline*.

As MDRNNs são o estado da arte para reconhecimendo de escrita cursiva. No artigo em que foi proposto o algoritmo estudado neste trabalho [5], foi feita a comparação com soluções clássicas como as baseadas em Modelo Oculto de Markov (*Hidden Markov Model*). Os experimentos foram feitos com imagens de escrita cusiva árabe, e embora os autores não conhecessem nada sobre a língua, obtiveram os melhores resultados. Mesmo com essa prova da capacidade de generalização das MDRNNs, é muito importante avaliar seu desempenho para reconhecimento de escrita cursiva em cenários de uso reais.

[1] Escrita cursiva é o estilo de escrita à mão voltada para escrita ágil.

A possibilidade de usar a capacidade das MDRNN sem a necessidade de conhecer sua implementação aumentaria os casos em que poderiam ser aplicados. A solução criada poderia ser integrada a quaisquer outros sistemas sem que o usuário precisasse antes compreender o seu funcionamento.

1.2 Objetivos e Metas

O principal objetivo deste trabalho é avaliar a eficiência do uso de MDRNN para reconhecimento *offline* de escrita cursiva em cenários de uso reais. Para isso pretende oferecer uma interface que facilite sua integração e uso e analisar a influência de mudanças na resolução das imagens na qualidade de sua classificação. Foram traçados os objetivos específicos:

- Realizar modificações necessárias no código publicado [3] para facilitar a integração de suas funcionalidades;
- Desenvolver módulo de pré-processamento que, a partir de uma imagem ou conjunto de imagens, prepara a entrada para treinamento ou classificação usando a MDRNN;
- Integrar módulo de pré-processamento ao código modificado da MDRNN;
- Testar o sistema para reconhecimento de imagens de escrita cursiva de meses do ano em cheques brasileiros com variação na resolução;
- Analisar resultados para validar o sistema proposto.

1.3 Estrutura do Documento

Os 5 capítulos que compõem este trabalho serão resumidos a seguir.

- **Capítulo 1: Introdução**

Conteúdo introdutório do trabalho. Neste capítulo serão explicados os problemas que o trabalho visa solucionar, as principais motivações para sua realização e os objetivos traçados para alcançar o resultado esperado.

- **Capítulo 2: Fundamentação Teórica**

Referencial teórico necessário para a compreensão do trabalho. São explicados conceitos de Redes Neurais, Redes Neurais Recorrentes, *Long Short Term Memory*, Multidimensionalidade e Multidirecionalidade. Também são descritos os principais pontos do funcionamento do código usado no trabalho.

- **Capítulo 3: Reconhecimento de Escrita Cursiva**

Neste capítulo é descrita a estratégia utilizada para o desenvolvimento do módulo de pré-processamento, para a realização das modificações no código original da MDRNN e para a integração que deu origem ao sistema proposto. Também é apresentada a abordagem utilizada para analisar a qualidade da classificação das redes criadas e os fatores que a influenciaram.

- **Capítulo 4: Experimentos e Resultados**

Este capítulo expõe e descreve os experimentos e discute os resultados obtidos pela utilização do sistema proposto em cenários de uso reais.

- **Capítulo 5: Conclusões e Trabalhos Futuros**

São apresentadas as conclusões feitas a partir da análise dos resultados obtidos e as contribuições do trabalho. Também são sugeridos trabalhos futuros que melhorem o que foi alcançado.

Capítulo 2

Fundamentação Teórica

Neste capítulo serão apresentados os conceitos básicos necessários para compreender este trabalho. Serão descritos conceitos de Rotulamento Supervisionado de Sequências, do inglês *Supervised Sequence Labelling*, Redes Neurais, Redes Neurais Recorrentes e *Long Short Term Memory*.

2.1 Rotulamento Supervisionado de Sequências

Primeiramente é importante compreender a tarefa que o sistema visa desempenhar. O Rotulamento Supervisionado de Sequências é um problema de aprendizado supervisionado em que o sistema deve classificar exemplos que são apresentados a ele de acordo com o conhecimento que foi previamente construído. Esses conceitos são detalhados a seguir.

Aprendizado supervisionado é o nome dado ao aprendizado de máquina quando um conjunto de exemplos de entrada e suas respectivas respostas são apresentados ao sistema. [8] Em outras palavras, para treinar o sistema são apresentados pares entrada-resposta. O aprendizado supervisionado é diferente do aprendizado por reforço, quando é apresentado um valor que indica o grau de acerto ou erro; e do aprendizado não supervisionado, em que nenhum sinal é fornecido e o algoritmo tenta descobrir sozinho a estrutura dos dados. [2]

Em aprendizado de máquina, rotulamento de sequências engloba as tarefas em que sequências de dados são transcritos em sequências de rótulos. Quando fornecidas para um algoritmo de treinamento supervisionado uma sequência de dados como entrada e suas transcrições como alvo, tem-se o rotulamento supervisionado de sequências. [2]

A principal diferença desse problema para os demais, tradicionalmente solucionados com classificação de padrões supervisionada, é que os elementos individuais dos dados não podem ser interpretados como independentes e identicamente distribuídos. Tanto os dados de entrada quanto os rótulos formam sequências fortemente correlacionadas. [2] No reconhecimento de escrita cursiva, por exemplo, um caractere escrito no meio de uma palavra foi produzido sofrendo influência do traçado anterior e atua sobre o caractere seguinte.

Para uma descrição formal de rotulamento supervisionado de sequências é necessário fazer algumas considerações:

- As sequências x de vetores de tamanho fixo e valores reais são as entradas;
- As sequências z de rótulos discretos definidos a partir de um alfabeto finito L são as respostas;
- Os pares entrada-resposta (x, z) são independentemente e identicamente distribuídos (o que nem sempre é verdade, como quando as sequências de entrada representam manuscritos ou um diálogo falado. Apesar disso, essa consideração é aceitável contanto que os limites da sequência sejam escolhidos cautelosamente);
- O espaço das entradas $\mathcal{X} = (\mathbb{R}^M)$ é o conjunto de todas as sequências de vetores de valores reais de tamanho M;
- O espaço de respostas $\mathcal{Z} = L^*$ é o conjunto de todas as sequências do alfabeto finito L de rótulos. Assim, L^* representa sequências de rótulos;
- S é um conjunto de exemplos de treinamento originados independentemente a partir de uma distribuição $D_{\mathcal{X}\times\mathcal{Z}}$. Cada um de seus elementos é um par de sequências (x, z);
- As sequências de alvos são, no máximo, do tamanho das sequências de entrada correspondentes. Por exemplo, dada a sequência de alvos $z = (z_1, z_2, \ldots, z_U)$ e a sequência de entradas $x = (x_1, x_2, \ldots, x_T)$, $U \leq T$.

A tarefa de rotulamento de sequência é usar S para treinar o algoritmo $h: \mathcal{X} \rightarrow \mathcal{Z}$ para rotular as sequências em um conjunto de testes $S' \subset D_{\mathcal{X}\times\mathcal{Z}}$, e $S \cap S' = \emptyset$, visando minimizar uma medida de erro escolhida. [2]

Em alguns casos podem ser aplicadas algumas restrições ao tipo de sequências de rótulos necessário. Essas restrições afetam tanto a escolha do algoritmo quanto as medidas de erro usadas para verificar seu desempenho. [2] Dessa maneira é possível distinguir os tipos o rotulamento de sequência: classificação de segmentos, classificação de sequências e classificação temporal.

Para a classificação de sequências, cada sequência de entrada é associada a uma única classe. Este é o tipo mais restritivo de classificação. A sequência de rótulos é obrigada a ter um tamanho unitário. [2] Um exemplo é o reconhecimento individual de uma letra manuscrita. Neste caso, quando as sequências de entrada puderem ser consideradas de tamanho fixo o problema pode ser reduzido a um problema de reconhecimento de padrões.

Em alguns casos é necessário fazer várias classificações para uma mesma sequência de entrada. Quando é necessário classificar cada letra em uma linha manuscrita, por exemplo. Quando as posições dos segmentos de entrada para as quais as classificações são solicitadas são conhecidas previamente, a tarefa é chamada de classificação de segmentos.

Esse tipo de classificação difere de simples reconhecimentos de padrão por levarem em consideração o contexto. Como mostra a Figura 1, essa informação é indispensável para a classificação de segmentos. É possível ler com facilidade a palavra ‘sequência’. Mas quando isoladas, as letras ‘u’ e ‘n’ não podem ser distintas.

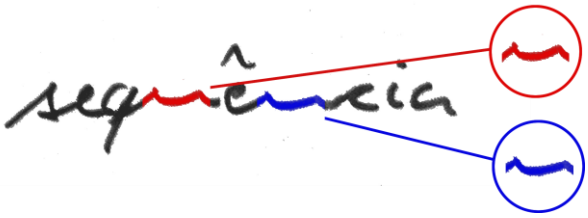

Figura 1. O contexto é importante para identificar as letras de uma palavra manuscrita.

A classificação temporal é o caso mais generalizado. Nele a única premissa considerada sobre as sequências de respostas é que seu tamanho é menor ou igual ao das sequências de entrada. A diferença para a classificação de segmentos é que na classificação temporal é necessário que o algoritmo decida onde a classificação

deve ser feita na sequência de entrada. Por não ser conhecido o alinhamento entre as sequências de entrada e as sequências de rótulos, a qualidade da classificação é feita medindo o número de substituições, inserções e remoções de rótulos que foram necessárias para transformar uma sequência na outra. [2] Como resultado tem-se a taxa de erro $E^{lab}(h, S')$, definida pela equação a seguir.

$$E^{lab}(h, S') = \frac{1}{Z} \sum_{(x,z) \in S'} LEV(h(x))$$

Onde Z é o tamanho total das sequências de respostas no conjunto de testes, e $LEV(p, q)$ é a Distância de Levenshtein entre as sequências p e q, ou seja, o número mínimo de operações que foi necessário para transformar p em q. [2]

A classificação temporal é o tipo de rotulamento de sequências utilizado no algoritmo estudado neste trabalho. As seções a seguir apresentarão os conceitos de Redes Neurais necessários para sua compreensão.

2.2 Redes Neurais

As redes neurais, ou redes neurais artificiais, são modelos que têm como metáfora o funcionamento do cérebro humano com suas redes neurais biológicas. [8] Em geral, sua estrutura básica é uma rede de unidades de processamento interconectadas. Essas unidades, os neurônios, estão dispostos em camadas (uma de entrada, pelo menos uma intermediária e uma de saída) responsáveis por garantir a memória e a não linearidade da rede.

Assim como nos neurônios biológicos, é necessário que um determinado valor de estímulo seja fornecido para que o impulso nervoso seja disparado. Esse valor é representado entre as ligações da rede como pesos (w).

Diversos foram os tipos de redes neurais que surgiram desde a criação do conceito. Uma importante distinção a ser feita é entre as redes que possuem conexões cíclicas e as que não possuem. As primeiras são as redes com *feedback*, com retropropagação, recursivas ou recorrentes. As redes neurais que não possuem

ciclos são as redes neurais com *feedforward* ou com pré-alimentação. [2] As seções a seguir descrevem esses conceitos.

2.2.1 Perceptron Multicamadas

As Perceptron Multicamadas, MLP (*Multilayer Perceptron*), são redes com pré-alimentação que possuem pelo menos três camadas, como ilustrado na Figura 2. Uma camada abriga os neurônios que representam a variável de entrada para o problema. Pelo menos uma camada intermediária é necessária para tornar a rede capaz de resolver problemas reais[2]. Por fim, uma camada de saída expõe a resposta da rede, representando a variável desejada. [8]

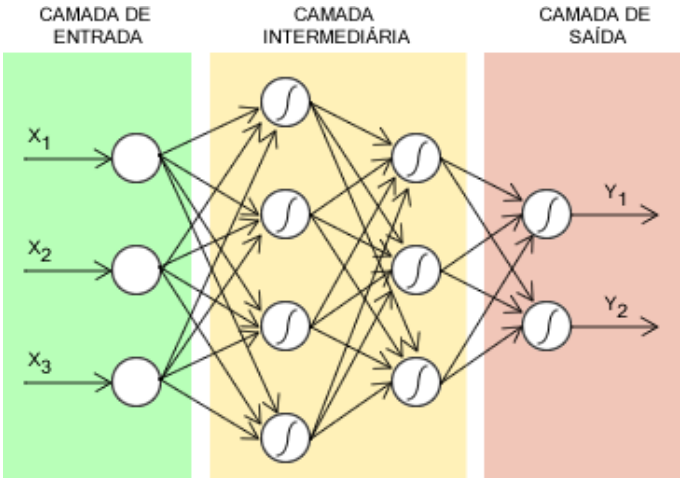

Figura 2. Perceptron Multicamadas

Em sua utilização, padrões de entrada são apresentados à camada de entrada e os valores resultantes das ativações são propagados pelas camadas intermediárias até a camada de saída (fase *forward*). Após obter a resposta na camada de saída os pesos da rede são reajustados visando minimizar a diferença entre a resposta esperada e o que foi calculado pela rede (fase *backward*).

- **Fase *forward***

[2] Os problemas reais normalmente não são linearmente separáveis, isto é, suas classes não podem ser distintas por apenas uma característica. Por exemplo, para um problema de classificação em que há duas classes, não é possível traçar uma reta que as separe graficamente.

É nesta fase que o impulso é propagado da camada de entrada até a camada de saída. Considerando uma MLP com I neurônios de entrada, apresentam-se os valores do vetor x como entrada. O primeiro passo é calcular a entrada líquida para os neurônios da camada intermediária, representada por net_h:

$$net_h = \sum_{i=1}^{I} w_{ih} x_i$$

Onde w_{ih} é o peso da conexão entre o neurônio da camada de entrada i para o neurônio da camada intermediária h.

É então calculado o valor de resposta do neurônio da camada intermediária para o impulso recebido, a partir da função de saída y_h:

$$y_h = \theta_h(net_h)$$

Onde θ_h é a função de ativação, normalmente sigmoidal. As mais usadas são a sigmoide logística $\sigma(x) = \frac{1}{1+e^{-x}}$, e a tangente hiperbólica $tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$. Suas principais características são a não linearidade e a capacidade de diferenciação. Elas permitem que as redes resolvam problemas mais complexos. Além disso possibilitam que seja usada a técnica do gradiente descendente, ou *backpropagation*, no treinamento das redes. Essa técnica será explicada a seguir.

Generalizando as equações da primeira camada intermediária para qualquer camada intermediária l, tem-se como entrada do neurônio h:

$$net_h = \sum_{h'=1}^{H_{l-1}} w_{h'h} y_{h'}$$

Onde H é o número de neurônios da camada l.

Os valores de entrada para a camada de saída de uma rede neural com m camadas escondidas são dados por:

$$net_k = \sum_{h=1}^{H_m} w_{hk}\, y_h$$

Assim como na camada intermediária, também é calculado o valor de saída da camada escondida:

$$y_k = \theta_k(net_k)$$

Onde θ_k é uma função de ativação independente de θ_h.

- **Fase *backward***

Nesta fase são usadas técnicas de gradiente descendente, como *backpropagation,* para reduzir a função objetivo diferenciável O. Métodos de gradiente descendente consistem em encontrar a derivada da função objetivo em relação a cada um dos pesos da rede e então ajustá-los na direção do coeficiente negativo. A técnica de *backpropagation* propaga o erro, distância entre o objetivo e o resultado obtido, de tal forma que os pesos possam ser reajustados.

O reajuste dos pesos da rede dependem de uma medida chamada sensibilidade. Ela define a capacidade da rede de classificar corretamente uma entrada. A sensibilidade para qualquer elemento da camada escondida h antes da última é dada pela equação a seguir.

$$\delta_h = \theta_h'(net_h) \sum_{h\prime=1}^{H_{l+1}} \delta_{h\prime} w_{hh\prime}$$

A partir da sensibilidade dos elementos da camada intermediária é possível calcular a diferencial para cada peso da rede como:

$$\frac{\partial O}{\partial w_{ij}} = \frac{\partial O}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}} = \delta_j y_i$$

O reajuste é feito até que a rede consiga alcançar uma medida de desempenho ou um número de iterações (épocas) limite.

2.2.2 Redes Neurais Recorrentes

As redes neurais apresentadas não possuem ciclos em suas conexões. As redes neurais recorrentes, *Recurrent Neural Networks* (RNN), não têm essa restrição e permitem conexões cíclicas entre seus neurônios. [2] A Figura 3 ilustra um exemplo desta arquitetura.

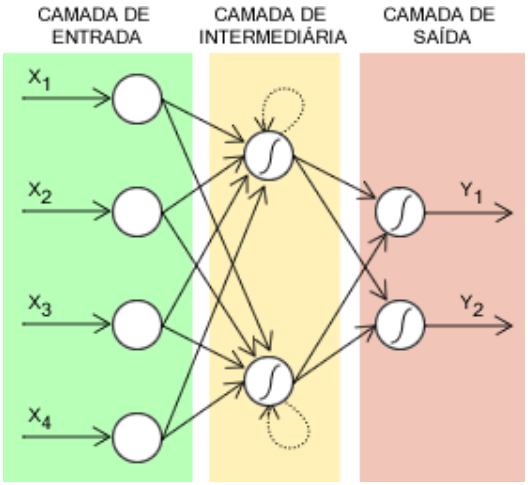

Figura 3. Rede Neural Recorrente. As setas pontilhadas representam as conexões recorrentes.

Evoluir o conceito de MLP para RNN é simples, mas as implicações para o aprendizado de sequências são enormes. Isso porque uma MLP limita-se a mapear vetores de entrada em vetores de saída enquanto uma RNN é capaz de usar toda a história de entradas passadas para fazer o mapeamento para saída. Sendo assim, as conexões de uma rede recorrente permitem que a 'memória' de entradas passadas persista nos impulsos internos da rede e influenciem em suas respostas. [2] Essa característica torna as RNNs robustas a distorções nos dados de entrada e permitem a utilização do contexto da entrada.

Assim como no treinamento de MLP, as redes recorrentes possuem a fase *forward* e *backward*:

- **Fase *forward***

A principal diferença para a fase forward das MLPs é que nas RNNs os impulsos que incidem na camada intermediária são provenientes da camada de entrada atual e da ativação da camada intermediária do passo anterior (a função de saída da camada intermediária persiste na rede).

$$net_h^t = \sum_{i=1}^{I} w_{ih} x_i^t + \sum_{h'=1}^{H} w_{h'h} y_{h'}^{t-1}$$

Onde x_i^t é o valor da entrada i no instante t, net_h^t é a entrada da camada intermediária no instante t e $y_{h'}^{t-1}$ é a função de saída do neurônio da camada intermediária h' no instante $t-1$.

A função de saída para o neurônio h no instante t é calculada aplicando a função de ativação diferenciável exatamente como nas MLPs:

$$y_h^t = \theta_h(net_h^t)$$

É importante notar que no instante $t = 1$ é a primeira vez que os exemplos são apresentados à rede. Sendo assim, os valores y_i^0 deveriam ser extraídos dos neurônios da camada intermediária em um estado anterior à apresentação de exemplos. Neste trabalho y_i^0 será sempre zero apesar de alguns pesquisadores terem descoberto que, em alguns casos, usar valores diferentes de zero aumenta a robustez a ruídos e estabilidade da RNN.

A entrada líquida dos neurônios da camada escondida é calculada a partir da função de saída dos neurônios da camada escondida:

$$net_k^t = \sum_{h=1}^{H} w_{hk} y_h^t$$

- **Fase *backward***

Nesta fase pode ser aplicado uma extensão da técnica usada anteriormente para MLPs. A *Backpropagation Through Time* (BPTT) consiste da aplicação repetida da regra da cadeia. A diferença no caso das redes recorrentes é que a função objetivo depende da ativação da camada intermediária tanto para a camada escondida, quanto para a camada intermediária no próximo instante $(t+1)$.

Sendo assim a sensibilidade da rede leva em consideração os dois fatores.

$$\delta_h^t = \theta'(net_h^t)\left(\sum_{k=1}^{K} \delta_k^t w_{hk} + \sum_{h'=1}^{H} \delta_{h'}^{t+1} w_{hh'}\right)$$

As diferenciais em relação aos pesos da rede podem ser calculadas considerando que os pesos de cada elemento da camada escondida são os mesmos em todos os instantes t:

$$\frac{\partial O}{\partial w_{ij}} = \sum_{t=1}^{T} \frac{\partial O}{\partial net_j^t} \frac{\partial net_j^t}{\partial w_{ij}} = \sum_{t=1}^{T} \delta_j^t y_i^t$$

2.3 Long Short Term Memory

Como apresentado na seção anterior, a principal característica das RNNs para a classificação de segmentos é a habilidade de usar informação contextual para realizar o mapeamento entre entradas e saídas. O problema é que a influência de uma entrada na camada intermediária e, consequentemente, na camada de saída, ou cresce exponencialmente ou é diluída enquanto circula pelas conexões recorrentes da rede.

Conhecido como problema da dissipação do gradiente (*vanishing gradient problem*), ele torna difícil para RNNs aprender tarefas que contenham mais que dez atrasos no tempo (*timesteps*) entre a apresentação de entradas relevantes e eventos esperados. [2] Na Figura 4 é possível observar a dissipação do gradiente com o tempo. À medida que novas entradas são apresentadas à rede, a ativação da camada intermediária é sobrescrita e a rede 'esquece' as primeiras entradas.

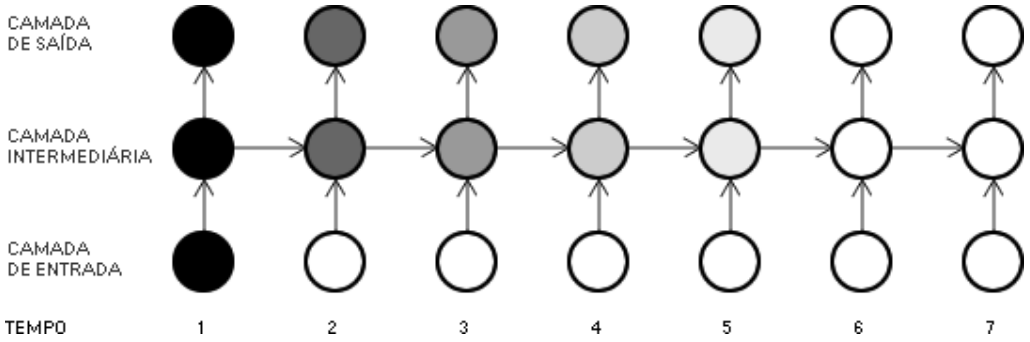

Figura 4. Problema da dissipação do gradiente. Os nós representam neurônios e seu conteúdo representa a sensibilidade para a entrada, quanto mais escuro o nó, mais sensível à entrada preta.

Dentre várias soluções propostas para reduzir os impactos da dissipação do gradiente em RNNs, a solução mais efetiva é a arquitetura *Long Short Term Memory* (LSTM), proposta por Hochreiter e Schmidhuber. [6]

2.3.1 Descrição da Arquitetura

A arquitetura LSTM consiste de um conjunto de sub-redes conectadas recorrentemente. Essas sub-redes, chamadas de blocos de memória, podem ser consideradas uma metáfora de *chips* de memória. Cada bloco possui uma ou mais células de memória autoconectadas e três unidades de multiplicação que definem a operação que deve ser realizada, as portas de entrada, saída e de esquecimento (*forget*). Voltando à analogia com *chips* de memória, essas operações podem ser interpretadas como as operações de escrita, leitura e de limpeza (*reset*) dos circuitos.

A única diferença estrutural de uma rede LSTM para uma RNN é que os elementos da camada intermediária deixam de ser neurônios com funções de ativação não lineares e passam a ser blocos de memória. A Figura 5 ilustra um bloco de memória que tem apenas uma célula de informação. O chaveamento das portas permite preservar a informação do bloco por muito tempo, evitando o problema da degradação de gradiente. Enquanto a porta de entrada estiver fechada, a ativação da célula de informação não será alterada, podendo ser acessada em um outro momento ativando a porta de saída. A informação é mantida na célula através de uma conexão recorrente de peso 1. [6]

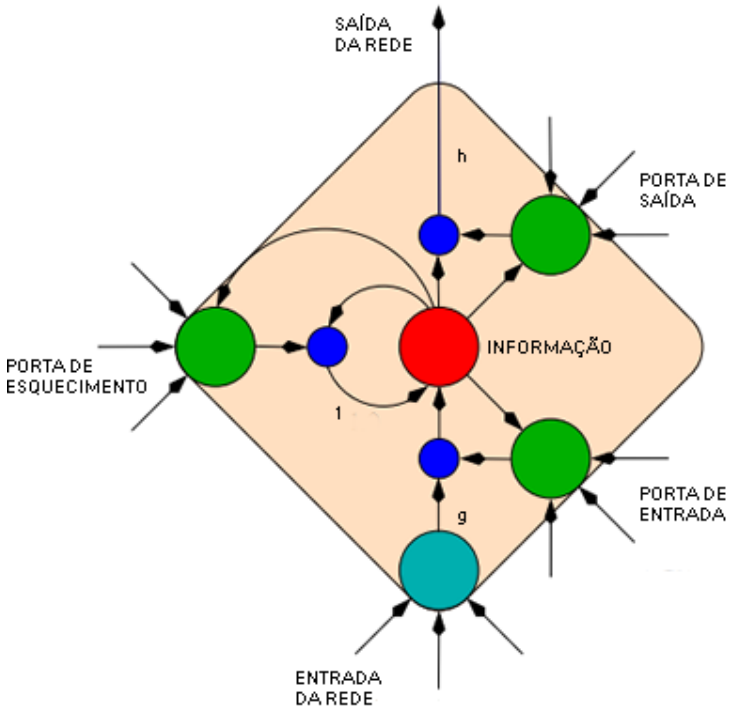

Figura 5. Estrutura de bloco de memória da arquitetura LSTM.

A Rede LSTM preserva o gradiente da informação como ilustrado na Figura 6. Quando a porta de esquecimento encontra-se aberta, a informação é preservada. A sensibilidade do bloco de memória pode ser controlada pela porta de saída sem afetar a célula de informação do bloco. Por fim, a entrada só afeta a informação quando a porta de entrada está aberta.

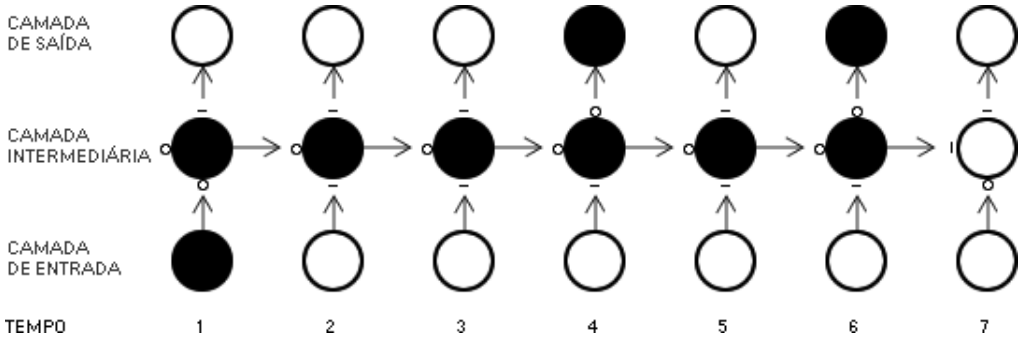

Figura 6. Portas dos blocos de memória preservam a informação nas redes LSTM. Para os blocos da camada intermediária, os círculos representam portas abertas e os traços portas fechadas. A porta em baixo do nó é a porta

de entrada, a superior é a porta de saída e a lateral é a porta de esquecimento.

O algoritmo da rede LSTM também possui as fases *forward* e *backward*. Para definir suas equações considera-se c uma entre as C células de um bloco de memória, s_c^t o estado da célula c no instante t, f a função de ativação das portas, g a função de ativação para a entrada da célula e h a função de ativação saída da célula. Da mesma forma que nas demais RNNs, no estado inicial ($t = 0$), considera-se que todos os estados e ativações são zero e que, no instante seguinte ao fim do processo ($t = T + 1$), a sensibilidade de todos os elementos é zero.

- **Fase *forward***

A ordem em que os passos são realizados é importante. Inicialmente é calculada a entrada líquida da porta de entrada (ι) no instante t do bloco e sua função de saída para esta entrada:

$$net_\iota^t = \sum_{i=1}^{I} w_{i\iota} x_i^t + \sum_{h=1}^{H} w_{h\iota} y_h^{t-1} + \sum_{c=1}^{C} w_{c\iota} s_c^{t-1}$$

$$y_\iota^t = f(net_\iota^t)$$

Onde I é o número de entradas e H é o número de blocos na camada escondida.

Para portas de esquecimento (ϕ) a entrada líquida e a função de saída são descritas de maneira semelhante:

$$net_\phi^t = \sum_{i=1}^{I} w_{i\phi} x_i^t + \sum_{h=1}^{H} w_{h\phi} y_h^{t-1} + \sum_{c=1}^{C} w_{c\phi} s_c^{t-1}$$

$$y_\phi^t = f(net_\phi^t)$$

Após os cálculos dessas funções de saída, é calculada a entrada líquida da célula de informação e o estado da célula para os impulsos provenientes das portas de esquecimento e de entrada.

$$net_c^t = \sum_{i=1}^{I} w_{ic} x_i^t + \sum_{h=1}^{H} w_{hc} y_h^{t-1}$$

$$s_c^t = y_\phi^t s_c^{t-1} + y_\iota^t g(net_c^t)$$

As mesmas equações das portas anteriores se aplicam para as portas de saída (ω):

$$net_\omega^t = \sum_{i=1}^{I} w_{i\omega} x_i^t + \sum_{h=1}^{H} w_{h\omega} y_h^{t-1} + \sum_{c=1}^{C} w_{c\omega} s_c^{t-1}$$

$$y_\omega^t = f(net_\omega^t)$$

Assim a saída do bloco de memória em questão é dada por:

$$y_c^t = y_\omega^t h(s_c^t)$$

Essa é a única saída de todo o bloco de memória. Vale à pena notar a forte dependência deste valor com a função de saída da porta de saída. A função de ativação sugerida para as portas é a sigmoide logística. Já para a entrada e a saída recomenda-se o uso da tangente hiperbólica. [2]

- **Fase *backward***

Nesta fase é utilizada a técnica de BPTT. Sabe-se que a sensibilidade de um elemento j é a diferença entre a função objetivo e sua entrada líquida ($\delta_j^t \stackrel{\text{def}}{=} \frac{\partial O}{\partial net_j^t}$) e que os erros relacionados à saída do bloco de memória são $e_c^t \stackrel{\text{def}}{=} \frac{\partial O}{\partial y_c^t}$ e $e_s^t \stackrel{\text{def}}{=} \frac{\partial O}{\partial s_c^t}$.

Sendo K a quantidade de elementos na camada de saída e k um índice dentre esses elementos, o primeiro passo é calcular o erro relacionado à saída do bloco de memória para então calcular a sensibilidade da porta de saída.

$$e_c^t = \sum_{k=1}^{K} w_{ck} \delta_k^t + \sum_{h=1}^{H} w_{ch} \delta_h^{t+1}$$

$$\delta_\omega^t = f'(net_\omega^t) \sum_{c=1}^{C} h(s_c^t) e_c^t$$

São então calculados os erros dos estados, necessários para o cálculo da sensibilidade das células e das portas de esquecimento e entrada.

$$e_s^t = y_\omega^t h'(s_c^t) e_c^t + y_\phi^{t+1} e_s^{t+1} + w_{c\iota} \delta_\iota^{t+1} + w_{c\phi} \delta_\phi^{t+1} + w_{c\omega} \delta_\omega^t$$

$$\delta_c^t = y_\iota^t g'(net_c^t) e_s^t$$

De maneira semelhante, as sensibilidades das portas de esquecimento e entrada são, respectivamente:

$$\delta_\phi^t = f'(net_\phi^t) \sum_{c=1}^{C} s_c^{t-1} e_s^t$$

$$\delta_\iota^t = f'(net_\iota^t) \sum_{c=1}^{C} g(net_c^t) e_s^t$$

2.4 Adicionando Multidimensionalidade e Multidirecionalidade

Informações de contexto são muito importante em alguns casos de rotulamento de sequência. Quando tratamos de dados unidimensionais, isso pode ser solucionado usando de Redes Neurais Recorrentes Bidirecionais. Neste tipo de rede duas camadas intermediárias percorrem a entrada em sentidos opostos.

Uma generalização deste tipo de rede para n dimensões exige 2^n camadas intermediárias em que cada uma começaria a percorrer a entrada de um dos cantos de sua forma n dimensional. A Figura 7 ilustra, para dados de duas dimensões, o tratamento de uma RNN bidimensional e multidirecional. As setas no interior indicam a direção da propagação da informação durante a fase *forward*.

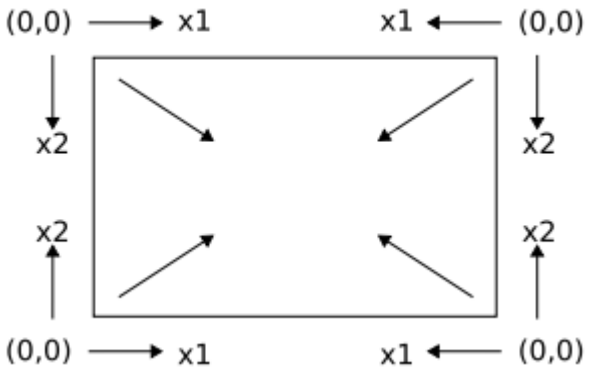

Figura 7. Quatro camadas intermediárias percorrem os dados em uma RNN bidimensional multidirecional.

A multidirecionalidade da rede permite que ela tenha ainda mais informação de contexto disponível em um neurônio. A Figura 8 compara a disponibilidade de contexto para uma MDRNN unidirecional, à esquerda, e uma MDRNN multidirecional, à direita.

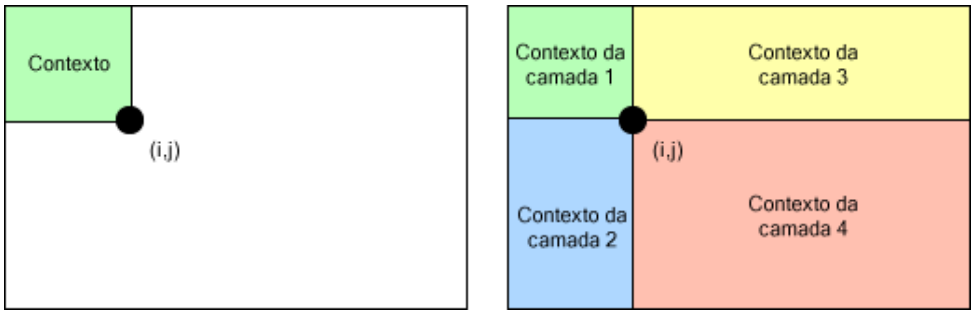

Figura 8. Disponibilidade de contexto para de acordo com a multidirecionalidade.

A multidimensionalidade é uma extenção do conceito de conexões recorrentes das RNNs. Deixa de haver apenas uma conexão recorrente para haver quantas forem as dimensões dos dados. Essas conexões permitem que a rede crie representações internas flexíveis do contexto, o que a torna mais robusta a perturbações locais nos dados. [4]

Uma MDRNN varre o dado de entrada de maneira que, quando alcança um ponto em uma sequência n-dimensional, ela já visitou todos os pontos dos quais necessita dos impulsos. A Figura 9 ilustra como o algoritmo visita os pontos de uma sequência de dados bidimensional. [5]

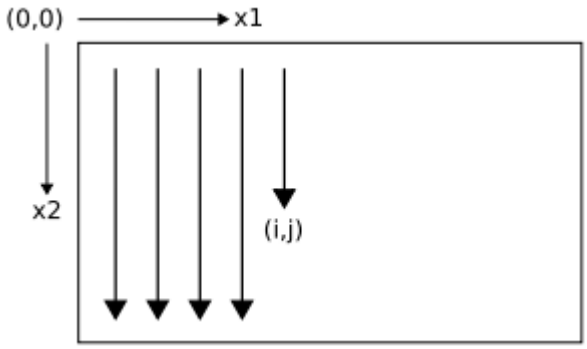

Figura 9. Como as MDRNNs visitam pontos em dados de duas dimensões.

Nas subseções a seguir serão descritos os conceitos formais da multidimensionalidade como aplica-los às redes neurais da arquitetura LSTM.

2.4.1 Equações de Multidimensionalidade

Para uma MDRNN considera-se net_j^p e y_j^p entrada da rede e função de ativação para o elemento j no ponto $p = (p_1, \dots, p_n)$ em uma sequência x de n dimensões. Nesse mesmo modelo w_{ij}^d representa o peso de uma conexão recorrente do neurônio i ao j na dimensão d. Considerando que a MDRNN possui I neurônios de entrada, K neurônios de saída e H neurônios na camada intermediária, e que a função de ativação dos neurônios da camada intermediária é θ_h, são definidas as equações para as fases *forward* e *backward*.

- **Fase *forward***

$$net_h^p = \sum_{i=1}^{I} x_i^p w_{ih} + \sum_{\substack{d=1: \\ p_d>0}}^{n} \left(\sum_{h\prime=1}^{H} y_{h\prime}^{p_d^-} w_{h\prime h}^d \right)$$

$$y_h^p = \theta_h(net_h^p)$$

Onde $p_d^- \stackrel{\text{def}}{=} (p_1, \dots, p_d - 1, \dots, p_n)$. Em outras palavras, para a conexão na dimensão d é usado o valor dos neurônios anteriores nesta dimensão. A Figura 10 ilustra essas conexões para uma MDRNN de duas dimensões.

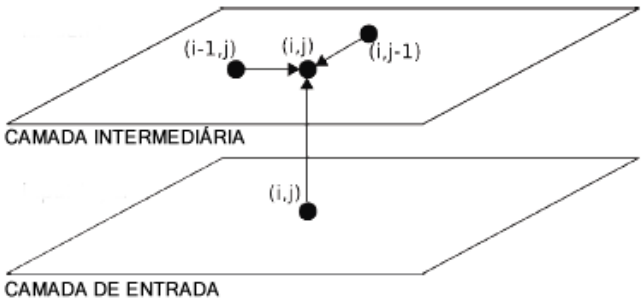

Figura 10. Conexões na fase *forward* em uma MDRNN de duas dimensões.

- **Fase *backward***

Nesta fase é usada a informação do neurônio da camada de saída e dos neurônios seguintes na mesma camada do neurônio em questão, como ilustrado na Figura 11. Essas são as conexões afetadas pelo impulso do neurônio.

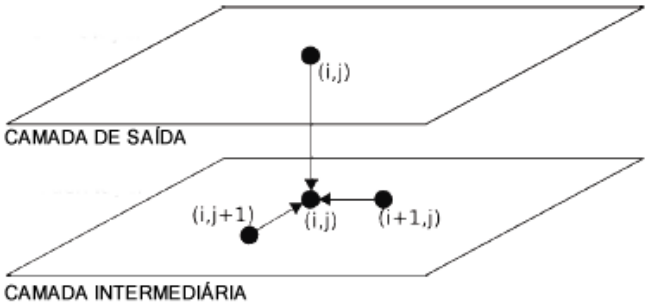

Figura 11. Conexões na fase backward em uma MDRNN de duas dimensões.

Sabendo que a sensibilidade para um neurônio j no ponto p é $\delta_j^p \stackrel{\text{def}}{=} \frac{\partial O}{\partial net_j^p}$, a sensibilidade para a camada uma camada intermediária h é definida por:

$$\delta_h^p = \theta_h'(net_h^p)\left(\sum_{k=1}^{K}\delta_k^p w_{hk} + \sum_{\substack{d=1: \\ p_{d<D_d-1}}}^{n}\sum_{h'=1}^{H}\delta_{h'}^{p_d^+} w_{hh'}^d\right)$$

Se houvesse apenas uma dimensão ($n = 1$) a equação anterior seria reduzida à equação de sensibilidade de redes neurais recorrentes descrita anteriormente.

2.4.2 Aplicando Multidimensionalidade à Arquitetura LSTM

A extensão do conceito de multidimensionalidade para redes neurais recorrentes da arquitetura LSTM define que, para n dimensões, são necessários n controles de memória. Como a memória de uma rede LSTM está relacionada à conexão recorrente da célula de informação a porta de esquecimento, são necessárias n conexões recorrentes e n portas de esquecimento.

É importante notar que há algumas peculiaridades no caso das LSTM multidimensionais. A porta de entrada ι está conectada à célula c com o mesmo peso $w_{c\iota}$ para todas as dimensões. Já a porta de esquecimento está conectada à célula c com um peso $w_{c(\phi,d)}$ para cada dimensão d. Levando em conta essas notações é possível transformar as equações da arquitetura LSTM para LSTM multidimensionais como mostrado a seguir.

- **Fase *forward***

A entrada líquida e a função de saída da porta de entrada de uma célula são calculadas pelas equações:

$$net_\iota^p = \sum_{i=1}^{I} x_i^p w_{i\iota} + \sum_{\substack{d=1;\\ p_d>0}}^{n} \left(w_{c\iota} s_c^{p_{\bar{d}}} + \sum_{h=1}^{H} y_h^{p_{\bar{d}}} w_{h\iota}^d \right)$$

$$y_\iota^p = f\left(net_\iota^p\right)$$

Para as portas de esquecimento tem-se:

$$net_{(\phi,d)}^p = \sum_{i=1}^{I} x_i^p w_{i(\phi,d)} + \sum_{\substack{d'=1;\\ p_{d'}>0}}^{n} \sum_{h=1}^{H} y_h^{p_{\bar{d}}} w_{h(\phi,d)}^{d'} + \begin{cases} w_{c(\phi,d)} s_c^{p_{\bar{d}}}, & p_d > 0 \\ 0, & CC \end{cases}$$

$$y_{(\phi,d)}^p = f\left(net_{(\phi,d)}^p\right)$$

A partir dos impulsos da porta de esquecimento e da porta de entrada tem-se a entrada da célula de informação do bloco. E a partir desta entrada é calculado estado atual da célula como descrito nas equações a seguir.

$$net_c^p = \sum_{i=1}^{I} x_i^p w_{ic} + \sum_{\substack{d=1;\\ p_d>0}}^{n} \sum_{h=1}^{H} y_h^{p_{\bar{d}}} w_{hc}^d$$

$$s_c^p = y_i^p g\left(net_c^p\right) + \sum_{\substack{d=1:\\ p_d>0}}^{n} s_c^{p_{\bar{d}}} y_{(\phi,d)}^p$$

Para as portas de saída:

$$net_\omega^p = \sum_{i=1}^{I} x_i^p w_{i\omega} + \sum_{\substack{d=1;\\ p_{d'}>0}}^{n} \sum_{h=1}^{H} y_h^{p_{\bar{d}}} w_{h\omega}^d + w_{c\omega} s_c^p$$

$$y_\omega^p = f\left(net_\omega^p\right)$$

Sendo assim a saída da célula é calculada pela equação:

$$y_c^p = y_\omega^p h\left(s_c^p\right)$$

- **Fase *backward***

Sabendo que $e_c^p \stackrel{\text{def}}{=} \frac{\partial O}{\partial y_c^p}$ e $e_s^p \stackrel{\text{def}}{=} \frac{\partial O}{\partial s_c^p}$ são obtidas as equações para a fase *backward*:

$$e_c^p = \sum_{k=1}^{K} \delta_k^p w_{ck} + \sum_{\substack{d=1: \\ p_d < D_d - 1}}^{n} \sum_{h=1}^{H} \delta_h^{p_d^+} w_{ch}^d$$

A sensibilidade da porta de saída pode ser calculada como:

$$\delta_\omega^p = g'\left(net_\omega^p\right) e_c^p h\left(s_c^p\right)$$

O erro do estado é dado por:

$$e_s^p = y_\omega^p h'\left(s_c^p\right) e_c^p + \delta_\omega^p w_{c\omega} + \sum_{\substack{d=1: \\ p_d < D_d - 1}}^{n} \left(e_s^{p_d^+} y_{(\phi,d)}^{p_d^+} + \delta_\iota^{p_d^+} w_{c\iota} + \delta_{(\phi,d)}^{p_d^+} w_{c(\phi,d)}\right)$$

A sensibilidade da célula e da porta de esquecimento são calculados através das equações:

$$\delta_c^p = y_\iota^p g'\left(net_c^p\right) e_s^p$$

$$\delta_{(\phi,d)}^p = \begin{cases} g'\left(net_{(\phi,d)}^p\right) s_c^{p_{\bar{d}}} e_s^p, & p_d < 0 \\ 0, & caso\ contrário \end{cases}$$

Por fim, a sensibilidade da porta de entrada pode ser calculada por:

$$\delta_\iota^p = f'\left(net_\iota^p\right) g\left(net_c^p\right) e_s^p$$

Na seção seguinte será explicado o uso do código baseado em redes neurais LSTM multidimensionais e multidirecionais que fundamenta este trabalho. Essa arquitetura possui grande capacidade de memorização (LSTM), tem acesso completo ao contexto (multidirecionalidade) e pode tratar imagens diretamente, sem que sejam transformadas em dados unidimensionais (multidimensionalidade).

2.5 Visão Geral da *rnnlib*

Quando publicado o artigo sobre o reconhecimento de escrita cursiva usando MDRNNs, os autores também disponibilizaram a implementação usada, chamada

rnnlib [3]. Esta seção descreverá o funcionamento básico desse código. Esse conhecimento é necessário para compreender o que será apresentado nos próximos capítulos.

2.5.1 Entradas

A *rnnlib* realiza tarefas como treinamento e classificação a partir de um conjunto de dados de entrada. Como foi criada para tratar problemas diferentes, como reconhecimento de fonemas e reconhecimento de escrita cursiva, todo tipo de dado é convertido para um formato de dados padrão antes de entrar no fluxo do sistema.

Esse gerenciamento de tipos de dados é feito utilizando *netCDF* [7], um conjunto de bibliotecas que permite a criação, o acesso e o compartilhamento de dados científicos. O acesso e a representação são baseados em arranjos e independente da máquina.

Como o algoritmo é de aprendizado supervisionado, dentro desses arquivos também devem estar os rótulos para cada elemento que será usado como entrada. Neste trabalho é considerada apenas a aplicação para reconhecimento de escrita cursiva. Sendo assim, associado a cada imagem da palavra que será usada no sistema, deve haver um rótulo.

2.5.2 Rótulos

O formato como é escrito o rótulo varia de acordo com o tipo de treinamento utilizado. A *rnnlib* oferece treinamentos por dicionário e alfabeto. No primeiro, o rótulo de uma entrada descreve a palavra inteira e a rede criada a partir dele irá fazer o rotulamento avaliando toda a palavra. No treinamento por alfabeto, o rótulo especifica cada letra presente na palavra de entrada. Dessa forma a rede tentará realizar o reconhecimento a partir dos caracteres da imagem de entrada.

Um rótulo é descrito por um arquivo com extensão "*.tru*" que contém o mesmo do arquivo de entrada. O conteúdo destes arquivos é, por exemplo, "*LBL:palavra;*", para redes treinadas por dicionário, e "*LBL:p|a|l|a|v|r|a|;*" para redes treinadas por alfabeto.

Os elementos presentes nos rótulos são listados em um arquivo de dicionário usado na execução das tarefas da *rnnlib*. Nele devem conter todas as palavras de rótulos, no caso de treinamento por dicionário, ou todas as letras que os rótulos possuem, no caso de treinamento por alfabeto.

2.5.3 Execução

A definição da tarefa que será desempenhada pela *rnnlib* é descrita em um arquivo de configuração. Neste arquivo estão especificados a tarefa (treinamento ou classificação), os arquivos de entrada e alguns parâmetros de configuração da rede.

Para realizar a tarefa de treinamento são usados três arquivos *netCDF* que contém os dados de treinamento, validação e teste. O arquivo de treinamento contém o conjunto de informações usadas para construir a estrutura da rede. O conjunto de validação é usado para otimizar qualidade de classificação da rede. O conjunto de testes é composto por elementos independentes usados apenas para avaliar o desempenho da rede, sem interferir em sua estrutura.

Ao fim de cada ciclo são calculadas métricas de classificação temporal que definem se os pesos devem sofrer reajuste. O treinamento termina quando a rede satisfizer a medida de desempenho requerida.

A tarefa de classificação depende de um arquivo *netCDF* contendo os dados de entrada, e uma rede previamente treinada para classifica-los. Independente do tipo de treinamento usado, a *rnnlib* responde rotulando as entradas.

No Capítulo 3 será apresentada a abordagem usada para construir uma biblioteca que utiliza a *rnnlib*. Por sua vez, a partir dessa biblioteca, serão construídas as redes que realizarão as análises propostas neste trabalho.

Capítulo 3

Reconhecimento de Escrita Cursiva

Neste capítulo é descrita a abordagem utilizada para a criação de uma biblioteca chamada *easyLabel*, que visa facilitar o uso e a integração das funcionalidades das redes LSTM multidimensionais.

Outra importante abordagem deste trabalho é a avaliação do classificador criado a partir do código estudado. Neste capítulo também é descrita a metodologia usada para analisar os fatores que influenciam na qualidade do reconhecimento de escrita cursiva usando a biblioteca *easyLabel*.

3.1 Biblioteca *easyLabel*

A biblioteca tem como objetivo tornar transparente ao seu usuário o funcionamento do algoritmo das MDRNNs. Sendo assim foram definidos como requisitos para sua construção:

- O sistema deve oferecer uma interface simples e direta para treinamento. O usuário deve fornecer um conjunto de imagens separadas em pastas que serão usadas como rótulos e definir o tipo de treinamento. Sendo notificado ao fim da criação da rede para a base especificada.
- O sistema deve oferecer uma interface simples e direta para classificação. O usuário deve fornecer uma imagem ou conjunto de imagens a serem classificadas e obter como resposta os rótulos que foram atribuídos a cada uma.
- O sistema deve ser desenvolvido em Java, de maneira que, ao portar a *rnnlib*, originalmente desenvolvida em Linux, para outro sistema operacional, ele mantenha seu funcionamento sem haver necessidade de modificação.

Para a execução das tarefas de treinamento e classificação usando a *rnnlib* são necessárias duas etapas. A etapa de pré-processamento prepara os dados para o formato esperado pela *rnnlib*. A etapa de processamento a executa e traduz a

resposta final para o usuário. Essas etapas formam os dois módulos presentes na biblioteca que podem ser usados independentemente.

3.1.1 Módulo de Pré-processamento

Neste módulo uma ou mais imagens são transformadas em arquivos *netCDF*, entradas esperadas pela *rnnlib*. Também é criado um arquivo de configurações contendo todos os parâmetros necessários para realizar a tarefa solicitada. Esse arquivo é a saída do módulo de pré-processamento que pode ser usada no módulo de processamento ou diretamente na *rnnlib*.

Como foi descrito anteriormente, a execução da *rnnlib* requer um arquivo de configuração que determina os parâmetros que serão usados. Este módulo também é responsável por traduzir a opção do usuário para a *rnnlib*. A Figura 12 ilustra seu modelo computacional. A seguir são descritas suas principais operações.

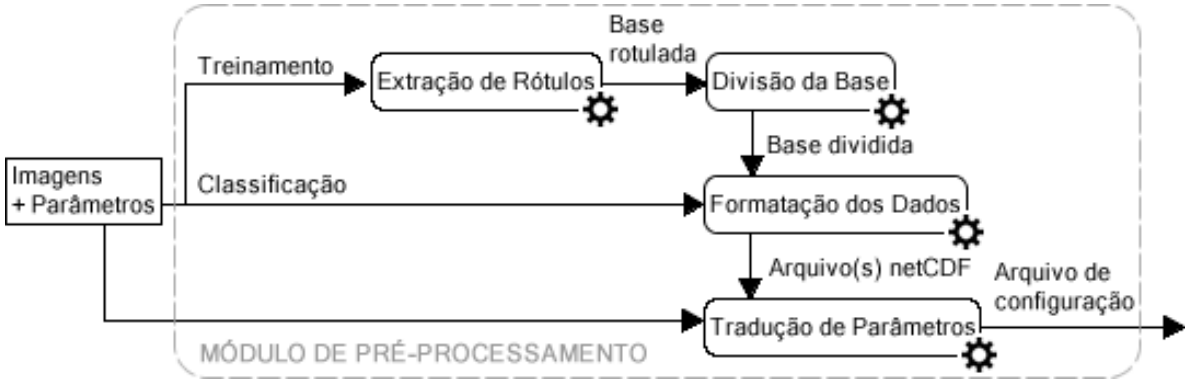

Figura 12. Modelo computacional do módulo de pré-processamento.

- **Tradução de Parâmetros**

É este elemento que cria o arquivo de configurações que ditará o funcionamento da *rnnlib* de acordo com os parâmetros do usuário. Seu papel é fundamental para permitir a transparência do funcionamento da *rnnlib* para o usuário.

Para realizar um treinamento com a *rnnlib*, por exemplo, seria preciso criar um arquivo de configuração com a tarefa "*transcription*", especificar os conjuntos de treinamento, teste e validação, dentre outros parâmetros. Graças à tradução de parâmetros o usuário precisa apenas indicar qual o tipo de tarefa e a rede que ela

tem como alvo. É possível treinar uma nova rede, continuar um treinamento, ou realizar uma classificação com no máximo três parâmetros.

- **Extração de Rótulos**

Este elemento assume que as imagens estão separadas por classes em pastas cujos nomes sejam os próprios rótulos. Ele então realiza a tarefa mecânica de extrair do nome das pastas os rótulos e criar os arquivos de rótulo para cada imagem da base fornecida.

Os rótulos extraídos também são usados para dar origem arquivo de dicionário que será usado pela rede para transcrição. Quando o treinamento for por dicionário, o próprio rótulo será incluso no dicionário criado. Já no treinamento por alfabeto, todas as letras dos rótulos extraídos são inclusos no arquivo de dicionário criado. Esse arquivo de dicionário reunirá todos os possíveis rotulamentos das entradas.

- **Divisão da Base**

Com a base devidamente rotulada, é feita a divisão entre treinamento, validação e teste, dividindo na proporção 50%, 25% e 25%, respectivamente. Este elemento cria cópias da base original e de seus rótulos que serão usados para a formatação dos dados na etapa seguinte.

- **Formatação dos Dados**

A formatação consiste apenas em transformar a imagem (e seus rótulos, caso a tarefa seja treinamento) em arquivos *netCDF*. Para esta tarefa são usados scripts em Python que acompanham o código original da *rnnlib*. Esses scripts organizam rótulos e imagens para serem inseridos em um arquivo *netCDF* com o uso de suas bibliotecas.

3.1.2 Módulo de Processamento

Este módulo é bem mais simples que o primeiro, mas não menos importante. Sua separação é para garantir que sejam usadas entradas criadas fora da biblioteca e que sejam criadas entradas para uso posterior, permitindo maior liberdade ao usuário.

Ele é constituído de um processo que executa a *rnnlib* para um determinado arquivo de configuração. Para treinamento, este módulo notifica o usuário quando o processo tiver fim e fornece o destino para a rede final. No caso de classificação, os rótulos atribuídos às entradas são interpretados pela biblioteca e fornecidos ao usuário em uma estrutura de dados.

Com o uso da *easyLabel*, a dificuldade de usar MDRNNs em um outro sistema se resume a importar a biblioteca e executar os métodos de treinamento e classificação. São necessários três parâmetros no caso de treinamento e dois no caso de classificação:

- *EasyLabel.train(imgDir, netPath, useDictionary)* – Onde *imgDir* é o diretório onde está a base de imagens que será usada para treinamento, *netPath* é o caminho para a nova rede que será criada ou para uma rede que já exista e deva ser atualizada, e *useDictionary* é um parâmetro booleano que especifica se o treinamento deve ser feito por dicionário (verdadeiro) ou por alfabeto (falso).
- *EasyLabel.classify(input, netPath)* – Onde *input* é o caminho para uma imagem ou para um diretório de várias imagens que devem ser classificadas pela rede que está no caminho *netPath*. No caminho da rede também deve estar o dicionário que deve ser usado para o rotulamento. Quando os dois módulos da *easyLabel* são usados, o usuário não precisa se preocupar com este detalhe. Este método responde com um mapeamento arquivo-rótulo em que a chave é um *String* que descreve o caminho original do arquivo, e o valor é um *String* contendo o rótulo que obteve maior escore na classificação.

3.1.3 Usabilidade

A facilidade de integração da *easyLabel* permite que os sistemas que a utilizem também sejam fáceis de usar pelo usuário final. O cenário mais simples é quando a aplicação apenas expõe as funcionalidades da biblioteca em uma interface gráfica.

A Figura 13 mostra a tela inicial de um aplicativo que utiliza a *easyLabel*. Neste caso ele está configurado para treinar uma rede. Como a opção '*Update if network exists.*' está selecionada, se já existir uma rede no diretório

‘*/home/easyLabel/network’*, ela será atualizada para tratar as novas entradas fornecidas.

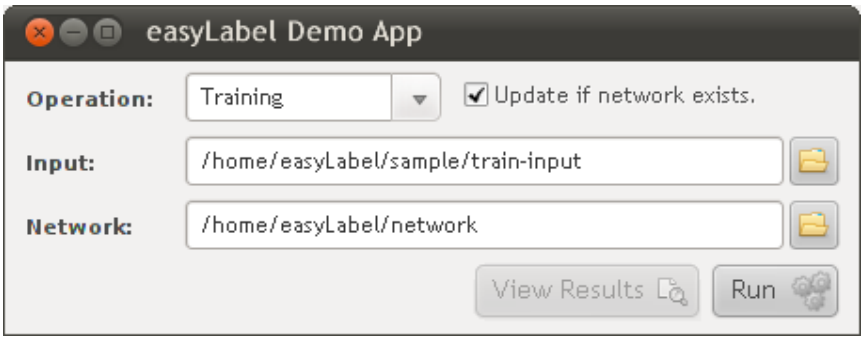

Figura 13. Aplicativo de demonstração configurado para treinamento.

O diretório escolhido como entrada para o treinamento deve conter os arquivos separados em outros diretórios, de acordo com os rótulos que devem receber. Na Figura 14, por exemplo, os arquivos dentro do diretório ‘abril’ serão rotulados com este nome.

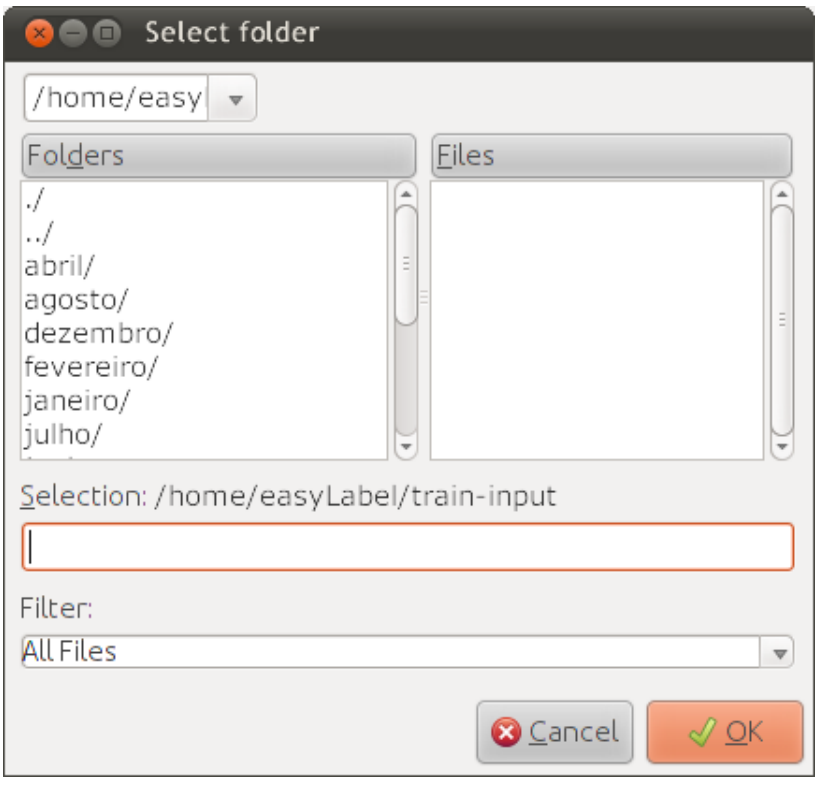

Figura 14. Diretório de entrada para treinamento.

O usuário pode visualizar o resultado do treinamento quando ele chegar ao fim. Esse resultado é um arquivo de texto contendo o tempo gasto no treinamento e o erro final do treinamento.

Com uma rede construída, o usuário pode rotular uma ou várias imagens. Ao selecionar a opção de '*Classification*' o sistema irá usar a rede especificada no campo '*Network*' para fazer a classificação. Um exemplo de seleção de um diretório de uma rede existente é ilustrado pela Figura 15.

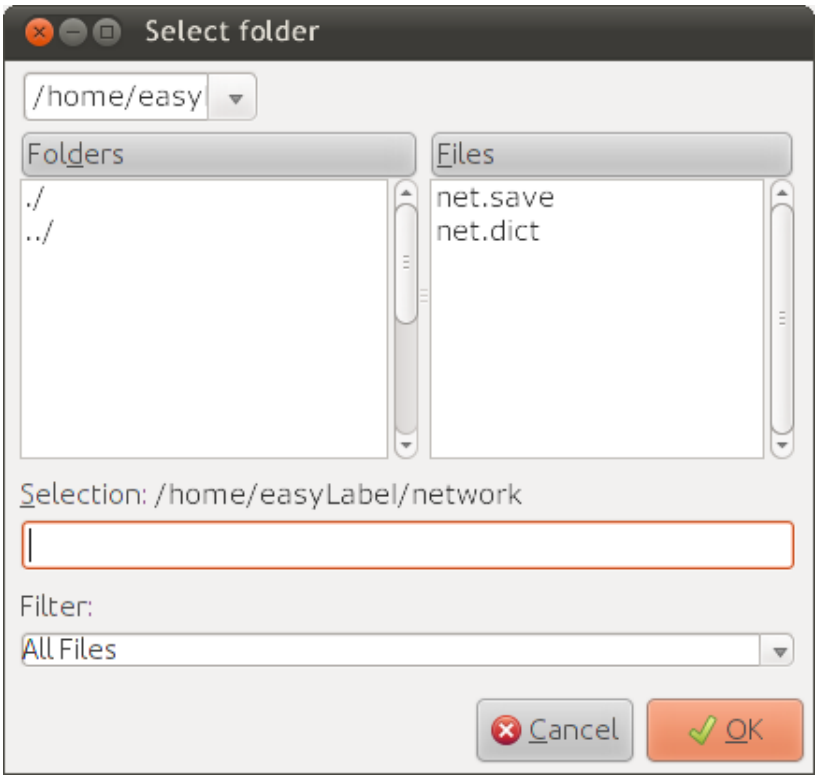

Figura 15. Seleção de um diretório que contém uma rede já treinada.

Ao fim da classificação o usuário pode visualizar, em um arquivo de texto, uma lista contendo o caminho completo para cada arquivo e o rótulo que lhe foi atribuído. A Figura 16 mostra o sistema quando a classificação termina.

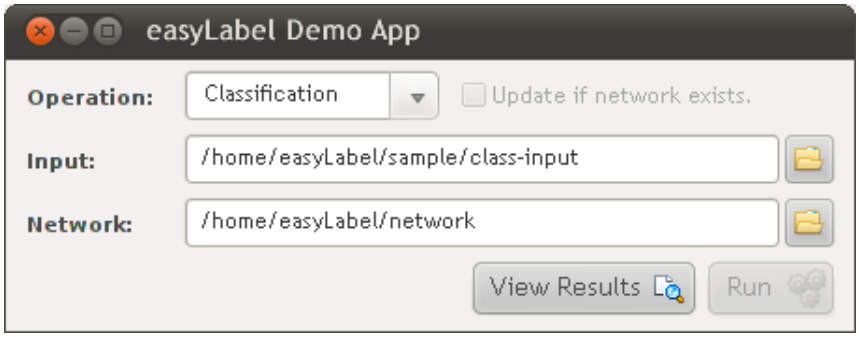

Figura 16. Aplicativo de demonstração ao fim de uma classificação.

Essa simples interface do aplicativo de demonstração é suficiente para fornecer ao usuário as funções da *easyLabel*, *rnnlib*. Consequentemente ele poderá usufruir do rotulamento de sequências das redes neurais LSTM multidimensionais.

3.2 Metodologia de Análise

Para avaliar a qualidade de reconhecimento o sistema foi testado em um cenário real. A digitalização de cheques bancários é um dos grandes problemas atuais de reconhecimento de escrita cursiva. Há diversos campos que variam de números até a assinatura do cliente. O campo 'mês' é um dos que possui características de escrita cursiva, como mostra a Figura 17.

Figura 17. Campo 'mês' de um cheque bancário em destaque.

O uso da *easyLabel* torna todo o processo de criação da rede transparente para o usuário. No caso de treinamento por dicionário, os meses são rotulados a partir de doze classes, uma para cada mês. No treinamento por alfabeto a classificação é feita baseada nas letras que formam os nomes dos meses. Para

usufruir da extração de rótulos suportada pela *easyLabel* é necessário colocar toda a base em um diretório separando as imagens de cada mês em um sub-diretório com o nome do mês.

As imagens usadas neste trabalho foram cedidas pela empresa *Stefanini Document Solutions*. Foram mil imagens para cada mês, proveninentes de digitalizações de cheques de diversos bancos brasileiros. A *easyLabel* divide aleatoriamente essa base nas proporções 50% treinamento, 25% validação e 25% teste.

Usar um cenário real permite a avaliação da qualidade do reconhecimento realizado. Mas também é importante avaliar quais os fatores que afetam essa qualidade. Neste trabalho foi estudada a influência dos parâmetros de treinamento: tipo de treinamento e diferença de resolução entre as imagens utilizadas para treinamento e para classificação.

Várias redes deverão ser criadas para fazer as análises. A influência do tipo de treinamento pode será avaliada comparando o resultado da classificação de duas redes criadas a partir da mesma base. Uma delas com treinamento por dicionário e outra por alfabeto. Já o impacto da diferença de resolução será comparado para os dois tipos de treinamento. Usar as redes criadas para classificar uma mesma base de teste com resolução original, com resolução reduzida a 80% da original e com resolução ampliadas a 120% da original permitirão avaliar o impacto desse fator.

São necessárias duas redes neurais para fazer a análise proposta neste trabalho. Para facilitar a visualização dos resultados será usada a notação: *tipo de treinamento_resolução* para identificar os resultados das classificações:

- Rotulamento de imagens a 80% da resolução original usando rede treinada por dicionário (*dicionário_reduzida*);
- Rotulamento de imagens na resolução original usando rede treinada por dicionário (*dicionário_original*);
- Rotulamento de imagens a 120% da resolução original usando rede treinada por dicionário (*dicionário_ampliada*);

- Rotulamento de imagens a 80% da resolução original usando rede treinada por alfabeto (*alfabeto_reduzida*);
- Rotulamento de imagens na resolução original usando rede treinada por alfabeto (*alfabeto_original*);
- Rotulamento de imagens a 120% da resolução original usando rede treinada por alfabeto (*alfabeto_ampliada*).

A comparação entre os resultados é feita observando tabelas de resultados. As tabelas apresentam a rotulação feita para cada classe em matrizes de confusão. Em um classificador ideal todos os resultados iriam convergir na diagonal principal. Isso significaria que todas as classes foram rotulada corretamente.

Para uma classe, a célula da diagonal principal da linha contém a quantidade de verdadeiro positivos, TP (*true positives*). As outras células da mesma linha contêm os falso negativos, FN (*false* negatives), e as células da mesma coluna contêm os falso positivos, FP (*false positives*). Células da diagonal principal de outras linhas representam os verdadeiro negativos, TN (*true negatives*).

- TP – o rótulo esperado foi atribuído à imagem;
- FN – foi atribuído um rótulo diferente do esperado para a imagem;
- TN – a imagem não pertence à classe avaliada e não recebeu seu rótulo;
- FP – a imagem pertence à classe avaliada mas não foi rotulada corretamente.

Para destacar informações importantes sobre os classificadores, as células das tabelas são destacadas em azul para representar a convergência, e vermelho para representar a maior confusão para a classe. Para cada tabela de resultados é calculada a taxa de acerto absoluto. O cálculo consiste em dividir o número de acertos totais relatados na matriz de confusão (soma dos valores da diagonal principal) pela quantidade total de elementos apresentados. A taxa de acerto de cada classe pode ser calculada dividindo o valor da diagonal principal pelo valor total de exemplos apresentados para a classe em questão.

No capítulo seguinte são apresentados os resultados de experimentos baseados na abordagem descrita anteriormente.

Capítulo 4

Experimentos e Resultados

Com o objetivo de avaliar o desempenho da classificação de escrita cursiva utilizando MDRNNs, a *easyLabel* foi testada em um cenário de uso real. Neste capítulo serão apresentados os resultados da classificação de imagens de meses em cheques bancários brasileiros seguindo a metodologia apresentada no capítulo anterior.

Dois classificadores criados com a *easyLabel* foram usados para rotular um mesmo conjunto de 1200 imagens, divididas em cem para cada classe. Essas imagens não participaram da construção dos classificadores. Na Seção 4.1 serão apresentadas as matrizes de confusão para os resultados dos classificadores treinados usando o tipo de treinamento por alfabeto. Na Seção 4.2 estão os resultados dos classificadores treinados dicionário. Por fim, a Seção 4.3 reunirá as principais inferências que podem ser feitas a partir dos resultados obtidos.

4.1 Classificadores Treinados por Alfabeto

Na Tabela 1 estão os resultados do rotulamento feito pela rede treinada por alfabeto para um conjunto de imagens na resolução original. O rotulamento convergiu corretamente para todas as classes e obteve 71.25% de acerto absoluto.

Tabela 1. Matriz de confusão do resultado *alfabeto_original*.

	JAN	FEV	MAR	ABR	MAI	JUN	JUL	AGO	SET	OUT	NOV	DEZ
JAN	69	5	2	1	3	11	0	0	2	2	2	2
FEV	18	72	1	0	0	0	1	0	1	4	2	1
MAR	0	0	73	0	25	1	0	0	0	1	0	0
ABR	0	0	3	70	13	1	2	2	2	2	0	0
MAI	3	0	17	6	61	8	1	2	0	0	0	0
JUN	5	0	1	1	2	56	19	0	3	12	1	0
JUL	0	0	0	0	5	13	79	0	0	2	0	0
AGO	1	0	9	3	4	5	2	72	3	0	0	1
SET	3	0	0	1	2	1	1	0	84	3	0	5
OUT	0	1	2	0	1	7	5	1	8	73	1	1
NOV	2	3	4	0	1	0	2	2	6	3	70	7
DEZ	2	6	0	1	0	0	2	3	3	2	5	76

Para a rede treinada por alfabeto classificando imagens em resolução reduzida foram obtidos os resultados descritos na Tabela 2. A taxas acerto absoluto de 57.25% indica que a qualidade do rotulamento foi pior a de *alfabeto_original*.

Tabela 2. Matriz de confusão do resultado *alfabeto_reduzida*.

	JAN	FEV	MAR	ABR	MAI	JUN	JUL	AGO	SET	OUT	NOV	DEZ
JAN	66	1	7	0	9	11	0	0	4	1	1	0
FEV	32	46	0	0	6	1	0	0	3	2	2	8
MAR	1	0	52	4	38	2	1	0	0	1	0	0
ABR	0	0	1	67	22	2	0	0	0	0	0	0
MAI	0	0	15	4	64	7	3	1	0	0	1	0
JUN	6	0	3	5	3	54	17	3	0	7	0	1
JUL	2	0	3	1	9	21	60	0	0	2	0	0
AGO	0	0	5	4	10	6	3	67	2	0	1	1
SET	6	0	5	8	5	9	2	3	51	3	1	6
OUT	1	1	7	5	9	7	6	1	6	53	4	0
NOV	7	0	9	0	6	3	1	2	8	9	52	2
DEZ	8	3	0	2	2	3	0	9	13	2	3	55

Apesar de os resultados terem convergido corretamente para todas as classes houveram grandes confusões na classificação. Para a classe ‘marco’ por exemplo, 38% de suas imagens foram rotuladas como ‘maio’. A classe ‘fevereiro’

teve uma taxa de acerto menor que 50%. Esse resultado indica que, para esta base de testes, o classificador se comportou como um classificador aleatório.

O resultado *alfabeto_ampliada* foi ligeiramente melhor que o *alfabeto_reduzida* como descrito na Tabela 3. O acerto absoluto foi de 61.08%.

Tabela 3. Matriz de confusão do resultado *alfabeto_ampliada*.

	JAN	FEV	MAR	ABR	MAI	JUN	JUL	AGO	SET	OUT	NOV	DEZ
JAN	50	22	1	2	2	12	1	2	1	1	4	2
FEV	10	75	1	0	1	1	0	0	2	3	3	3
MAR	1	1	45	3	34	8	1	0	1	4	1	0
ABR	0	0	6	61	17	2	2	3	5	3	0	0
MAI	6	1	21	5	52	8	2	2	0	1	1	0
JUN	3	1	0	0	2	63	14	3	2	10	1	1
JUL	0	0	0	1	4	26	65	0	0	4	0	0
AGO	3	0	3	3	9	8	2	55	2	3	5	7
SET	3	1	0	2	2	5	3	0	63	8	6	7
OUT	0	4	3	1	1	5	6	0	5	62	11	2
NOV	2	6	3	0	1	1	1	0	0	4	74	8
DEZ	2	11	0	0	0	0	0	3	2	3	11	68

Assim como no resultado *alfabeto_reduzida* teve taxas de acerto muito baixas para algumas classes.

4.2 Classificadores Treinados por Dicionário

A Tabela 4 reúne os resultados do rotulamento para a rede treinada por dicionário classificando imagens na resolução original. Esse foi o melhor resultado de classificação com taxa de acerto de 77.25%.

Tabela 4. Matriz de confusão do resultado *dicionário_original*.

	JAN	FEV	MAR	ABR	MAI	JUN	JUL	AGO	SET	OUT	NOV	DEZ
JAN	75	5	3	0	2	4	0	1	2	1	6	1
FEV	17	68	0	0	1	1	0	0	1	5	6	1
MAR	0	1	71	2	16	4	2	0	1	1	2	0
ABR	0	0	4	78	9	3	1	2	2	1	0	0
MAI	1	1	11	3	80	2	0	1	1	0	0	0
JUN	3	1	1	1	0	80	7	2	2	2	0	1
JUL	0	0	0	1	3	18	75	0	0	3	0	0
AGO	3	0	0	1	4	9	2	68	0	2	1	5
SET	1	0	0	0	1	3	0	0	90	2	1	2
OUT	0	1	1	0	2	3	3	0	6	84	1	0
NOV	0	1	1	0	0	0	0	1	2	3	89	1
DEZ	1	3	3	0	1	1	0	2	12	1	7	69

O resultado da classificação feita pela rede treinada por dicionário de imagens reduzidas a 80% da resolução original obteve os resultados descritos na Tabela 5.

Tabela 5. Matriz de confusão do resultado *dicionário_reduzida*.

	JAN	FEV	MAR	ABR	MAI	JUN	JUL	AGO	SET	OUT	NOV	DEZ
JAN	61	3	1	0	14	6	2	2	7	1	0	3
FEV	26	43	3	0	2	4	0	0	8	2	3	9
MAR	1	0	41	3	43	1	2	2	1	3	1	2
ABR	3	0	3	78	13	1	2	0	0	0	0	0
MAI	6	0	7	3	72	2	5	3	1	1	0	0
JUN	10	0	1	6	9	40	18	4	5	6	0	1
JUL	4	0	0	6	7	9	69	2	3	0	0	0
AGO	1	0	5	7	11	8	5	59	3	1	0	0
SET	6	0	1	4	9	5	4	1	60	1	2	7
OUT	1	2	9	8	5	5	4	1	18	44	1	2
NOV	5	0	10	2	11	6	0	2	14	10	30	10
DEZ	3	2	5	0	6	5	1	16	5	1	3	53

Com uma taxa de acerto absoluto de apenas 54.17% e apresentando convergências para rótulos errados e baixas taxas de acerto individuais, esse resultado mostra a maior sensibilidade, das redes treinadas por dicionário, a redução na resolução das imagens.

O resultado *dicionário_ampliada* também não possui boas taxas. A Tabela 6 descreve sua matriz de confusão em que pode ser observada a taxa de acerto absoluto de 51.75%, pior entre os demais resultados.

Tabela 6. Matriz de confusão do resultado *dicionário_ampliada*.

	JAN	FEV	MAR	ABR	MAI	JUN	JUL	AGO	SET	OUT	NOV	DEZ
JAN	44	26	3	0	0	2	0	3	8	1	1	12
FEV	9	75	0	0	0	2	1	0	4	0	0	9
MAR	1	4	39	2	17	5	3	1	10	11	5	2
ABR	1	1	3	70	6	2	6	4	2	3	1	1
MAI	5	2	24	11	37	6	3	4	0	2	5	1
JUN	6	4	4	6	2	33	5	11	6	14	2	6
JUL	2	1	1	8	1	22	52	2	4	1	0	0
AGO	3	0	5	2	2	8	3	41	5	5	7	19
SET	3	2	1	4	0	8	2	0	55	7	7	10
OUT	0	10	2	2	1	5	2	1	10	56	7	3
NOV	0	6	7	0	0	2	0	0	9	5	52	19
DEZ	2	10	1	1	0	4	0	2	5	1	7	67

4.3 Compreensão dos Resultados

A melhor taxa de acerto absoluta foi de 77.25%. A rede treinada por dicionário no cenário em que as imagens estavam na mesma resolução que as usadas para treinamento comprovaram a eficiência da solução.

Apesar da taxa de acerto absoluta menor para as redes treinadas por alfabeto, elas provaram ser menos sensíveis a diferenças na resolução da imagem. Por esse motivo o treinamento por alfabeto é mais adequado quando não há controle sobre a resolução das imagens usadas no treinamento e classificação.

O classificador criado usando dicionário é ideal quando há controle sobre as imagens que serão usadas para treinamento e classificação. Apesar de ser mais sensível a mudanças de resolução, as redes treinadas por dicionário apresentam resultados melhores quando usadas para rotular imagens da mesma resolução que as usadas para seu treinamento.

No próximo e último capítulo são feitas as considerações finais sobre este trabalho, o que foi alcançado e o que pode ser melhorado.

Capítulo 5

Conclusões e Trabalhos Futuros

Este capítulo reúne as contribuições alcançadas com este trabalho e sugere possíveis continuações.

5.1 Contribuições

Para avaliar o quão eficiente é o reconhecimento de escrita cursiva usando MDRNNs foi desenvolvido um sistema que facilita a integração e o uso desse poderoso algoritmo. A *easyLabel* possibilita que qualquer sistema possível de utilizar bibliotecas Java faça uso das do rotulamento realizado pelas MDRNNs. A facilidade para integração, uso e demonstração proporciona a aproximação da sociedade e da academia a este tipo de solução. Outras aplicações, análises sobre a classificação e melhorias poderão ser elaborados com maior facilidade.

A eficiência do reconhecimento do campo de meses do ano em cheques bancários comprovou a eficiência da abordagem de MDRNNs para o reconhecimento de escrita cursiva. O estudo sobre a influência da alteração de resolução das imagens de treinamento e de classificação permitiu identificar para quais situações cada tipo de treinamento é mais adequado. Redes treinadas por dicionário têm maior taxa de acerto, mas são mais sensíveis a mudanças na resolução das imagens. O oposto acontece com as treinadas por alfabeto. Outras análises podem ser feitas a fim de identificar outros casos em que os tipos de treinamento se comportam de maneira diferente.

5.2 Trabalhos Futuros

Com o desenvolvimento do trabalho surgiram idéias para continuação tanto sobre a biblioteca quando sobre a análise da classificação:

- Portar a biblioteca para o sistema operacional Microsoft Windows. Essa tarefa requer alterar a maneira como são criados os arquivos *netCDF* no módulo de pré-processamento. O principal objetivo é que o código central da *rnnlib* possa ser usada tanto em Windows quanto em Linux.
- Estudar a qualidade do reconhecimento de escrita cursiva quando a rede é treinada a partir de uma base artificial de imagens.
- Identificar a influência de outras propriedades da imagem sobre a qualidade de classificação.

Bibliografia

[1] DE CARVALHO, J. V.; SAMPAIO, M. C.; MONGIOVI, G. **Utilização de Técnicas de "*Data Mining*" para o Reconhecimento de Caracteres Manuscritos**. Universidade Federal da Paraíba. Campina Grande, PB. 2000.

[2] GRAVES, A. ***Supervised Sequence Labelling with Recurrent Neural Networks****. Technischen Universität München*. Munique, Alemanha. 2008.

[3] GRAVES, A. *Github Social Coding*. ***rnnlib***, 2009. Disponivel em: <https://github.com/meierue/RNNLIB>. Acesso em: 02 Fevereiro 2011.

[4] GRAVES, A.; FERNÁNDEZ, S.; SCHMIDHUBER, J. ***Multi-Dimensional Recurrent Neural Networks****. Istituto Dalle Molle di Studi sull'Intelligenza Artificiale.* Manno, Suiça. 2007. (IDSIA-04-07).

[5] GRAVES, A.; SCHMIDHUBER, J. ***Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks****. International Conference on Document Analysis and Recognition.* Washington, DC, EUA: [s.n.]. 2009.

[6] HOCHREITER, S.; SCHMIDHUBER, J. *Long Short-Term Memory.* ***Neural Computation****,* 15 Novembro 1997. 1735-1780.

[7] UNIDATA PROGRAM CENTER. NetCDF (Network Common Data Form). **Site da Unidata**. Disponivel em: <http://www.unidata.ucar.edu/software/netcdf/>. Acesso em: 5 Maio 2011.

[8] VALENÇA, M. J. S. **Fundamentos das Redes Neurais**. 2ª. ed. Olinda, PE: Livro Rápido, 2010.