CENTRO UNIVERSITÁRIO DE BRASÍLIA -UniCEUB

CURSO DE ENGENHARIA DE COMPUTAÇÃO

LUCAS MESQUITA NOGUEIRA

AUTENTICAÇÃO BIOMÉTRICA EM REDES DE ACESSO RESTRITO

Orientador: Prof. Francisco Javier De Obaldía

Brasília

Dezembro, 2012

LUCAS MESQUITA NOGUEIRA

AUTENTICAÇÃO BIOMÉTRICA EM REDES DE ACESSO RESTRITO

Trabalho apresentado ao Centro Universitário de Brasília (UniCEUB) como pré-requisito para a obtenção de Certificado de Conclusão de Curso de Engenharia de Computação.

Orientador: Prof.MsC Francisco Javier de Obaldía

Brasília

Novembro, 2012

LUCAS MESQUITA NOGUEIRA

AUTENTICAÇÃO BIOMÉTRICA EM REDES DE ACESSO RESTRITO

Trabalho apresentado ao Centro Universitário de Brasília (UniCEUB) como pré-requisito para a obtenção de Certificado de Conclusão de Curso de Engenharia de Computação.
Orientador: Prof. MsC Francisco Javier de Obaldía

Este Trabalho foi julgado adequado para a obtenção do Título de Engenheiro de Computação, e aprovado em sua forma final pela Faculdade de Tecnologia e Ciências Sociais Aplicadas - FATECS.

_____

Prof. Abiezer Amarilia Fernandes
Coordenador do Curso

Banca Examinadora:

_____

Prof. Francisco Javier de Obaldía, Mestre.
Universidade de Brasília
Orientador

_____

Prof. Flávio Antonio Klein, Mestre.
Universidade de Brasília

_____

Prof. Leonardo Pol Suarez, Mestre.
Universidade Católica de Brasília

DEDICATÓRIA

Dedico esta monografia a todos os engenheiros que desenvolveram o interesse pela teoria da probabilidade e o aprendizado de máquinas. Que as próximas grandes inovações surjam de nossas cabeças.

AGRADECIMENTOS

Agradeço aos meus professores pelo conhecimento transmitido, sem o qual não seria possível completar este projeto. Agradeço também a meus pais e irmãos pelo apoio incondicional.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS E ABREVIATURAS

DAV	Detecção da Ativação da Voz
EM	*Expectation Maximization*
FFT	*Fast Fourier Transform*
FIR	*Finite Ipulse Response*
HMM	*Hidden Markov Model*
IDCT	*Inverse Discrete Time Cosine Transform*
RNA	Redes Neurais Artificiais
SSE	*Sum of Squared Error*

RESUMO

Este projeto final de engenharia consiste na implementação de um sistema de reconhecimento de voz. Visando extender o paradigma de digitação de senhas para a autenticação, cujas limitações e variedade de métodos de golpes com base neste paradigma motivaram este trabalho, o sistema de reconhecimento de voz proposto efetua a identificação de um usuário em duas etapas: uma de reconhecimento do emissor da voz em si (o locutor) e uma de reconhecimento do conteúdo do sinal vocal (a fala). A implementação destes dois módulos, feita usando a linguagem R de programação, é fundamentada em algoritmos de aprendizagem de máquina, particularmente as Redes Neurais Artificiais e o Modelo Oculto de Markov, onde a primeira classe de modelos é implementada completamente sem código de terceiros neste trabalho. Além destes dois módulos, também faz parte da implementação um módulo dedicado à extração de parâmetros de interesse do sinal de voz através da aplicação do método Mel-Cepstrum. O sistema obteve sucesso no reconhecimento em duas etapas, com uma taxa de acerto significativamente superior à de um método mais simples onde cadastros são selecionados aleatoriamente não importando, o sinal de voz de entrada.

Palavras Chave: Redes Neurais, Modelo Oculto de Markov, Mel-Cepstrum, Aprendizado de Máquina, Sinal de voz, Reconhecimento de voz, Autenticação.

ABSTRACT

This engineering final project consists on implementing a voice recognition system. Envisaging an extension to the password-typing autenthication paradigm, whose limitations and the variety of attack methods against which motivated this work, the voice recognition system herein proposed performs user identification in two stages: one for recognizing the source of the voice signal itself (the speaker) and one for the recognition of this signals contents (the speech). These two modules implementations, done using the R programming language, are grounded on machine learning algorithms, particularly Neural Netwroks and Markov Hidden Models, where the first class of models is implemented without any code from third parties in this work. Other than these two modules, the implementation of this system also includes a module for feature extraction by applying the Mel-Cepstrum method to a voice signal. The system successfully recognized users with the two stage method, having had a success rate significantly superior to that of a simples method where profiles are randomly selected without regard to the inputed signal.

CAPÍTULO 1 - INTRODUÇÃO

1.1 - Motivação

Redes de comunicação são uma parte integrante do modo em que interagimos com o mundo hoje. Pessoas ao redor do mundo as utilizam todos os dias, manipulando os mais variados tipos de informação: de notícias e jogos a transações bancárias, mensagens e dados profissionais. Alguns destes tipos de informação, como notícias em um portal na internet, têm, em geral, como objetivo, chegar ao maior número de pessoas possível; com pouca ou nenhuma restrição. Isto porém, não é verdade para várias outras aplicações das redes de comunicação. Obviamente, não é desejável, por exemplo, que os dados contidos em suas transações financeiras realizadas na internet caiam nas mãos de pessoas mal-intencionadas; ou que um concorrente tenha acesso à informações internas da sua empresa. Nestes casos, deseja-se restringir o acesso aos dados que transitam em uma rede. Uma maneira de se conseguir isto, é restringindo o acesso ao próprio ambiente que contém a informação, ou o recurso, que se deseja proteger; seja este ambiente o local físico onde estão os dados ou a rede onde eles trafegam.

Métodos como o uso de senhas e perguntas secretas existem para garantir esta restrição, ou exclusividade, no acesso à redes e seus recursos. Quando um usuário insere a sua senha em uma plataforma de internet banking, por exemplo, o sistema de segurança estará na verdade validando a alegação de que a identidade - fornecida mediante o número da conta- realmente pertence àquele usuário; baseando-se no fato que apenas ele deve ter conhecimento da própria senha e, assim, restingindo e garantindo exclusividade no acesso aos recursos daquela conexão de rede ao usuário em questão. No entanto, a idéia de verificar um parâmetro do qual somente um usuário – ou um grupo de usuários – tenha conhecimento não é nova: seitas e grupos secretos da antiguidade requeriam que uma senha fosse dita por alguém que quisesse entrar em um recinto como forma de identificar membros autorizados a participar de suas reuniões. Também não é nova, por outro lado, a idéia de roubar esta informação para obter acesso que, de outra forma, não seria autorizado. Por mais que existam formas cada vez mais sofisticadas de proteger senhas e restringir acesso, como a autenticação em duas camadas SecurID da RSA, não é impossível que indivíduos ou organizações mal-intencionados consigam obtê-las. Sendo assim, formas ainda mais personalizadas de securização, tais como RFID e identificação biométrica, surgem como alternativa às senhas

1.2 – Objetivo Geral do Trabalho

Neste contexto, este trabalho tem por objetivo apresentar uma implementação do paradigma biométrico de securização baseado no reconhecimento da voz do usuário. Com duas camadas de autenticação, o sistema proposto identifica o usuário, baseado em parâmetros do sinal de voz; e verifica se o conteúdo do que foi dito por ele coincide com os registros prévios: uma espécie de senha vocal, ao invés de escrita ou digitada.

1.3 – Objetivos Específicos do Trabalho

Tendo em vista a exposição de métodos e técnicas necessários à viabilização do objetivo de implementar o sistema descrito acima, este trabalho também apresenta uma descrição da adaptação de algoritmos e conceitos de *Machine Learning* e processamento de sinais ao problema específico do reconhecimento da fala, mais especificamente: modelos de redes neurais para classificação de usuários quanto à sua identidade; o modelo oculto de Markov para modelagem linguistica do sinal de voz; e técnicas de pré-processamento do sinal de voz, para obter as características que deverão ser *inputs* do sistema de reconhecimento da fala. Considerando esta descrição do sistema a ser implementado, é importante ressaltar que os testes envolvidos na execução do projeto foram realizados em um ambiente controlado, com um baixo nível de ruído. No entanto, a camada de pré-processamento do sinal de voz conta com etapas de filtragem, que visam minimizar a perda de qualidade no sinal e proporcionar o funcionamento em ambientes ruidosos.

1.4 – Justificativa e Relevância do Tema

A biometria como forma de autenticação oferece um nível personalizado de securização, chegando a oferecer, em alguns casos, parâmetros quase únicos para a identificação de um usuário qualquer, como é o exemplo da autenticação por impressões digitais. Neste sentido, a identificação por sinal de voz aparece como uma alternativa prática ao paradigma da verificação de identidades por meio da digitação senhas – cujos problemas serão discutidos mais adiante, no capítulo 2 -, por não necessitar de aparelhagens sofisticadas como em outros exemplos do paradigma biométrico. No futuro, iniciativas como a descrita neste trabalho podem ser extendidas para o desenvolvimento de comandos personalizados de voz, onde máquinas, como um carro por exemplo, ou serviços on-line, como o *internet-banking*, responderiam exclusivamente aos comandos ditos por um determinado usuário.

1.5 – Escopo do trabalho e Resultados Esperados

Dado o que foi dito acima, o escopo deste trabalho se limita a implementar um sistema capaz de realizar o reconhecimento da identidade e do conteúdo fala de um usuário sobre o qual a informação (sinal de voz) tenha sido previamente obtida e armazenada; não se extendendo de maneira alguma a oferecer formas eficientes de armazenamento destes dados, ou a comunicação - com uma rede real- do resultado efetivo da autenticação do usuário pelo sistema. O escopo deste projeto envolve, no entanto, a apresentação de evidências estatísticas, colhidas durante a etapa de testes, de que o sistema é efetivo na validação da identidade de um usuário. Isto será feito através de uma comparação com um sistema *baseline*, mais simples, onde um cadastro é escolhido aleatoriamente entre todos os disponíveis, não importando a natureza do sinal de voz obtido.

1.6 – Estrutura do Trabalho

No capítulo seguinte, o leitor encontrará uma breve descrição da solução proposta, assim como uma descrição superficial dos métodos que serão utilizados e de que forma eles se moldam a esta solução. O capítulo 3 contém a metodologia teórica por trás destas técnicas, indo desde suas definições aos detalhes que devem ser observados para uma implementação efetiva. É neste capítulo que o leitor encontrará os fundamentos matemáticos que compõem a solução final. No capítulo 4, uma descrição aprofundada da aplicação dos métodos ao problema de reconhecimento de voz será apresentada, além dos detalhes da implementação do sistema proposto. A discussão dos resultados obtidos, bem como a descrição das condições em que foram obtidos e a comparação estatística com o sistema *baseline* de seleção aleatória, será apresentada no capítulo 5. Finalmente, o capítulo 6 contém a conclusão baseada nos resultados expostos no capítulo 5 e sugestões para trabalhos futuros.

CAPÍTULO 2 - APRESENTAÇÃO DO PROBLEMA

É comum uso de senhas como parâmetro identificador de um usuário que deseje entrar em uma rede ou usar um serviço. O usuário cria uma sequência de caracteres válidos, que deverá se associar unicamente ao seu cadastro, baseado em qualquer lógica que ele desejar; podendo gerar esta sequencia aleatoriamente, usar a sua data de nascimento; palavras ou uma mistura de letras e caracteres especiais

Esta simples regra de negócio no entanto, não é, de forma alguma, nova ou exclusiva das redes de comunicação: sociedades secretas da antiguidade já usavam a idéia de um código secreto que, sendo conhecido somente por membros do grupo, identificava aqueles indivíduos que eram autorizados a entrar em um recinto ou participar de uma reunião. Também não são novas, nem exclusivas das redes, as formas de tentar obter estes códigos e senhas de maneira ilícita. No caso das redes de comunicação, existe uma variedade de golpes e métodos para se conseguir acesso à zonas e serviços restritos utilizando senhas de outros usuários sem o conhecimento destes. Entre os mais notórios, temos os métodos de *phishing*, onde alguém mal-intencionado cria uma cópia idêntica, com um endereço muito similar, de uma página onde o usuário deva entrar com sua senha -como um serviço de internet banking- visando roubá-la; métodos de *keylogging*, onde, através de um malware instalado no computador da vítima, o atacante consegue mapear eventos do teclado e pesquisar, nestes *logs* gerados pelo programa, padrões que possam ser senhas digitadas por aquela vítima. Hackers podem também simplesmente testar senhas simples (como datas de nascimento ou senhas simples como “123456”) quando conseguem acesso à dados cadastrais de usuários do sistema.

Pensando nestas e outras formas de ataque, empresas de segurança ao redor do mundo desenvolveram contra-medidas para proteger senhas de usuários contra ações de pessoas e organizações mal-intencionadas: desde alertas contra potenciais páginas de *phisinhg* e sistemas de medição da qualidade de uma senha, até tecnologias mais sofisticadas de autenticação por senha. Mesmo assim, atacantes continuam a encontrar formas de ludibriar usuários para obter suas senhas, ou mesmo de burlar os sistemas de segurança que as protegem. Um exemplo famoso de um sistema sofisticado que foi burlado é o SecurID da RSA. Este sistema consiste na distribuição de *tokens*, físicos ou virtuais à usuários de redes e serviços restritos. Cada um destes *tokens* exibe uma sequencia, gerada aleatoriamente, de 6 números, que deve mudar a cada minuto e serve como uma segunda camada para senha do usuário, já que a autenticação é feita baseada na *string* formada pela própria senha concatenada com a sequencia numérica exibida naquele minuto. Pensando no fato que cada

token individual possui um número único, chamado *seed*, no qual a geração da sequencia aleatoria é baseada, atacantes perceberam que poderiam obter a senha de um usuário se houvesse uma maneira de saber qual é o *seed* associado àquele *token*. Foi então que, aproveitando-se de uma falha por parte da RSA, hackers conseguiram acesso à estes números e o sistema SecurID foi burlado.

Visando oferecer uma camada de securização personalizada da senha de um usuário, a proposta apresentada neste trabalho busca, na voz, um parâmetro de validação da identidade de alguém que solicite acesso à uma rede ou serviço. Diferentemente das soluções citadas anteriormente, a identificação por sinal de voz se baseia em um parâmetro que se diferencia em cada ser humano, isto é, diversas pessoas podem ter uma mesma sequencia de caracteres como senha, mas cada uma delas pode apresentar diferenças na forma de dizê-la devido a combinações de fatores anatômicos envolvidos na produção da voz e até mesmo hábitos da fala de cada indivíduo. Tais parâmetros podem ser usados para diferenciar usuários e efetuar o reconhecimento de indivíduos (CAMPBELL 1997).

O sistema proposto substitui a digitação da senha pela captação do sinal vocal, de forma que a autenticação se dá em duas camadas, de maneira análoga ao sistema SecurID. A senha deverá então ser dita por um usuário através de um microfone. Este sinal captado passa em seguida por uma etapa de pré-processamento, onde é filtrado para diminuir a influências de ruído na análise espectral subsequente. Nesta etapa de pré-processamento, busca-se identificar as características do sinal que serão usadas nas etapas de identificação: o timbre da voz do usuário, as frequencias dominantes do sinal e a duração da senha que é dita; e a separação do sinal captado em fonemas, que serão modelados de maneira a reconhecer a senha dita pelo usuário.

Para identificar os parâmetros que serão usados tanto no reconhecimento do usuário como no de sua senha, será usada a metodologia descrita em (NILSSON & EJNARSSON 2002), com técnicas descritas em (CAMPBELL 1997) e (RABINER 1989). Esta metodologia envolve uma “adequação” do sinal de voz captado à extração das características usadas pelos algoritmos de reconhecimento. Tal adequação é feita através de sucessivas transformações que, feitas da maneira correta, resultam no chamado domínio *mel-cepstrum*, no qual a ênfase em frequencias mais elevadas é reduzida. Uma vez que o sinal esteja representado neste domínio específico, são extraídas características em forma de vetores, que serão os inputs dos modelos de reconhecimento. Para esta aplicação, as características são: os coeficientes *mel-cepstrum*, a primeira e a segunda derivadas da trajetória formada por estes coeficientes do domínio em questão, e uma medida da energia do sinal.

A etapa de reconhecimento do usuário, ou de reconhecimento do *falante*, será efetuada utilizando um classificador baseado em redes neurais artificais. De posse de um banco de dados de treinamento –que será obtido através da repetição da senha pelo usuário-, uma rede neural artificial pode ser treinada e utilizada como mecanismo de discriminação entre a identidade que o usuário afirma ter e outros cadastros registrados no sistema (CAMPBELL 1997), utilizando os parâmetros obtidos na etapa de pré-processamento. Existem diferentes formas de estimação dos parâmetros de uma rede neural. Para este trabalho, o algoritmo *backpropagation*, juntamente com o método dos gradientes conjugados para a otimização da função de erro, foi escolhido devido à simplicidade na sua implementação.

A camada de reconhecimento da senha do usuário, ou de reconhecimento da *fala*, será realizada através do modelo oculto de Markov (HMM). Este modelo, de base estocástica, é capaz de estimar qual a probabilidade de que uma certa sequencia de observações ocorra dado o próprio modelo e seus parâmetros. De forma análoga à camada de reconhecimento do falante, este modelo será treinado, isto é, seus parâmetros serão estimados, utilizando dados de um banco de dados de treinamento. Com um modelo treinado para cada senha dita ao sistema, será possível identificar qual é o modelo que maximiza a probabilidade de que uma determinada sequencia de observações ocorra (RABINER 1989, KOLLER 2008), ou seja, que uma determinada senha esteja sendo dita, e assim, determinar se aquela é a senha correta. Para treinar o modelo, foi escolhido o algoritmo de Baum-Welch, um caso particular do algoritmo *Expectation maximization*.

CAPÍTULO 3 – BASES METODOLÓGICAS PARA RESOLUÇÃO DO PROBLEMA

Este capítulo traz as bases teóricas sobres as quais se apoiam os métodos utilizados na obtenção das características do sinal, no reconhecimento do falante e no reconhecimento da fala. A matemática envolvida em cada uma das etapas será apresentada em detalhe, porém as demonstrações dos teoremas que possibilitam o uso das técnicas serão omitidas, sendo referenciadas e apontadas conforme necessário.

3.1 – REDES NEURAIS ARTIFICIAIS

Baseando-se nos princípios biológicos que regem o funcionamento do cérebro humano, as redes neurais artificiais podem ser definidas como um processador de dados massivamente distribuído e que possui uma propensão natural a armazenar informações experimentais, tornando-as disponíveis para uso (HAYKIN, 1994).

Concretamente, este processador massivamente distribuído pode ser representado como um grafo (figura 3.1), onde cada nó é uma unidade abstrata de processamento – isto é, que realiza uma operação matemática com dados – chamada “neurônio”; e cada arco representa uma quantidade chamada de “peso”. Através da combinação destes pesos e dos cálculos realizados nos neurônios, as redes neurais são capazes de aproximar , com o nível de precisão desejado, qualquer função contínua (HORNIK, 1991), aparecendo como um forte candidato quando se pensa em modelos não-lineares.

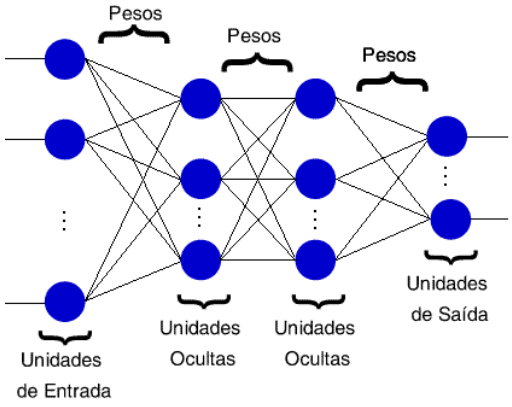

Figura 3.1: Representação gráfica de redes neurais

Com base nesta propriedade, estes modelos podem ser aplicados à solução de diversos problemas, tais como previsão (Zhang, Patuo & Hu, 1997), regressão e classificação; este útlimo sendo o problema de interesse deste trabalho, que deverá usar características de um

sinal de voz X para gerar como saída algo que valide ou não a identidade do usuário Y, cuja voz serviu de entrada. Existem dieferentes formas de se construir e organizar um modelo de redes neurais artificiais para resolver problemas de modelagem não-lineares, tais como funções de base radial ou modelos neurais de séries temporais. Nesta aplicação, porém, será utilizada aquela que é uma das formas mais populares e efetivas de implementação de redes neurais artificiais: o Perceptron. Para melhor visualizar os conceitos envolvidos na explicação do que é um Perceptron, será considerado que esta forma de rede neural artificial consiste em uma estrutura organizada contendo os seguintes elementos:

- Uma camada de entrada de dados (ou de *inputs:* X na notação aqui adotada)
- N camadas "ocultas", consistindo de *k* neurônios cada
- Uma camada de saída de dados (ou *outputs:* Y na notação adotada) com M neurônios

Onde cada neurônio de cada camada está conectado com todos os neurônios da camada seguinte e recebe o sinal enviado por todos os neurônios da camada antecedente.

Nesta representação, cada neurônio é composto por duas partes: armazenamento e ativação. A parte de armazenamento de um determinado neurônio *j* é dada por:

$$z_j = \sum_{i=1}^{n} W_{ij} X_i \qquad (3.1)$$

Onde W_{ij} é o peso ou arco que conecta o *input i* ao neurônio *j*, e X_i é o i-ésimo dos exemplos contido nos dados de treinamento.

A parte de ativação de um neurônio é responsável pela transformação dos *inputs* em uma quantidade chamada "nível de ativação", que definirá o sinal que deverá ser transferido para a camada seguinte da rede neural. Esta parte do neurônio é dada por:

$$a_j = f(z_j) \qquad (3.2)$$

Onde $f(x)$ é a chamada "função de ativação", que deve ser uma função contínua e derivável por razões que serão mostradas mais adiante. Além disso, para algumas aplicações, inclusive a deste trabalho, é desejável que a função de ativação esteja definida somente em um dado intervalo, sendo igual a zero fora dele (mais detalhes sobre esta característica serão dados na subseção a seguir).

Para aplicar redes neurais artificias à solução de problemas, como o de classificação que é o tema deste trabalho, é necessário obter os valores dos pesos W_{ij} para que o sinal de entrada seja alterado enquanto se propaga entre as camadas do modelo e a relação entre entrada e saída possa ser aproximada. A forma de obter ou, mais especificamente, de estimar estas quantidades é através do chamado "treinamento" da rede neural artificial. Este processo consiste em apresentar uma massa de dados ao modelo de forma que este possa "aprender"

qual conjunto de parâmetros se ajusta melhor a este conjunto de dados – que nada mais é do que pares de entrada e saída {X,Y} já observados - chamado de *training set (*no caso da presente aplicação, X são as características da voz e Y é o usuário*).* Neste contexto, o treinamento -ou aprendizado- do modelo é realizado através da otimização de alguma função de custo, também chamada de função de erro ou de perda de informação. Assim, de maneira semelhante à um modelo regressão, o conjunto de pesos W_{ij} que melhor ajusta o conjunto de dados de treinamento é aquele que minimiza esta função de custo. Para a presente aplicação das rede neurais artifciais, a função de custo escolhida foi a soma de quadrados do erro (SSE).

Tendo-se escolhido uma função de custo, é necessário apenas otimizá-la com relação aos parâmetros W_{ij} para se treinar o modelo e seguir para as etapas de aplicação à solução de problemas. No entanto, os métodos de otimização, em geral, necessitam da derivada da função em questão como entrada para que esta possa ser minimizada, o que não é uma tarefa trivial quando se trata de redes neurais. Para obter as derivadas desta função altamente complexa, uma das opções é a aplicação do algoritmo *Backpropagation* ou alguma de suas variações.

3.1.1 – Perceptron e o Algoritmo Backpropagation

Desenvolvido na da década de 70 (WERBOS, 1974), o algoritmo *Backpropagation* surgiu como uma solução ao problema da estimação dos parâmetros de uma rede neural e possibilitou que estes modelos fossem finalmente aplicados à solução de problemas práticos não-lineares. Este algoritmo é, na verdade, uma generalização do método delta de propagação de erros, muito usado em estudos experimentais onde se emprega alguma função matemática de medidas sujeitas a erro experimental. Levando em conta a versão particular do método delta, pode-se visualizar a idéia por trás do algoritmo mais geral, o *backpropagation*: propaga-se, da camada de saída até a camada de entrada, a medida de erro (soma dos quadrados do erro), que resulta da função de custo ao inserirmos os *inputs* na rede, de forma que estas quantidades passam, não pelas funções de ativação de cada neurônio, mas por suas derivadas, de forma semelhante ao método delta para funções não-matriciais. Desta propagação do erro de trás para frente – daí o nome *Backpropagation* – resulta uma medida do erro geral da rede, ou seja, da porção do erro, medido pela função de custo, devido à estimação dos parâmetros W_{ij}. É a partir destes erros que será possível otimizar a função de custo através de uma atualização iterativa dos parâmetros da rede, seja por meio de um método de Descida de Gradiente ou variações do método de Newton-Raphson ou métodos Quasi-Newton. No

entanto, antes de apresentar a aplicação das medidas de erro à otimização e, particularmente, ao método dos gradientes conjugados, que será usado para o presente trabalho; é necessário entender alguns aspectos técnicos do algoritmo *Backpropagation.*

Matematicamente, o algoritmo é implementado através da execução dos seguintes passos:

- Propaga-se o sinal de entrada pela rede neural da seguinte forma:
  - $z^{(1)} = W^{(1)}X$
  - $a^{(1)} = f(z^{(1)})$
  - $z^{(2)} = W^{(2)}a^{(1)}$
  - $a^{(n)} = f(z^{(n)})$ (3.3)
  - $z^{(n+1)} = W^{(n+1)}a^{(n)}$ (3.4)

  Sucessivamente até a camada de saída, onde:
  - $a^{(output\ )} = f(z^{(output\ )})$
- Em seguida, calcula-se o sinal de erro e efetua-se a retropropagação na rede:
  - $\delta^{(output\ )} = -(Y - a^{(output\ )}) * f'(z^{output\ })$, onde Y são os valores correspondentes aos X contidos no conjunto de dados de treinamento;
  - $\delta^{(output\ -1)} = ((W^{(N)})^T \delta^{(output\ )}) * f'(z^N)$

  E assim succesivamente até chegar à primeira camada oculta:
  - $\delta^{(output\ -N)} = ((W^{(N-(N-1))})^T \delta^{(output\ -N)}) * f'(z^{(N-(N-1))})$ (3.5)

  Prossegue-se então para a obtenção das derivadas da função de custo:
  - $\nabla_{W^{(n)}} F_{custo} = \delta^{(n+1)}(a^{(n)})^T$ (3.6)

Onde $\nabla_{W^{(n)}} F_{custo}$ é o gradiente (vetor das derivadas) da função de custo F (a soma de quadrados do erro neste caso) com relação aos parâmetros W da n-ésima camada. Daqui em diante, $\nabla_{W^{(n)}} F_{custo}$ será escrito simplesmente como ∇W para facilitar a leitura. O leitor deve apenas tomar o cuidado de não entender isto como o gradiente de alguma função W, e sim o gradiente da função de custo da rede neural com relação aos seus parâmetros W.

Sabendo como se calcula o gradiente da função de custo com relação aos parâmetros de cada camada, pode-se seguir para a etapa de otimização, onde estes parâmetros do modelo serão estimados em um processo iterativo de minimização do erro: o método dos gradientes conjugados, que será descrito a seguir.

3.1.2 – O Método dos Gradientes Conjugados

Quando se deseja encontrar um ponto extremo de uma função definida em $f: \mathbb{R} \rightarrow \mathbb{R}$, em geral, basta calcular a derivada desta função e encontrar quais coordenadas igualam este resultado a zero. Com um pouco de análise, é possível descobrir se estas coordenadas são as de um ponto de máximo ou mínimo, global ou local. Este processo é chamado de otimização. A solução de problemas de otimização, simples quando tratamos de funções de dimensões baixas, torna-se complexa conforme é extendida para funções de dimensão mais elevada. No entanto, algoritmos computacionais foram desenvolvidos ao longo do tempo para aproximar pontos extremos de funções complicadas e de dimensão elevada, onde uma solução analítica seria impraticável. Um dos mais populares entre estes algoritmos, e que servirá como ponto de partida para a descrição do método dos gradientes conjugados, é o método da descida de gradientes. (LUENBERGER, 1997)

Imagine que se deseje otimizar uma função de custo de uma rede neural para encontrar o conjunto de parâmetros que minimize uma medida de erro, como o SSE por exemplo. A complexidade desta tarefa dependerá do número de parâmetros envolvidos, quantidade esta que aumenta consideravelmente conforme a arquitetura da rede se torna mais complexa, pois para cada novo neurônio - ou camada inteira de neurônios- que é inserido, deve-se conectar este a todos os outros da camada seguinte. Sendo assim, considere uma rede de 3 camadas: uma de entrada, uma oculta e uma de saída, com 4, 5 e 1 nós, respectivamente. Nesta configuração, a tarefa de otimizar a função de custo envolve o cálculo da derivada com relação a 25 parâmetros, que devem ser considerados simultâneamente para se encontrar o mínimo desta função. Se fosse possível visualizar, em um gráfico, a medida de erro (SSE) como função destes 25 parâmetros, provavelmente veria-se algo como picos e vales distribuídos de uma forma altamente complicada sobre um "plano". É neste contexto que se aplica o método da descida de gradientes. Imagine que foi escolhido um ponto aleatoriamente neste espaço complexo de picos e vales. A idéia por trás do método é simples: encontrar a direção, à partir deste ponto inicial selecionado aleatoriamente, pela qual é possível descer mais rapidamente de um pico em direção a uma região mais baixa e, em seguida, dar um passo de um tamanho arbitrário nesta direção. Repete-se este processo até a convergência – ou seja, um ponto de onde não é mais possível descer para uma região mais baixa – ou até que um número máximo de iterações seja atingido. Concretamente, o método é implementado seguindo a seguinte relação de recorrência para atualizar a matriz dos parâmetros do modelo:

$$W_{k+1} = W_k - \alpha \nabla W_k \qquad (3.7)$$

Onde α é a chamada taxa de aprendizagem, ou o tamanho de cada passo dado por iteração; e ∇W_k é o gradiente da função de erro com relação aos parâmetros da rede durante a iteração k. O primeiro passo, W_1, é inicializado aleatoriamente.

Por funcionar com um tamanho de cada "passo" arbitrário e constante na direção de descida mais rápida, às vezes é necessária uma grande quantidade de iterações para se atingir um ponto de mínimo, ou o algoritmo pode acabar levando a um mínimo local muito rapidamente (SCHEWCHUK, 1994), este podendo ser uma solução ruim para o problema que se deseja resolver. Foi para tentar remediar estes problemas que derivou-se um algoritmo similar: o método dos gradientes conjugados.

Um problema do algoritmo de descida de gradientes é que ele pode dar passos, sempre do mesmo tamanho, em uma mesma direção durante várias iterações seguidas. Isto pode levar a resultados indesejados, tais como a demora na convergência ou a convergência rápida para um mínimo local ruim, citados anteriormente. A idéia por trás do método dos gradientes conjugados funciona de forma similar ao método da descida de gradientes, porém a direção de descida não é aquela que oferece o maior decréscimo do erro em um só passo: toma-se uma direção *ortogonal* à direção tomada no passo imediatamente anterior e que, dentre as possibilidades de passos ortogonais, leva ao maior decréscimo do erro. Esta pequena mudança faria com que fosse mais difícil convergir rapidamente para um mínimo local e levaria a soluções melhores para problemas de otimização (SCHEWCHUK, 1994). O problema é que não é tão simples encontrar uma direção ortogonal à anterior sem que uma série de condições sejam satisfeitas e um número de parâmetros seja conhecido. Além disso, precisaria-se conhecer a solução (as coordenadas do ponto de mínimo) de antemão para que esta idéia pudesse ser aplicada na prática. Para remediar esta limitação e permitir a aplicação parcial desta idéia, o método dos gradientes conjugados usa o conceito de A-ortogonalidade, também chamado de conjugalidade, ao invés de simplesmente ortogonalidade (SCHEWCHUK, 1994). Dois vetores X_i e X_j são ditos A-ortogonais, ou conjugados, se:

$$X_i^T \, \mathrm{A} X_j = 0 \qquad (3.8)$$

Onde *A* é uma matriz quadrada, simétrica e positivamente definida. Ao encontrar uma matriz *A* que satisfaça (3.8), é possível então minimizar funções que contenham as chamadas formas quadráticas, isto é, funções de n variáveis da forma:

$$f(x) = a - b^T x + \frac{1}{2} x^T A x \qquad (3.9)$$

No entanto, observou-se que a mesma idéia poderia ser extendida à otimização de qualquer função não-linear que seja diferenciável de segunda-ordem, pois (3.9) poderia ser interpretado como a expansão desta função em série de Taylor com dois termos, onde *A* seria

a matriz Hessiana (das segundas derivadas) da função em questão sendo minimizada pelo método (CHAN, WONG & LAM, 2000). Usando esta interpretação, o método dos gradientes conjugados é aplicado neste trabalho para minimizar a função de erro com relação aos parâmetros da rede neural: uma função não-linear de várias variáveis, cuja propriedade de diferenciabilidade de segunda ordem não será demonstrada aqui. Recomenda-se a leitura da descrição em (HAGAN & MENHAJ, 1994) para mais detalhes sobre esta propriedade e fomas de se aproximar a matriz Hessiana.

O método dos gradientes conjugados é então implementado matematicamente através da execução da seguinte relação recursiva, similar à descrita em (3.7):

$$W_{k+1} = W_k + \alpha_k d_k \tag{3.10}$$

Onde d_k é, por sua vez, dado pela relação recursiva:

$$d_{k+1} = -\nabla W_{k+1} + \beta_k d_k \tag{3.11}$$

Em (3.10), a taxa de aprendizado α não é constante como em (3.7), sendo definida através da aplicação de um método de otimização unidimensional à seguinte função:

$$f(W_k + \alpha d_k) \tag{3.12}$$

Onde a função f é a própria função de erro sendo minimizada: o SSE no caso presente.

No caso do parâmetro β_k, que é incluído em (3.11) para evitar oscilações durante cada iteração do método, foram propostas diferentes maneiras, por diferentes autores, para se obtê-lo. Aqui, foi utilizada a fórmula de Fletcher e Reeves, conforme a metodologia usada em (CHAN, WONG & LAM, 2000):

$$\beta_k \frac{\nabla W_{k+1}^T \nabla W_{k+1}}{\nabla W_k^T \nabla W_k} \tag{3.13}$$

Finalmente, juntando as peças apresentadas anteriormente durante esta subseção, torna-se possível treinar um Perceptron através de retropropagação (*backpropagation*) usando o método dos gradientes conjugados. O algoritmo que consiste da junção destas três partes pode ser descrito da seguinte forma:

1. Inicialize aleatoriamente os parâmetros W da rede neural, conforme a arquitetura escolhida para a rede a ser treinada. Limite a geração deste valores aleatorios a um intervalo de valores pequenos, como por exemplo [-1,1].
2. Insira os valores de entrada do conjunto de dados de treinamento e realize a propagação pela rede, conforme (3.3) e (3.4).
3. Realize a retropropagação do resultado conforme (3.5) e (3.6) para obter o gradiente da rede neural.

4. Defina $d_1 = \nabla_W F_{custo}$, conforme obtido no passo anterior.
5. Atualize os parâmetros W conforme (3.10), (3.11),(3.12) e (3.13) durante n iterações, onde n é o numero de parâmetros da rede, calcule o SSE com os parâmetros atualizados (some o quadrado das diferenças entre os *outputs* obtidos com (3.3) e (3.4) e os valores Y do conjunto de dados de treinamento).
6. Se após n iterações o SSE tiver atingido um valor tolerado, ou seja, um mínimo aceitável foi alcançado, pare o algoritmo. Senão, volte ao passo 2 usando os pesos W atualizados e repita todo o processo até que o SSE atinja um valor tolerado ou até o número máximo de iterações.

O desvio no passo 6 é necessário devido à natureza da aproximação usada no método, descrita em (3.9). Como não se está usando realmente uma forma quadrática, mas uma expansão em série de Taylor de uma função não-linear, a propriedade de A-ortogonalidade se torna mais fraca conforme o número de iterações cresce. Assim, para reestabelecer a propriedade, reinicia-se o vetor gradiente a cada n iterações. Recomenda-se a leitura de (SCHEWCHUK, 1994) para mais detalhes.

Ao final da aplicação do algoritmo, um ponto de mínimo, local ou global, da função de custo deverá ter sido encontrado. Estes valores serão usados na solução do problema de classificação, ou seja, deverão mapear *inputs* não vistos pela rede neural, durante a etapa de treinamento, ao *output* esperado, isto é, ao usuário correto.

3.2 – PRÉ-PROCESSAMENTO E OBTENÇÃO DAS CARACTERÍSTICAS DO SINAL

Para que seja possível reconhecer, dado um sinal de voz de entrada, a identidade do locutor e o conteúdo de sua fala, é necessário obter o tipo correto de característica deste sinal para a utilização com os modelos de Redes Neurais Artificiais e o Modelo Oculto de Markov. Estas duas classes de modelo são, por definição, sistemas de reconhecimento de padrão. Sendo assim, o reconhecimento propriamente dito se dá através de uma predição, por parte destes modelos, dado as características do sinal de entrada. Tais características devem então conter informações tanto sobre a voz do usuário, em si, quanto sobre o conteúdo do sinal, no sentido de mapear, com qualidade, as mudanças que ocorrem espectralmente quando uma palavra está sendo dita, quando uma pausa acontece, ou quando diferentes fonemas são pronunciados.

Seguindo a metodologia usada em (NILSSEN & EJNARSSON, 2002), as características que contêm a informação necessária à tarefa de reconhecimento da fala – e do locutor – podem ser extraídas através da criação de vetores de quantidades espectrais específicas, obtidas através da chamada representação cepstral do sinal de entrada. As transformações necessárias para se chegar à esta representação não são de forma alguma triviais e, além disto, o sinal deve ser pré-processado para apresentar certas qualidades antes de se poder extrair as características desta representação de forma correta. Assim, as próximas subseções tratam da metodologia para adequar o sinal à representação cepstral, além da técnica específica de transformação para esta representação. Vale lembrar que a captação de sinais, em geral, nas aplicações práticas, não se dá em ambientes controlados, onde é possível obter sinais com uma alta relação sinal-ruído e realizar análises precisas das características deste sinal; de maneira que a utilização de filtros deverá fazer parte do pré-processamento. Técnicas de filtragem, no entanto, não estão no escopo deste trabalho e estas não serão discutidas explicitamente. Para mais detalhes sobre a aplicação de filtros, recomenda-se a leitura de (HAYKIN & VEEN, 2001).

3.2.1 – Pré-Processamento do Sinal

Tendo como objetivo a adequação do sinal à representação cepstral, a metodologia de pré-processamento do sinal se dividirá nas seguintes etapas:

- Filtragem e pré-ênfase
- Detecção da Ativação da Voz
- Definição de Frames e Janelas

A primeira parte, coberta em menos detalhe por este trabalho, consiste, primeiramente, na filtragem do sinal para eliminar ruídos indesejados, isto é, para se obter uma alta relação sinal-ruído. No passo seguinte, chamado pré-ênfase, aplica-se um filtro linear para reduzir a ênfase nas frequências mais baixas do sinal. Isto é desejável pois a forma como o trato vocal humano funciona resulta em uma redução da energia do sinal nas frequências mais altas e, consequentemente, na perda desta informação caso esta ênfase nas frequencias mais baixas não seja tratada. Este efeito é tratado através da aplicação de um filtro linear adequado. Na presente aplicação, optou-se por um filto FIR, seguindo a metodologia apresentada em (NILSSEN & EJNARSSON, 2002), que tem a seguinte forma:

$$H(z) = 1 - 0.95z^{-1} \qquad (3.14)$$

Cuja aplicação no domínio do tempo é dada por:

$$s_{pe}(n) = \sum_{k=0}^{M-1} h(k)s(n-k) \qquad (3.15)$$

A segunda parte da etapa de pré-processamento é chamada Detecção da Ativação da Voz. Como o próprio nome deste passo indica, procura-se definir e separar as amostras do sinal onde há sinal de voz e onde não há. Para que esta tarefa possa ser executada, é necessária uma estimativa do chamado "ruído de fundo", já que a amplitude do sinal não será zero mesmo que o usuário esteja em silêncio, devido a sons captados provindos de outras fontes.

Seguindo a mesma metodologia dos passos anteriores, deverá ser efetuado o cálculo de duas medidas para a estimação do ruído de fundo:

$$P_{S_{pe}}(m) = \frac{1}{L} \sum_{i=m-L+1}^{m} s_{pe}^2(i) \qquad (3.16)$$

$$Z_{S_{pe}}(m) = \frac{1}{L} \sum_{i=m-L+1}^{m} \frac{sgn\left(S_{pe}(i)\right) - sgn(S_{pe}(i-1))}{2} \qquad (3.17)$$

Onde $sgn(S_{pe}(n))$ é simplesmente o sinal da função $S_{pe}(n)$ no tempo *n*, ou seja:

$$sgn(x) = \begin{cases} 1, & x \geq 0 \\ -1, & x < 0 \end{cases}$$

A medida $Z_S(m)$ em (3.17), chamada de taxa de cruzamento do eixo *x*, denota simplesmente se houve passagem do sinal S_{pe} de um lado para outro do eixo das abscissas durante um bloco *L* de amostras do sinal. Esta medida servirá na identificação de momentos de silêncio na captação da voz, já que nestes momentos a transição entre positivo e negativo tende a ser mais frequente (NILSSEN & EJNARSSON, 2002). Já a medida em (3.16), é uma medida da potência do sinal à curto prazo, ou seja, durante o mesmo bloco *L* de amostras.

De posse destas duas funções, pode-se construir uma função, cujas propriedades estatísticas podem ser utilizadas para estimar o ruído de fundo. Esta função é dada por:

$$W_{S_{pe}}(w) = P_{S_{pe}}(w)\left[1 - Z_{S_{pe}}(w)\right]S_c \qquad (3.18)$$

Onde S_c é um parâmetro de escala, definido experimentalmente (ver o capítulo 4), para evitar valores muito pequenos de *W*. Usando esta função, pode-se derivar uma estimativa para o ruído de fundo:

$$t_W = \mu_W + \alpha\delta_W \qquad (3.19)$$

Onde μ_W é a média e δ_W a variância de $W_{Spe,}$ calculados usando os primeiros L blocos de amostras, e α é um parâmetro determinado experimentalmente (veja o Capítulo 4).

Considerando esta estimativa do ruído de fundo, é possível, finalmente, definir a chamada função de deteção ativação da voz, que definirá quando uma determinada observação do sinal contém realmente informação da voz do usuário ou se é um momento de silêncio. Com esta função, pode-se estimar em que momento uma palavra começa ou termina, definindo com precisão o espaço de tempo em que o sinal deve ser analisado para a extração das características que serão reconhecidas mais adiante. A função de detecção da ativação da voz é dada por:

$$DAV(m) = \begin{cases} 1, & W_{S_{pe}} \geq t_w \\ 0, & W_{S_{pe}} < t_w \end{cases} \qquad (3.20)$$

Para então definir quais observações se tratam da voz do usuário e não apenas de ruído de fundo (teoricamente momentos de silêncio), basta efetuar a multiplicação da função de detecção da voz pelo o sinal pré-enfatizado S_{pe} e desconsiderar os blocos onde o resultado é igual a zero. A figura 3.2 contém um exemplo da aplicação desta função a um sinal de voz.

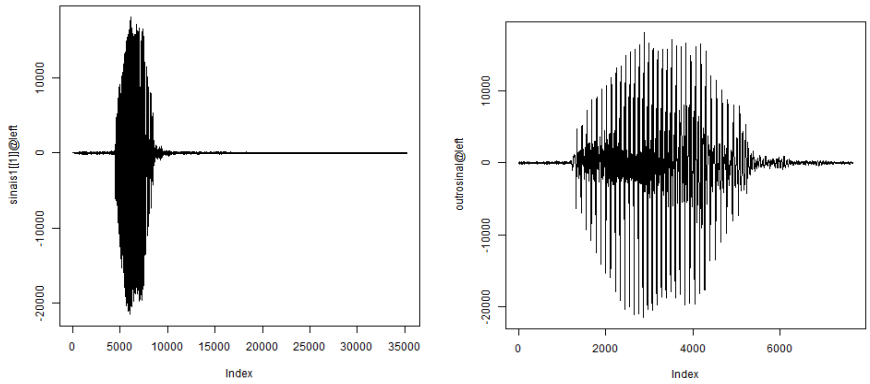

Figura 3.2: Detecção da Ativação da Voz

Finalmente, a terceira e última etapa de pré-processamento do sinal é a definição de *frames* e a aplicação da chamada Janela de Hamming. A definição de frames consiste simplesmente na divisão do sinal $x_1(n)$ – resultado da aplicação da função de detecção da ativação da voz ao sinal pré-enfatizado – em *K* blocos de um tamanho arbitrário, separados por um número também arbitrário de amostras. Após a definição destes frames, aplica-se uma função de "janelamento" à cada um deles, tendo a Janela de Hamming sido escolhida para esta aplicação devido à sua simplicidade relativa às janelas de Kaiser, cuja aplicação envolve a estimação de parâmteros de uma função de Bessel, ou às janelas de Blackman, que podem assumir diferentes formas, estando a busca por uma configuração ótima para esta aplicação fora do escopo do projeto. A Janela de Hamming é então dada por:

$$w(k) = 0.54 - 0.46\cos\left(\frac{2\pi k}{K-1}\right) \qquad (3.19)$$

A aplicação desta função de Janelamento ao sinal de voz, dividido em frames, faz com que as diferenças nas propriedades estatísticas entre um frame e seu antecessor sejam diminuídas,

isto é, a correlação entre os blocos aumenta; aproximando o sinal de um processo estacionário e melhorando a qualidade do reconhecimento efetuado pelo sistema (RABINER, 1978). Isto é necessário devido, mais uma vez, à forma como o sinal de voz é gerado pelo trato vocal humano.

3.2.2 – Mel Cepstrum: Extração das Características do Sinal

A voz humana é gerada por um processo físico. Primeiro, na caixa torácica, ocorre a chamada etapa de excitação, onde o diafragma se comprime e relaxa produzindo uma movimentação de ar com diferentes energias, dependendo do sinal que se deseje produzir. Em seguida, esta onda mecânica, ou sinal acústico, gerado pela movimentação do ar passa pelo trato vocal onde é alterado para produzir a voz particular daquele indivíduo. Todo este processo, explicado aqui apenas superficialmente, pode ser representado pela convolução do sinal acústico gerado pela movimentação do diafragma com o "filtro" do trato vocal:

$$s(n) = b_0 u(n) * h(n) \qquad (3.20)$$

Ou, no domínio da frequência:

$$S(z) = b_0 U(z) H(Z) \qquad (3.21)$$

Onde *u(n)* é o sinal acústico gerado na caixa torácica e *h(n)* é o filtro aplicado pelo trato vocal. Para efetuar análises sobre a forma do filtro H(z), no entanto, é desejável que a equação (3.21) seja linear, de forma que a separação entre U(z) e H(z) possa ser feita de forma simples. Aplicando a função log aos dois lados da equação (3.21) obtém-se então:

$$\log(S(z)) = \log\big(b_0 U(z)\big) + \log(H(Z)) \qquad (3.22)$$

Este efeito aditivo obtido com a aplicação da função log é conservado se for aplicada a transformada inversa de Fourier ou a transformada inversa do cosseno para retornar ao domínio do tempo, graças a uma propriedade algébrica chamada homomorfismo. A este resultado (o logarítimo de S(z) após a aplicação de uma transformada inversa), chama-se de "Cepstrum".

O método Cepstrum, no contexto de reconhecimento de voz, consiste em uma forma de emular o processo de greação da voz e encontrar a expressão aproximada para o filtro *h(n)*, que representa o trato vocal, baseado nas informações contidas no próprio sinal de voz. No entanto, para que esta aproximação funcione corretamente, é necessário que o processamento seja realizado de uma maneira análoga à forma da percepção humana dos sons. Resultados experimentais sugerem que nós, seres humanos, percebemos as variações entre diferentes frequências de sons de uma forma não-linear (RABINER & JUANG, 1993), ao contrário da

escala utilizada normalmente em cálculos envolvendo sinais (Hertz), que é linear. Assim, para realizar a aproximação correta do filtro *h(n)*, o sinal $x_1(n)$ será transformado para uma escala mais adequada: a escala Mel.

A conversão de frequências na escala tradicional de Hertz para a escala Mel pode ser feita através da aplicação da seguinte fórmula:

$$f_{mel} = 2595\ log_{10}\left(1 + \frac{f_{hertz}}{700}\right) \quad (3.24)$$

E a transformação inversa, de Mel para Hertz, é dada por:

$$f_{hertz} = 700 * 10^{\frac{f_{mel}}{2595}} - 1 \quad (3.25)$$

Na prática, o método cepstrum e a escala mel são usados para se obter os chamados coeficientes cepstrais, quantidades espectrais do sinal na escala mel, os quais são usados diretamente na obtenção dos vetores de *inputs* da rede neural e do modelo oculto de Markov. Assim, a mudança entre as escalas e a construção do espectro na escala mel, para se obter K coeficientes mel-espectrais, é implementada através da aplicação de um banco de K filtros triangulares passa-faixa ao sinal |S(z)| (OLIVEIRA, 2001) –utiliza-se o valor absoluto para evitar cálculos com números complexos. Para possibilitar o projeto destes filtros, é necessário, no entanto, encontrar a frequência central para cada um deles. Isto é feito simplesmente aplicando (3.24) aos extremos da faixa de frequências onde se quer analisar o sinal de voz e dividindo este intervalo em K partes iguais. Estes pontos, obtidos da divisão da faixa de frequências na escala mel, são então transformados de volta para a escala original usando (3.25) e são as frequências centrais de cada um dos K filtros, onde a largura de cada um deles é igual a duas vezes a distância entre a sua frequência central e a do filtro antecedente, sendo que o centro antecedente ao primeiro filtro será setado como zero. Assim, somando os valores obtidos com cada um dos K filtros, obtem-se K coeficientes mel-espectrais m_k. Concretamente:

$$m_k = \sum_{n=0}^{N-1} |X_1(n,j)| T_k(n) \quad (3.26)$$

Onde $X_1(n,j)$ é a transformada de Fourier do frame j com a aplicação da janela de Hamming – dado pela equação (3.19)-, e T_k é o k-ésimo filtro triangular.

Finalmente, aplica-se a função logaritmo a cada coeficiente mel-espectral m_k, para se obter, conforme (3.22), uma versão linearizada e separável dos coeficientes. Esta operação resulta nos chamados coeficientes mel-cepstrais-reais: uma sequencia real e par no domínio da frequencia, pré-requisitos para se aplicar a transformada do cosseno. A este resultado,

aplica-se então a Transformada Inversa Discreta do Cosseno para se obter os coeficientes no domínio cepstral:

$$c_s(n,j) = \sum_{n=0}^{N-1} \alpha_k \log(m_k) cos(\frac{\pi(2n+1)k}{2N}) \qquad (3.27)$$

Este domínio tem a propriedade de indexar as intensidades das frequencias por *n*, isto é, as frequencias mais altas estão nos coefiecientes onde *n* é alto e as mais baixas nos coeficientes onde *n* é baixo. Finalmente, para obter a aproximação do filtro *h(n)* do trato vocal no domínio cepstral, aplica-se uma função chamada de "alavancagem", que é análoga à filtragem no domínio cepstral. Seguindo ainda a metodologia proposta por (NILSSEN & EJNARSSON, 2002), esta função de alavancagem é dada por:

$$l(n) = \begin{cases} 1 + \frac{L-1}{2} sen\left(\frac{\pi n}{L-1}\right) & , n = 0,1,..L-1 \\ 0 & , c.c \end{cases} \qquad (3.28)$$

Assim, os coeficientes cepstrais do filtro *h(n)* são dados aproximadamente por:

$$c_h(n,j) \approx c_s(n,j) l(n) \qquad (3.29)$$

Este resultado contém a primeira porção do vetor de características que serão passadas como inputs às camadas de reconhecimento. É também a partir deste resultado que algumas das outras características são obtidas: os coeficientes delta e os coeficientes de aceleração.

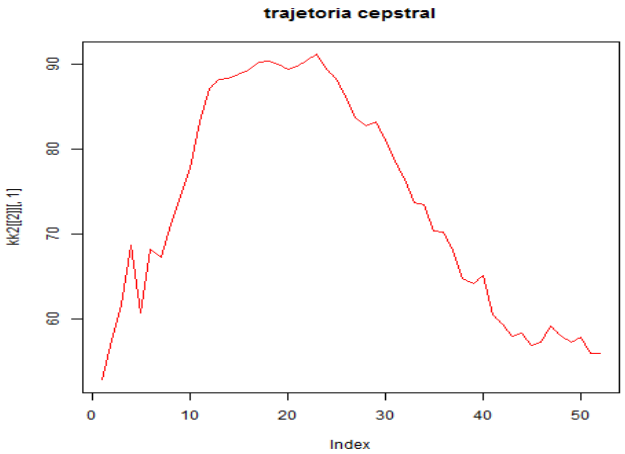

Figura 3.3: Trajetória cepstral de um sinal

Para que os vetores de características contenham informações completas sobre o sinal de voz captado, é necessário se aprofundar na análise cepstral descrita acima. Neste sentido, serão adicionadas mais três medidas aos vetores: uma medida da energia do sinal janelado, os

coeficientes delta-cepstrais e os coeficientes de aceleração, que são as derivadas de primeira e segunda ordem, respectivamente, da trajetória formada pelos coeficientes $c_h(n,j)$ no domínio cepstral. A primeira adição ao vetor de características, a medida de energia, é obtida simplesmente aplicando a fórmula à seguir ao sinal janelado $x_1(n)$:

$$E_m = \log\left(\sum_{i=0}^{K-1} x_1^2(i,j)\right) \tag{3.30}$$

Já para os coeficientes delta e de aceleração, é necessário primeiro calcular uma aproximação da trajetória formada pelos coeficientes cepstrais. Para esta aplicação, a forma escolhida para aproximar esta trajetória, seguindo ainda (NILSSEN & EJNARSSON, 2002), foi o ajuste de um polinômio de grau 2, pelo método dos mínimos quadrados, à segmentos descontínuos de trajetória, formados por blocos de um tamanho arbitrário P (P=3 nesta aplicação). As etapas exatas de cálculo serão omitidas aqui, porém é possível demonstrar (NILSSEN & EJNARSSON, 2002) que aproximações para as derivadas de primeira e sugunda ordem do polinômio ajustado da forma descrita acima podem ser obtidas por:

$$\delta^1 \approx \frac{\sum_{p=-P}^{P} c_h(n,j+p)p}{\sum_{p=-P}^{P} p^2} \tag{3.31}$$

$$\delta^2 \approx 2\frac{\sum_{p=-P}^{P} p^2 \sum_{p=-P}^{P} c_h(n,j+p) - (2P+1)\sum_{p=-P}^{P} c_h(n,j+p)p^2}{\left(\sum_{p=-P}^{P} p^2\right)^2 - (2P+1)\sum_{p=-P}^{P} p^4} \tag{3.32}$$

Assim, construindo uma grande matriz a partir de j vetores menores (um para cada frame *j* do sinal de voz) que contenham todas estas quantidades, isto é, a medida de energia E_m, os coeficientes cepstrais c_h, os coeficientes delta-cepstrais δ^1, e os coeficientes de aceleração δ^2, se obtem uma matriz de *inputs* para os modelos de reconhecimento do locutor e da fala.

3.3 – O MODELO OCULTO DE MARKOV

Modelos para o estudo de sinais, em geral, podem ser divididos em duas grandes categorias: modelos determinísticos e modelos estatísticos (RABINER, 1989). No caso dos modelos determinísticos, emprega-se funções matemáticas como a senóide, ou uma soma de exponenciais para caracterizar a forma de um sinal. Já no caso dos modelos estatísticos, assume-se que um processo estocástico (de natureza aleatória) caracteriza a forma de um dado sinal e, sendo assim, estes são modelados empregando funções probabilísticas. Baseado no conceito de cadeias de Markov, um tema importante em teoria da probabilidade, o Modelo Oculto de Markov está na categoria dos modelos estatísticos.

Estudada desde os anos 60, esta classe de modelos era inicialmente domínio de matemáticos teóricos, que publicavam resultados importantes em veículos não muito conhecidos pelos engenheiros. Por conta disto, a aplicação prática, além de técnicas de programação que permitem o desenvolvimento de pesquisas sobre o tema, começaram a ser explorados com a ênfase necessária somente nos anos 80 (RABINER, 1989). Um dos usos práticos que resultou destas pesquisas foi justamente o reconhecimento da voz. Algoritmos como o método de Baum-Welch para se "treinar" um Modelo de Markov, ou o Algoritmo de Viterbi para se determinar a probabilidade de uma sequência de estados condicionada ao modelo treinado, possibilitam a aplicação desta classe de modelos à solução de problemas práticos.

O Modelo Oculto de Markov será usado nesta aplicação para modelar o sinal de voz visando identificar o seu conteúdo, ou seja, que palavra está sendo dita por um usuário. Isto é feito aplicando a teoria das cadeias de Markov, juntamente com técnicas de programação, à descrição da sequência de observações formada por um vetor de *inputs* como os descritos na seção anterior. Se estes vetores forem tratados como sequências temporais de observações sobre informações espectrais do sinal de voz de um usuário, e se, além disso, for assumido que estas sequências são na verdade regidas por processos estocásticos para os quais é possível estimar parâmetros; então se pode aplicar o Modelo Oculto de Markov para modelar um banco de palavras específicas individualmente e, usando algoritmos associados, determinar qual modelo de palavra maximiza a probabilidade de observar-se a sequência produzida pelo sinal de voz e, assim, determinar o que está sendo dito por um usuário que busque autenticação em uma rede restrita. No entanto, para descrever o funcionamento desta família de modelos em detalhe, é necessário o entendimento do conceito de cadeias de Markov.

3.3.1 – Cadeias de Markov

Considere um sistema que possa ser descrito como estando, em um dado tempo *t*, em um de *Q* diferentes estados. Este estado em que o sistema se encontra em um dado instante, é denotado por q_t. Diz-se que este sistema tem a chamada propriedade de Markov de primeira ordem se a probabilidade da mudança para um estado q_t depender somente do estado em que o sistema estava no instante imediatamente anterior q_{t-1}. Isto é:

$$P(q_t = j \mid q_{t-1} = h, q_{t-2} = i, \dots, q_{t-n} = k) = P(q_t = j \mid q_{t-1} = h) \qquad (3.33)$$

Esta propriedade (3.33) é então definida como a probabilidade de transição entre quaisquer dois estados *i* e *j* , Isto é:

$$a_{ij} = P(q_t = j \mid q_{t-1} = i) \tag{3.34}$$

Onde estas probabilidades satisfazem às condições:

$$a_{ij} \geq 0, \qquad \forall\, i, j$$

$$\sum_{j=1}^{N} a_{ij} = 1, \qquad \forall\, i$$

Assim, as transições entre todos os estados possíveis para o sistema podem ser resumidas na chamada matriz de transição:

$$A = \begin{bmatrix} a_{11} & \dots & a_{1N} \\ \vdots & \ddots & \vdots \\ a_{N1} & \dots & a_{NN} \end{bmatrix} \tag{3.35}$$

Para descrever a chamada Cadeia de Markov, que é uma sequência de observações em diferentes estados de um sistema que tenha uma propriedade de Markov, basta conhecer a matriz de transição entre os estados e a probabilidade inicial para cada estado, isto é, a probabilidade de que o primeiro estado do sistema seja aquele estado em particular. Estas probabilidades podem ser escritas na forma do seguinte vetor:

$$\pi = \begin{bmatrix} P(q_1 = i) \\ \vdots \\ P(q_1 = N) \end{bmatrix} \tag{3.36}$$

Onde estas probabilidades inciais devem satisfazer a condição:

$$\sum_{i=1}^{N} \pi_i = 1 \tag{3.37}$$

Para melhor entender o conceito de cadeias de Markov de primeira ordem, considere o exemplo da figura 3.4. Neste exemplo, um bêbado frequenta três bares diferentes durante a semana.

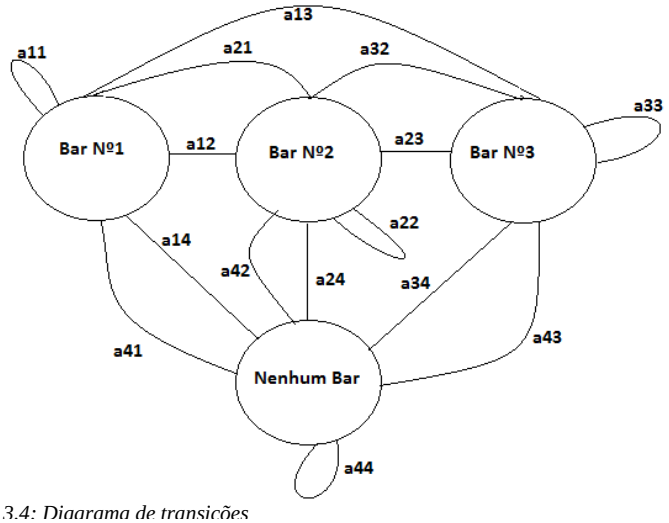

Figura 3.4: Diagrama de transições

Considere que este bêbado escolhe apenas um bar por dia para ir beber e não pode visitar dois bares no mesmo dia. Este bêbado, porém, pode mudar ou não, aleatoriamente, o

bar em que vai consumir seus drinks de um dia para o outro, além de poder simplesmente não aparecer em bar algum de forma também aleatória. Imagine que o comportamento do bêbado foi observado por tempo suficiente para saber quais são as probabilidades de transição entre os bares conforme o diagrama da figura 3.4. Este diagrama mostra as probabilidades de transição entre os estados (bares), sendo que todos estão conectados entre si. Cada estado possui uma probabilidade de transição relativa a ele e outras probabilidades de transição para todos os outros estados a partir dele. Aém disso, cada estado possui uma probabilidade de transição com fim e origem nele próprio: esta é a probabilidade de o bêbado simplesmente não trocar de local de um dia para o outro, a
chamada probabilidade de permanência (ou persistência) de um dado estado em uma cadeia de Markov.

Utilizando um modelo de Markov, é possível descrever as probabilidades de uma certa sequência de observações referente à opção de bar escolhida pelo bêbado, por exemplo, de que ele visite os bares 1,2,3,2,2,nenhum, nesta ordem, em uma determinada semana. Este cálculo é efetuado através dos seguintes passos:

- Define-se a sequência temporal de estados **O=**(1,2,3,2,2,0), onde 0 significa que o bêbabdo não foi à bar algum naquele dia.
- Define-se a probabilidade π do estado inicial, que neste exemplo será o bar de número 1.
- Define-se a matriz $\boldsymbol{A}$ das probabilidades de transição conforme a figura 3.2.
- Calcula-se a probabilidade da sequencia **O** de estados ser observada dada a cadeia de markov definida por π e $\boldsymbol{A}$**,** isto é $P(\boldsymbol{O}|A,\pi) = P(1,2,3,2,2,0|A,\pi)$.

Partindo do fato que este processo é uma cadeia de Markov de primeira ordem e, portanto, possui uma propriedade de Markov de primeira ordem, calcula-se a probabilidade acima como:

$$P(\boldsymbol{O}|A,\pi) = P(1)P(2|1)P(3|2)P(2|3)P(2|2)P(0|2) \tag{3.38}$$

$$P(\boldsymbol{O}|A,\pi) = \pi_1 a_{12} a_{23} a_{32} a_{22} a_{24} \tag{3.39}$$

Este exemplo, no entanto, descreve um modelo de Markov dito explícito, pois os estados neste caso são determinísticos (observa-se diretamente que o bêbado está em um dos bares 1, 2 ou 3, ou nenhum bar). Para caracterizar um modelo de Markov dito oculto, é necessário que o modelo contenha estados probabilísticos, ou seja, não se conhece o estado em que se encontra o sistema, mas apenas observações que podem ter vindo de qualquer um

destes estados de forma aleatória. Dois processos estocásticos estão envolvidos nesta definição: as observações, produzidas de forma aleatória dado um estado fixo; e as mudanças de estados, que não se pode observar diretamente, daí o nome "oculto". Para elucidar este conceito, considere novamente o exemplo do bêbado. Imagine que ele, sempre que chega em um dos bares, pede para o garçom surpreendê-lo e trazer uma bebida selecionada aleatoriamente entre as seguintes opções: cachaça, cerveja ou vodka. No caso em que não vai a bar algum, ele seleciona, também aleatoriamente, uma destas três bebidas de um armário em sua casa.

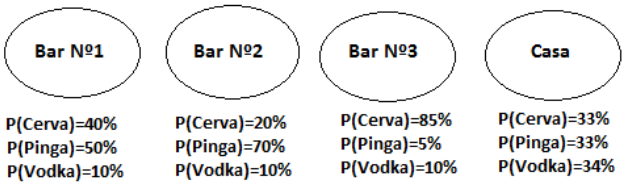

Figura 3.5:Distribuição de Probabilidade de Observações

Além disso, a disponibilidade destas bebidas em cada um dos quatro casos não é a mesma, isto é, caso vá ao bar de número um, o bêbado não receberá cerveja com a mesma probabilidade de caso tivesse ido ao bar de número dois. A figura 3.5 mostra uma distribuição das probabilidades para as bebidas em cada bar deste exemplo.

Considere que o homem diga agora apenas qual bebida consumiu, sem dizer em qual bar ele esteve, mas que se saiba que seus hábitos não mudaram e, portanto, as probabilidades de transição entre os bares continuam as mesmas do exemplo anterior. Note que agora, se o problema for calcular a probabilidade de que uma determinada sequência de bebidas seja consumida pelo bêbado em uma determinada semana, é preciso ser levado em conta, além das probabilidades de transição entre bares, as distribuições de probabilidade que descrevem as chances do bêbado receber uma das três bebidas caso vá a um determinado bar. Neste caso em paticular, como a matriz de transição $\boldsymbol{A}$ já é conhecida, assim como as probabilidades iniciais π, basta definir uma distribuição de probabilidade B_j para as bebidas que ele pode consumir em um bar (estado) j para caracterizar um modelo oculto de Markov, onde B é dado por:

$$B_j = P(\boldsymbol{o}_t = v_k | q_t = j) \tag{3.40}$$

Isto é, a probabilidade de se observar, no tempo t, a bebida v_k dado que o estado atual q_t é o bar j. Assim, um modelo oculto de Markov λ pode ser completamemte caracterizado como:

$$\lambda = (A, B, \pi) \tag{3.41}$$

Tendo caracterizado o modelo oculto de Markov, é possível calcular as probabilidades de se observar determinadas sequências de bebidas durante a semana sem que seja

diretamente observado em que bar o bêbado esteve. Os métodos para tal serão descritos mais adiante.

Voltando ao problema do reconhecimento de voz, que é uma aplicação mais sofisticada do que o problema do exemplo, é necessário fazer uma modificação na definição da distribuição *B* das observações para um estado *j*. Devido à natureza das medições usadas neste processo, a utilização de distribuições de probabilidade discretas para as observações em cada estado (como é o caso do exemplo do bêbado) acarretaria em uma degradação da informação contida nos vetores de entrada do modelo (RABINER & JUANG, 1993).

Para evitar este inconveniente, aplica-se uma distribuição contínua para descrever as probabilidades de ocorrência destas observações, mesmo que elas sejam discretas por definição, já que são medições de quantidades em tempo discreto. No entanto, a escolha da família de distribuições a ser utilizada deve ser feita observando a forma de funcionamento do algoritmo de Baum-Welch, que será utilizado para treinar o modelo. Este algoritmo contém um passo que, de forma análoga ao que ocorre no Método dos Gradientes Conjugados (seção 3.1), otimizará a função de densidade de probabilidade (*pdf*) para estimar os parâmetros que melhor se ajustam aos dados. Sendo assim, a distribuição escolhida deve ter uma *pdf* que seja log-côncava (isto é, o logarítmo da função deve ser uma função côncava) ou que seja elipticamente simétrica (RABINER & JUANG, 1993). Felizmente, existe uma família de distribuições largamente conhecida e estudada que satisfaz estas condições: a distribuição Normal.

No caso do reconhecimento de voz, não se aplica diretamente uma curva da família Normal de distribuições à descrição das probabilidades de ocorrência das observações. Ao invés disto, aplica-se uma mistura de distribuições. Isto é feito devido, à natureza das quantidades espectrais medidas, que podem apresentar formas complexas e multi-modais quanto à distribuição das probabilidades de observação (RABINER, 1989 e NILSSEN & EJNARSSON, 2002). Esta mistura de distribuições, conhecida como "Mistura Gaussiana" no caso de distribuições normais é escrita como:

$$b_j(\boldsymbol{O}_t) = \sum_{k=1}^{M} c_{jk}\, b_{jk}(\boldsymbol{O}_t), \qquad j = 1,2,\dots N \tag{3.42}$$

Onde *M* é o número de componentes da mistura e os pesos de mistura c_{jk} devem satisfazer às seguintes condições:

$$c_{jk} \geq 0 \tag{3.43}$$

$$\sum_{k=1}^{M} c_{jk} = 1 \tag{3.44}$$

Em (3.42), b_{jk} é uma distribuição normal multivariada com vetor de médias μ_k e matriz de vâriancia-covariância Σ_k.

Seguindo (NILSSEN & EJNARSSON, 2002), a matriz Σ_k será substituída por apenas seus elementos diagonais, ou seja, as variâncias, para facilitar a implementação computacional.

Uma observação importante a ser feita neste ponto é que não existe apenas um tipo de Modelo Oculto de Markov. Dependendo da aplicação sendo desenvolvida, é possível estabelecer restrições adicionais ao modelo básico descrito até aqui. A descrição dos variados tipos de modelo está fora do escopo deste trabalho, que se preocupará somente em descrever o chamado modelo de Bakis, ou modelo esquerda-direita. Este tipo de modelo estabelece o paradigma temporal da aplicação, pois restringe a transição entre estados apenas a transições da esquerda para a direita, isto é, não é possível retomar um estado que já passou (RABINER, 1989). Matematicamente, esta restrição é dada por:

$$a_{ij} = 0, \qquad \forall j < i \tag{3.45}$$

Com esta restrição, torna-se possível caracterizar o Modelo Oculto de Markov tal qual ele será aplicado ao problema de reconhecer o conteúdo da fala de um usuário do sistema.

Retoma-se agora o segundo exemplo do bêbado, onde os estados (bares) não são diretamente observáveis e, mesmo assim, deseja-se calcular a probabilidade de se observar uma certa sequência de bebidas consumidas pelo homem. Neste exemplo, a matriz de transição, as probabilidades iniciais para cada estado, e as distribuições de probabilidade das observações para cada estado são conhecidas. Assim, com o Modelo Oculto de Markov completamente caracterizado, será possível calcular a probabilidade de observação de uma dada sequencia se for possível resolver três problemas fundamentais: primeiro treinar (isto é, estimar os parâmetros de) modelos usando observações disponíveis sobre o consumo do bêbado; em seguida, encontrar uma sequência de estados que maximize a chance de que o modelo treinado resulte nas observações disponíveis, isto é, "revelar" a parte oculta do modelo; e, finalmente, calcular a probabilidade, dado o modelo treinado, de observar o que de fato foi observado.

3.3.2 – O Método Baum-Welch

Para resolver o primeiro problema – o de treinamento do modelo – este trabalho emprega o chamado método Baum-Welch. Este método, que estima os todos os parâmetros do Modelo Oculto de Markov (probabilidades iniciais, matriz de transição e coeficientes de mistura, médias e variâncias da mistura gaussiana) utilizando somente as observações como

entrada, é uma versão particular de uma família de algoritmos mais geral conhecida como *Expectation Maximization* e será descrito aqui sob esta ótica mais geral. Para começar, define-se algumas variáveis úteis como se segue:

$$\alpha_t(i) = P(O_1, O_2, \dots, O_t, q_t = S_i|\lambda) \tag{3.46}$$

$$\beta_t(i) = P(O_{t+1}, O_{t+2}, \dots, O_T|q_t = S_i, \lambda) \tag{3.47}$$

$$\gamma_t(i) = P(q_t = S_i|O, \lambda) \tag{3.48}$$

$$\xi_t(i,j) = P\big(q_t = S_i, q_{t+1} = S_j\big|O, \lambda\big) \tag{3.49}$$

A equação (3.46) define a variável α como a probabilidade conjunta de observar-se a sequência parcial até o tempo *t* e o estado S_i no tempo *t*, dado o modelo λ. (3.47) define a variável β como a probabilidade de observar-se a sequencia parcial, a partir do tempo *t+1* até o fim em *T*, dados o estado S_i no tempo *t* e o modelo λ. (3.48) define a variável γ como a probabilidade de o processo se encontrar no estado S_i no tempo *t* dados a sequência completa de observações e o modelo λ. Finalmente, (3.49) define ξ como a probabilidade de se observar uma transição entre os estados S_i e S_j dados a sequência completa de observações e o modelo λ.

Com um pouco de manipulação algébrica, é possível demonstrar (RABINER, 1989) que as variáveis γ e ξ são combinações das outras duas, podendo ser escritas como:

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)} \tag{3.50}$$

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}\, b_j(O_{t+1})\beta_{t+1}(j)}{\sum_{j=1}^{N}\sum_{i=1}^{N} \alpha_t(i)a_{ij}\, b_j(O_{t+1})\beta_{t+1}(j)} \tag{3.51}$$

Considerando (3.50) e (3.51), procede-se para a primeira etapa do algoritmo *expectation maximization*, a chamada etapa E. Esta etapa, que corresponde a *expectation* no nome do algoritmo, consiste, como o nome indica, em obter estimativas dos valores esperados, ou as expectâncias, dos parâmetros que deseja-se estimar. Deriva-se então estimativas para os valores esperados de cada um dos parâmetros (***A,π*** e B) do modelo. Este trabalho não contém os passos analíticos necessários para construir estas estimativas, mas é possível demonstrar que elas são obtidas através seguintes fórmulas (RABINER, 1989, NILSSEN & EJNARSSON, 2002):

$$\hat{\pi}_i = \frac{\alpha_1(i)\beta_1(i)}{\sum_{i=1}^{N} \alpha_T(i)} \tag{3.52}$$

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{3.53}$$

$$\gamma_t(j,k) = \gamma_t(j) \frac{c_{jk} N(\mathrm{o}_{\boldsymbol{t}}, \mu_{jk}, \Sigma_{jk})}{\sum_{k=1}^{M} c_{jk} N(\mathrm{o}_{\boldsymbol{t}}, \mu_{jk}, \Sigma_{jk})} \tag{3.54}$$

Onde $N(\mathrm{o}_{\boldsymbol{t}}, \mu_{jk}, \Sigma_{jk})$ é o quantil de uma distribuição normal multivariada, com vetor de médias μ_{jk} e matriz de variância Σ_{jk}, correspondente ao vetor de observações $\mathbf{o}_t$.

$$\hat{c}_{jk} \frac{\sum_{t=1}^{T} \gamma_t(j,k)}{\sum_{k=1}^{M} \sum_{t=1}^{T} \gamma_t(j,k)} \tag{3.56}$$

$$\widehat{\mu}_{jk} = \frac{\sum_{t=1}^{T} \gamma_t(j,k)\, \boldsymbol{o}_t}{\sum_{t=1}^{T} \gamma_t(j,k)} \tag{3.57}$$

$$\hat{\Sigma}_{jk} = \frac{\sum_{t=1}^{T} \gamma_t(j,k)\, (\boldsymbol{o}_t - \mu_j)((\boldsymbol{o}_t - \mu_j)'}{\sum_{t=1}^{T} \gamma_t(j,k)} \tag{3.58}$$

Com estas formulas, chamadas de fórmulas de reestimação, termina-se a etapa E do algoritmo e passa-se à etapa M, a etapa de maximização.

Para melhor elucidar a razão dos nomes “fórmulas de reestimação” e “etapa de maximização”, é preciso entender que o objetivo final do algoritmo é encontrar o conjunto de parâmetros que maximize a probabilidade de observar o conjunto de observações **O**. Concretamente, isto significa que se deseja otimizar a seguinte probabilidade:

$$P^* = P(\boldsymbol{O}|\lambda) \tag{3.59}$$

Pensando na solução deste problema específico, foi demonstrado nos anos 60 (BAUM & PETRIE, 1966), que ao tomar-se um modelo inicial λ^1, e aplicar as fórmulas de reestimação (3.52 a 3.58) de maneira iterativa, isto é, realimentando as fórmulas com seus próprios resultados para obter os parâmetros de um novo modelo λ^2, então uma das duas opções é verdadeira: ou a probabilidade P* (equação 3.59) é maior para λ^2, ou então o modelo λ^1 já é um ponto crítico de P*, isto é, um máximo local ou global. Em outras palavras, ao reestimar-se os parâmetros utilizando como entrada os parâmetros de um modelo inicial, a probabilidade P*, também chamada de função de verossimilhança do modelo λ, pode ser maximizada de maneira iterativa, similar ao que acontece no método dos gradientes conjugados apresentado na seção 3.1. Neste sentido, reestima-se os parâmetros até que P* encontre um ponto de máximo (ou seja, que não aumente mais). O conjunto de parâmetros resultante deste processo chama-se conjunto de estimativas de máxima verossimilhança e é o

resultado final da fase de treinamento do modelo, isto é: é com base nestes parâmetros que o modelo será utilizado para reconhecer e classificar novas entradas que ainda não foram vistas por ele.

3.3.3 – O Algoritmo Forward-Backward

Tendo descrito o método para se treinar um modelo com base em dados observados, procede-se agora para a solução do terceiro problema: calcular a probabilidade de observar o que foi de fato observado, dado o modelo treinado com base nestes dados. Este problema é essencialmente a formulação técnica do problema específico do reconhecimento da fala, pois é a solução deste problema que viabiliza a "triagem" de um banco de modelos quando se quer que o sistema reconheça uma palavra a partir do sinal de voz de entrada. Em outras palavras, o sistema treinará X modelos para X palavras diferentes e, quando confrontado com uma palavra qualquer, após a etapa de treinamento, deverá calcular a probabilidade de observar o sinal correspondente à palavra em questão dado cada um dos modelos já treinados. Aquele que maximizar esta probabilidade deverá ser o modelo treinado para reconhecer a própria palavra em questão e, por tanto, a palavra sendo dita é aquela modelada pelo modelo escolhido, caracterizando assim o sistema de reconhecimento.

Para efetuar este cálculo, utiliza-se o chamado Algoritmo *Forward-Backward*: um algoritmo baseado na aplicação do princípio algébrico da indução para calcular probabilidades em uma cadeia de Markov a partir de um conjunto de valores iniciais.

Para se definir o algoritmo *Forward*, utiliza-se a idéia da variável definida em (3.46), a chamada variável *forward*. A partir desta definição, é possível projetar um algoritmo que, baseado em um estado inicial, pode calcular por indução a probabilidade que desejamos encontrar, isto é:

$$P^{obs} = P(\boldsymbol{O}|\lambda) \tag{3.60}$$

Define-se então o algoritmo *Forward* da seguinte forma (RABINER, 1989):

1. Inicializa-se a variável $\alpha_1(i)$ como:

$$\alpha_1(i) = \pi_i b_i(\mathbf{o_1}), \qquad 1 \leq i \leq N \tag{3.61}$$

2. Calcula-se $\alpha_{t+1}(i)$ por indução, ou seja:

$$\alpha_{t+1}(i) = b_j(\mathbf{o}_{t+1}) \sum_{i=1}^{N} \alpha_t(i)\, a_{ij}, \qquad 1 \leq j \leq N \tag{3.62}$$

3. Repete-se o passo anterior até o tempo final T. Soma-se as variáveis *forward:*

$$P(\boldsymbol{O}|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{3.63}$$

É importante observar que o este processo não é utilizado apenas para calcular a probabilidade das observações dado o modelo, mas também para treinar o próprio modelo já que a variável forward aparece na etapa E do método Baum-Welch. O algoritmo forward só não foi apresentado antes neste trabalho para manter a ordem lógica de se treinar o modelo e, somente em seguida, utilizá-lo para computar as probabilidades de observações.

Uma outra forma de calcular estas probabilidades é através do chamado algoritmo *Backward*. A rigor, somente o algoritmo *Forward* é necessário para se calcular P(**O|λ**), mas, assim como a variável *forward* α, a variável *backward* β também é necessária na etapa de treinamento do modelo e, por isso, será tratada aqui.

Utilizando a idéia da variável *backward*, definida em (3.47), um método similar ao algoritmo *forward* pode ser construído:

1. Inicializa-se $\beta_T(i)$ arbitrariamente com o valor 1. Ou seja:
$$\beta_T(i) = 1, \quad 1 \leq i \leq N \tag{3.64}$$
2. Calcula-se $\beta_t(i)$ por indução. Isto é:
$$\beta_t(i) = \sum_{\mathrm{j}=1}^{\mathrm{N}} \beta_{t+1}(i) a_{ij}\, b_j(\boldsymbol{o}_{t+1}), \qquad 1 \leq j \leq N \tag{3.65}$$
3. Repete-se o passo anterior até o tempo *t=1*. Soma-se as variáveis *backward*.

O conjunto formado pelo método Baum-Welch e o algoritmo Forward-Backward, oferece uma plataforma teórica completa para lidar com o problema de reconhecimento de voz proposto neste trabalho. A solução para o segundo problema específico, o problema de "revelar" a porção oculta do modelo, não será discutida aqui, pois não interfere na obtenção dos resultados buscados. Esta solução, que pode ser encontrada com a aplicação do Algoritmo de Viterbi (RABINER, 1989), tem utilidade prática quando os estados do modelo têm algum significado físico no contexto do problema sendo estudado, como por exemplo, saber qual é a sequência de bares mais provável de ter sido visitada pelo bêbado no exemplo do início desta subseção.

CAPÍTULO 4 – DESENVOLVIMENTO DO SISTEMA DE AUTENTICAÇÃO BIOMÉTRICA EM REDES DE ACESSO RESTRITO

Visando oferecer uma forma de securização personalizada, o sistema proposto extende o paradigma da verificação de identidades por senha com a adição de um parâmetro biométrico: a voz. Este capítulo contém as descrições das três camadas que compõem este sistema além de detalhes técnicos de suas implementações.

4.1 – Apresentação Geral do Modelo Proposto

A topologia do sistema de reconhecimento de voz proposto pode ser dividida em três camadas: a camada de captação e pré-processamento do sinal, a camada de reconhecimento do locutor e a camada de reconhecimento da fala (figura 4.1). A primeira aplica os conceitos vistos na seção 3.2 para transformar o sinal de voz, que será captado por um microfone comum, e adequá-lo à extração das características que serão usadas nas camadas de reconhecimento. Esta camada precede necessariamente as outras duas.

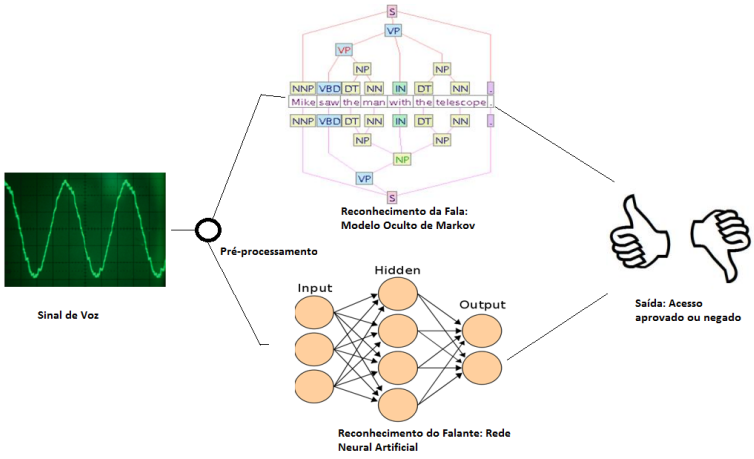

Figura 4.1: Topologia do Sistema

A segunda camada aplica o conceito de Redes Neurais Artificiais, como visto na seção 3.1, para classificar as características, extraídas do sinal na camada anterior, como pertencentes ou não à voz de um usuário que esteja tentando autenticar-se em uma rede restrita. A classificação é baseada em um modelo previamente treinado com a voz do usuário em questão e mais um banco de vozes de outras pessoas para que seja feita a distinção. Esta etapa pode ocorrer paralelamente, ou em qualquer ordem, em relação à terceira etapa: o reconhecimento da fala. Esta terceira etapa aplica os conceitos do Modelo Oculto de Markov,

vistos na seção 3.3, à definição do conteúdo do sinal de voz, conforme as características extraídas na etapa de pré-processamento. De forma similar à etapa de reconhecimento do locutor, o modelo aplicado nesta etapa também deve ser previamente treinado com um banco de palavras que pode ser formado simplesmente por repetições de uma palavra por parte de diferentes usuários do sistema.

4.2 – Descrição das Etapas do Modelo

Para oferecer uma visão mais detalhada da topologia vista na figura 4.1, esta subseção se dedica à apresentação de explicações aprofundadas do funcionamento de cada uma das três etapas, individualmente. O conteúdo desta subseção é uma extensão à prática dos conceitos vistos no capítulo 3. Sendo assim, para uma descrição das minúcias técnicas da implementação do modelo - tais como detalhes sobre a programação e o ambiente de testes – recomenda-se ver a subseção 4.3.

A primeira camada da topologia do sistema é a camada de captação e pré-processamento de dados. Nesta etapa, o sinal de voz do usuário, contendo a sua senha, é captado por meio de um microfone comum. Em seguida, este sinal é passado ao módulo de pré-processamento do sistema, que contém uma etapa de filtragem, uma etapa de pré-ênfase, e uma etapa de deteção da ativação de voz.

A etapa de filtragem, coberta apenas superficialmente no capítulo 3, consiste na aplicação de um algoritmo de redução de ruído para tentar conseguir uma melhor relação sinal-ruído. Isto é necessário pois nem sempre é possível captar o sinal de voz em um ambiente 100% controlado, o que acarreta em sons provindos de outras fontes indesejadas sendo captados pelo sistema. Este ruído pode alterar significativamente o desempenho do sistema, pois altera os parâmetros que são extraídos do sinal com os quais o reconhecimento é efetuado.

Em seguida, tendo o filtro sido aplicado ao sinal de entrada, passa-se à fase de pré-ênfase. Nesta etapa, o sinal é novamente filtrado, com a diferença de que o filtro é projetado para diminuir a ênfase nas frequências mais baixas. A aplicação deste filtro, que é da classe dos filtros lineares FIR (eq. 3.14), resulta em um sinal cuja amplitude é mais homogeneamente distribuída entre as faixas de frequências altas e baixas. Isto é necessário devido ao fato de que existe informação sobre a voz do usuário, disponível nas frequências mais altas do sinal, que seria simplesmente perdida caso a ênfase nas frequências mais baixa fosse conservada. Com a etapa de pré-ênfase, parâmetros que permitem a identificação do usuário podem ser buscados com mais facilidade em toda a faixa de frequências analisada pelo sistema. Mais detalhes serão expostos no capítulo seguinte.

Na etapa seguinte de pré-processamento do sinal, o conjunto de dados resultante da operação de pré-ênfase é passado pela função de detecção da ativação da voz. Esta etapa tem por objetivo o isolamento e a separação das amostras que de fato contêm informação sobre a voz do usuário (e que, portanto, são de interesse para a aplicação), daquelas em que não há informação alguma sobre a voz do usuário (e que portanto podem ser descartadas). Para entender melhor como isto é feito, imagine que o usuário veja em uma tela algo que lhe confirme que ele deve começar a pronunciar a sua senha naquele momento. Entre a emissão desta permissão e o início da fala do usuário, deve haver um pequeno espaço de tempo onde o sistema captará apenas "silêncio". No entanto, partindo do princípio de que esta situação não se passa em um ambiente 100% isolado, haverá conteúdo sonoro sendo captado pelo sistema nestes momentos de silêncio: o chamado ruído de fundo. A função de detecção de ativação da voz (eq. 3.20), baseia-se em uma estimativa da intensidade máxima deste ruído de fundo (eq. 3.19) que é calculada com base no próprio sinal captado pelo sistema. Se a amplitude de uma amostra for menor do que a estimativa da intensidade do ruído de fundo, então ela é considerada como um instante de "silêncio" e é descartada. Por outro lado, se a amplitude de uma amostra for maior do que a do ruído de fundo, então ela é considerada como parte do sinal de voz. Tendo sido aplicada a função de detecção da ativação da voz ao sinal pré-enfatizado, passa-se o resultado para a próxima e última etapa de pré-processamento: a divisão em frames e a aplicação de uma função de janelamento.

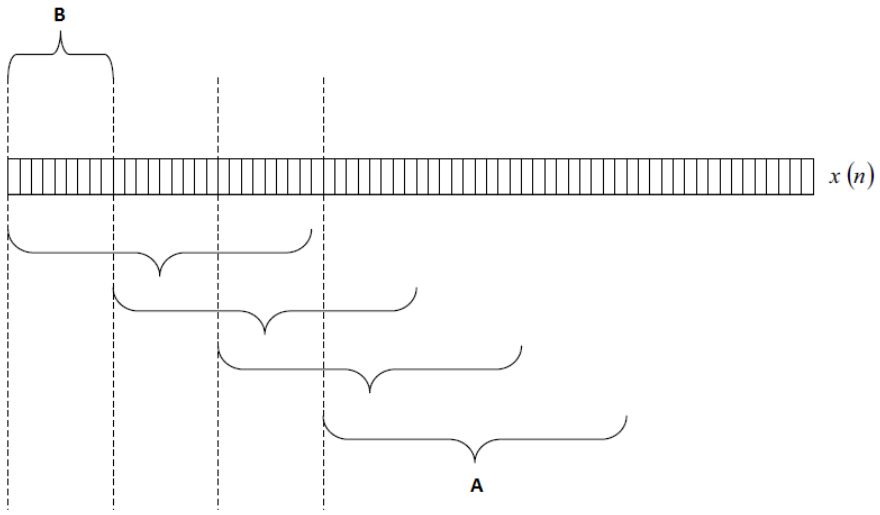

Figura 4.2:Divisão em Frames

A divisão do sinal em frames deve ser feita nesta etapa como forma de preparação ao método que será aplicado para extrair as características desejadas do sinal. Esta divisão é feita

simplesmente tomando blocos de um tamanho arbitrário *A*, separados por blocos de tamanho *B*, conforme a figura 4.2 (adaptada de NILSSEN & EJNARSSON,2002).

Em seguida, aplica-se a cada um dos blocos uma função de janelamento: a janela de Hamming (eq. 3.21) neste caso. A aplicação deste tipo de função aos frames formados no passo anterior serve para diminuir o efeito da descontinuidade criada pela divisão do sinal em blocos. O efeito que se deseja conseguir com este janelamento é um aumento da correlação estatística interblocos, isto é, que os parâmetros estatísticos do sinal em cada frame não variem demasiadamente de bloco para bloco. Isto é desejável pois o Modelo Oculto de Markov, da forma como é implementado neste trabalho, funciona melhor com os chamados "processos estacionários" (RABINER, 1989), que são processos estocásticos onde os parâmetros estatísticos não variam com a passagem do tempo.

O módulo seguinte do sistema é responsável pela extração propriamente dita das características utilizadas no reconhecimento através do método Mel-Cepstrum. Este método consiste em emular o processo de produção da voz baseado em informações do próprio sinal de voz captado. Para tal, são calculados os chamados "cepstrums" que, por sua vez, devem ser transferidos para a escala *mel*: uma escala mais adequada à forma como as variações nas frequências sonoras são percebidas pelos seres humanos. O cálculo dos coeficiente *mel-cepstrais* (a "matéria-prima" com a qual constrói-se os vetores de características desejadas do sinal) é então efetuado através dos seguintes passos, descritos no capítulo 3 e resumidos na figura 4.3: computar a transformada de Fourier do sinal janelado; tomar o valor absoluto do sinal transformado; projetar e aplicar K filtros triangulares ao sinal transformado; somar os blocos de sinais filtrados; tomar o logarítimo natural do resultado; aplicar a transformada discreta do cosseno; aplicar uma função de "alavancagem" ao sinal retransformado.

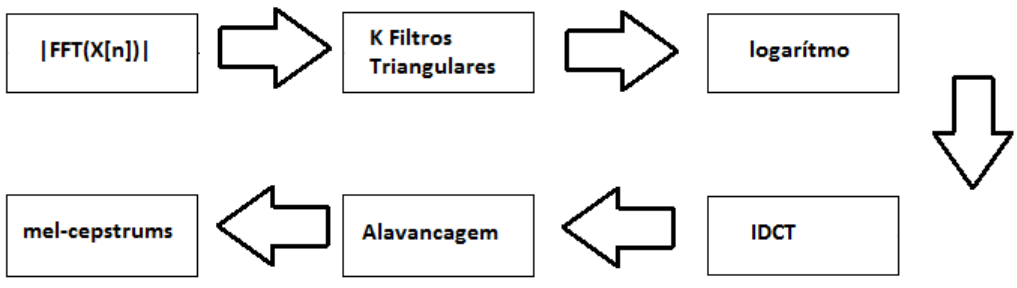

Figura 4.3: Diagrama de Fluxo (obtenção das características)

Tendo calculado os coeficientes *mel*-cepstrais, procede-se para o último módulo de pré-processamento do sinal. Neste módulo, obtêm-se os vetores de características a partir dos coeficientes obtidos no módulo anterior. Estas características são: uma medida de energia do

sinal janelado (eq. 3.30); os coeficientes *mel*-cepstrais puros; os coeficientes delta-cepstrais; os coeficientes de aceleração. Os dois coeficientes que ainda não haviam sido calculados até este ponto, ou seja, coeficientes delta-cepstrais e de aceleração são, respectivamente, as derivadas de primeira e segunda ordem da trajetória formada pelos coeficientes *mel-cepstrais* puros no domínio cepstral. A forma de aproximar estas derivadas foi descrita no capítulo 3, subseção 3.2. Finalmente, uma vez obtidas as quantidades que representam as características de interesse do sinal de voz, procede-se à construção dos vetores de *inputs* das duas camadas seguintes da topologia do modelo. Para montar o *input* do Modelo Oculto de Markov, basta criar-se J vetores com todas as medidas obtidas no passo anterior, onde cada um dos J vetores corresponde a uma observação $\mathbf{o}_t$, onde J é número de blocos (*frames*) em que o sinal foi dividido. No caso dos *inputs* da Rede Neural Artificial, cada componente do vetor corresponde a uma única observação e, portanto, a um *input* único. Para incluir todas as observações, basta organizar estes valores em uma matriz.

Uma vez que os vetores de inputs tenham sido obtidos e definidos, segue-se para qualquer uma das camadas de reconhecimento, isto é, do falante ou da fala. No caso da camada de reconhecimento do falante, aplica-se o conceito de redes neurais artificiais para modelar a relação entre os *inputs* descritos acima e a identidade de um usuário. Isto é feito a partir do "treinamento" de uma rede neural artificial com dados de usuários, onde já se saiba quais *inputs* provêm de qual usuário.

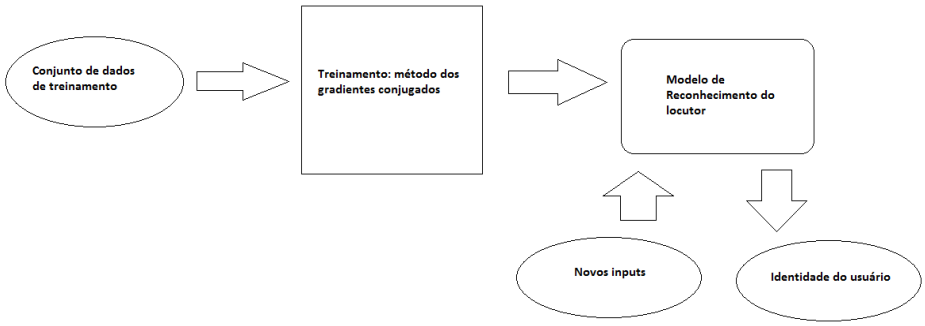

Figura 4.4: Diagrama Rede Neural

Com dados deste tipo, é possível fazer com que o modelo "aprenda" quais vetores de características correspodem a cada usuário e, assim, quando for inserido um novo conjunto de dados para os quais não se sabe quem é o usuário correspondente, o modelo deverá ser capaz de reconhecer a identidade da pessoa responsável pela produção de tais *inputs* com base no que "aprendeu" na fase de treinamento. Existem várias opções de algoritmos eficientes com os quais é possível executar o treinamento de um modelo de rede neural artificial. Nesta aplicação, optou-se pelo *método dos gradientes conjugados*, onde um processo iterativo

otimiza os parâmetros do modelo de acordo com o conjunto de dados de entrada. O processo é resumido pela figura 4.4.

No caso da camada de reconhecimento da fala, aplica-se o Modelo Oculto de Markov para modelar a relação entre os *inputs* e uma determinada palavra que tenha sido dita por um usuário. De maneira análoga ao processo utilizado na camada de reconhecimento do falante, é possível treinar um modelo de forma que este seja capaz de mapear quais *inputs* correspondem a qual palavra e, assim, quando confrontado com um conjunto de dados novo, ou seja, que não fez parte do conjunto de dados de treinamento, o modelo deverá ser capaz de associar seu conteúdo à uma palavra que já tenha sido modelada. É importante notar que a plataforma constituída pelo Modelo Oculto de Markov, o método de Baum-Welch e os algoritmos Forward e Backward – diferentemente, das redes neurais artificiais, não mapeiam uma relação direta entre *inputs* e a palavra que está sendo dita. Ao invés disso, atribui-se uma *probabilidade* de que tal palavra esteja sendo dita dado que o modelo correto é o modelo em questão. Neste contexto, ajusta-se um Modelo Oculto de Markov para um banco de diferentes palavras e, quando um usuário disser a sua senha ao sistema, este varrerá o banco de Modelos Ocultos de Markov, calculando, para cada um deles, qual é a probabilidade de observar-se o vetor de características derivado da palavra dita pelo usuário. Aquele que obtiver a maior probabilidade condicional de se observar as características em questão, deverá ser aquele associado a palavra que de fato foi dita pelo usuário, efetuando assim o reconhecimento da fala. O processo é resumido pela figura 4.5.

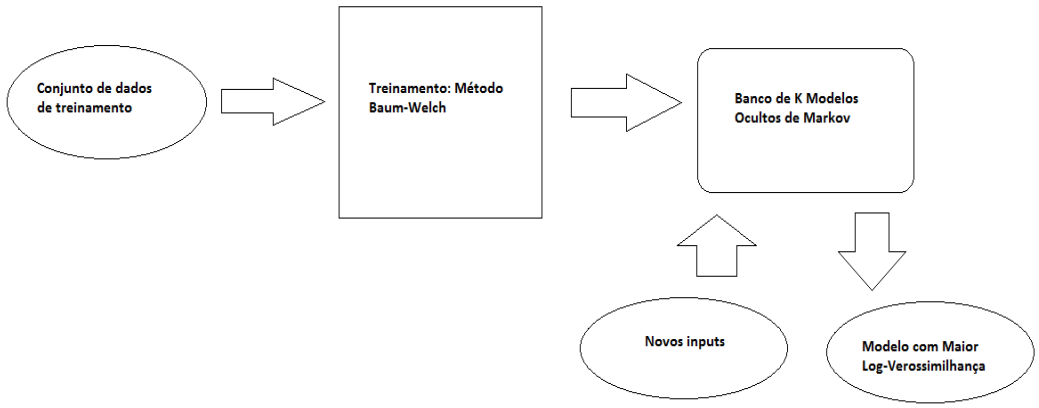

Figura 4.5: Diagrama HMM

Finalmente, o acesso à rede restrita será concedido ao usuário do sistema somente caso as duas camadas de reconhecimento concordem. Em outras palavras, um usuário pode até dizer a senha corretamente, mas caso ele não seja o usuário associado àquela conta , o acesso será negado. O mesmo acontece caso um usuário esqueça a sua própria senha ou não a diga corretamente.

4.3 - Descrição da Implementação

Todo o sistema foi implementado utilizando a linguagem e o sistema de análises R. Esta plataforma open-source é utilizada pela comunidade científica para análises e formulações de modelos estatísticos. No entanto, por oferecer um ambiente eficiente para cálculos matriciais semelhante ao do MATLAB, é possível desenvolver aplicações específicas para outras áreas além da estatística.

Este trabalho tem como maior contribuição para a engenharia justamente a implementação de um tipo específico de aplicação, o reconhecimento de voz, para a plataforma R. Uma contribuição menos abrangente é a implementação completa em linguagem R de uma plataforma vetorizada de treinamento e análise de modelos de Redes Neurais Artificiais. Aqui, será explicado de forma detalhada como proceder para replicar esta implementação, sendo que os detalhes de cada parte do sistema serão expostos seguindo uma ordem modular diferente da utilizada nas subseções anteriores: primeiro o módulo de pré-processamento, em seguida o módulo de reconhecimento da fala e, finalmente, o módulo de reconhecimento do locutor.

O primeiro módulo do programa, o de pré-processamento e obtenção das características do sinal de voz, foi construído com base em três bibliotecas open-source da linguagem R: as bibliotecas *seewave* (SUEUR, 2012), *signal* (SHORT, 2012) e *tuneR* (LIGGER, 2012). Estas bibliotecas contêm funções gerais para o estudo de sinais, tendo toda a parte referente à filtragem e pré-ênfase do sinal sido implementada utilizando funções internas destas bibliotecas.

A leitura do sinal foi feita com uma frequência de amostragem de 16khz e apenas um canal através um microfone simples em um ambiente relativamente silencioso (um quarto fechado durante uma madrugada no meio da semana). Captou-se então a senha dos usuários, sendo que esta deveria sempre ser uma palavra simples como um número de 1 a 3. Para a filtragem inicial, utilizou-se um filtro simples de médias móveis com uma janela de tamanho 10, cuja saída foi passada ao filtro FIR. A detecção da ativação da voz é então feita com base na estimativa do ruído de fundo, implementada conforme descrito no capítulo 3, e obtida com a constante de escala S_c igual a 100. Já a sua aplicação ao sinal é feita efetuando uma expansão dos blocos, de tamanho L igual a 320 (blocos de 20ms), conforme o valor da função de detcção dentro deste. Para melhor entender este mecanismo, recomenda-se a análise do código disponível no APÊNDICE A. Em seguida, a divisão em frames é feita com blocos de amostras de tamanho igual a 320 amostras e espaçamento entre os blocos de 100 amostras. A aplicação da janela de hamming é feita através da função *windowing* da biblioteca *signal*, que

é chamada diretamente de dentro da função *melfcc* da biblioteca *tuneR* (ver o código em anexo, APÊNDICE A linha 96).

O módulo de obtenção das características requer cuidado na implementação prática, pois é necessário observar alguns detalhes para que o programa retorne os valores buscados de forma eficiente. O primeiro deles é a aplicação da função *fft* para o cálculo da transformada de Fourier. Aqui, é necessário observar que o algoritmo usado nesta função padrão da linguagem R só é eficiente caso o tamanho do vetor de observações ao qual a função é aplicada for uma potência de 2. Para isto, aplica-se uma função verificadora desta condição, onde caso o tamanho não seja potência de 2, o vetor é completado com 0s (zeros) até atingir um tamanho que satisfaça a condição. O segundo detalhe é a escolha da quantidade de filtros triangulares para compor o banco. Esta quantidade K é igual ao número de coeficientes cepstrais que serão obtidos ao final do módulo, por tanto, se K for muito elevado, pode-se provocar o chamado *overfitting* das camadas de reconhecimento. *Overfitting* é o nome dado ao problema que ocorre quando informação demais sobre o fenômeno sendo modelado é inserida em um modelo, de forma que este é capaz de reconhecer com quase 100% de precisão qualquer vetor de observações que pertença ao conjunto de treinamento, mas é incapaz de generalizar esta precisão para observações que sejam diferentes do conjunto de treinamento. Nesta aplicação, K igual a 20 se mostrou uma quantidade razoável. A biblioteca *tuneR* contém métodos que implementam todas estas transformações de forma eficiente, em particular, este trabalho utiliza a função *melfcc* para a obtenção dos coeficientes cepstrais. A obtenção dos coeficientes delta e de aceleração é implementada simplesmente seguindo as definições dadas no capítulo 3 (ver código em APÊNDICE A linha 100).

O segundo módulo, o de reconhecimento da fala, é feito completamente com base na biblioteca RHmm (TARRAMASCO & BAUER, 2012), através de suas funções que implementam os algoritmos descritos no capítulo 3. A função *HMMset* constrói um objeto a partir de uma lista de parâmetros iniciais e vetores de observação. Os vetores de observações são a saída do módulo anterior, cuja forma pode ser vista no código em anexo. Já as estimativas iniciais são geradas aleatoriamente com a função *runif*, que gera números aleatórios seguindo uma distribuição uniforme. Uma vez criado o objeto, invoca-se o método HMM*Fit* para treinar o modelo com o algoritmo EM. Nesta aplicação, utilizou-se 2 estados e 4 misturas por estado para modelar todoas as palavras. A saída será um novo objeto da classe *HMM* contendo as estimativas de máxima verossimilhança no lugar dos valores aleatórios iniciais. Para calcular a probabilidade condicional das observações dado o modelo, basta invocar o método *forwardBackward* e obter a probabilidade buscada como resultado. É

importante ressaltar neste ponto que a implementação do algoritmo forward-backward contida na bliblioteca RHmm é extremamente sensível à precisão da máquina onde roda o programa. Sendo assim, o cálculo da soma das variáveis alfa (eq 3.63) pode ser impreciso devido ao fato da plataforma R setar qualquer valor menor do que a precisão da máquina como zero. Como os vetores de entrada são longos, os valores da densidade de probabilidade para as observações na porção final destes vetores tende a ser pequena o bastante para acabar sendo setada como zero. Para remediar este problema, este trabalho aplica a Log-Verossimilhança como medida para realizar a triagem dos modelos com relação ao sinal de entrada que se deseje reconhecer.

A terceira e última camada da topologia a ser descrita é o módulo de reconhecimento do locutor. Para esta tarefa, foi desenvolvida uma plataforma vetorizada completa de treinamento e análise de redes neurais em linguagem R.

Uma implementação vetorizada de redes neurais aqui significa que o algoritmo backpropagation foi escrito com base em operações matriciais ao invés de atualizações iterativas em cada camada do modelo através de *loops* do tipo *for*. Esta alteração diminui a complexidade do algoritmo uma vez que todo o conjunto de dados de treinamento é lido e processado como um todo, apenas uma vez em uma única matriz, ao invés de N leituras individuais, onde N é o tamanho do conjunto de dados de treinamento. Assim, as matrizes de parâmetros referentes a cada camada da rede neural, são armazenadas em uma lista, onde podem ser atualizadas individualmente, desta vez utilizando *loops* do tipo *for* para acessar cada posição da lista (ver código no APÊNDICE B linha 124 para mais detalhes). Embora seja possível posicionar todos os parâmetros em uma grande matriz de forma que não seja necessário atualizar cada camada individualmente em um *loop for* sobre uma lista, nesta aplicação optou-se por esta implementação menos eficiente devido a incertezas quanto ao tamanho que estas matrizes poderiam tomar e que, caso fossem muito grandes, poderiam acarretar em problemas com o sistema operacional relacionados ao uso de memória.

As funções de ativação foram definidas como métodos internos aos objetos criados pelas funções *matrixForwardPropagation* e *matrixBackPropagation*, sendo possível escolher entre as funções logística, tangente hiperbólica ou softmax através da passagem de um argumento à função-pai.

O método dos gradientes conjugados emprega esta implementação vetorizada do *backpropagation* em seu funcionamento, com a parte iterativa sendo levada a cabo por um *loop for* de, no máximo, mil repetições. É nesta etapa que se pode enxergar a elegância de uma implementação vetorizada: caso se tivesse optado pela implementação literal do

algoritmo descrito no capítulo 3, o método de treinamento efetuaria um número elevado de operações dentro de cada uma das mil iterações, resultando em uma complexidade algorítimica grande.

Na etapa de definição do tamanho do "passo" tomado em uma direção *A*-ortogonal à anterior (eq. 3.12), um método de otimização unidimensional deve ser aplicado para se encontrar o tamanho ótimo. O algoritmo escolhido para esta aplicação foi o método de busca em secção áurea (MOLER, 2004). A implementação do algoritmo incluída no programa do método dos gradientes conjugados, foi feita completamente com base em operações básicas da plataforma R, sem o uso de bibliotecas de terceiros e, portanto, também é uma contribuição de engenharia deste trabalho, embora menos abrangente que a plataforma completa para redes neurais.

Finalmente, uma vez que um modelo seja treinado com o conjunto de dados de treinamento e, em seguida, confrontado com um novo conjunto de dados, o modelo deverá apresentar como saída uma matriz de binários, com K colunas – onde K é o número de usuários registrados no sistema - onde um bit setado indicará que aquele bloco de tempo foi associado com o usuário da k-ésima coluna. Ao final, o sistema realizará uma soma simples de cada uma das colunas. Aquela que obtiver o maior resultado, deverá a coluna associada ao usuário que efetivamente produziu o sinal de voz de entrada.

CAPÍTULO 5 - APLICAÇÃO PRÁTICA DO MODELO PROPOSTO

Este capítulo trata das condições específicas em que o sistema foi implementado, aplicado e testado. Além disso, esta seção trata dos resultados obtidos por estes testes, bem como alguns dos problemas que aconteceram durante a aplicação. Aqui também será feita uma rápida discussão das possíveis áreas concretas de aplicação deste sistema, o custo financeiro total envolvido e uma avaliação geral da aplicação e da implementação com base nos resultados obtidos.

5.1 - Apresentação da Área de Aplicação do Modelo

O título deste trabalho sugere a aplicação do modelo à autenticação em ambientes de rede, porém não está limitada apenas a este contexto. De fato, em qualquer aplicação onde seja desejada a identificação do usuário com base em sua voz, acompanhada de um comando vocal ou uma senha, o modelo propsoto poderia aparecer como solução. Alguns exemplos práticos são a automatização de funções domésticas por comando de voz, aplicações à robótica ou mesmo operações bancárias por telefone. No entanto, é importante ressaltar que testes foram realizados apenas em um ambiente pouco ruidoso, estando a extensão ao caso de ambientes com alto grau de interferência fora do escopo deste estudo.

5.2 – Descrição da Aplicação do Modelo

O sistema segue a topologia apresentada na figura 4.1. Na primeira camada, a de pré-processamento e extração das características do sinal, utiliza-se um microfone convencional (neste trabalho, utilizou-se o microfone embutido em um MacBook Pro) para realizar captação do sinal de voz. Para abrir o canal onde foram registrados os arquivos, dois métodos foram utilizados: a função *Record* da biblioteca *seewave* e o software *Audacity*, sendo que o segundo método foi mantido para os testes finais. Uma vez registrado um sinal de voz, este é salvo no sistema de arquivos com o formato "*.*wav*". A amostragem foi feita com apenas um canal (mono) em 16Khz com 16 bits PCM para todos os sinais de voz captados. Já no ambiente do R, faz-se uma chamada à função *ReadWave* para carregar os dados do sinal em um objeto da classe *Wave*.

Em seguida, executa-se uma chamada à função *filter* para realizar a filtragem inicial do sinal, com o detalhe de que o argumento não deve ser diretamente o objeto criado pela função *ReadWave*, mas apenas a sequencia temporal de amplitudes, extraída com o método @left. Passa-se este resultado à função DAV para se estimar o ruído de fundo e aplicar a

função de detcção da ativação da voz ao sinal, o resultado será um novo objeto da classe *Wave* contendo o sinal filtrado. Neste ponto, o sinal está pronto para o método Mel-Cepstrum, implementado através da função *melfcc* do pacote *tuneR.* Esta função, com os argumentos passados conforme o código anexo, que pode ser encontrado no APÊNDICE A linha 96, segue a sequência de transformações exposta no capítulo 3 e resulta em uma matriz de tamanho MxN contendo os coeficientes cepstrais, onde M é o número de blocos de 20ms em que o sinal é dividido e N é o número desejado de coeficientes cepstrais (20 nesta aplicação).

De posse da matriz de coeficientes cepstrais, aplica-se a função *findDelta* para calcular-se as derivadas de primeira e segunda ordem dos coeficientes. Em seguida, uma chamada à função *buildInputs*, dependendo do argumento *type*, que pode ser igual à "NN" ou "HMM", fará a adequação dos coeficientes à entrada de cada um dos modelos. Por fim, uma chamada às funções *callNeuralNet* ou *callMarkov* efetua a aplicação dos métodos de treinamento aos dados obtidos. De posse dos modelos treinados, pode-se efetuar uma nova chamada à estas funções passando o conjunto de dados desejado como argumento para que seja realizado o reconhecimento.

Este processo foi realizado com leituras de sinais de voz das palavras "um", "dois" e "três" durante uma madrugada de uma quarta-feira, buscando minimizar efeitos de ruído indesejado no sistema. Cada uma das palavras palavras foi gravada em três repetições por parte de dois usuários diferentes para a etapa de treinamento. Em seguida, o reconhecimento foi realizado com novas entradas gravadas *ad hoc*, processo o qual foi repetido 36 vezes para cada palavra.

Finalmente, para efeito de teste da concordância entre as duas camadas, referente ao caso de uso da autenticação em uma rede restrita, assumiu-se que a senha do autor fosse a palavra "um", e a senha do outro usuário a palavra "dois". Uma bateria de testes, de 36 repetições, foi realizada separadamente exclusivamente para este caso de uso.

5.3 – Resultados da Aplicação do Modelo

O sistema de reconhecimento da fala foi capaz de reconhecer, com base nos conjuntos de treinamento de tamanho igual N=6, as palavras "um" e "dois" com 97,2% de precisão.

Tabela 5.1: Estatísticas de teste RNA

	Reconhecimento do Locutor	
	Redes Neurais	Baseline
Acertos	96	33
Erros	12	74

Tabela 5.2: Estatísticas de teste HMM

	Reconhecimento da Fala	
	Modelo de Markov	Baseline
Acertos	102	51
Erros	6	57

A palavra “três” obteve um índice de acerto de 88,88%, resultando em um índice global de acerto estimado em 94,44%, com um intervalo de confiança, à 5% de significância, de 89% até 100%. Para efeito de comparação, o modelo *baseline*, uma simples seleção aleatória uniforme (isto é, cada palavra tem 33,33...% de chance de ser selecionada), obteve índices de acerto bem menos expressivos, próximos dos 33% esperados.

Já o sistema de reconhecimento do locutor obteve um índice de acerto estimado de 88,88%, com um intervalo de confiança, também com 5% de significância, entre 82,40% e 100%. O modelo simples *baseline* obteve índices próximos aos 50% esperados. As tabelas 5.1 e 5.2 resumem as estatísticas de teste observadas. Em ambos os casos, um teste Z uni-caudal à direita para proporções (com aplicação da correção de Yates para a continuidade) rejeita, para um nível de significância de 5% , a hipótese nula de inferioridade do método em relação ao caso *baseline*, resultando em *p-valores* menores que 0.0001 nos dois casos.

Contudo, considerando a topologia do sistema proposto, conforme pode ser visto na figura 4.1, a autenticação só é realizada caso haja concordância entre as duas camadas de reconhecimento. Portanto, a avaliação do desempenho global do sistema deve levar em conta o índice de acerto na autenticação considerando as duas camadas, que, na segunda bateria de testes, foi igual a 86,11% com intervalo de confiança, a 5% de significância, entre 69,71% e 94,77%.

5.4 – Custos do Modelo Proposto

Este sistema não envolve quase custo algum de implementação, já que a codificação dos métodos de reconhecimento foram completamente feitas com base em uma plataforma de computação 100% open-source. O único custo é o do microfone, neste caso embutido no computador utilizado para testes. Poderia-se mencionar a licença do software *audacity*, usado para a leitura dos sinais de voz, como uma fonte de custo financeiro para a implementação, no entanto, este programa pode ser facilmente substituído por um equivalente open-source sem perda alguma, pois apenas suas funções simples de escrita em formato *wav* e de interface de captação de voz foram utilizadas para esta aplicação.

5.3 – Avaliação Global do Modelo

De maneira geral, o sistema cumpre os objetivos propostos: o reconhecimento tanto da voz como do conteúdo do sinal funciona bem. Algumas vantagens e ganhos técnicos desta implementação são: o fato de ser baseada em uma plataforma open-source com uma comunidade ativa de contribuidores como a plataforma R; a codificação vetorizada de redes neurais; e a prórpia natureza biométrica do sistema, com relação ao paradigma de escrita de senhas.

No entanto, como a linguagem utilizada é uma linguagem interpretada e a plataforma R depende de um interpretador reconhecidamente lento, o sistema perde em desempenho, relativamente a uma linguagem compilada como o C por exemplo, no que tange o tempo de execução dos métodos. Um outro ponto fraco desta implementação é a ausência de uma interface gráfica que sirva como meio para a captação do sinal sem a mudança manual entre aplicações.

Alguns pontos forte são: a portabilidade do código, pois a plataforma R existe em diferentes arquiteturas e sistemas operacionais e a possibilidade de extensão dos usos do código vetorizado para redes neurais.

CAPÍTULO 6 - CONCLUSÃO

6.1 - Conclusões

Neste trabalho, foi proposto um sistema de verificação de identidade baseado em duas camadas associadas ao sinal de voz. Conclui-se que a implementação aqui descrita atinge os objetivos gerais propostos, reconhecendo tanto o usuário como o conteúdo de sua fala com sucesso significativamente maior do que uma simples seleção aleatória. Além disso, a implementação obteve um índice satisfatório de concordância entre as duas camadas, autenticando o usuário corretamente na maior parte do tempo.

Os objetivos específicos propostos também foram alcançados, pois a adaptação de algoritmos de aprendizado de máquina foi feita com sucesso. Mais especificamente, as implementações da função de ativação da voz e do método dos gradientes conjugados para redes neurais funcionaram com sucesso, o qual é confirmado não somente pelas taxas de acerto no reconhecimento de usuários, mas também por análises individuais das saídas destas funções (ver o APÊNDICE A).

Dado o sucesso técnico da aplicação, ressalta-se também o sucesso do ponto de vista financeiro, pois o modelo implementado é 100% open-source e gratuito. Apesar das limitações da plataforma, no contexto do desempenho computacional, que ainda pode ser largamente melhorado, a implementação, tal qual descrita neste trabalho, oferece uma ótima relação custo-benefício caso haja interesse em extensões mais práticas deste projeto.

6.2 - Sugestões para Trabalhos Futuros

Embora este projeto implemente uma plataforma bastante abrangente para o reconhecimento de voz, aplicações específicas do modelo, com exceção da autenticação combinada por duas camadas, não são tratadas aqui. Além disso, o trabalho não se trata de um estudo completo sobre as variadas opções de técnicas e algoritmos que podem ser utilizados para implementar um sistema de reconhecimento de voz. Sendo assim, algumas sugestões para trabalhos futuros são apresentadas a seguir.

Integração do sistema com um banco de dados: Sugere-se um trabalho onde o sistema de reconhecimento de voz possa ser implementado de forma completa e comercial. Isto pode ser alcançado através da integração do sistema proposto com um banco de dados, onde cadastros e informações de usuários seriam armazenados. Com uma arquitetura adequada, pode-se otimizar o processo de autenticação, gerando um potencial grande para aplicações práticas onde a autenticação de usuários seja um requisito.

Aprofundamento no estudo de Redes Neurais: No presente trabalho, apenas uma arquitetura simples de redes neurais artificiais foi estudada, pois o foco estava no desenvolvimento de uma implementação vetorizada do método dos grandientes conjugados. Assim, uma sugestão interessante de trabalho futuro é um estudo mais abrangente do impacto da escolha da arquitetura de rede neural na qualidade do sistema de reconhecimento. Poderia-se explorar, por exemplo, arquiteturas como as Redes de Função de Base Radial, Mapas Auto-Organizantes ou arquiteturas com salto de camada. Outros exemplos de aprofundamento são: o estudo do impacto da escolha da função de ativação, a inclusão de regularização bayesiana no treinamento e a escolha de outras técnicas de treinamento, como o algoritmo de Levenberg-Marquadt, o algoritmo BFGS ou técnicas baseadas em arrefecimento simulado.

Aprofundamento no estudo de Modelos de Markov: De maneira semelhante ao caso das redes neurais, este trabalho se concentra no uso de uma forma relativamente básica do modelo oculto de Markov. Uma sugestão para um trabalho futuro seria então a exploração de técnicas e problemas mais sofisticados relacionados a esta classe de modelos, como por exemplo: como fazer o sistema reconhecer frases inteiras? Ou como implementar um sistema inteligente de análise sintática ou semântica com base no sinal de voz? Outras opções de trabalhos futuros podem envolver o estudo do impacto da escolha de outros métodos que o mel-cepstrum para a extração de características. Ou ainda, a escolha de outras estruturas que não sejam uma mistura gaussiana para modelar a probabilidade de emissão de um dado estado. Além disso, pode-se explorar outras implementações do algoritmo *expectation maximization* ou implementações de um algoritmo de treinamento baseado no algoritmo de Viterbi.

BIBLIOGRAFIA

BAUM, Leonard; PETRIE, Ted. **Statistical Inference for Probabilistic Functions of Finite State Markov Chains**. Annals of Mathematical Statistics: revista do Institute of Mathematical Statistics, [S.l],vol 37, n. 6, 1966.

CAMPBELL, Joseph. **Speaker Recognition:** A tutorial. Preceedings of the IEEE: revista do IEEE, [S.l.], vol. 85, n. 9, 1997.

CHAN, Man-Chung et al. Financial Time Series Forecasting By Neural Network Using Conjugate Gradient Learning Algorithm And Multiple Linear Regression Weight Initialization**. Computing in Economics and Finance: revista da Society for Computational Economics, Barcelona, n. 61, 2000.**

HAGAN, Martin; MENHAJ, Mohammad. **Training Feedforward Networks With the Marquadt Algorithm**. IEEE Transactions on Neural Networks, [S.l.], vol. 5, n. 6, 1994.

HAYKIN, Simon. **Neural Networks**: A Comprehensive Foundation. NewYork: Macmillan, 1994.

HAYKIN, Simon; VEEN Barry. **Sinais e Sistemas**. Porto Alegre: Artmed, 2001.

HORNIK, Kurt. **Approximation Capabilities of Multilayer Feedforward Networks**. Neural Networks: revista especializada em redes neurais, vol. 4, n. 2, Oxford, 1991.

KOLLER, Daphne; FRIEDMAN, Nir. **Probabilistic Graphical Models**. Cambrige, USA: MIT Press, 2008.

LUENBERGER, David. **Optimization by Vector Spaces Methods.** New York: John Wiley & Sons, 1997.

MOLER, Cleve. **Numerical Computing With MATLAB.** [S.l.], Society for Industrial and Applied Mathematics, 2004.

NILSSON, Mikael; EJNARSSON, Marcus. **Speech Recognition Using Markov Hidden Model:** Performance Evaluation in Noisy Environment. 2001. Trabalho de conclusão de curso (Mestrado em Engenharia Elétrica) – Department of Telecommunications and Signal Processing, Blekinge Institute of Technology, Sweden, 2002.

OLIVEIRA, Marcos Paulo. **Verificação automática do locutor, dependente do texto,utilizando sistemas híbridos MLP/HMM**. 2001. Dissertação de Mestrado, Instituto Militar de Engenharia, Rio de Janeiro, 2001.

RABINER, Lawrence. **A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition**. Preceedings of the IEEE: revista do IEEE, [S.l.], vol. 77, n. 2, 1989.

RABINER, Lawrence; JUANG, Biing-Hwang. **Fundamentals of Speech recognition**.New York: Prentice Hall, 1993.

SCHEWCHUK, Johnatan. **An Introduction to the Conjugate Gradient Method Without the Agonizing Pain**. Pittsburgh: Carnegie Mellon University's School of Computer Science, 1994.

WERBOS, Paul. **Beyond Regression: New Tools For Prediction and Analysis in Behavioral Sciences.** 1974. Tese de Doutorado, Harvard University, Cambridge, USA, 1974.

ZHANG et al. **Forecasting With Neural networks:** The State of Art. International Journal of Forecasting, n. 14, Oxford, 1997.

APÊNDICE A – Sistema de Reconhecimento de Voz

```
##================================================================
##Sistema de Reconhecimento de voz: versão 0.1 Novembro de 2012
##Autor: Lucas Mesquita Nogueira
##RA: 2081748/9
## '#' marca comentários
##
##
##================================================================

1 library(signal)  #carrega bibliotecas necessárias
2 library(seewave)
3 library(tuneR)
4 library(RHmm)
5 source("nnet.R")
6 library(RODBC)

7 setwd("#insira o diretório onde se encontram as amostras de voz")

8 ReadWave<-function(dir){ #função para ler amostras de sinais das palavras "um", "dois" e
"três", seis vezes cada um

9       rootdir<-getwd()
10      setwd(dir)
11      sinais1<-vector("list",6)
12      sinais2<-vector("list",6)
13      sinais3<-vector("list",6)

14      for(i in 1:6){
15              if(i==1){
16                      signalnumber<-"
17              }
18              else{
```

```
19                      signalnumber<-i
20              }
21              filename<-paste("um",signalnumber,".wav",sep='')
22              sinais1[[i]]<-readWave(filename=filename)
23      }

24      for(i in 1:6){
25              if(i==1){
26                      signalnumber<-"
27              }
28              else{
29                      signalnumber<-i
30              }
31              filename<-paste("dois",signalnumber,".wav",sep='')
32              sinais2[[i]]<-readWave(filename=filename)
33      }

34      for(i in 1:6){
35              if(i==1){
36                      signalnumber<-"
37              }
38              else{
39                      signalnumber<-i
40              }
41              filename<-paste("tres",signalnumber,".wav",sep='')
42              sinais3[[i]]<-readWave(filename=filename)
43      }
44      return(sinais=list(sinais1=sinais1,sinais2=sinais2,sinais3=sinais3))
45 }

46 DAV<-function(sinal){ #função de detecção da ativação da voz
47      blocksize<-320  #tamanho do frame 'hardcoded' para testes, pode ser passado como
argumento no futuro
        Sc<-1000
```

```
48      sinal<-sinal@left
49      tamanho<-ceiling(length(sinal)/blocksize)
50      potencia<-rep(0,tamanho)
51      potencia[1]<-(1/blocksize)*sum((sinal[1:blocksize])^2) #potencia a curto prazo do
sinal
52      for(i in 2:tamanho){
53              potencia[i]<-(1/blocksize)*sum((sinal[(((i-1)*blocksize)+1):(i*blocksize)])^2)
54      }
55      potencia[is.na(potencia)]<-potencia[(length(potencia)-2)]
56      taxacruzamento<-rep(0,tamanho)
57      taxacruzamento[1]<-(1/blocksize)*sum(abs(sign(sinal[blocksize:blocksize*2])-
sign(sinal[1:blocksize]))/2) #taxa de cruzamento do eixo X
58      for(i in 2:length(taxacruzamento)){
59              taxacruzamento[i]<-
(1/blocksize)*sum(abs(sign(sinal[((i*blocksize)+1):((i+1)*blocksize)])-sign(sinal[(((i-
1)*blocksize)+1):(i*blocksize)]))/2)
60      }
61      taxacruzamento[is.na(taxacruzamento)]<-taxacruzamento[(length(taxacruzamento)-2)]
62      W<-potencia*(1-taxacruzamento)*Sc
63      W2<-W[1:10]
64      ruidodefundo<-mean(W2,na.rm=T)+sd(W2,na.rm=T) #estimativa do ruído de fundo
65      W[W<ruidodefundo]<-0
66      W[W>ruidodefundo]<-1
67      if(W[1]==0){ #bloco de aplicação da função ao sinal completo
68              DAVfinal<-rep(0,blocksize)
69      }
70      else{
71              DAVfinal<-rep(1,blocksize)
72      }
73      for(j in 2:length(W)){
74              if(W[j]==0){
75                      DAVfinal<-c(DAVfinal,rep(0,blocksize))
76              }
77              else{
```

```
78                      DAVfinal<-c(DAVfinal,rep(1,blocksize))
79              }
80      }
81      if((length(sinal)-length(DAVfinal))!=0){
82              cond<-(length(sinal)-length(DAVfinal))
83              if(cond<0){
84                      DAVfinal<-DAVfinal[1:(length(DAVfinal)+cond)]
85              }
86              else{
87                      DAVfinal<-c(DAVfinal,rep(DAVfinal[length(DAVfinal)],cond))
88              }
89      }
90      voz<-sinal*DAVfinal
91      voz<-voz[abs(voz)>0]
92      voz<-Wave(left=voz,samp.rate=16000,bit=16)
93      return(voz)
94 }


95 melCepstrum<-function(sinal){
96      ceps<-melfcc(sinal,preemph=0.95,numcep=20,frames_in_rows=T) # função 'melfcc'
97 impelementa o método mel-cepstrum completo
98      return(ceps)
99 }

100 FindDeltas<-function(cepstrums,acceleration=TRUE){ #cálculo das derivadas da
trajetoria no dominio cepstral

101     if(acceleration==FALSE){
102             input<-t(cepstrums)
103             delta<-deltas(input,w=3)
104             delta<-t(delta)
105             return(delta)
106     }
```

```
107     else{
108             input<-t(cepstrums)
109             delta<-deltas(input,w=3)
110             delta<-t(delta)
111             accel<-deltas(delta)
112             return(list(delta=delta,accel=accel))
113     }
114 }
115 buildInputs<-function(cepstrum,derivatives,type,user){ #adequação dos inputs a cada
modelo
116     hmmInputs<-function(cepstrum,derivatives){
117             inputs<-cepstrum
118             for(i in 1:length(cepstrum)){
119                     inputs[[i]]<-
rbind(inputs[[i]],derivatives[[i]])##$delta)##,derivatives[[i]]$accel)
120             }
121             return(inputs)
122     }
123     nnInputs<-function(cepstrum,derivatives){
124             extract24first<-function(cepstrum){ ##extração das 24 primeiras observações
para NeuralNets
125                     cepstrum<-cepstrum[1:24,]
126             }
127             cepstrum<-lapply(cepstrum,extract24first)
128             Y<-c(c(rep(1,24*2*3),rep(0,24*2*3)),c(rep(0,24*3*2),rep(1,24*3*2)))
129             Y<-matrix(Y,nrow=288,2)
130             derivatives<-lapply(cepstrum,findDeltas)
131             X<-rep(0,0)
132             for(i in 1:length(cepstrum)){
133                     X<-
rbind(X,rbind(cepstrum[[i]],derivatives[[i]]$delta,derivatives[[i]]$accel))
134             }
135             ##X<-t(X)
136             return(inputs=list(X=X,Y=Y))
```

```
137     }
138     inputs<-switch(type,
                  "hmm"=hmmInputs(cepstrum,derivatives),
                  "nn"=nnInputs(cepstrum,derivatives)
                  )
139 }
140 callMarkov<-function(inputs,type,models=NULL){
141     getLLH<-function(models){
142             LLH<--1000000000
143             for(i in 1:length(models)){
144                     LLH<-models[[i]]$LLH
145             }
146     }
147     predictMarkov<-function(inputs,models){
148             lapply(models,forwardBackward,list(object=inputs))
149             LLH<-getLLH(models)
150             maxprob<-which.max(LLH)
151     }
152     markov<-switch(type,
"fit"=HMMFit(inputs,nStates=2,nMixt=4,dist="MIXTURE",control=list(verbose=1))),
                  "predict"=predictMarkov(inputs,models)
                )
153     return(markov)
154 }
155 callNeuralNet<-function(inputs,users,type,model=NULL){
156     fitNNnet(inputs,users){
157             W<-InitializeWeights(c(20,1,10,users))
158             output<-
ConjugateGradient(inputs$X,inputs$Y,W,lambda=0,"sigmoid","linear",2,maxIter=1000)
159     }
160     out<-switch(type,
        "fit"=fitNNet(inputs,users),
        "predict"=predict(inputs$X,model))
        return(out)
```

```
161 }
```

APÊNDICE B – Implementação Vetorizada de Redes Neurais Artificiais

```
##================================================================
##Implementação vetorizada de Redes Neurais: versão 0.1 Novembro de 2012
##Autor: Lucas Mesquita Nogueira
##RA: 2081748/9
## '#' marca comentários
##
##
##================================================================


1 InitializeWeights<-function(netDesign,initMethod){ #inicialização aleatória dos pesos

2       if(initMethod=="random"){
3               Wcirc<-vector("list",(netDesign[2]+1))
4               Wcirc[[1]]<-runif(netDesign[3]*(netDesign[1]+1),-1,1)
5               Wcirc[[1]]<-matrix(ncol=netDesign[3],nrow=(netDesign[1]+1),Wcirc[[1]])

7               if(netDesign[2]!=1){
8                       for(i in 2:(length(Wcirc)-1)){
9                               Wcirc[[i]]<-runif(netDesign[3]*(netDesign[3]+1),-1,1)
10                              Wcirc[[i]]<-
matrix(ncol=netDesign[3],nrow=(netDesign[3]+1),Wcirc[[i]])
11                      }
12              }
13              Wcirc[[length(Wcirc)]]<-runif((netDesign[3]+1)*(netDesign[4]),-1,1)
14              Wcirc[[length(Wcirc)]]<-
matrix(ncol=netDesign[4],nrow=(netDesign[3]+1),Wcirc[[length(Wcirc)]])
15      }
16      return(list(Wcirc=Wcirc))
17 }
```

```
18MatrixForwardPropagation<-
function(X,W,myActFunction,myOutputFunction=actFunction){ #etapa de propagação dos inputs

19      if(myActFunction=="sigmoid"){ #definição das funções de ativação e suas derivadas
20              actFunction<-function(x)1/(1+exp(-x))
21      }
22      else{
23              if(myActFunction=="tanh"){
24                      actFunction<-function(x)(exp(2*x)-1)/(exp(2*x)+1)
25              }
26              else{
27                      if(myActFunction=="softmax"){
28                              actFunction<-function(x)exp(x)/sum(exp(x))
29                      }
30                      else{
31                              print("invalid Activation Function")
32                      }
33              }
34      }

35      if(myOutputFunction=="sigmoid"){
36              outputFunction<-function(x)1/(1+exp(-x))
37      }
38      else{
39              if(myOutputFunction=="linear"){
40                      outputFunction<-function(x)x
41              }
42              else{
43                      if(myOutputFunction=="tanh"){
44                              outFunction<-function(x)(exp(2*x)-1)/(exp(2*x)+1)
45                      }
46                      else{
47                              if(myOutputFunction=="softmax"){
```

```
48                                      outputFunction<-function(x)exp(x)/sum(exp(x))
49                              }
50                              else{
51                              print("invalid Activation Function")
52                              }
53                      }
54              }
55      }


56      dimhidden<-(length(W$Wcirc)) #forward propagation começa aqui
57      a<-vector("list",dimhidden)
58      abias<-vector("list",dimhidden)
59      z<-vector("list",dimhidden)
60      a[[1]]<-X
61      bias<-rep(1,ncol(X))
62      abias[[1]]<-rbind(bias,a[[1]])
63      for(i in 1:dimhidden){
64              z[[i]]<-t(-W$Wcirc[[i]])%*%abias[[i]]
65              if(i!=dimhidden){
66                      a[[i+1]]<-actFunction(z[[i]])
67                      bias<-rep(1,ncol(a[[i+1]]))
68                      abias   [[i+1]]<-rbind(bias,a[[i+1]])
69              }
70      }
71      h<-outputFunction(z[[dimhidden]])

72      return(list(a=abias,z=z,h=h))
73 }

########### Back Propagation (Also Matrix Implementation) ###########
#####################################################################
```

```
74MatrixBackPropagation<-
function(Y,W,forwardResults,myActFunction,myOutputFunction,lambda){  #retropropagação
do sinal de erro

75      if(myActFunction=="sigmoid"){ #funções de ativação e suas derivadas
76                      actFunction<-function(x)1/(1+exp(-x))
77                      actPrime<-function(x)actFunction(x)*(1-actFunction(x))
78      }
79      else{
80              if(myActFunction=="tanh"){
81                      actFunction<-function(x)(exp(2*x)-1)/(exp(2*x)+1)
82                      actPrime<-function(x)1-((actFunction(x))^2)
83              }
84              else{
85                      if(myActFunction=="softmax"){
86                              actFunction<-function(x)exp(x)/sum(exp(x))
87                              actPrime<-function(x)actFunction(x)-(actFunction(x)^2)
88                      }
89                      else{
90                              print("invalid Activation Function")
91                      }
92              }
93      }

94      if(myOutputFunction=="sigmoid"){
95              outputFunction<-function(x)1/(1+exp(-x))
96              outPrime<-function(x)actFunction(x)*(1-actFunction(x))
97      }
98      else{
99              if(myOutputFunction=="linear"){
100                     outputFunction<-function(x)x
101                     outPrime<-function(x)1
102             }
103             else{
```

```
104                     if(myOutputFunction=="tanh"){
105                             outFunction<-function(x)(exp(2*x)-1)/(exp(2*x)+1)
106                             outPrime<-function(x)1-((actFunction(x))^2)
107                     }
108                     else{
109                             if(myOutputFunction=="softmax"){
110                                     outputFunction<-function(x)exp(x)/sum(exp(x))
111                                     outPrime<-function(x)actFunction(x)-(actFunction(x)^2)
112                             }
113                             else{
114                             print("invalid Activation Function")
115                             }
116                     }
117             }
118     }


119     netlength<-length(W$Wcirc) #backpropagation começa aqui
120     gradients<-vector("list",netlength)
121     delta<-vector("list",netlength)
122     delta[[netlength]]<- -(Y-forwardResults$h)*outPrime(forwardResults$z[[netlength]])
123     gradients[[netlength]]<-
delta[[netlength]]%*%t(forwardResults$a[[netlength]])+lambda*t(W$Wcirc[[netlength]])
124     for(i in (netlength-1):1){
125             W$W[[i+1]]<-as.matrix(W$Wcirc[[i+1]][2:nrow(W$Wcirc[[i+1]]),])
126             delta[[i]]<- W$W[[i+1]]%*%delta[[i+1]]*actPrime(forwardResults$z[[i]])
127             gradients[[i]]<-
delta[[i]]%*%t(forwardResults$a[[i]])+lambda*t(W$Wcirc[[i]])
128     }
129      return(list(gradients=gradients))
130 }
```

```
GoldenSearch<-function(a,b,tol,W,X,Y,D,myActFunction,myOutputFunction,maxIter=1000){ #método de busca em secção áurea
    Wa<-W
    Wb<-W
    for(j in 1:maxIter){
        intwidth<-abs(a-b)
        if(abs(intwidth)<tol){
            break
        }

        else{
            c1<-0.618*a+(1-0.618)*b
            c2<-(1-0.618)*a+0.618*b

            for(i in 1:length(W$Wcirc)){
                Wa$Wcirc[[i]]<-W$Wcirc[[i]] + c1*(D[[i]])
                Wb$Wcirc[[i]]<-W$Wcirc[[i]] + c2*(D[[i]])
            }
            fwpassc1<-MatrixForwardPropagation(X,Wa,myActFunction,myOutputFunction)
            fwpassc2<-MatrixForwardPropagation(X,Wb,myActFunction,myOutputFunction)
            costc1<-sum((fwpassc1$h-Y)^2)
            costc2<-sum((fwpassc2$h-Y)^2)
            if(costc1<costc2){
                b<-c2
            }
            else{
                a<-c1
            }
        }
    }
    alpha<-(a+b)/2
```

```
161     return(alpha)
162}

163ConjugateGradient<-
164             function(X,Y,W,lambda,myActfunction,myOutputFunction,niter,maxIter=1000){
#gradientes conjugados!

165     neglist<-function(x)x*(-1)
166     err<-rep(0,(maxIter))
167     netlength<-length(W$Wcirc)
168     for(k in 1:maxIter){
169
170             forward<-
171MatrixForwardPropagation(X,W,myActFunction,myOutputFunction) #1ª etapa
172             err[k]<-sum((forward$h-Y)^2)
173             if(err[k]<0.00000001)break
174             back<-
175MatrixBackPropagation(Y,W,forward,myActFunction,myOutputFunction,lambda)
176             D<-lapply(back$gradients,t)
177             D<-lapply(D,neglist)
178             alpha<-
GoldenSearch(0,1,0.0001,W,X,Y,D,myActFunction,myOutputFunction)
179             for(i in 1:netlength){
180                     W$Wcirc[[i]]<-W$Wcirc[[i]]+(alpha*D[[i]])
181             }

182             for (j in 2:niter){ #otimização iterativa

183                     g<-back$gradients
184                     forward<-
185MatrixForwardPropagation(X,W,myActFunction,myOutputFunction)
186                     err[k]<-sum((forward$h-Y)^2)
187                     back<-
188MatrixBackPropagation(Y,W,forward,myActFunction,myOutputFunction,lambda)
```

```
189                     for(i in 1:netlength){
190                             beta1<-(t(back$gradients[[i]])%*%(back$gradients[[i]]-
g[[i]]))/(t(g[[i]])%*%g[[i]])
191                             D[[i]]<-t(back$gradients[[i]])+beta1%*%D[[i]]
192                     }
193                     alpha<-
194GoldenSearch(0,1,0.0001,W,X,Y,D,myActFunction,myOutputFunction)  #cálculo  do
tamanho de alpha: o tamanho do "passo"

195                     for(i in 1:netlength){
196                             W$Wcirc[[i]]<-W$Wcirc[[i]]+(alpha*D[[i]])
197                     }
198             }
199     ##err<-err[k]
200     }
201     return(list(W=W,err=err))
202}
```