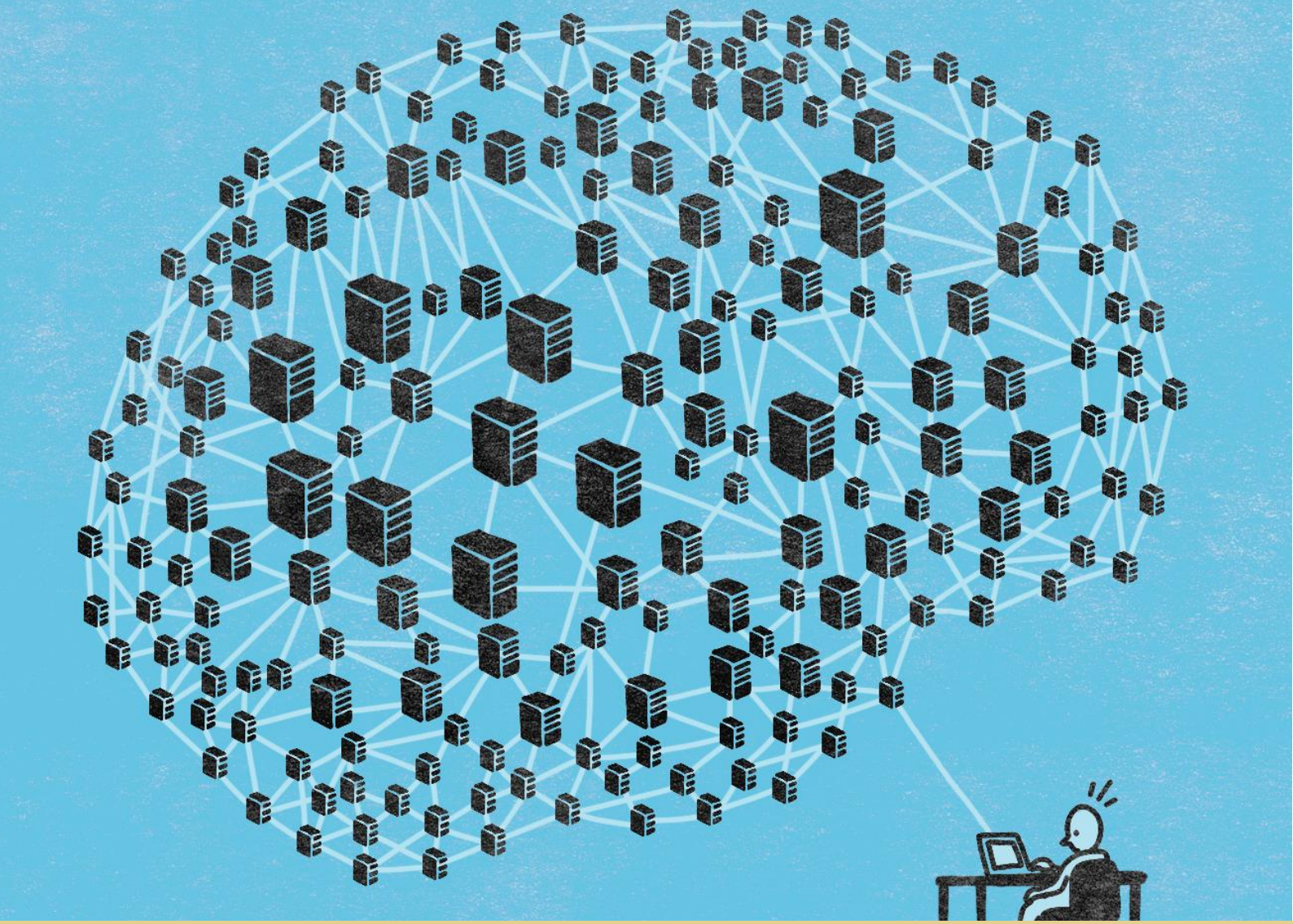

Guilherme de Abreu Franco

Reconhecedor de fonemas em português europeu baseado em redes neuronais

Setembro de 2016

· U C ·

Universidade de Coimbra

Faculdade de Ciências e Tecnologia, Universidade de Coimbra

Mestrado Integrado em Engenharia Electrotécnica e de Computadores

Reconhecedor de fonemas em português europeu baseado em redes neuronais

Guilherme de Abreu Franco

Orientador:
Fernando Manuel dos Santos Perdigão

Co-orientador:
Gabriel Falcão Paiva Fernandes

Júri:

Presidente - Luís Alberto da Silva Cruz

Membro - Fernando Santos Perdigão

Membro - Carla Alexandra Calado Lopes

Setembro de 2016

Agradecimentos

Em primeiro lugar, quero agradecer à minha família pelo apoio incondicional que sempre mostraram para comigo, não só durante o percurso académico, como também em todas as outras situações, boas e más, com que me deparei ao longo da vida.

Quero agradecer especialmente também ao Professor Doutor Fernando Perdigão pela orientação, disponibilidade e motivação fundamentais ao desenvolvimento desta tese, pela prontidão em esclarecer quaisqueres dúvidas acerca do projecto, pelos conhecimentos transmitidos, pelo tempo dispendido a favor desta tese, e pela fomentação do interesse no tema deste trabalho.

Quero deixar também um agradecimento sincero ao Professor Doutor Gabriel Falcão pela ajuda prestada, e prontidão em apoiar o desenvolvimento desta tese.

Por fim, queria agradecer aos meus colegas de laboratório Jorge Proença, João Amaro e Hugo Ferreira pelo bom ambiente e ajuda fundamental em algumas fazes críticas desta tese.

Quero também agradecer à Universidade de Coimbra, ao Instituto de Telecomunicações, e ao Centro de Pesquisa CUDA da Nvidia do Laboratório de processamento de Imagem por disponibilizarem as instalações e equipamento necessários à realização deste trabalho.

Resumo

Nesta tese aborda-se o problema de reconhecimento automático de fonemas em português europeu através do uso de redes neuronais profundas (DNNs – Deep Neural Networks) em conjugação com modelos de Markov não observáveis (HMM). Implementam-se soluções para esse mesmo problema, explorando diversas arquitecturas para as DNNs usando vários tipos de parâmetros.

Começa-se com a implementação de reconhecedores simples, em que as redes neuronais têm apenas uma ou duas camadas escondidas.

Posteriormente implementa-se um sistema que usa três contextos temporais (esquerdo, central e direito), baseado no reconhecedor desenvolvido pela Universidade de Brno.São ainda feitas tentativas de melhoria do desempenho das redes implementadas.

O desenvolvimento desses sistemas é feito usando o framework CNTK, criado pela Microsoft, que possibilita o treino de forma eficiente dos sistemas em placas gráficas (ou GPUs – Graphics Processing Units).

O sistema de GPUs usado é disponibilizado pelo centro de pesquisa CUDA da NVidia, localizado no laboratório de processamento de imagem do Instituto de Telecomunicações da Universidade de Coimbra.

Palavras-Chave : DNN, Reconhecimento de Fonemas, CNTK, GPU, HMM

Abstract

In this thesis, the problem of automatic European Portuguese phoneme recognition using deep neural networks (DNNs) in conjunction with hidden Markov models (HMMs) is tackled. Solutions to that same problem are implemented, exploring several DNN architectures while using several kinds of parameters.

A simple recognizer is implemented at first, in which the DNNs have only one or two hidden layers.

Then, a system with three contexts is implemented (left, central and right), based on the recognizer developed by Brno University. Attempts are made in order to improve the performance of the implemented networks.

The networks development is achieved through the use of Microsoft's CNTK framework, which makes training using graphics processing units (GPUs) possible.

The used GPU system is provided by NVidia's CUDA research center located at the image processing laboratory, at University of Coimbra's Telecommunications Institute.

Key-Words : DNN, Phoneme recognition, CNTK, GPU, HMM

Índice

Lista de Figuras ii
Lista de Tabelas iv
Lista de Acrónimos v
1 - Introdução 1
Capítulo 2 - Redes Neuronais 2
2.1 Estrutura de uma Rede Neuronal Artificial 2
2.2 Treino de uma DNN 4
2.3 Parâmetros de treino 7
2.3.1 Learning Rate 7
2.3.4 Momentum 7
2.3.2 Mini-Batch Size 8
2.3.3 Época 8
2.3.5 Especialização vs Generalização 8
2.3.6 Droput Rate 9
2.4 Exemplo simples 10
2.5 Modelos de Markov Não Observáveis 14
2.6 Posteriorgrama 15
Capítulo 3 - Testes e Experiências Iniciais 17
3.1 CNTK 17
3.2 Base de Dados 18
3.3 MFCCs 19
3.4 Testes com uma camada 20
3.5 Teste com duas camadas 21
Capítulo 4 - Arquitectura com contexto temporal 22
4.1 Três contextos usando MFCCs 22
4.2 TRAPs 25
Capítulo 5 - Melhorias 30
5.1 Adição e remoção de ruído 30
5.2 Melhoria do sistema TRAP 32
Capítulo 6 - Conclusão 35
Referências: 36
Apêndice 1 37
Apêndice 2.1 38
Apêndice 2.2 43

Apêndice 2.3 46
Apêndice 3 48
Apêndice 4 49

Lista de Figuras

Figura 1 Modelo de um perceptrão. Editado de [20]. ……..……2

Figura 2. Modelo de um MLP com uma camada escondida Editado de [21] …….4

Figura 3. Superfície de erro para quando só há dois pesos a ajustar,
como descrito na equação (9) **……..6**

Figura 4. Evolução do erro durante o treino e teste da uma DNN,
em que é explícito o fenómeno de *overfitting.* **……9**

Figura 5. Pontos usados para o treino da DNN exemplo, com respectiva classe colorida……..10

Figura 6. Pontos usados para o teste da DNN exemplo, com respectiva classe colorida……11

Figura 7. ArgMax para a classe Verde…….12

Figura 8. ArgMax para a classe Vermelho…….….12

Figura 9. ArgMax para a classe Amarelo…….13

Figura 10. Matriz de confusão para a DNN exemplo tomada na secção 2.4…….13

Figura 11. Sequência de estados com respectivas probabilidades de transição usado na
descodificação de Viterbi de um vector de estados proveniente de uma DNN……..…15

Figura 12. Posteriorgrama da saída de uma DNN, com 120 estados
(3 estados para cada um dos 40 fonemas) ……16

Figura 13. Posteriorgrama da saída de uma DNN, com 120 estados(3 estados
para cada um dos 40 fonemas), em que as saídas foram concatenadas 3 a 3……..16

Figura 14. Traçado do ArgE para o conjunto de treino……21

Figura 15. Arquitectura do reconhecedor com contexto esquerdo, central e direito
que faz uso de MFCCs……..…23

Figura 16. Desempenho da melhor rede obtida com contexto central usando MFCCs…….24

Figura 17. Esquema da aquisição dos contextos temporais e classificadores por banda
do sistema TRAP original. Editado de [8]. ……..…25

Figura 18. Aquisição dos contextos esquerdo e direito da saída de um banco de
filtros MEL. ……..…26

Figura 19. Arquitectura do reconhecedor com contexto esquerdo e direito
baseado no sistema TRAP. ……..27

Figura 20. Desempenho da rede com 39 fonemas, usando TRAPs com
contexto direito e esquerdo……...28

Figura 21. Evolução do CorrE e AccE durante o teste para a rede com 40
fonemas e uso de TRAPs com contexto esquerdo e direito descrita na secção 4.2…..29

Figura 22. Filtro Chebyshev passa-baixo (módulo e fase) usado para colorir o ruído……..30

Figura 23. Ruído branco (a azul), e ruído colorido (a vermelho), onde é evidente a ausência
das componentes de alta frequência no ruído colorido…….31

Figura 24. Evolução dor CorrE e AccE ao longo das épocas de treino para o reconhecedor com
TRAPs, adição de contexto central, e descodificação com *free*-phone *loop*……33

Figura 25. Evolução dor WER e AER ao longo das épocas de treino para o reconhecedor com TRAPs, adição de contexto central, e descodificação com *free*-phone *loop*……33

Figura 26. Evolução do CorrE e AccE ao longo das épocas de treino para o reconhecedor com TRAPs, expansão artificial da base de dados, e descodificação com *free*-phone *loop*…….34

Lista de Tabelas

Tabela 1. Comparação dos tempos de treino de uma época usando a mesma DNN, para as diferentes máquinas usadas……...17

Tabela 2. Comparação de erros de teste e treino para os mesmos parâmetros de treino, variando o nº de nodos escondidos……..20

Tabela 3. Resultados dos testes feitos para a rede com contexto central usando MFCCs…….....24

Lista de Acrónimos

DNN: Deep Neural Network

HMM: Hidden Markov Model

ANN: Artificial Network

MLP: Multi-Layer Perceptron

SGD: Stochastic Gradient Descent

CNTK: Convolutional Network Toolkit

CPU: Central Processing Unit

GPU: Graphics Processing Unit

MFCC: Mel-Frequency Cepstral Coefficients

TRAP: Temporal Pattern

DFT: Discrete Fourier Transform

DCT: Discrete Cossine Transform

MLF: Master Label File

IT: Instituto de Telecomunicações

1 - Introdução

O reconhecimento de fala como tarefa computacional tem sido alvo de crescente atenção nos últimos anos, não só pela importância que lhe está associada, mas também pelo rápido aumento do poder de processamento e investimento em investigação na área de tecnologias de informação e comunicação.

O desenvolvimento de um sistema que seja capaz de reconhecer fonemas com uma taxa de sucesso alta representa um desafio de elevado grau de dificuldade. Um método que já deu provas que cumpre os requisitos é o treino e uso de DNNs - Deep Neuronal Networks, e uso dessas mesmas redes em conjunto com HMMs em tarefas de reconhecimento de fala.

Nesta tese aborda-se o problema do reconhecimento de fonemas, dado o sinal acústico da fala. Fonema é uma unidade básica abstracta que serve para descrever os diferentes sons de uma dada língua. Podemos assumir que as palavras de uma dada língua são constituídas pela concatenação de fonemas. Assim tal como diferentes sequências de palavras formam frases diferentes, diferentes sequências de fonemas formam palavras diferentes. A realização acústica de um fonema é designado por fone. Assim, quando falamos de reconhecimento de fonemas, podemos dizer que o sinal acústico contém fones que se pretendem associar às respetivas classes, os fonemas.

Os reconhecedores de fonemas podem ser de diversos tipos, contudo, um paradigma usual e omnipresente nos sistemas de reconhecimento de fala são baseados em redes neuronais. As redes neuronais estimam a probabilidade a posteriori dos fonemas, dado o sinal de fala.

Capítulo 2 - Redes Neuronais

2.1 Estrutura de uma Rede Neuronal Artificial

Uma Rede Neuronal Artificial (ANN) é um modelo computacional inspirado na estrututa do sistema nervoso presente na maior parte dos animais. ANNs são frequentemente usadas em tarefas de reconhecimento de padrões e classificação, como é o caso do reconhecimento de fala. O componente principal de uma ANN é o perceptrão, que é um modelo de um neurónio. É este componente que é responsável pela tomada de decisões por parte da ANN:

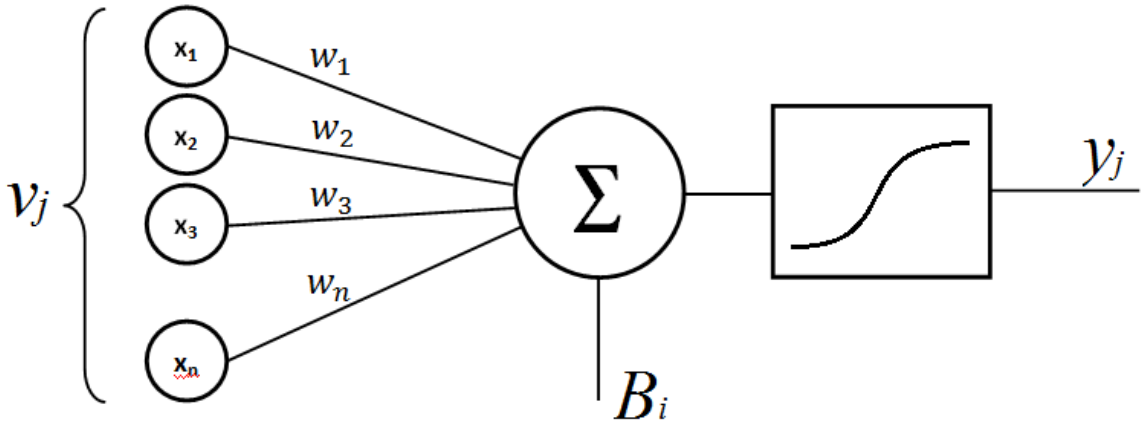

Figura 2 Modelo de um perceptrão. Editado de [20].

O perceptrão é constituído por um conjunto de entradas ($x_1..x_n$) que são multiplicadas por um conjunto de pesos ($w_1..w_n$), e depois são somadas para formar uma saída composta, definida como 'ativação' do neurónio. A essa soma é depois aplicada uma função de activação, que funciona como um limite de decisão a partir do qual a saída do perceptrão é alta ou baixa (true ou false). No caso da figura 1, a derivada desta função é contínua e definida para todo o seu domínio, condição necessária para o treino da rede. Normalmente a função de ativação é *sigmóide* ou *softmax*, em que a função sigmóide é usada nas camadas escondidas da rede (a função *softmax* usa-se normalmente na saída da rede), por ser a função que melhor substitui a função degrau (a derivada desta não é contínua), dada pelas seguintes equações, respectivamente

$$y_j = \frac{1}{1-e^{-v_j}} \quad (1)$$

$$y_j = \frac{e^{v_j}}{\sum_{j=1}^{n} e^{-v_j}} \quad (2)$$

Em que v_j é o parâmetro de entrada da função, y_j a saída, e *n* é o número de saídas da rede Ainda relativamente à figura anterior, à soma das entradas pesadas é ainda somado um peso extra (bias), que permite deslocar a função de activação para a esquerda ou para a direita (diminuir ou aumentar o limiar de decisão, respectivamente). A saída *y* de um perceptrão *i* é então descrita como:

$$y_i = f(x^\top w + B_i) = f(w^\top x + B_i) \quad (3)$$

onde *x* é o vetor de entradas e *w* é o vetor de pesos. Para mais que uma saída, podemos considerar todos os vetores de pesos numa matriz, vindo

$$\boldsymbol{y} = f(\boldsymbol{w}^\top \boldsymbol{x} + \boldsymbol{B}) \quad (4)$$

em que B_i é o Bias do perceptrão *i*, e *f()* é a função de activação

Tipicamente, uma rede neuronal é formada por um número elevado de perceptrões (nodos), tipicamnte na ordem das centenas de milhar ou até mesmo milhões, e tem uma topologia com mais que uma camada, pelo que se denomina por perceptrão multi-camada (MLP - Multi-Layer Perceptron). Por exemplo, uma das redes testadas para o reconhecimento de fonemas tem 195 entradas, 1000 nodos escondidos, e 120 saídas. Os nodos estão organizados em paralelo, e um conjunto de nodos organizados em paralelo denomina-se camada. Neste caso referido, a rede tem uma camada escondida:

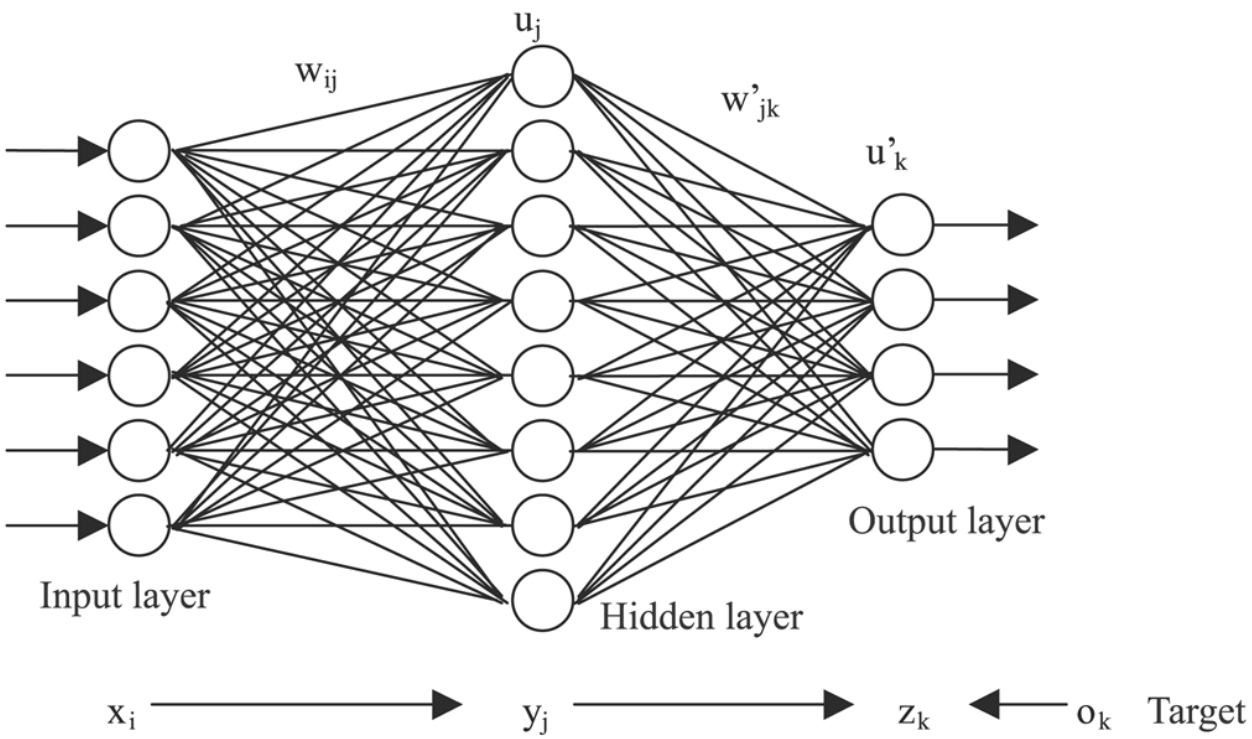

Figura 2. Modelo de um MLP com uma camada escondida Editado de [21]

Estas redes têm a particularidade em que o fluxo dos dados é sempre da entrada para a saída, daí serem denominadas redes "*feed-forward*". Existem também redes ditas recorrentes, em que existem ligações de posteriores para camadas anteriores. Redes recorrentes não são abordadas neste trabalho.

Uma rede DNN é usualmente uma rede *feed-forward*, com várias camadas escondidas.

2.2 Treino de uma DNN

A aprendizagem por parte de uma DNN é feita através do ajuste dos pesos e dos *bias* através de propagações sucessivas de dados etiquetados pela rede (dados que além de conterem vários valores para as entrada, contêm também o valor desejado de saída, que deve ser gerado pela rede. Este tipo de treino é designado por treino supervisionado, pois sabe-se à priori quais os valores que a rede deve gerar. Existe outro tipo de treino designado por treino não-supervisionado, que não é abordado nesta tese, em que a rede é treinada usando dados não etiquetados.

Quando a rede acaba de gerar os valores para um dado conjunto de dados, é feita uma comparação das saídas que gerou com as saídas desejadas. Isto gera um erro, que é retropropagado através da rede, por forma a determinar a influência de cada peso no erro

resultante, e assim determinar o valor dos novos pesos. Este processo é feito através de um algoritmo de retropropagação (backpropagation), usualmente na forma de gradiente estocástico descendente (*SGD - Stochastic Gradient Descent*).

Primeiro, é necessário determinar o erro entre a previsão obtida e o valor correcto. A forma mais comum de calcular esse erro é através do erro quadrático. Seja d_j as saídas desejadas, y_j as saídas calculada pela rede, em que *p* é o número total de saídas. A função de erro $\mathcal{E}$ é, neste caso:

$$\varepsilon = \frac{1}{2}\sum_{j=1}^{p}(d_j - y_j)^2 \tag{5}$$

Queremos um conjunto de pesos que minimize ε, isto é, que façam com que a saída calculada pela rede seja, tanto quanto possível, igual à saída correcta ($d \approx y$). Não existindo uma solução formal definida para o problema, efectuam-se ajustes aos pesos de forma iterativa, no sentido oposto ao gradiente do erro em função dos pesos ($d\mathcal{E}/dw$), por forma a que a solução se aproxime para um mínimo (local):

$$\Delta w_i = -r\frac{\partial \mathcal{E}}{\partial w_i} = -r\delta x_i \tag{6}$$

onde *r* é a constante de aprendizagem (*learning rate*), δ é o erro da retropropagação após passar pela função e activação, e x_i é o conjunto de entradas.

Como a última camada tem como função de activação a função softmax, é conveniente usar como função objectivo o critério da entropia cruzada mínima, em vez do erro quadrático mínimo [9]:

$$\varepsilon = \sum_{j=1}^{p} d_j * \log\left(\frac{d_j}{y_j}\right) \tag{7}$$

Isto significa que o gradiente do erro na última camada passa a ser:

$$\frac{\partial \mathcal{E}}{\partial w} = -\sum_{j=1}^{p}\frac{d_j}{dw} * \frac{\partial y_j}{\partial w} \tag{8}$$

No entanto, o algoritmo de retropropagação mantém-se inalterado.

No caso mais simples em que só existem 2 pesos, a variação dos pesos Δw necessária seria:

$$\Delta w = -r\left(\frac{\partial \mathcal{E}}{\partial w_1}i + \frac{\partial \mathcal{E}}{\partial w_2}j\right) \tag{9}$$

em que i e j são os vectores unitários do sistema de eixos coordenados e r é um factor de escala (*learning rate*). Graficamente, estamos a encontrar a direcção a partir de (w_1, w_2) que converge para o mínimo o, avançando nessa direcção proprocionalmente a r (ver figura 3).

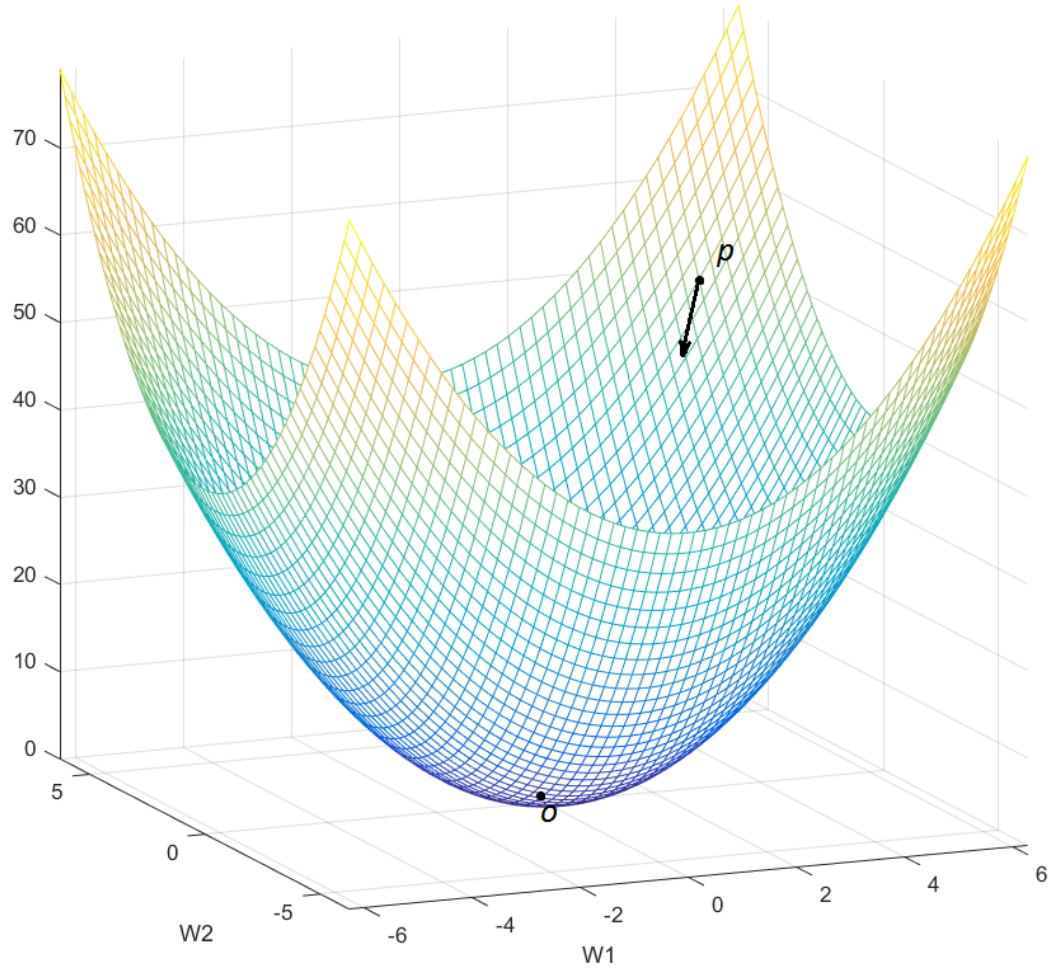

Figura 3. Superfície de erro para quando só há dois pesos a ajustar, como descrito na equação (9)

Para uma DNN com uma camada escondida com n_i entradas, n_h nodos na camada escondida e n_o saídas, o número de pesos n_w e bias n_B é

$$n_w = n_i \times n_h + n_h \times n_o \quad (10)$$

$$n_B = n_h + n_o \quad (11)$$

Por exemplo, para uma rede com 2 entradas, 10 nodos escondidos e 3 saídas, $n_w = 50$ e $n_B = 30$. Mesmo para uma topologia simples, há um grande número de mínimos locais de P, para os quais os pesos e Bias podem convergir, aumentando exponencialmente a complexidade

do problema, pois não se sabe à partida a forma da súperfície de *n* dimensões que representa a função a minimizar.

Foi visto que o treino supervisionado requer dados etiquetados. Em reconhecimento de fala, esses dados encontram-se normalmente sob a forma de um ficheiro MLF. Este ficheiro contém um conjunto de locuções, e para cada locução apresenta uma sequência de fonemas que é usada como referência, em que cada fonema tem associado um tempo de início e fim (ver apêndice 3).

2.3 Parâmetros de treino

2.3.1 Learning Rate

Como foi visto atrás, o *learning* rate é o parâmetro que determina a dimensão do avanço dos pesos sempre que estes são actualizados. No caso da figura 3, se o *learning rate* fôr muito grande, o ponto *p* avança para lá do ponto mínimo *o*, e terá que recuar na próxima iteração. Caso tenha um valor reduzido, são necessáris várias iterações para se chegar ao valor mínimo. A escolha do *learning* rate adequado prova-se então de grande importância para o treino correcto da rede. Existem algoritmos que vão ajustando o learning rate ao longo do treino, ao invés de se escolher um valor fixo para o mesmo. São implementados à definindo um limite superior e inferior para a diferença obtida no erro de treino. Caso a diferença de erro de treino diminua abaixo do limite inferior, o *lerning rate* diminui segundo um factor pré-estabelecido. Caso a diferença de erro de treino aumente acima do limite superior, o *lerning rate* aumenta segundo um factor também pré-estabelecido.

2.3.4 Momentum

Momentum (μ) [3] é um valor escalar associado ao peso (importância) que o Δw da época anterior tem, aquando do cálculo do Δw actual:

$$w_i' = w_i - r * \Delta w_i + \mu * \Delta w_{i-1} \quad , \quad i = \text{época actual} \tag{12}$$

Corresponde à influência que a variação de pesos da época anterior tem no cálculo dos novos pesos. Este método de controlo difere do ajuste automático do *learning rate* na medida em que não o ajusta com base no valor do erro quadrático ou mínima entropia cruzada, mas sim tendo em conta a variação dos pesos na época anterior.

2.3.2 Mini-Batch Size

Em inglês, batch refere-se a uma quantidade de algo (informação, por exemplo) que é processado de uma só vez [21]. O *mini-batch size* (MBsize) é o tamanho do conjunto de dados que se toma para se calcular um gradiente (Δw) parcial. A cada *mini-batch size* amostras, os pesos são actualizados. Por exemplo, se um conjunto de treino tiver um milhão de exemplares (tramas), pode-se fazer progressos mais rapidamente analizando uma fracção desse milhão de exemplares (por exemplo 100) de cada vez para chegar a um valor do gradiente final mais rapidamente [23], e assim actualizar os pesos de forma mais eficaz, em vez de actualizar esses pesos só após a nálise da totalidade de amostras no conjunto de treino. Este parâmetro torna-se bastante relevante para bases de dados de treino de grandes dimensões pois permite uma eficiência computacional maior, especialmente em GPUs que conseguem fazer cálculos matriciais em paralelo. Para um MBsize de grandes dimensões, a variância da actualização do gradiente é reduzida, o que possibilita a escolha de um *learning rate* maior, e assim converge-se mais rapidamente para uma solução óptima. No entanto, não se pode aumentar o *learning rate* para além de um limite que depende da suavidade da função (consultar [1] e [2]).

2.3.3 Época

Uma época corresponde ao treino da DNN com a totalidade de amostras de treino.

2.3.5 Especialização vs Generalização

O objectivo de treinar uma rede neuronal é fazer com que esta se generalize, i.e. consiga prever um conjunto de situações que não tenham sido apresentadas durante o treino. Teoricamente, se o número de nodos for suficientemente elevado, a rede pode atribuir a cada peso um valor que mapeie cada amostra presente no conjunto de treino à saída correcta, atingindo uma taxa de

acerto de 100% para o conjunto de treino, mas falhando as previsões feitas quando lhe é apresentada uma entrada que não tenha estado presente no treino. Para isso, existem dois conjuntos de dados: comjunto de treino e conjunto de teste. O conjunto de treino serve para a rede ser treinada, enquanto que o conjunto de teste serve para avaliar o desempenho da mesma, apresentando-lhe casos com que ela não se deparou durante o treino. No caso limite, a rede adapta-se perfeitamente ao conjunto de treino (0% erro), mas falha redondamente durante o teste. Este fenómeno chama-se *overfitting* (diz-se que a rede se está a especializar), em que cada nodo da rede aprende a identificar cada trama individual do conjunto de treino. Graficamente, isto corresponde à descida do erro de treino e à subida do erro de teste (Figura 4). A melhor rede é aquela em que o erro de teste atinge um mínimo, situação que é oposta à especialização, ou seja generalização.

Na figura 4 vê-se o traçado dos erros de treino e teste em simultâneo para uma rede com 2500 nodos escondidos. Note-se o início da especialização da rede por volta da época 25:

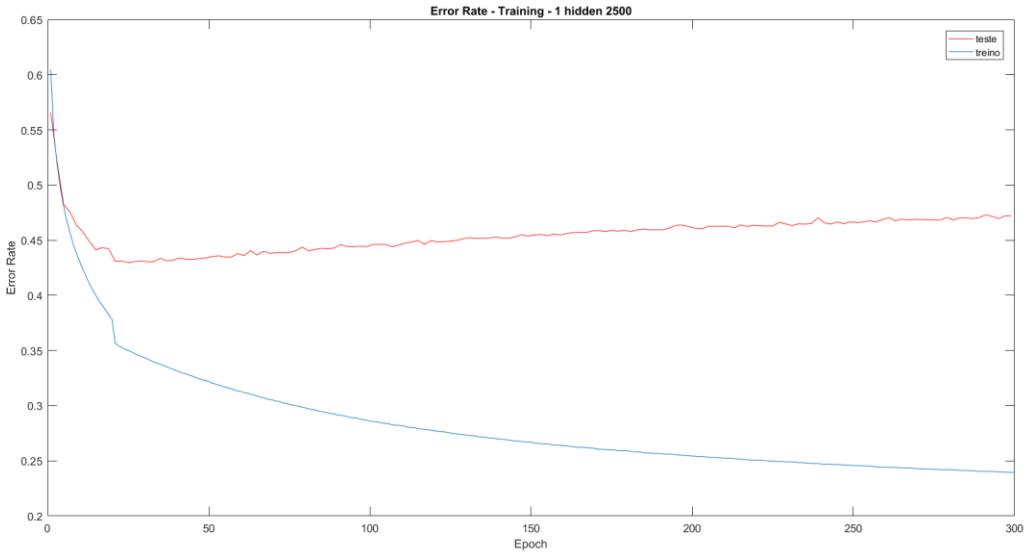

Figura 4. Evolução do erro durante o treino e teste da uma DNN, em que é explícito o fenómeno de *overfitting.*

2.3.6 Droput Rate

Dropout [4] é um método que consiste em temporariamente remover da rede nodos, por forma a que esta não se especialize. Antes de cada época, cada nodo tem uma probabilidade de ser treinado, ou ser temporariamente removido. Essa probabilidade denomina-se *dropout rate*. Este

método torna-se mais relevante em DNNs de grandes dimensões, visto que estas se podem especializar mais facilmente.

2.4 Exemplo simples

Para ilustrar o treino de uma rede neuronal, tomemos como exemplo uma rede com 2 entradas e três saídas. É-lhe fornecida como entrada o par de coordenadas (x,y), e a saída correspondente pode ser uma de três classes: verde, amarelo ou vermelho. A rede deve aprender a identificar correctamente a classe (côr) com base apenas nas coordenadas fornecidas. Para isso é-lhe fornecida uma lista de treino contendo 20000 pontos etiquetados (coordenadas e respectiva classe), tendo estes sido gerados de forma aleatória. Esta solução foi implementada através do MatLab.A figura abaixo mostra os pontos usados para o treino, com a respectiva cor:

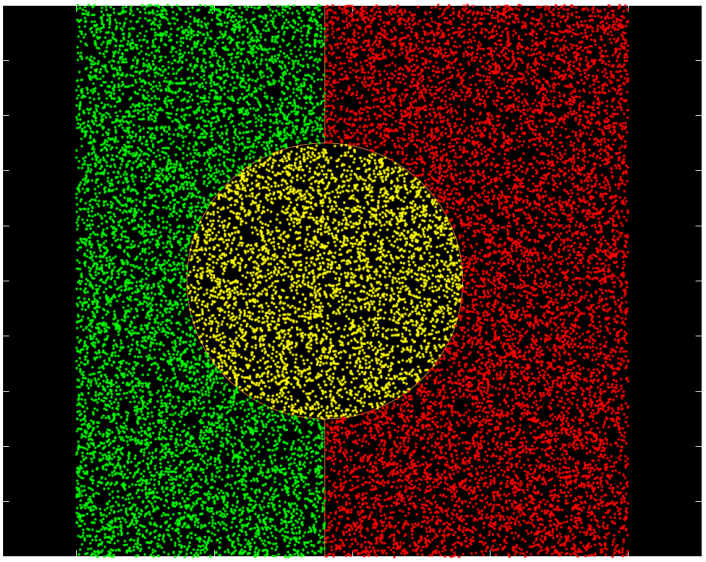

Figura 5. Pontos usados para o treino da DNN exemplo, com respectiva classe colorida

A rede lê os pontos desta lista e ajusta os pesos por forma a se aproximar dos valores correctos, durante várias épocas. Após atingir um número máximo de épocas ou uma outra condição de paragem (normalmente se o erro no treino praticamente não diminuir durante x épocas), o treino acaba. Para se verificar o desempenho da rede, fornece-se à mesma uma lista de teste, que idealmente não deve conter nenhum ponto que esteja presente na lista de treino. Os pontos de teste são os seguintes:

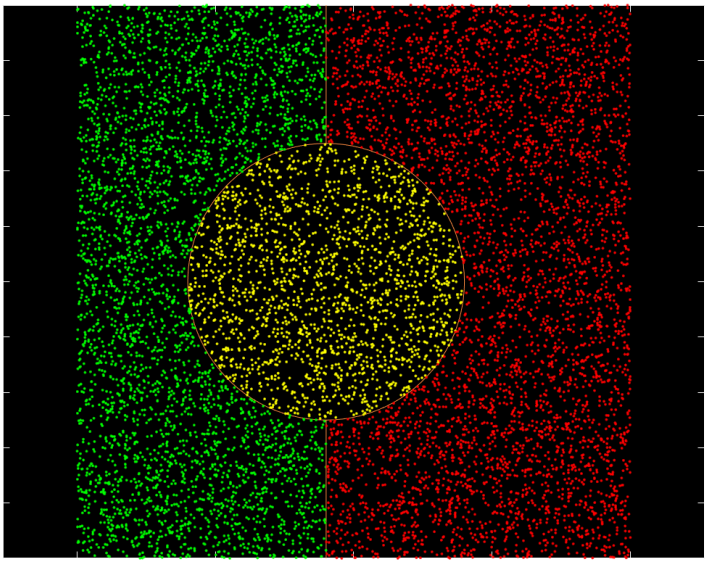

Figura 6. Pontos usados para o teste da DNN exemplo, com respectiva classe colorida

Quando estes pontos são propagados pela rede, a entrada é classificada com uma determinada certeza: cada uma das três saídas da rede ‘vota’ em qual será a classe correcta para um dado ponto com uma certa probabilidade. A soma das probabilidades para as três saídas é igual a 1. Uma forma de avaliar o desepenho da rede consiste em assumir que a saída com maior probabilidade ganha, e a entrada é classificada como pertencendo a essa classe. A este método de decisão chama-se ArgMax. De seguida mostra-se o ArgMax para as três classes em separado, sendo a cor azul correspondente a 0% de certeza para a respectiva classe e a vermelha/castanha a 100%:

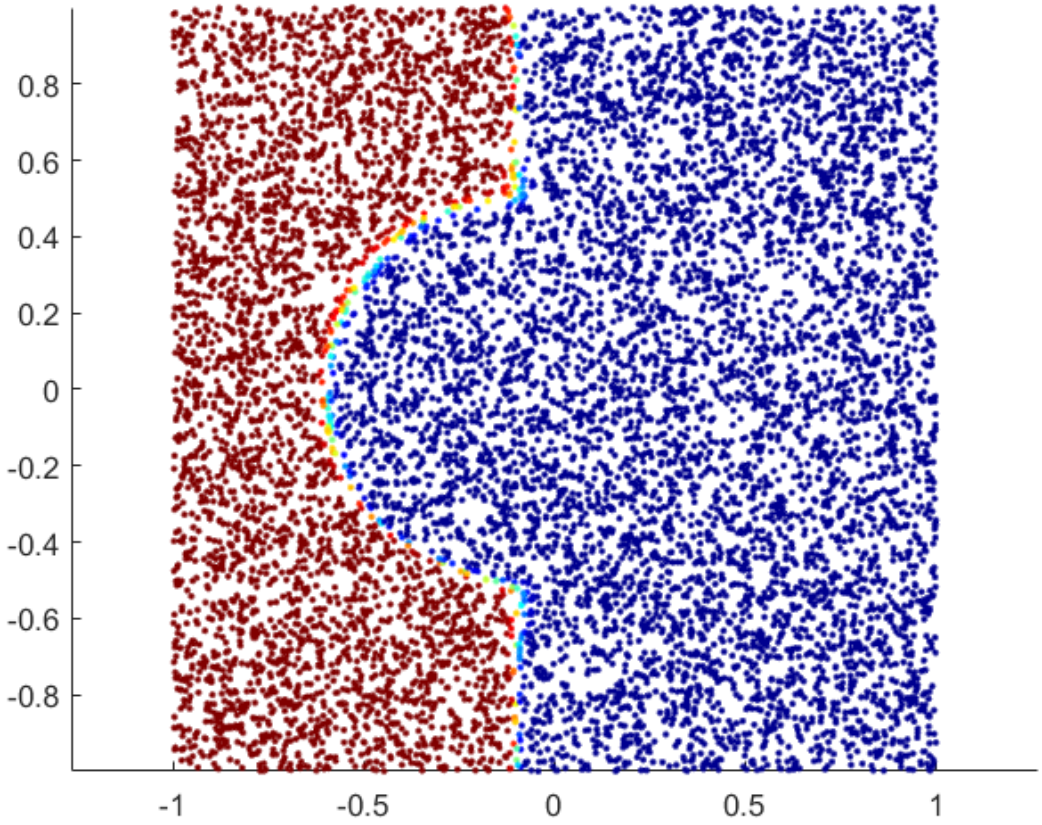

Figura 7. ArgMax para a classe Verde

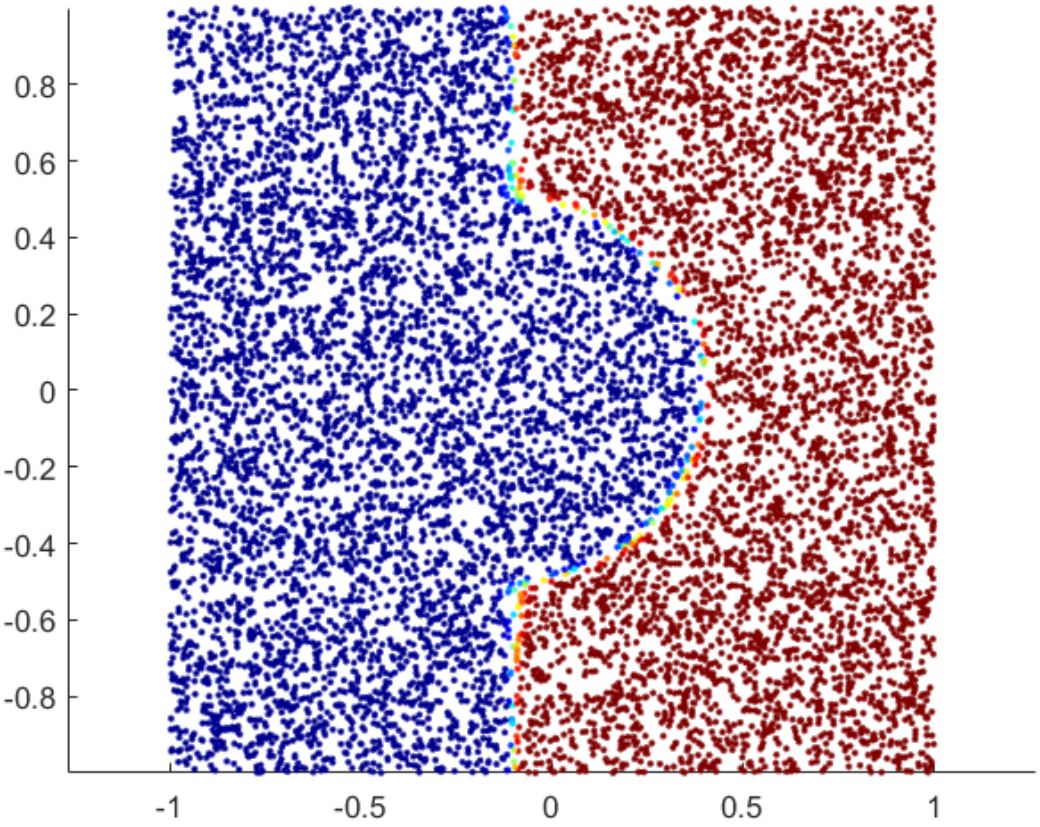

Figura 8. ArgMax para a classe Vermelho

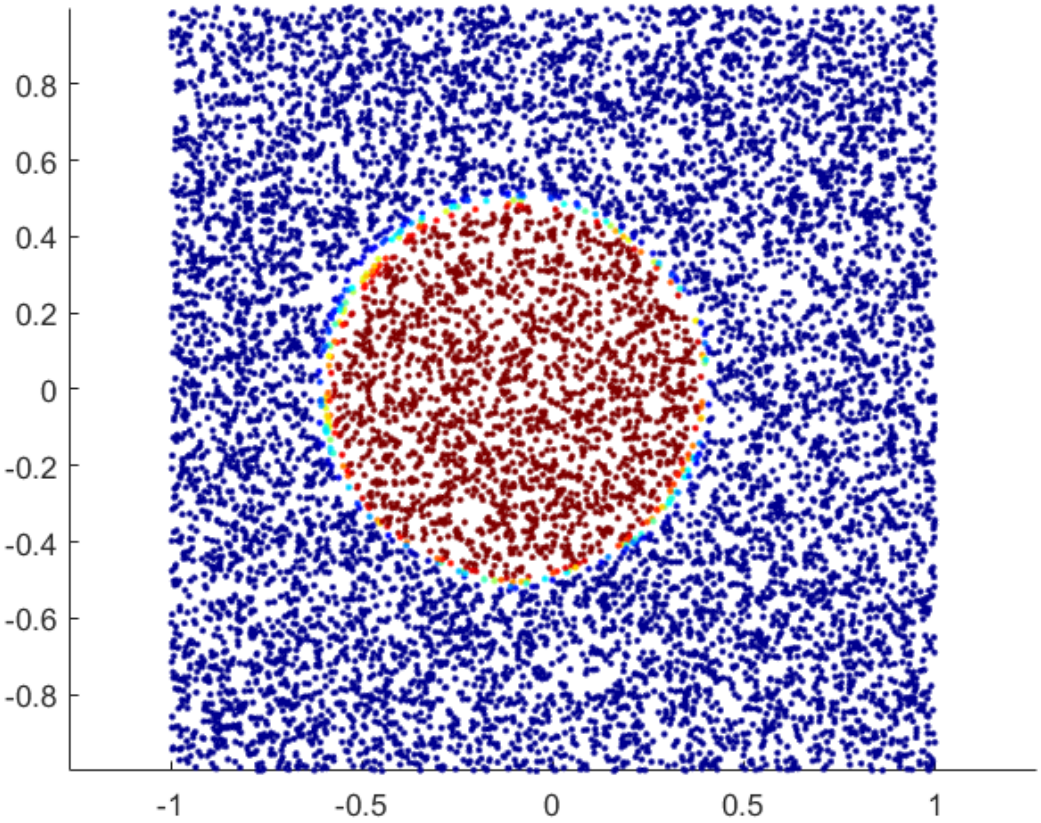

Figura 9. ArgMax para a classe Amarelo

Como se pode constatar, as regiões onde há menos certeza são nas fronteiras (devido à decisão gradual da sigmóide), o que leva a classificações erradas por parte da rede. Uma forma comum de avaliar o comportamento da rede é através da matriz de confusão. Esta matriz mapeia as classes de referência (correctas) das entradas, e compara-as com as classes identificadas pela rede no conjunto de teste.

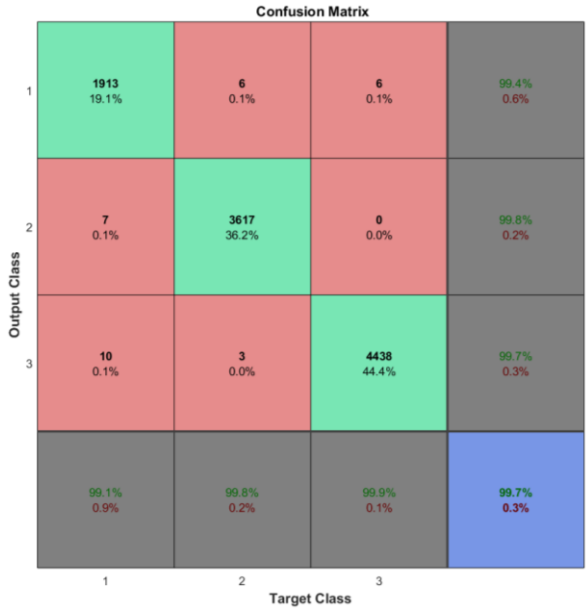

Figura 10. Matriz de confusão para a DNN exemplo tomada na secção 2.4

Na linha inferior da figura 10 a cinzento estão as classes de referência (1-verde, 2-vermelho, 3-amarelo), e na coluna direita a cinzento estão as classes previstas pela rede. A vermelho estão as previsões erradas, e a verde as certas. Por exemplo, a classe 3 (amarelo) foi identificada como sendo da classe 1 (verde) 10 vezes (0,1%), como sendo da classe 2 (vermelho) 3 vezes (<0,1%), e estando correcta 4438 vezes (44,4% do total das amostras – verde, vermelho e amarelo). No quadrado azul está a percentagem de acertos total: 99,7%. É de notar que esta percentagem de erro (0,3%) tem um valor reduzido dada a natureza simples do problema de classificação apresentado, e os dados de treino e teste abrangem praticamente todos os valores possíveis. O problema de reconhecimento de fala é muito mais complexo, e um dos maiores obstáculos é não só arranjar uma base de dados grande o suficiente para representar com fidelidade a fala humana, como também a forma como essa informação é fornecida à rede.

Usualmente os HMM usam probabilidades a priori dos dados acústicos em cada estado. Com redes neuronais são usadas probabilidades a posteiori dos estados (de fonemas) dadas as observaçoes acústicas.

Os HMM modelam a sequência temporal da fala (sequência dos fonemas) enquanto a rede neuronal estima as probabilidades dos fonemas em cada instante. Para a rede ser aplicada aos modelos HMM, esta tem de ter 3 saídas por cada fonema, correspondentes aos estados do modelo HMM de cada fonema.

Para as redes treinadas nesta tese, optou-se pelo uso de 40 fonemas como alfabeto fonético do português europeu (ver apêndice 1). Sendo necessários 3 estados por fonema, há 120 saídas para as redes treinadas, correspondendo às entradas do descodificador de Viterbi.

2.5 Modelos de Markov Não Observáveis

Um modelo de Markov não observável (Hidden Markov Model - HMM) é um modelo estocástico construído a partir de observações conhecidas, modelado em estados que não são obseráveis [15]. No contexto do problema, o objectivo é encontrar a sequência de estados (de fonemas, neste caso) que maximiza a probabilidade dessa mesma sequência se encontrar no modelo feito a partir das locuções conhecidas [14]. Este problema é designado por descodificação, e é solucionado através do algoritmo de Viterbi [16].

A saída de cada rede contém 120 nodos, correspondentes ao número de estados necessários a serem usados pelo HMM aquando a descodificação. Cada nodo representa um estado dos 40 fonemas usados (cada fonema tem 3 estados possíveis) [apêndice 1]. É importante o treino com

estados e não fonemas directamente, pois a saída da rede é ligada a um descodificador de Viterbi (Free Phone Loop), que faz uso desses mesmos estados:

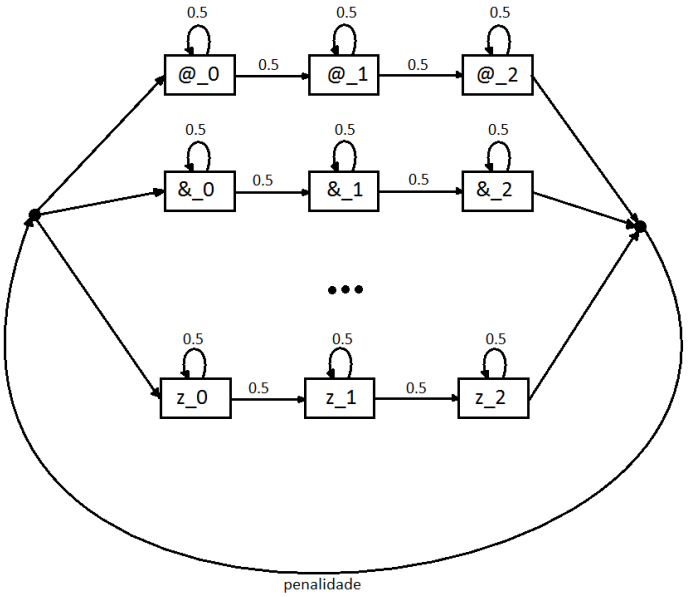

Figura 11. Sequência de estados com respectivas probabilidades de transição usado na descodificação de Viterbi de um vector de estados proveniente de uma DNN.

2.6 Posteriorgrama

Uma forma comum e intuitiva de visualizar a saída de uma rede, especialmente no campo do reconhecimento de fala, é através de posteriorgramas, que indicam as diferentes saídas da rede (eixo das ordenadas, representando os fonemas) em função das tramas de uma locução (eixo das abcissas). Na figura 12, zonas a azul correspondem a baixa probabilidade de fonemas (estados), enquanto que zonas a vermelho/castanho representam elevada probabilidade de fonemas (estados).

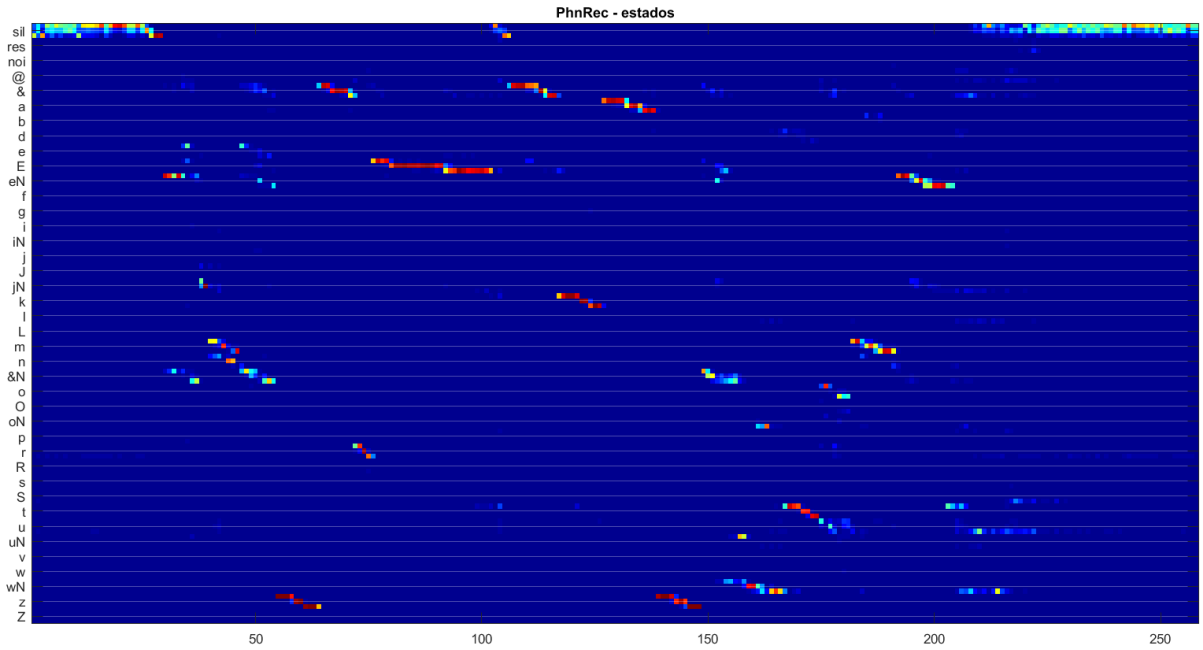

Figura 12. Posteriorgrama da saída de uma DNN, com 120 estados (3 estados para cada um dos 40 fonemas)

Como foi mencionado na secção 1.1, cada saída da rede corresponde à votação de cada nodo, sendo a soma total igual a 1. Isto torna possível a obtenção das saídas da rede em termos de fonemas, e não estados. Para isso, basta somar cada conjunto de três saídas, cujo resultado é um conjunto de 40 valores que correspondem ao peso da votação para cada estado, e cujo resultado também soma a 1:

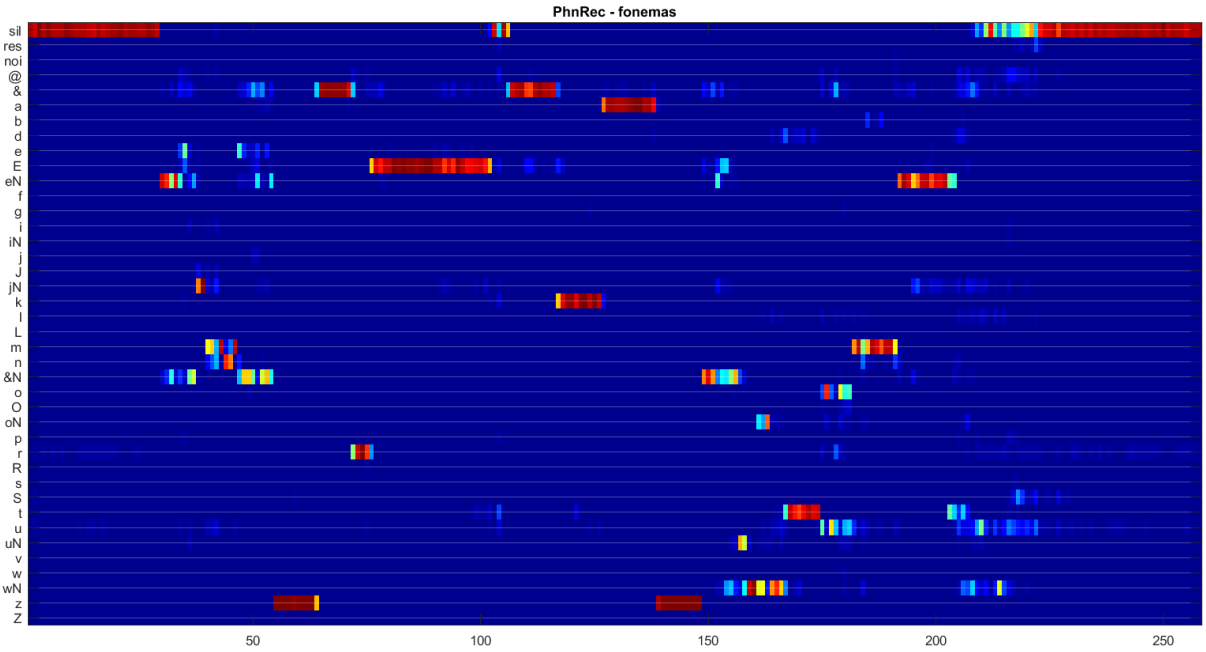

Figura 13. Posteriorgrama da saída de uma DNN, com 120 estados (3 estados para cada um dos 40 fonemas), em que as saídas foram concatenadas 3 a 3.

A transcrição da frase é : “Em Nazaré, a casa é um tormento.”, que traduzido para fonemas fica: “sil &N jN n & z & r E sil & k a z & E uN t u r m eN t u sil”

Capítulo 3 - Testes e Experiências Iniciais

3.1 CNTK

CNTK [10] é o acrónimo de *Computational Network Toolkit* relativo a uma *framework* desenvolvida pela Microsoft que permite a implementação, treino e teste de redes neuronais para aplicações no reconhecimento de fala, imagem, ou outro tipo de dados definidos pelo utilizador. Apesar de se encontrar ainda em desenvolvimento, foi escolhida como ferramenta para a criação das redes neuronais definidas nesta tese devido à sua flexibilidade e à possibilidade de se treinarem as redes usando clusters de placas gráficas, o que reduz o tempo de treino significativamente quando comparado com o tempo numa CPU com 4 núcleos (*cores*). O cluster de placas gráficas usado (denominado *windows9*) foi disponibilizado pelo centro de pesquisa CUDA da Nvidia [24] localizado no laboratório de processamento de imagem do I.T. Os detalhes da arquitectura do cluster de GPUs podem ser consultados no apêndice 4, no final deste documento. Segue-se uma comparação dos tempos de treino por época das máquinas utilizadas:

Máquina	Tempo de treino (s/época)
Windows9 (cluster de placas gráficas, S.O. Linux)	4.4
IT-Buntu (processador dual-core, S.O. Linux)	544
Hades (processador quad-core,, S.O. Linux)	124
Santiago (processador quad-core, S.O. Linux)	235
Portátil pessoal (processador quad-core, S.O. Windows)	1690

Tabela 1. Comparação dos tempos de treino de uma época usando a mesma DNN, para as diferentes máquinas usadas

O CNTK é fornecido em formato *open-source*, tanto para *Windows*, como para *Linux*. O código-fonte foi compilado nas máquinas Linux, embora sejam disponibilizadas versões pré-compiladas. Em máquinas Windows (computador pessoal) foi descarregada e compilada a solução *Visual Studio*, por forma a se poder analisar o código fonte (escrito em C++), pois a informação no manual de utilizador muitas vezes era insuficiente, e nos fóruns de discussão também não havia informação relevante.

3.2 Base de Dados

A base de dados utilizada é o resultado da junção de três bases de dados distintas (6 horas de fala no total):

-Tecnovoz: Esta base de dados foi criada para servir de apoio ao projecto homónimo, e é constituída por 78 locutores femininos (36.62%) e por 135 locutores masculinos (63.38%), tendo uma duração de 2.9 horas de fala na sua totalidade.

-Controlo: Contém locuções de comandos falados usados para controlo de fala. Tem 3 locutores femininos (48.86%) e 4 locutores masculinos (57.14%), perfazendo 1.9 horas de fala.

- Telejornal: Locuções extraídas de telejornais. Estes ficheiros de áudio apresentam uma gama dinâmica mais reduzida que a restante base de dados (sofrem de algum grau de 'clipping'). É composta por 237 locutores femininos (71,818%) e 93 locutores masculinos (28.182%), com uma duração total de 1.2 horas de fala.

Análise de Performance

O método de medição da performance da rede é feito através dos parâmetros conhecidos como *accuracy* e *correctness* (ambos em percentagens), provenientes do descodificador de Viterbi:

$$\%correctness = \frac{N-D-S}{N} \times 100 \quad (13)$$

$$\%accuracy = \frac{N-D-S-I}{N} \times 100 \quad (14)$$

N é o total de fonemas presentes no conjunto de locuções etiquetadas de referência (nº de etiquetas no conjunto de teste), D é o número de deleções ocorridas (fonemas não identificados), S o número de substituições (fonemas identificados incorrectamente), e I o número de inserções (fonemas inseridos a mais).

Nesta tese usa-se como medida de desempenho da rede as taxas de erro, e não de acerto:

$$\%correctness\_error = \text{CorrE} = 100 - \%correctness \quad (15)$$

$$\%accuracy\_error = \text{AccE} = 100 - \%accuracy \quad (16)$$

Outra métrica menos comum usada em alguns pontos desta tese é a percentagem de acerto absoluta por trama, proveniente da saída da rede.

$$\%argmax\_error = \text{ArgE} = \ (1 - \frac{H}{N_s}) \times 100 \quad (17)$$

onde *H* corresponde ao número de tramas previstas coincidentes com a referência e N_s ao número total de tramas. *H* é obtido tomando o valor máximo do vector de saída da DNN e comparando com a referência correspondente. ?

3.3 MFCCs

Uma das maneiras mais comuns de fornecer dados de entrada à rede é através de coeficientes MFCC – *MEL Frequency Cepstral Coeficients*, [5]. Estes coeficientes são obtidos partindo do sinal original, e aplicando-lhe uma janela de Hamming de comprimento 25 ms que avança a cada 10 ms, produzindo várias tramas. De seguida aplica-se a transformada de Fourier, e passa-se o sinal resultante por um banco de filtros passa-banda com resposta em frequência triangular e com frequências centrais que são equidistantes entre si na escala MEL. Esta é uma escala não-linear que está adaptada à percepção não linear do sistema auditivo humano às diferentes alturas melódicas (somos mais sensíveis variações de frequência quando estas são baixas do que quando são altas; por exemplo, conseguimos notar mais se um instrumento musical está desafinado se este operar em baixas frequências). Depois de obtidas as potências dos sinais vindos do banco de filtros MEL, é-lhes aplicado o logaritmo, pois o ouvido humano percepciona a 'loudness' (volume do som, em termos simples) nessa mesma escala logarítmica. A seguir, tomam-se esses conjuntos de logaritmos e aplicasse-lhes a transformada cosseno discreta (DCT). A DCT decompõe um dado vetor de dados em termos de uma soma de cossenos oscilando a diferentes frequências [6]:

$$c_k = \sqrt[2]{\frac{2}{N}} \sum_{j=1}^{N} \log(m_j) \times \cos(k \frac{\pi}{2N} * (2j - 1)) \quad , \quad k = 1,\ldots,N\text{-}1 \quad (18)$$

onde c_k são os coeficientes MFCC, *N* é o número de filtros do banco de filtros MEL, e m_j são as energias provenientes desse mesmo banco de filtros.

Os MFCCs (c_k) são as amplitudes do espectro resultante. Também se costuma adicionar a esse vector de coeficientes o logaritmo da energia (de cada frame), e os parâmetros delta e delta-delta, que são calculados através da regressão linear dos valores c_k já obtidos.

3.4 Testes com uma camada

Por forma a analisar o funcionamento do CNTK, efectuaram-se no início testes em que a rede neuronal tem apenas uma camada escondida e toma como parâmetros de entrada coeficientes MFCC com contexto temporal de 15 tramas (7 tramas passadas, 7 tramas futuras, e 1 trama central). O parâmetro a variar é a dimensão da camada escondida. Efectuaram-se testes para 500, 1000, 1500, 2000, 2500, 3000, 5000 e 6000 nodos na camada escondida, sempre com o mesmo número de épocas, tamanho do *mini-batch*, e *learning rate*.As redes testadas possuem 195 entradas e 120 saídas. Os resultados mostram que o erro (*argmax* da saída da DNN) de teste varia pouco com a arquitectura da rede, se bem que o melhor resultado seja o da rede com 2000 nodos escondidos (ver tabela 2).

Nodos Escondidos	ArgE (treino)	ArgE (teste)
500	42%	44%
1000	39%	43%
2000	39%	42%
2500	39%	43%
3000	38%	42%
5000	38%	42%
6000	38%	43%

Tabela 2. Comparação de erros de teste e treino para os mesmos parâmetros de treino, variando o nº de nodos escondidos

3.5 Teste com duas camadas

Como exercício académico e a título de curiosidade, implementou-se também uma rede com 2 camadas escondidas, tendo como base o melhor resultado obtido para apenas uma camada. Usaram-se MFCCs na entrada, com contexto temporal de 15 tramas, análogomante aos testes descritos na secção 3.4. A rede foi treinada com 200 épocas para cada camada. O tamanho do *mini-batch* e do *learning rate* foi variando ao longo do treino (por forma a se analisar o efeito que estas variações têm no desempenho da rede) da seguinte forma:

-*Mbsize* = 1024 para as primeiras 130 épocas, e 256 para as restantes

-*learning rate* = 0.8 durante as primeiras 20 épocas, 0.3 para as 130 seguintes (até à 150), e 0.15 para as restantes (até à 200)

O influência no erro de treino foi evidente e interessante, conforme se pode verificar na Figura 14.

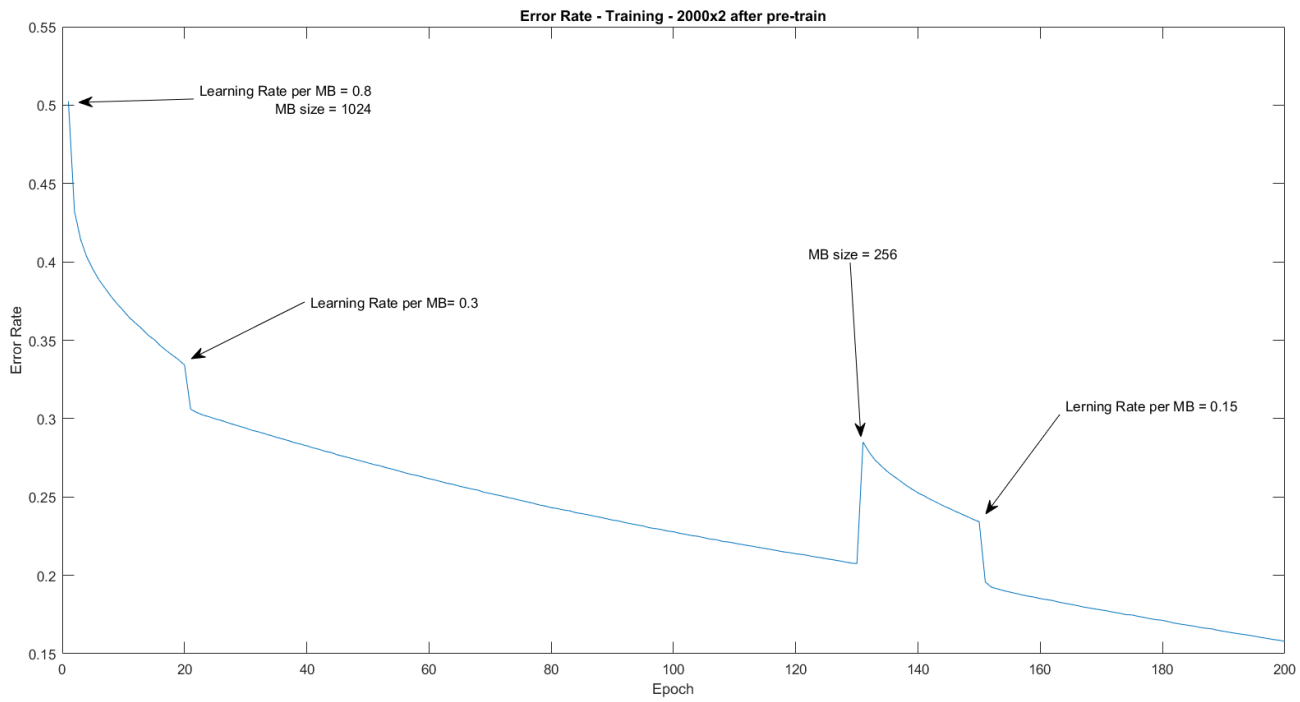

Figura 14. Traçado do ArgE para o conjunto de treino

As saídas da rede foram fornecidas a um descoddificador de Viterbi (free-phone loop), obtendo um CorrE = 37.50% e AccE = 32.76%.

Capítulo 4 - Arquitectura com contexto temporal

4.1 Três contextos usando MFCCs

Por forma a se ter um CorrE e AccE de referência, reproduziu-se o treino feito numa tese anterior [13], usando a mesma base de dados, e lista de ficheiros de treino e teste, no software da Universidade de Brno (PhnRec). Obtiveram-se resultados equivalentes aos obtidos na tese referida em [13], com um CorrE = 33.32% e um AccE = 30.08%. É de notar que esta rede usa 39 fonemas (o fonema *noi* não foi usado), e não 40 como as redes implementadas nesta tese, pelo que a taxa de erro é mais baixa quando comparada com a de 40 fonemas. No entanto, grande parte do desenvolvimento inicial deu-se usando 40 fonemas, pelo que as experiências posteriores também foram feitas usando 40 fonemas. Em todo o caso, é possível adaptar qualquer rede desenvolvida por forma a usar 39 fonemas de forma relativamente fácil.

Como primeira aproximação ao problema optou-se por se implementar uma rede com estrutura semelhante à da rede PhnRec, adicionando um contexto central, e utilizando parâmetros *MFCC* como entrada. O objectivo é avaliar o quão próximo se consegue chegar em termos de sucesso no reconhecimento de fonemas relativamente à arquitectura com TRAPs e contexto usada pelo reconhecedor da Universidade de Brno.

Os MFCCs são fornecidos à rede e divididos no seu contexto respectivo: 15 tramas passadas com trama presente para o contexto à esquerda, 15 tramas futuras com trama presente para o contexto à direita (31 tramas de contexto). Testou-se também 8 tramas passadas e futuras com a trama presente para o contexto central (17 tramas de contexto). Cada rede de contexto é treinada para reconhecer 120 estados de fonemas. Seguidamente, as saídas destas três redes são concatenadas num único vector que é fornecido como entrada de uma rede final de fusão, que também é treinada para reconhecer 120 estados. O esquema de processamento está indicado na Figura 15.

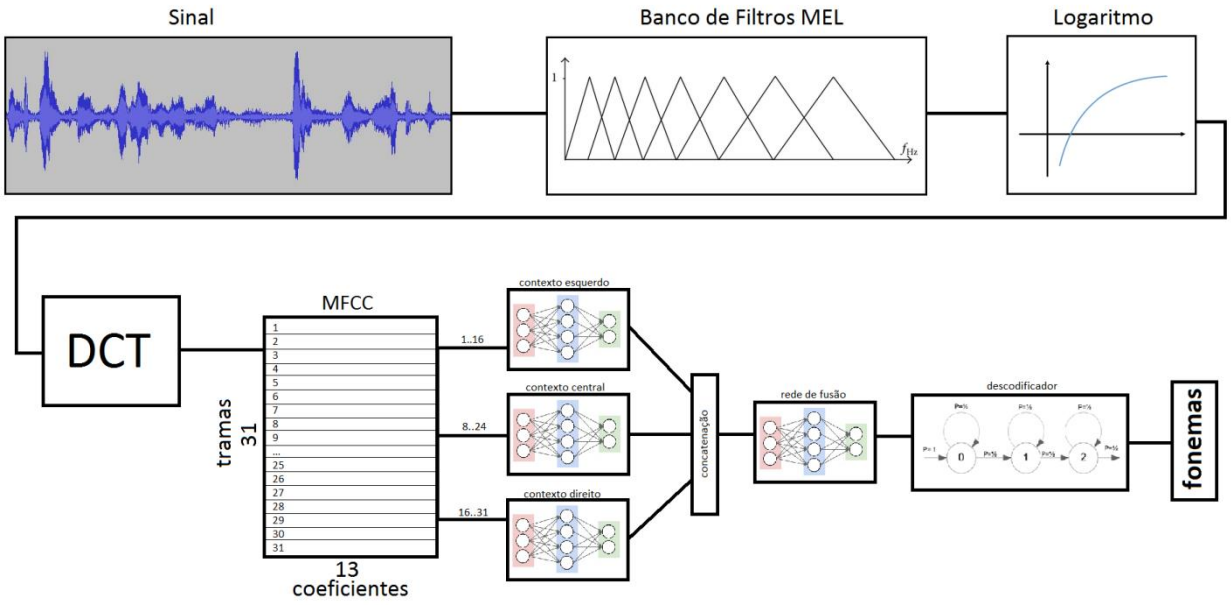

Figura 15. Arquitectura do reconhecedor com contexto esquerdo, central e direito que faz uso de MFCCs

O treino das 4 redes (de contexto passado, central e futuro, e a rede de fusão) é feito em separado Fizeram-se testes com variação do *mini-batch size*, *learning rate*, e *momentum*. Caso o valor do erro por amostra ($\%absolute\_error$, equação ()) não melhorasse, não se efectuavam teste no descodificador de Viterbi para essa rede, pelo facto de os testes no descodificador demorarem muito mais tempo em comparação com o treino a rede.

O melhor resultado obtido no descodificador de Viterbi foi para um *mini-batch size* = 128 *learning rate* = 0.1 e *momentum* = 0.1, com CorrE = 35.05% e AccE = 32.99%.

O melhor resultado relativo para o ArgE obtém-se para um *mini-batch size* = 256 *learning rate* = 0.1 e *momentum* = 0.1, com ArgE = 28.56%.

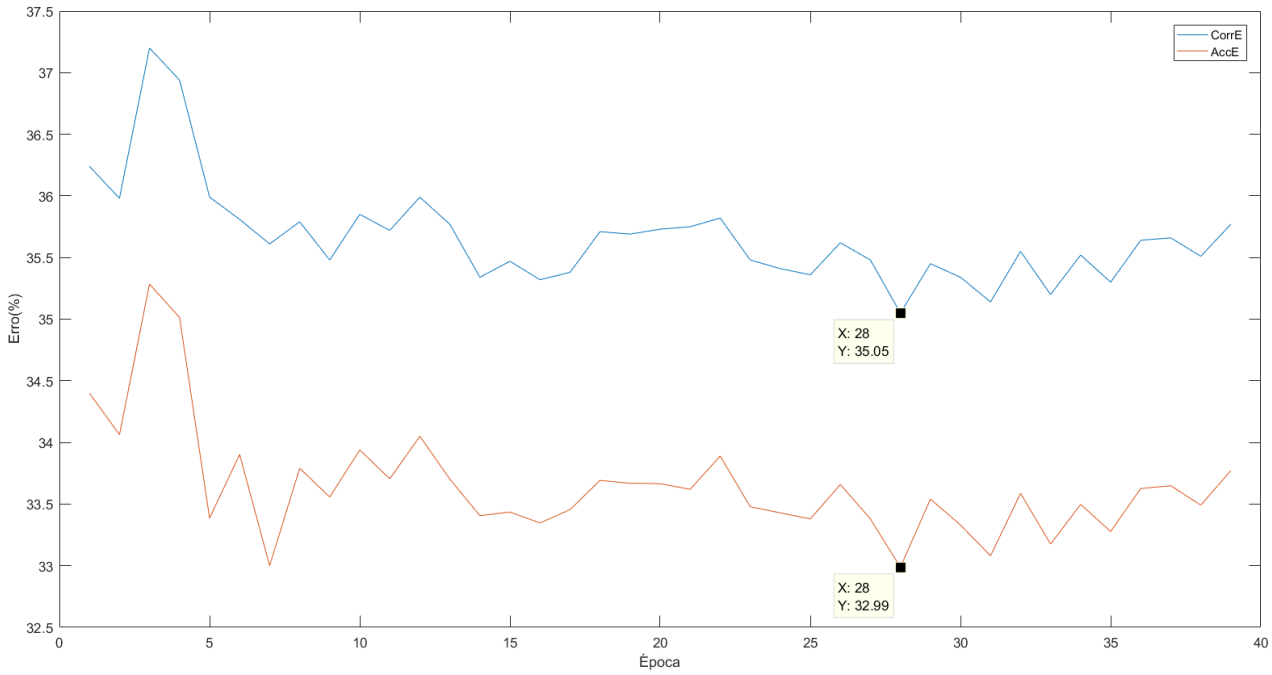

Figura 16. Desempenho da melhor rede obtida com contexto central usando MFCCs

Mini-batch size	*Learning Rate*	*Momentum*	CorrE	*ArgE*
256	0.5	0.9	37.51%	28.62%
256	0.1	0.1	35.93%	28.56%
128	0.1	0.1	35.05%	28.74%
64	0.8	0.0	-	32.53%
128	0.9	0.1	-	31.43%
30000	0.9	0.1	-	33.78%
200	0.8	0.1	-	32.05%
256-128	0.1	0.1	35.41%	28.71%

Tabela 3. Resultados dos testes feitos para a rede com contexto central usando MFCCs

Os testes feitos (ver tabela 2) apontam que um tamanho de *mini-batch* reduzido, bem como um *lerning rate* e *momentum* reduzidos beneficiam o desempenho da rede, pelo que foram tomados estes valores como ponto de partida para os treinos feitos posteriormente.

4.2 TRAPs

O sistema de análise espectral do sinal de fala, designada por TRAP foi desenvolvido pela universidade de Brno e toma como entradas os parâmetros vindos de um banco de filtros MEL aos quais se aplica o logaritmo. A aquisição prévia do sinal é idêntica à dos parâmetros MFCC: ao sinal original aplica-se uma janela de Hamming de comprimento 25 ms com avanço de 10 ms, e aplica-se a Tranformada de Fourier. O sinal resultante é fornecido ao banco de filtros em escala MEL. A diferença reside no facto de se tomar cada banda crítica do banco de filtros e amostrar-se temporalmente aplicando novamente uma janela de Hamming para cada banda crítica. Este vector representa a evolução da energia para cada banda, dando maior ênfase à amostra actual. Cada vector de cada banda crítica passa posteriormente por um classificador (rede neuronal), e as saídas de cada classificador são concatenadas num novo vector maior que por sua vez é fornecido a uma rede neuronal final:

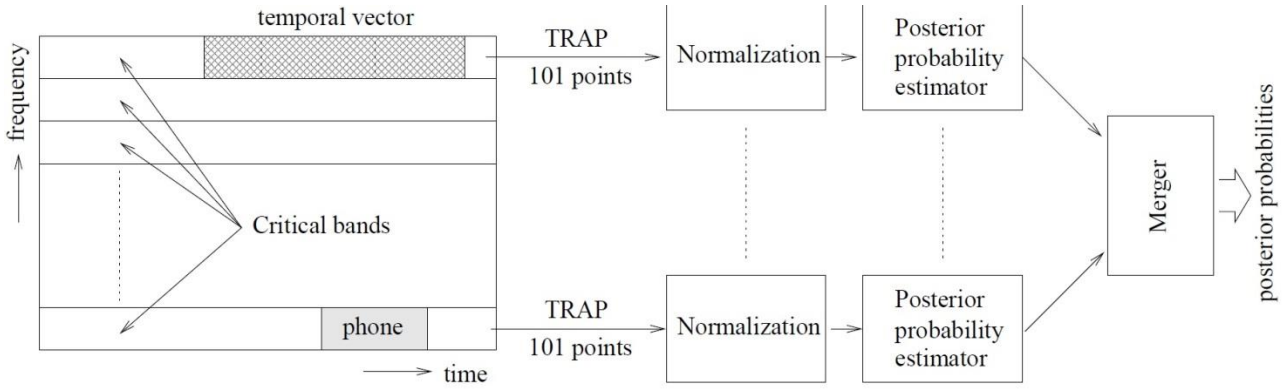

Figura 17. Esquema da aquisição dos contextos temporais e classificadores por banda do sistema TRAP original. Editado de [8].

O sistema de reconhecimento de fonemas desenvolvido pela Universidade de Brno [7] (PhnRec) é formado por uma arquitectura híbrida entre Hidden Markov Models (HMM) e ANNs. Tem como base o sistema TRAP, em que se usam vectores que descrevem a evolução temporal da densidade espectral das várias bandas críticas, provenientes do banco de filtros MEL. Depois de obtidos os logaritmos das energias provenientes dos 15 bancos de filtros MEL, normalizam-se as 15 bandas (subtração da média log-espetral) usando a média ao longo da locução para cada banda. A normalização da média e variância permite a remoção de factores que não sejam relevantes para o processamento dos sinais de fala, tal como a resposta em frequência dos

microfones e variação do meio de comunicação. Obtém-se assim uma matriz de dimensões 15 × nºframes dessa locução. De seguida são extraídos vectores temporais longos com duração de 310*ms*, que correspondem a 31×15 valores que caracterizam a evolução das energias das bandas críticas ao longo do tempo vizinho à trama presente. Cada vector de cada banda é multiplicado por uma janela de Hamming, ao longo das tramas, dando mais ênfase à trama presente. O contexto temporal é então dividio em duas partes: contexto esquerdo (15 tramas passadas mais a presente) e contexto direito (15 tramas futuras mais a presente):

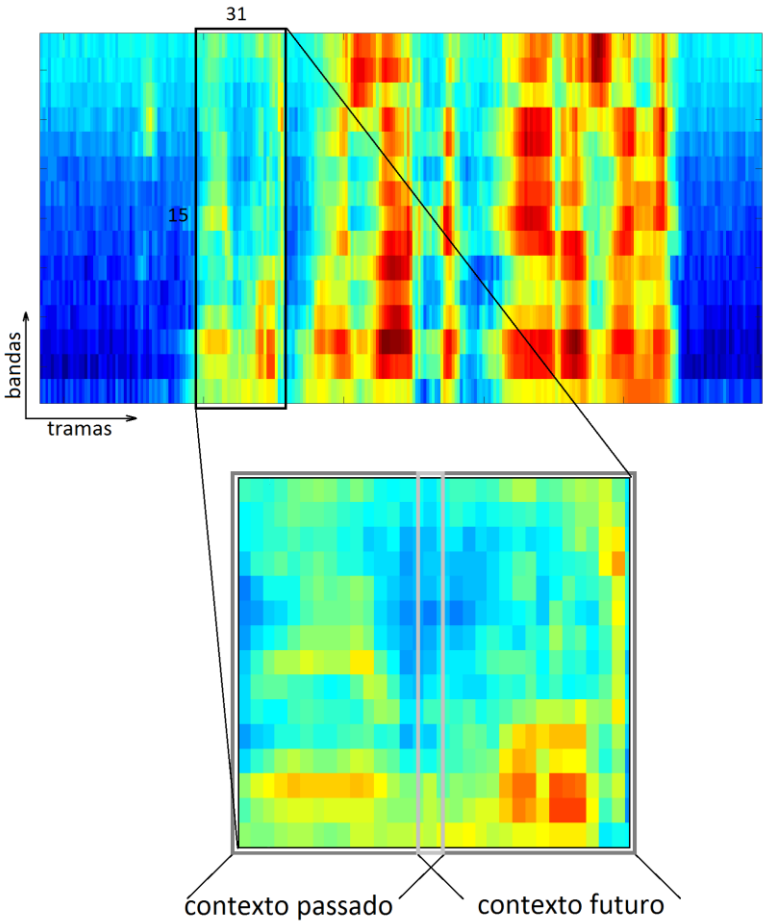

Figura 18. Aquisição dos contextos esquerdo e direito da saída de um banco de filtros MEL.

A cada um desses dois contextos é aplicada a DCT por forma a reduzir o contexto de 240 coeficientes (15 bandas críticas × 16 tramas temporais) para 165 coeficientes (15 bandas críticas × 11 coeficientes DCT), e aplicasse-lhes a normalização da média e variância ao longo de toda a base de dados de treino. Cada um dos dois contextos passa por uma ANN com apenas uma camada escondida de 1500 nodos. As saídas dessas duas redes são concatenadas, normaliza-se a média e a variância, e o vector resultante passa por uma rede final de fusão também apenas com

uma camada escondida com 1500 nodos. Por fim, a saída da rede de fusão é fornecida a um classificador de Viterbi, por forma a obter a sequência de fonemas.

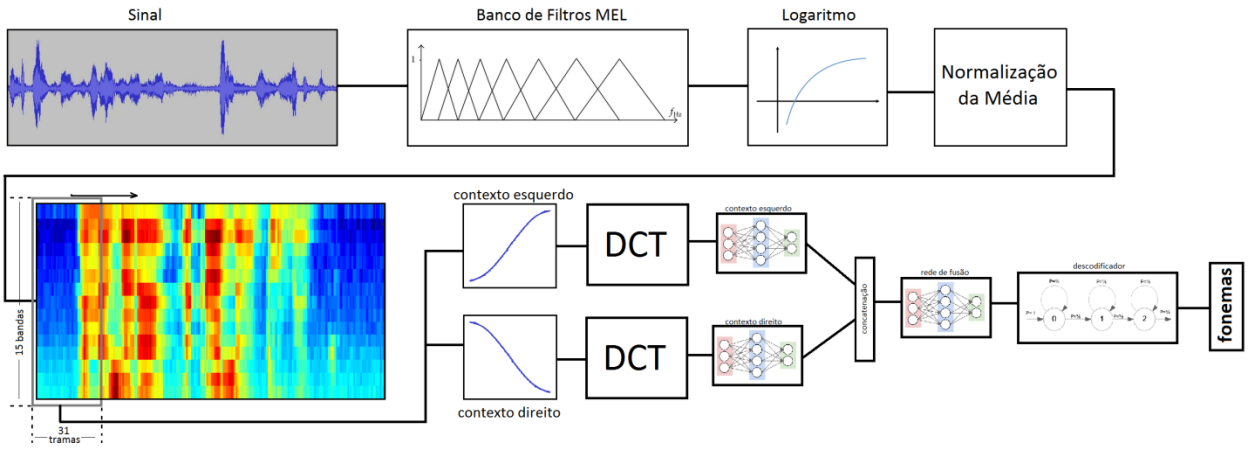

Figura 19. Arquitectura do reconhecedor com contexto esquerdo e direito baseado no sistema TRAP.

A primeira fase consiste em implementar a mesma arquitectura do PhnRec usando as ferramentas do CNTK. Para tal, fundiu-se o fonema *noi* com o fonema *res* para obter um ficheiro MLF de referência com 39 fonemas. Neste caso, deixa de haver 120 estados, e passam a existir 117. O vector de entrada da rede de fusão passa a conter 234 valores, correspondendo à concatenação dos 2×117 valores da saída de cada rede de contexto (passado e futuro).
Criaram-se os *scripts* necessários para a definição da arquitectura da rede, e para o treino e teste da mesma.
O treino é feito em três etapas separadas: treino do contexto esquerdo (passado), treino do contexto direito (futuro), e treino da rede de fusão. Optou-se por se definir a rede completa e congelar/descongelar as camadas necessárias consoante a etapa do treino (consultar apêndice 2.1, 2.2 e 2.3). Por exemplo, para treinar o contexto passado congela-se o contexto futuro e a rede de fusão, para treinar o contexto futuro congelam-se as duas outras redes, e assim sucessivamente.
Alternativas a este procedimento que foram ponderadas mas não implementadas (por se considerarem menos práticas) são:

- Reconstruir a rede à medida que as etapas do treino são terminadas. A rede começa por conter apenas o contexto passado. Terminado o treino do mesmo, adiciona-se o contexto futuro. No fim de se treinarem os dois contextos, adiciona-se à rede o contexto de fusão. Este método é mais complicado de implementar, e não tem benefícios relativamente à implementação escolhida.

- Criar três redes separadas e guardar os valores de saída das duas primeiras. Esta implementação envolve propagar os dados de treino pela rede esquerda e direita, e criar uma base de dados que contenha as saídas concatenadas das duas redes. Depois propagam-se esses dados pela rede de fusão por forma a obter os dados de saída. Embora a definição de cada rede individual seja mais simples, o facto de se ter que guardar esses dados extra considerou-se desvantajoso demais para se implementar este método de treino, já que o método seguido até ao momento era similar ao escolhido.

Os resultados obtidos foram satisfatórios, mostrando mesmo melhorias (CorrE) relativamente aos resultados obtidos para referência (CorrE = 33.32% e AccE = 30.08%, secção 4.1), com CorrE = 33.23% e AccE = 30.52% na época 36.

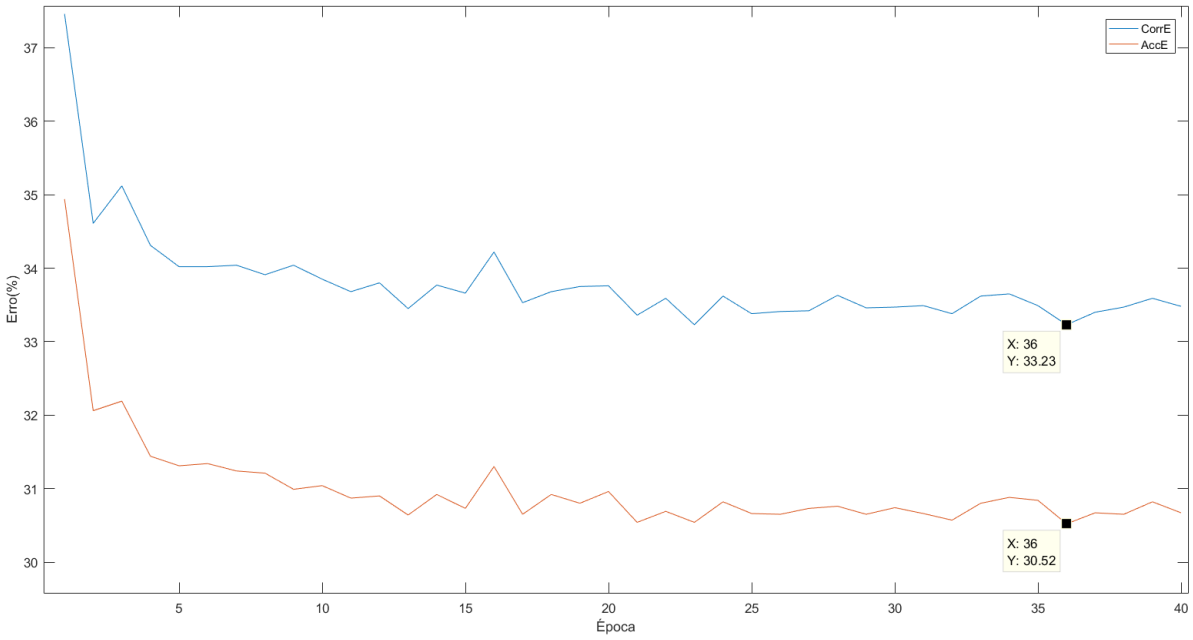

Figura 20. Desempenho da rede com 39 fonemas, usando TRAPs com contecto direito e esquerdo

O treino com 39 fonemas assegurou que o CNTK consegue reproduzir os resultados do PhnRec com fidelidade, pelo que se passou posteriormente à implementação da mesma rede utilizando 40 fonemas.

A rede (40 fonemas) foi treinada usando um ajuste automático do *learning rate* e do *mini-batch size*, com 100 épocas para cada classificador e para a rede de fusão. Os melhores resultados obtidos para o CorrE e AccE foram:

CorrE = 34.33% , AccE = 32.11% , época 23

CorrE = 34.53% , AccE = 32.05% , época 13

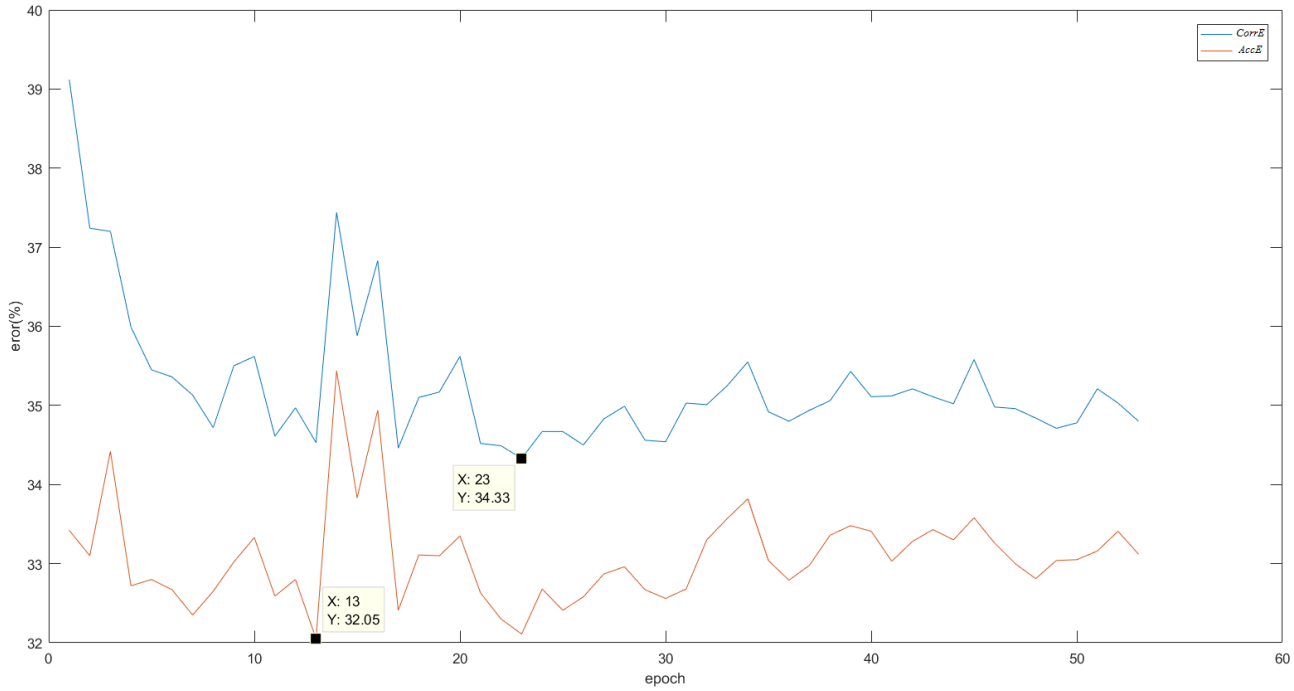

Figura 21. Evolução do CorrE e AccE durante o teste para a rede com 40 fonemas e uso de TRAPs com contexto esquerdo e direito descrita na secção 4.2

Tendo em conta os resulados obtidos na figura 20, conclui-se que o uso de 40 fonemas em vez de 39 reduz o desempenho do sistema, como foi mencionado no início da secção 4.1.

Capítulo 5 - Melhorias

5.1 Adição e remoção de ruído

Para a rede com duas camadas escondidas descrita no capítulo 2 expandiu-se a base de dados corrompendo os ficheiros áudio com ruído colorido, partindo de ruído branco e aplicando-lhe um filtro passa-baixo por forma a que a sua resposta em frequência deixe de ser uniforme.O ruído resultante é conhecido como colorido, em analogia à radiação electromagnética, que é branca quando contém todo o espectro visível, e possui cor quando assim não o é.

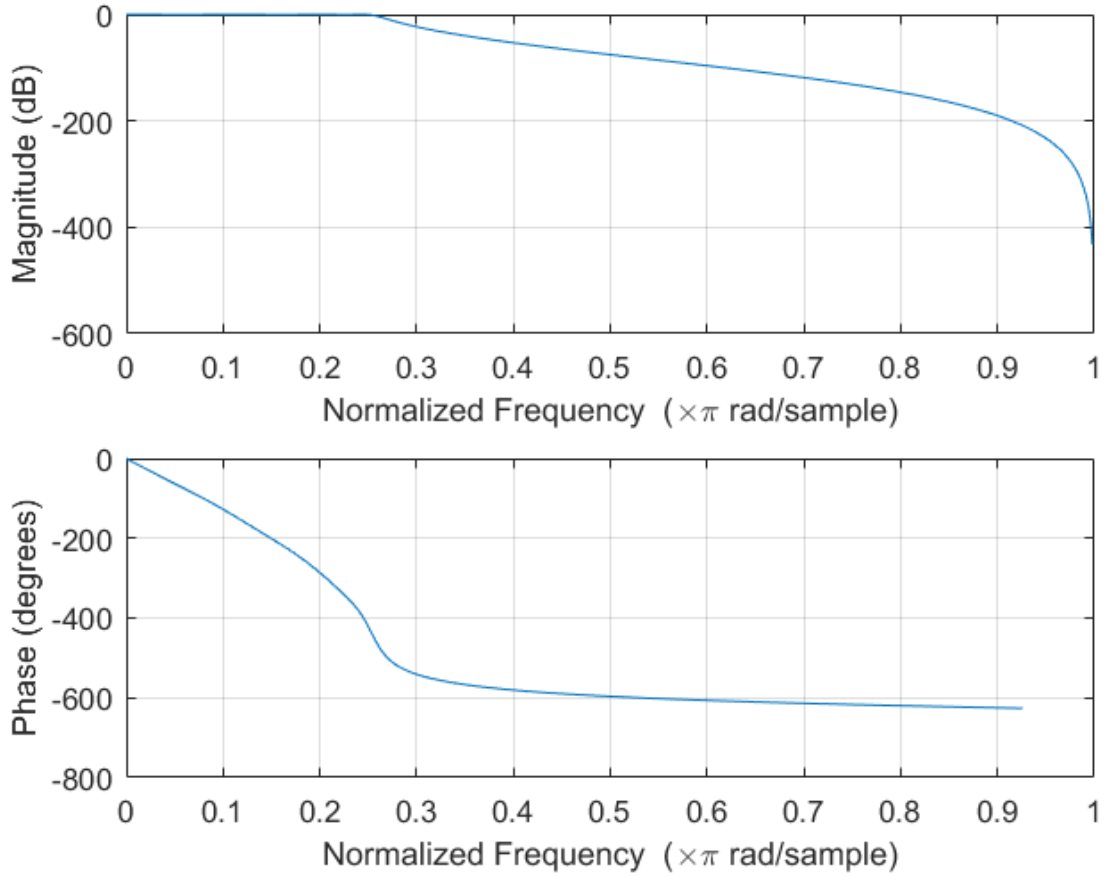

Figura 22. Filtro Chebyshev passa-baixo (módulo e fase) usado para colorir o ruído

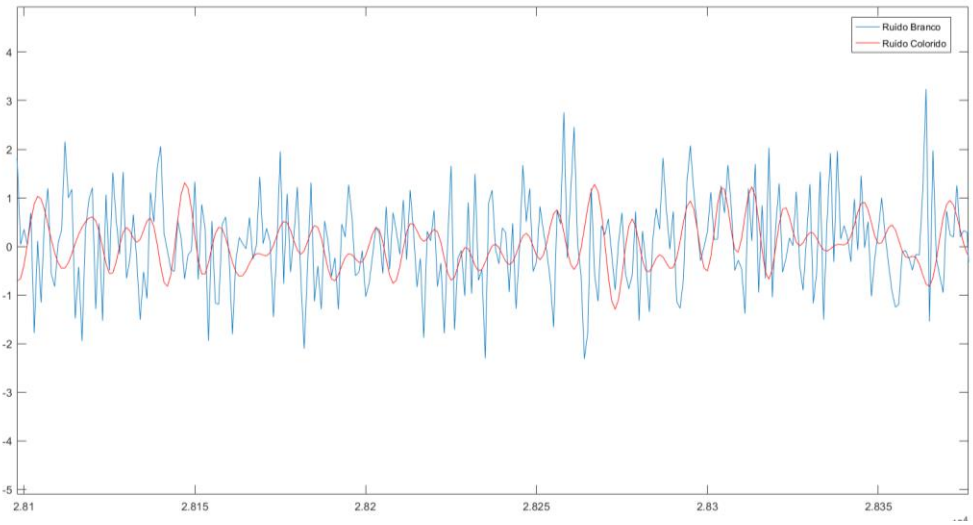

Figura 23. Ruído branco (a azul), e ruído colorido (a vermelho), onde é evidente a ausência das componentes de alta frequência no ruído colorido

Outro método de expansão da base de dados foi através da subtracção espectral, em que o ruído original presente no sinal é estimado e subtraído no domínio espectral por forma a obter um sinal sem ruído [12]. Este método foi implementado e descrito em [13], com mais detalhe, pelo que aqui é apresentada uma versão resumida do processo.

O primeiro passo consiste na remoção das componentes do sinal que se situem abaixo de 150 Hz, removendo assim o ruído proveniente da rede eléctrica e outras componentes de baixa frequência captados pelo microfone. Foi utilizado um filtro *Butterworth* devido à ausência de *ripple* na região da banda passante, e valor praticamente nulo na banda de rejeição. De seguida procedeu-se ao cálculo da energia através do processamento de termo curto do sinal, descrito na secção 2.3, em que se tomaram DFTs do sinal com janelas de Hamming de comprimento 25 ms e avanço 10 ms, aplicando o logaritmo da energia das tramas no final. Empiricamente considerou-se que abaixo do limiar *l* seria ruído:

$$l = m_1 + 0.2 * (m_2 - m_1) \tag{19}$$

em que m_1e m_2 são as medianas dos valores abaixo do 1º quartil e acima do 3º quartil do logaritmo energia das tramas.

Obtidas as tramas que são consideradas ruído, obtém-se a potência média do ruído P_N. O passo seguinte consiste na filtragem de Wiener com o filtro *H*,

$$H = \frac{P_S}{P_S + P_N} \tag{20}$$

fazendo uso da potência do sinal P_S que não foi considerado como sendo ruído.
Esta filtragem permite a obtenção de uma versão menos ruidosa do espectro de potência do sinal estimado.
Por fim, procede-se à reconstrução do sinal, adicionando a fase do sinal original ao módulo e sintetizando o sinal usando o método de sobreposição e soma (*overlap-add*)[18].

Estes dois métodos de expansão da base de dados triplicam os dados de treino e teste, sem ser necessária a criação de novos ficheiros MLF, pois não há deslocação dos fones no domínio do tempo, não sendo necessário portanto uma nova etiquetagem para cada locução.

Testou-se ainda, além do *free phone loop*, a descodificação num bigrama [19] (em que são obtidas as probabilidade de sequências dos fonemas aos pares). Foi obtido um resultado para o unigrama (*free phone loop*) de CorrE = 35.65% e AccE = 31.02%. Com a bigrama estimada a partie da base de dados de treini, obteve-se um resultado , CorrE = 32.95% e AccE = 29.56%
Em teoria, treinar a rede usando locuções com e sem ruído faz com que ela aprenda a filtrar o ruído, melhorando a sua performance. Os resultados demonstram essa hipótese, pois o erro diminui em 1.85% para o CorrE e 1.74% para o AccE (valores absolutos).

5.2 Melhoria do sistema TRAP

À rede implementada no capítulo 3, em que se usam TRAPs com contexto temporal, adicionou-se um contexto central, em que se extraem 8 tramas passadas e futuras em conjunto com a central aquando do cálculo dos parâmetros MFCC. A rede passa a ter uma configuração semelhante à configuração descrita na secção 3.1, com a diferença de as features serem fornecida sob a forma de TRAPs, e não MFCCs puros.

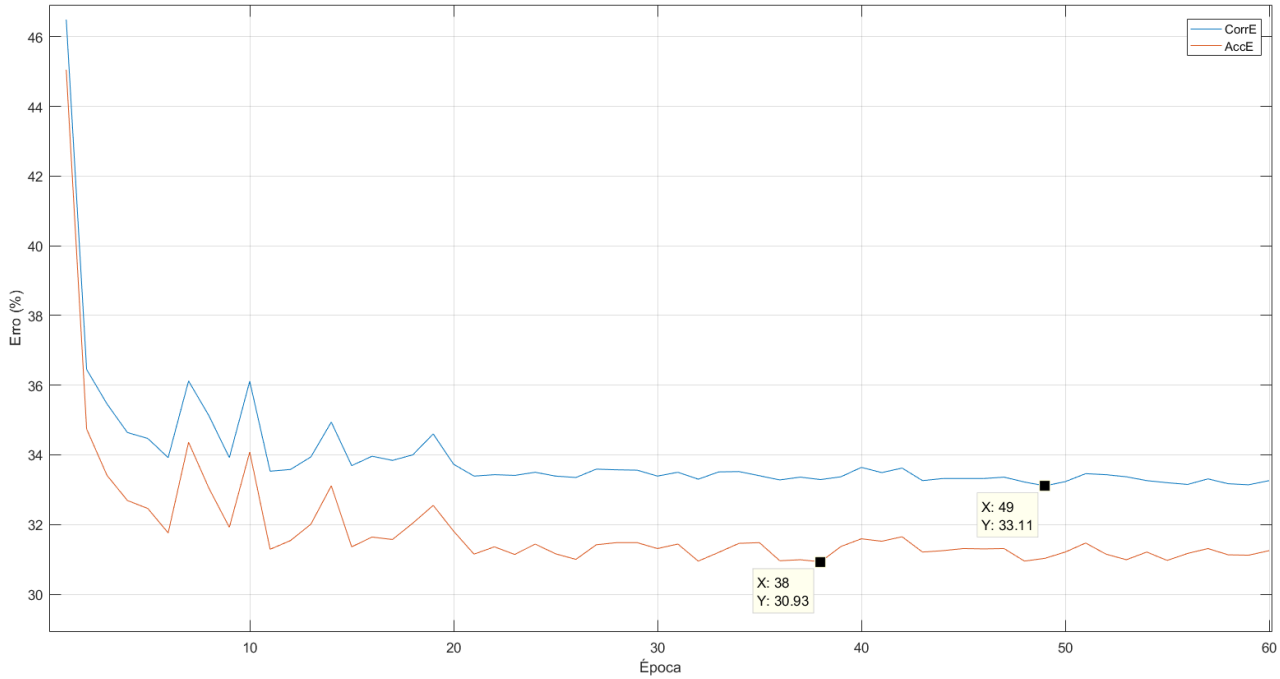

Figura 24. Evolução do CorrE e AccE ao longo das épocas de treino para o reconhecedor com TRAPs, adição de contexto central, e descodificação com *free*-phone *loop*

O melhor CorrE = 33.11% foi obtido na época 49, e AccE = 30.93% na época 38.

Testou-se ainda a rede com um bigrama, obtendo um CorrE = 32.99% (época 52) e um AccE = 27.33% (época 58).

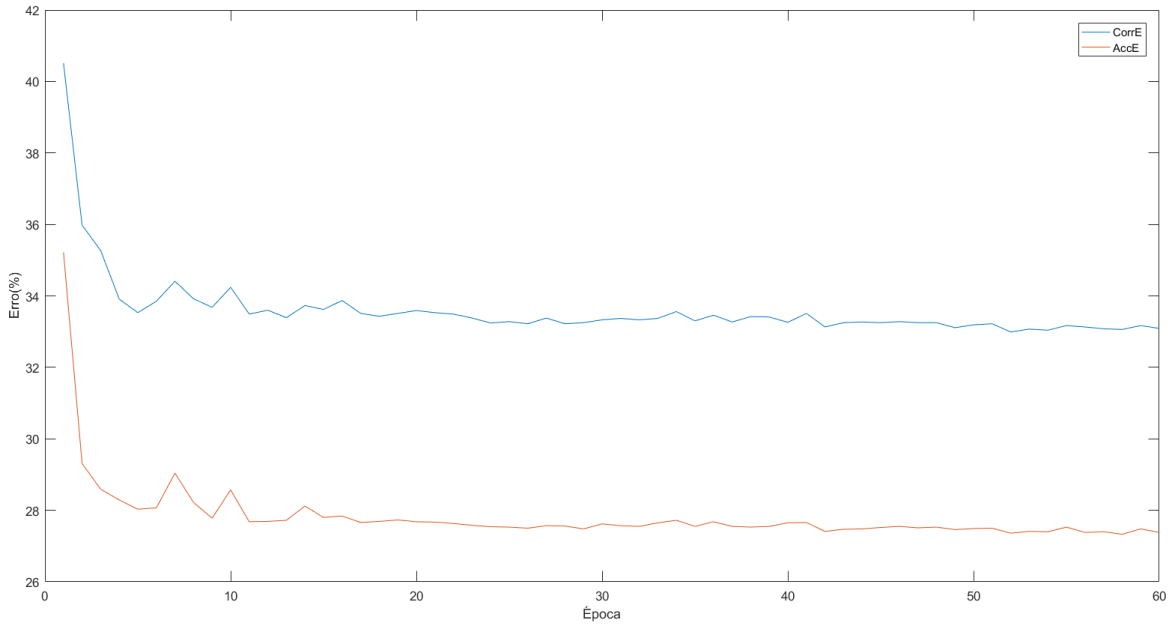

Figura 25. Evolução dor CorrE e AccE ao longo das épocas de treino para o reconhecedor com TRAPs, adição de contexto central, e descodificação com *bigrama*.

Treinou-se ainda a mesma rede, sem o contexto central, com a base de dados expandida artificialmente através da adição e remoção de ruído, analogamente à secção 5.1. Obtiveram-se também melhorias relativmente com CorrE = 33.85% (época 50) e ArgE = 32.29% (época 59).

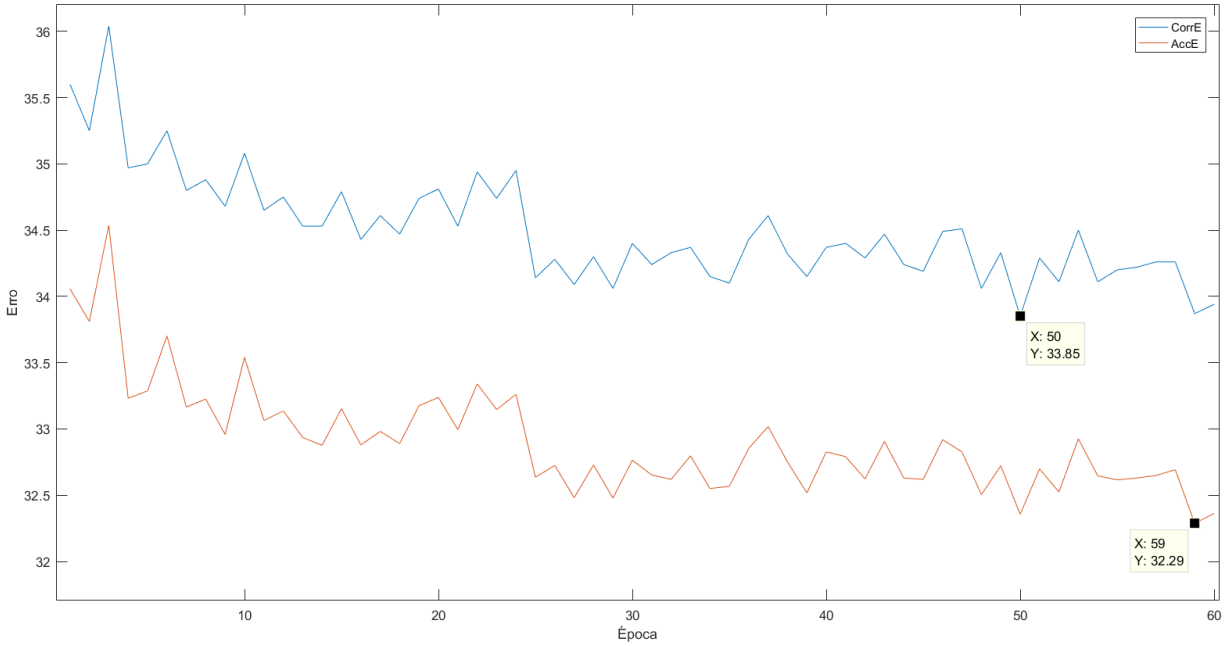

Figura 26. Evolução do CorrE e AccE ao longo das épocas de treino para o reconhecedor com TRAPs, expansão artificial da base de dados, e descodificação com *free*-phone *loop*

Capítulo 6 - Conclusão

O objectivo principal deste trabalho é a implementação de um sistema híbrido DNN/HMM capaz de reconhecer fonemas, e foi alcançado através das várias arquitecturas descritas.
Outra meta importante era a obtenção de uma rede com uma taxa de performance equiparável à obtida pela rede PhnRec treinada previamente com o software Universidade de Brno, que foi atingida com sucesso.
A aplicação de técnicas que melhoram o desmpenho dos reconhecedores também trouxe resultados positivos.
Futuramente, a expansão da base de dados pode ser uma mais-valia no desenvolvimento de um reconhecedor ainda melhor, e a implementação de redes neuronais recorrentes seria uma hipótese interessante a investigar pela mesma razão.
Resumidamente, foi um trabalho que fomentou grandemente o interesse em redes neuronais, e possibilitou ainda mais o aumento do interesse pessoal não só no reconhecimento mas também no processamento de fala e áudio em geral.

Referências:

[1] https://www.quora.com/Intuitively-how-does-mini-batch-size-affect-the-performance-of-stochastic-gradient-descent

[2] - arXiv:1606.04838

[3] - *Rumelhart, David E.; Hinton, Geoffrey E.; Williams, Ronald J. (8 October 1986). "Learning representations by back-propagating errors". Nature.* ***323*** *(6088): 533–536. doi:10.1038/323533a0.*

[4] - https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

[5] - http://recognize-speech.com/feature-extraction/mfcc

[6] - https://en.wikipedia.org/wiki/Discrete_cosine_transform

[7] - http://speech.fit.vutbr.cz/software/phoneme-recognizer-based-long-temporal-context

[8] - Petr Schwarz, Jan Černoký, "Phoneme Recognition Based On Long Term Context", Ph.D. Thesis, Brno University of Technology, 2008

[9] - http://neuralnetworksanddeeplearning.com/chap3.html

[10] - https://github.com/Microsoft/CNTK/wiki

[11] – Young, Steve; Evermann, Gunnar; Gales, Mark; Hain, Thomas; Kershaw, Dan; Liu, Xunying; Moore, Gareth; Odell, Julian; Ollason, Dave; Povey, Dan; Ragni, Anton; Valtchev, Valtcho; Woodland, Phil; Zhang, Chao; "The HTK Book" (December 2015), pág 231

[12] - http://dsp-book.narod.ru/304.pdf

[13] - Luis Castela, Fernando Perdigão,"'Pesquisa de Fala", Universidade de Coimbra, 2015

[14] - http://www.cs.cmu.edu/~roni/11761/Presentations/hmm-for-asr-whw.pdf

[15] - https://www.ll.mit.edu/publications/journal/pdf/vol03_no1/3.1.3.speechrecognition.pdf

[16] - https://en.wikipedia.org/wiki/Viterbi_algorithm#Algorithm

[17] - http://speech.fit.vutbr.cz/software/phoneme-recognizer-based-long-temporal-context

[18] - https://en.wikipedia.org/wiki/Overlap%E2%80%93add_method

[19] - https://en.wikipedia.org/wiki/N-gram#Examples

[20] - https://github.com/cdipaolo/goml/tree/master/perceptron

[21] - http://www.extremetech.com/extreme/215170-artificial-neural-networks-are-changing-the-world-what-are-they

[22] - http://dictionary.cambridge.org/pt/dicionario/ingles/batch

[23] - https://pt.coursera.org/learn/machine-learning/lecture/9zJUs/mini-batch-gradient-descent

[24] - https://developer.nvidia.com/academia/centers/university-coimbra

Apêndice 1: Lista de fonemas usados

FONEMA	TIPO	EXEMPLO	TRANSCRIÇÃO FONÉTICA	CLASSE
@	vogal fechada	pretende	p r @ t eN d @	vogal
&	vogal fechada	lagoa	l & g o &	vogal
a	vogal aberta	autor	a w t o r	vogal
e	vogal fechada	evitar	e v i t a r	vogal
E	vogal aberta	esta	E S t &	vogal
eN	vogal nasalada	entrada	eN t r a d &	vogal
i	vogal aberta	radia	R & d i k &	vogal
iN	vogal nasalada	interior	iN t @ r i o r	vogal
j	semi vogal	noite	n o j t @	vogal
jN	vogal nasalada	em	&N jN	vogal
&N	vogal nasalada	grandes	g r & N d @ S	vogal
o	vogal fechada	outro	o t r u	vogal
O	vogal aberta	própria	p r O p r i &	vogal
oN	vogal nasalada	contos	k oN t u S	vogal
u	vogal aberta	uma	u m &	vogal
uN	vogal nasalada	um	uN	vogal
w	semi vogal	causa	k a w z &	vogal
wN	vogal nasalada	são	s &N wN	vogal
b	plosiva sonora	ibérica	i b E r i k&	consoante
d	plosiva sonora	proferidas	p r u f @ r i d & S	consoante
f	fricativa surda	semáforos	s @ m a f u r u S	consoante
g	plosiva sonora	algumas	a l g u m & S	consoante
J	nasal	espanha	@ S p a J &	consoante
k	plosiva surda	comitiva	k u m i t i v &	consoante
l	líquidas/laterais	plano	p l & n u	consoante
L	líquidas/laterais	trabalho	t r & b a L u	consoante
m	nasal	américa	& m E r i k &	consoante
n	nasal	centena	s eN t e n &	consoante
p	plosiva surda	poeira	p u & j r &	consoante
r	líquidas/laterais	praça	p r a s &	consoante
R	líquidas/laterais	regional	R @ Z i u n a l	consoante
s	fricativa surda	concelhia	k oN s @ L i &	consoante
S	fricativa surda	buracos	b u r a k u S	consoante
t	plosiva surda	forte	f O r t @	consoante
v	fricativa sonora	viária	v i a r i &	consoante
z	fricativa sonora	meses	m e z @ S	consoante
Z	fricativa sonora	laranjas	l & r &N Z & S	consoante
noi	ruído	ruído	noi	silêncio/ruído
sil	silêncio	silêncio	sil	silêncio/ruído
res	ruído	respiração	res	silêncio/ruído

Apêndice 2.1: Script de treino da rede : PhnRec_TrainTraps_39.cntk

```
###############################################################################
#                          Rede TRAPs com 39 fonemas                          #
###############################################################################

RootDir = ".."

ConfigDir = "$RootDir$/Config_TrapsV2"
DataDir = "$RootDir$/Data_Traps"
OutputDir = "$RootDir$/Output"
ModelDir  = "$OutputDir$/Models_Traps_39"
FilesDir = "$DataDir$/Files"

command = Init:SetPast:TrainPast:SetFuture:TrainFuture:SetMerger:TrainMerger

precision = "float"
traceLevel = 0
verbosity = 0
stderr= "/homelocal/gfranco/CNTK/PhnRec/Output/"

#######################################
#  TRAINING CONFIG                    #
#######################################

#========= DEFINIÇÕES GLOBAIS=========#

    SGD = [
        epochSize = 0
        minibatchSize = 512
        learningRatesPerMB = 0.9
        momentumPerMB = 0.1
        maxEpochs = 0

autoAdjust = {
        autoAdjustLR = "adjustAfterEpoch"
        autoAdjustMinibatch = true

        reduceLearnRateIfImproveLessThan = 0.0001
        learnRateDecreaseFactor = 0.618
        increaseLearnRateIfImproveMoreThan = 5
        learnRateIncreaseFactor = 1.382
        loadBestModel = true
        learnRateAdjustInterval = 1
        useCVSetControlLRIfCVExists = true
        useEvalCriterionControlLR = false

        numMiniBatch4LRSearch = 500
        minibatchSizeTuningFrequency = 2
        minibatchSizeTuningMax = 2048
        minibatchSearchCriterionErrorMargin = 1
      }
    ]

reader = [
```

```
        readerType = "HTKMLFReader"
             readMethod = "blockRandomize"
             miniBatchMode = "partial"
             randomize = "auto"
             features = [
                    dim = 330
                    scpFile = "$FilesDir$/Train_framemode.scp"
             ]
             labels = [
                    #mlfFile = "$FilesDir$/PT3BDs_ref_phonestate_3_expanded.mlf"
                    mlfFile = "$FilesDir$/PT3BDs_ref_phonestate_39.mlf"
                    labelDim = 117
                    labelMappingFile = "$FilesDir$/dict_phone_states_39.txt"
             ]
   ]
]

#=============================================================================
#=============================================================================
#=============================================================================
#Inicializa a rede

Init = [
    action = "train"
    modelPath = "$ModelDir$/Rede_PhnRec.dnn"

    BrainScriptNetworkBuilder = (new ComputationNetwork [
             include "$ConfigDir$/Rede_PhnRec_39.bs"
       ])

#=============================================================================

SetPast = [
       action = edit
       PhnRecModel = "$ModelDir$/Rede_PhnRec.dnn"
       PastModel = "$ModelDir$/Past/Past.dnn.0"
       editpath = "$ConfigDir$/Set_Past.mel"
]

#=============================================================================
TrainPast = [
action = "train"

modelPath = "$ModelDir$/Past/Past.dnn"

BrainScriptNetworkBuilder = (new ComputationNetwork [
             include "$ConfigDir$/Rede_PhnRec_39.bs"
       ])

    SGD = [

        epochSize = 0
        minibatchSize = 128
        learningRatesPerMB = 0.9
        momentumPerMB = 0.1
        maxEpochs = 100

autoAdjust = {
        autoAdjustLR = "adjustAfterEpoch"
        autoAdjustMinibatch = true
        reduceLearnRateIfImproveLessThan = 0.0001
```

```
        learnRateDecreaseFactor = 0.618
        increaseLearnRateIfImproveMoreThan = 5
        learnRateIncreaseFactor = 1.382
        loadBestModel = true
        learnRateAdjustInterval = 1
        useCVSetControlLRIfCVExists = true
        useEvalCriterionControlLR = false

        numMiniBatch4LRSearch = 500
        minibatchSizeTuningFrequency = 2
        minibatchSizeTuningMax = 2048
        minibatchSearchCriterionErrorMargin = 1
             }
    ]
    reader = [
        readerType = "HTKMLFReader"
             readMethod = "blockRandomize"
             miniBatchMode = "partial"
             randomize = "auto"
             features = [
                    dim = 330
                    scpFile = "$FilesDir$/Train_framemode.scp"
             ]
             labels = [
                    #mlfFile = "$FilesDir$/PT3BDs_ref_phonestate_3_expanded.mlf"
                    mlfFile = "$FilesDir$/PT3BDs_ref_phonestate_39.mlf"
                    labelDim = 117
                    labelMappingFile = "$FilesDir$/dict_phone_states_39.txt"
             ]
   ]
]

#====================================================================
SetFuture = [
       action = edit
       CentralModel = "$ModelDir$/Past/Past.dnn"
       FutureModel = "$ModelDir$/Future/Future.dnn.0"
       editpath = "$ConfigDir$/Set_Future.mel"
]

#====================================================================
TrainFuture = [
    action = "train"

modelPath = "$ModelDir$/Future/Future.dnn"
BrainScriptNetworkBuilder = (new ComputationNetwork [
             include "$ConfigDir$/Rede_PhnRec_39.bs"
       ])

    SGD = [
        epochSize = 0
        minibatchSize = 128
        learningRatesPerMB = 0.9
        momentumPerMB = 0.1
        maxEpochs = 100

autoAdjust = {
        autoAdjustLR = "adjustAfterEpoch"
        autoAdjustMinibatch = true

        reduceLearnRateIfImproveLessThan = 0.0001
```

```
        learnRateDecreaseFactor = 0.618
        increaseLearnRateIfImproveMoreThan = 5
        learnRateIncreaseFactor = 1.382
        loadBestModel = true
        learnRateAdjustInterval = 1
        useCVSetControlLRIfCVExists = true
        useEvalCriterionControlLR = false

        numMiniBatch4LRSearch = 500
        minibatchSizeTuningFrequency = 2
        minibatchSizeTuningMax = 2048
        minibatchSearchCriterionErrorMargin = 1
             }
    ]

    reader = [
        readerType = "HTKMLFReader"
             readMethod = "blockRandomize"
             miniBatchMode = "partial"
             randomize = "auto"
             features = [
                    dim = 330
                    scpFile = "$FilesDir$/Train_framemode.scp"
             ]
             labels = [
                    #mlfFile = "$FilesDir$/PT3BDs_ref_phonestate_3_expanded.mlf"
                    mlfFile = "$FilesDir$/PT3BDs_ref_phonestate_39.mlf"
                    labelDim = 117
                    labelMappingFile = "$FilesDir$/dict_phone_states_39.txt"
             ]
   ]
]

#=============================================================================

SetMerger = [
       action = edit
       FutureModel = "$ModelDir$/Future/Future.dnn"
       MergerModel = "$ModelDir$/Merger/Merger.dnn.0"
       editpath = "$ConfigDir$/Set_Merger.mel"
]

#=============================================================================
TrainMerger = [
    action = "train"
      modelPath = "$ModelDir$/Merger/Merger.dnn"
    BrainScriptNetworkBuilder = (new ComputationNetwork [
             include "$ConfigDir$/Rede_PhnRec_39.bs"
      ])
    SGD = [
       epochSize = 0
        minibatchSize = 128
        learningRatesPerMB = 0.9
        momentumPerMB = 0.1
        maxEpochs = 100
             autoAdjust = {
        autoAdjustLR = "adjustAfterEpoch"
        autoAdjustMinibatch = true

        # for autoAdjustLR = "adjustAfterEpoch":
        reduceLearnRateIfImproveLessThan = 0.0001
```

```
        learnRateDecreaseFactor = 0.618
        increaseLearnRateIfImproveMoreThan = 5
        learnRateIncreaseFactor = 1.382
        loadBestModel = true
        learnRateAdjustInterval = 1
        useCVSetControlLRIfCVExists = true
        useEvalCriterionControlLR = false

        # for autoAdjustMinibatch = true:
        numMiniBatch4LRSearch = 500
        minibatchSizeTuningFrequency = 2
        minibatchSizeTuningMax = 2048
        minibatchSearchCriterionErrorMargin = 1
             }
    ]

    reader = [
        readerType = "HTKMLFReader"
             readMethod = "blockRandomize"
             miniBatchMode = "partial"
             randomize = "auto"
             features = [
                   dim = 330
                   scpFile = "$FilesDir$/Train_framemode.scp"
             ]
             labels = [
                   #mlfFile = "$FilesDir$/PT3BDs_ref_phonestate_3_expanded.mlf"
                   mlfFile = "$FilesDir$/PT3BDs_ref_phonestate_39.mlf"
                   labelDim = 117
                   labelMappingFile = "$FilesDir$/dict_phone_states_39.txt"
             ]
   ]
]
```

Apêndice 2.2: Scripts de modificação de rede : Set_Past.mel, Set_Future.mel e Set_Merger.mel

```
#================================
#      CONTEXTO PASSADO
#================================
mP = LoadModel("$PhnRecModel$", format="cntk")
setDefaultModel(mP)

#===== Liberta Passado =====#

SetProperty(mP.HLast_P,output,true)
SetProperty(mP.CE_P,criterion,true)
SetProperty(mP.EvalErrorPrediction_P,evaluation,true)
Setproperty(mP.W0_P,computeGradient,true)
Setproperty(mP.B0_P,computeGradient,true)
Setproperty(mP.WLast_P,computeGradient,true)
Setproperty(mP.BLast_P,computeGradient,true)

#===== Bloqueia Central =====#

SetProperty(mP.HLast_C,output,false)
SetProperty(mP.CE_C,criterion,false)
SetProperty(mP.EvalErrorPrediction_C,evaluation,false)
Setproperty(mP.W0_C,computeGradient,false)
Setproperty(mP.B0_C,computeGradient,false)
Setproperty(mP.WLast_C,computeGradient,false)
Setproperty(mP.BLast_C,computeGradient,false)

#===== Bloqueia Futuro =====#

SetProperty(mP.HLast_F,output,false)
SetProperty(mP.CE_F,criterion,false)
SetProperty(mP.EvalErrorPrediction_F,evaluation,false)
Setproperty(mP.W0_F,computeGradient,false)
Setproperty(mP.B0_F,computeGradient,false)
Setproperty(mP.WLast_F,computeGradient,false)
Setproperty(mP.BLast_F,computeGradient,false)

#===== Bloqueia Merger =====#

SetProperty(mP.HLast_M,output,false)
SetProperty(mP.CE_M,criterion,false)
SetProperty(mP.EvalErrorPrediction_M,evaluation,false)
Setproperty(mP.W0_M,computeGradient,false)
Setproperty(mP.B0_M,computeGradient,false)
Setproperty(mP.WLast_M,computeGradient,false)
Setproperty(mP.BLast_M,computeGradient,false)

#===== Guarda o Modelo =====#

SaveModel(mP, "$PastModel$", format="cntk")



#================================
```

```
#      CONTEXTO FUTURO
#================================

mF = LoadModel("$CentralModel$", format="cntk")
setDefaultModel(mF)

#===== Bloqueia Passado =====#

SetProperty(mF.HLast_P,output,false)
SetProperty(mF.CE_P,criterion,false)
SetProperty(mF.EvalErrorPrediction_P,evaluation,false)
Setproperty(mF.W0_P,computeGradient,false)
Setproperty(mF.B0_P,computeGradient,false)
Setproperty(mF.WLast_P,computeGradient,false)
Setproperty(mF.BLast_P,computeGradient,false)

#===== Bloqueia Central =====#

SetProperty(mF.HLast_C,output,false)
SetProperty(mF.CE_C,criterion,false)
SetProperty(mF.EvalErrorPrediction_C,evaluation,false)
Setproperty(mF.W0_C,computeGradient,false)
Setproperty(mF.B0_C,computeGradient,false)
Setproperty(mF.WLast_C,computeGradient,false)
Setproperty(mF.BLast_C,computeGradient,false)

#===== Liberta Futuro =====#

SetProperty(mF.HLast_F,output,true)
SetProperty(mF.CE_F,criterion,true)
SetProperty(mF.EvalErrorPrediction_F,evaluation,true)
Setproperty(mF.W0_F,computeGradient,true)
Setproperty(mF.B0_F,computeGradient,true)
Setproperty(mF.WLast_F,computeGradient,true)
Setproperty(mF.BLast_F,computeGradient,true)

#===== Bloqueia Merger =====#

SetProperty(mF.HLast_M,output,false)
SetProperty(mF.CE_M,criterion,false)
SetProperty(mF.EvalErrorPrediction_M,evaluation,false)
Setproperty(mF.W0_M,computeGradient,false)
Setproperty(mF.B0_M,computeGradient,false)
Setproperty(mF.WLast_M,computeGradient,false)
Setproperty(mF.BLast_M,computeGradient,false)

#===== Guarda o Modelo =====#

SaveModel(mF, "$FutureModel$", format="cntk")


#================================
#      CONTEXTO DE FUSÃO
#================================

mM = LoadModel("$FutureModel$", format="cntk")
setDefaultModel(mM)

#===== Bloqueia Passado =====#

SetProperty(mM.HLast_P,output,false)
```

```
SetProperty(mM.CE_P,criterion,false)
SetProperty(mM.EvalErrorPrediction_P,evaluation,false)
Setproperty(mM.W0_P,computeGradient,false)
Setproperty(mM.B0_P,computeGradient,false)
Setproperty(mM.WLast_P,computeGradient,false)
Setproperty(mM.BLast_P,computeGradient,false)

#===== Bloqueia Central =====#

SetProperty(mM.HLast_C,output,false)
SetProperty(mM.CE_C,criterion,false)
SetProperty(mM.EvalErrorPrediction_C,evaluation,false)
Setproperty(mM.W0_C,computeGradient,false)
Setproperty(mM.B0_C,computeGradient,false)
Setproperty(mM.WLast_C,computeGradient,false)
Setproperty(mM.BLast_C,computeGradient,false)

#===== Bloqueia Futuro =====#

SetProperty(mM.HLast_F,output,false)
SetProperty(mM.CE_F,criterion,false)
SetProperty(mM.EvalErrorPrediction_F,evaluation,false)
Setproperty(mM.W0_F,computeGradient,false)
Setproperty(mM.B0_F,computeGradient,false)
Setproperty(mM.WLast_F,computeGradient,false)
Setproperty(mM.BLast_F,computeGradient,false)

#===== Liberta Merger =====#

SetProperty(mM.HLast_M,output,true)
SetProperty(mM.CE_M,criterion,true)
SetProperty(mM.EvalErrorPrediction_M,evaluation,true)
Setproperty(mM.W0_M,computeGradient,true)
Setproperty(mM.B0_M,computeGradient,true)
Setproperty(mM.WLast_M,computeGradient,true)
Setproperty(mM.BLast_M,computeGradient,true)

#===== Guarda o Modelo =====#

SaveModel(mM, "$MergerModel$", format="cntk")
```

Apêndice 2.3: Script de definição da treino da rede : Rede_PhnRec_39.bs

```
FeatDim = 330            # 11 coeficientes x 15bandas x 2 contextos
MergerDim = 234          #concatenação das saídas dos 2 contextos
LabelDim = 117           #39 fones * 3 estados
HiddenDim = 1500         #número de unidades da camada escondida

features = Input (FeatDim, tag='feature')
labels   = Input (LabelDim, tag='label')

#======== Contexto Passado ========#

Feat_P = Slice (0, 165, features, axis=1)

W0_P = Parameter(HiddenDim, 165)
B0_P = Parameter(HiddenDim, 1)

WLast_P = Parameter(LabelDim, HiddenDim)
BLast_P = Parameter(LabelDim, 1)

MeanOfFeatures_P = Mean (Feat_P)
InvStdOfFeatures_P = InvStdDev(Feat_P)
MVNormalizedFeatures_P = PerDimMeanVarNormalization(Feat_P, MeanOfFeatures_P,
InvStdOfFeatures_P)

Times0_P = Times(W0_P,MVNormalizedFeatures_P)
Plus0_P = Plus(Times0_P,B0_P)
H0_P = Sigmoid(Plus0_P)

TimesLast_P = Times(WLast_P,H0_P)
PlusLast_P = Plus(TimesLast_P,BLast_P)
HLast_P = Softmax(PlusLast_P,tag='output')

LogSoftmax_P = Log(HLast_P)
Prior_P = Mean(labels)
LogOfPrior_P = Log(Prior_P)
ScaledLogLikelihood_P = Minus(LogSoftmax_P, LogOfPrior_P)

CE_P = CrossEntropyWithSoftmax(labels, PlusLast_P, tag='criterion')
EvalErrorPrediction_P = ErrorPrediction(labels, PlusLast_P, tag='evaluation')

#======== Contexto Futuro ========#

Feat_F = Slice (165, 330, features, axis=1)

W0_F = Parameter(HiddenDim, 165)
B0_F = Parameter(HiddenDim, 1)

WLast_F = Parameter(LabelDim, HiddenDim)
BLast_F = Parameter(LabelDim, 1)

MeanOfFeatures_F = Mean (Feat_F)
InvStdOfFeatures_F = InvStdDev(Feat_F)
MVNormalizedFeatures_F = PerDimMeanVarNormalization(Feat_F, MeanOfFeatures_F,
InvStdOfFeatures_F)

Times0_F = Times(W0_F,MVNormalizedFeatures_F)
```

```
Plus0_F = Plus(Times0_F,B0_F)
H0_F = Sigmoid(Plus0_F)

TimesLast_F = Times(WLast_F,H0_F)
PlusLast_F = Plus(TimesLast_F,BLast_F)
HLast_F = Softmax(PlusLast_F)

LogSoftmax_F = Log(HLast_F)
Prior_F = Mean(labels)
LogOfPrior_F = Log(Prior_F)
ScaledLogLikelihood_F = Minus(LogSoftmax_F, LogOfPrior_F)

CE_F = CrossEntropyWithSoftmax(labels, PlusLast_F)
EvalErrorPrediction_F = ErrorPrediction(labels, PlusLast_F)

#============ Merger ============#

Input_M = RowStack(HLast_P:HLast_F)

W0_M = Parameter(HiddenDim, MergerDim)
B0_M = Parameter(HiddenDim, 1)

WLast_M = Parameter(LabelDim, HiddenDim)
BLast_M = Parameter(LabelDim, 1)

MeanOfFeatures_M = Mean (Input_M)
InvStdOfFeatures_M = InvStdDev(Input_M)
MVNormalizedFeatures_M = PerDimMeanVarNormalization(Input_M, MeanOfFeatures_M,
InvStdOfFeatures_M)

#Times0_M = Times(W0_M,MVNormalizedFeatures_M)
Times0_M = Times(W0_M,Input_M)
Plus0_M = Plus(Times0_M,B0_M)
H0_M = Sigmoid(Plus0_M)

TimesLast_M = Times(WLast_M,H0_M)
PlusLast_M = Plus(TimesLast_M,BLast_M)
HLast_M = Softmax(PlusLast_M)

LogSoftmax_M = Log(HLast_M)
Prior_M = Mean(labels)
LogOfPrior_M = Log(Prior_M)
ScaledLogLikelihood_M = Minus(LogSoftmax_M, LogOfPrior_M)

CE_M = CrossEntropyWithSoftmax(labels, PlusLast_M)
EvalErrorPrediction_M = ErrorPrediction(labels, PlusLast_M)
```

Apêndice 3: Exemplo de parte de um ficheiro MLF

```
"*/F00_1029p399c1.lab"
0       1700000     sil
1700000     2700000     sil
2700000     3400000     u
3400000     4000000     p
4000000     4800000     r
4800000     5100000     u
5100000     6000000     Z
6000000     7100000     E
7100000     7800000     t
7800000     8100000     u
8100000     8400000     @
8400000     9000000     S
9000000     9700000     t
9700000     10700000    a
10700000    11300000    &
11300000    12100000    v
12100000    13100000    &
13100000    13600000    l
13600000    14700000    i
14700000    16100000    a
16100000    16700000    d
16700000    17000000    u
17000000    17400000    &N
17400000    18300000    jN
18300000    19400000    s
19400000    20200000    e
20200000    20700000    r
20700000    21600000    k
21600000    22000000    &
22000000    22500000    d
22500000    22800000    @
22800000    23400000    uN
23400000    24300000    m
24300000    24600000    i
24600000    25200000    L
25200000    25800000    &N
25800000    26400000    wN
26400000    26900000    d
26900000    27500000    @
27500000    28600000    k
28600000    30200000    oN
30200000    31300000    t
31300000    31600000    u
31600000    33400000    S
33400000    33700000    res
33700000    36300000    sil
36300000    37400000    i
37400000    38100000    p
38100000    38700000    r
```

Apêndice 4: Características das GPUs utilizadas

Device 0: "GeForce GTX TITAN X"

```
  CUDA Driver Version / Runtime Version          7.5 / 7.5
  CUDA Capability Major/Minor version number:    5.2
  Total amount of global memory:                 12287 MBytes (12884180992 bytes)
  (24) Multiprocessors, (128) CUDA Cores/MP:     3072 CUDA Cores
  GPU Max Clock rate:                            1076 MHz (1.08 GHz)
  Memory Clock rate:                             3505 Mhz
  Memory Bus Width:                              384-bit
  L2 Cache Size:                                 3145728 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(65536), 2D=(65536, 65536), 3D=(4096,
4096, 4096)
  Maximum Layered 1D Texture Size, (num) layers  1D=(16384), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(16384, 16384), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 2 copy engine(s)
  Run time limit on kernels:                     Yes
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Disabled
  Device supports Unified Addressing (UVA):      Yes
  Device PCI Domain ID / Bus ID / location ID:   0 / 3 / 0
  Compute Mode:
    < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

==========================================================================
```

```
Device 1: "Tesla K40c"

  CUDA Driver Version / Runtime Version          7.5 / 7.5
  CUDA Capability Major/Minor version number:    3.5
  Total amount of global memory:                 12288 MBytes (12884705280 bytes)
  (15) Multiprocessors, (192) CUDA Cores/MP:     2880 CUDA Cores
  GPU Max Clock rate:                            745 MHz (0.75 GHz)
  Memory Clock rate:                             3004 Mhz
  Memory Bus Width:                              384-bit
  L2 Cache Size:                                 1572864 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(65536), 2D=(65536, 65536), 3D=(4096,
4096, 4096)
  Maximum Layered 1D Texture Size, (num) layers  1D=(16384), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(16384, 16384), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 2 copy engine(s)
  Run time limit on kernels:                     No
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Disabled
  Device supports Unified Addressing (UVA):      Yes
  Device PCI Domain ID / Bus ID / location ID:   0 / 4 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >
> Peer access from GeForce GTX TITAN X (GPU0) -> Tesla K40c (GPU1) : No
> Peer access from Tesla K40c (GPU1) -> GeForce GTX TITAN X (GPU0) : No

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 7.5, CUDA Runtime Version
= 7.5, NumDevs = 2, Device0 = GeForce GTX TITAN X, Device1 = Tesla K40c
Result = PASS


================================================================================
```

Device 2: "GeForce GTX TITAN"

  CUDA Driver Version / Runtime Version   8.0 / 7.5
  CUDA Capability Major/Minor version number: 3.5
  Total amount of global memory: 6082 MBytes (6377046016 bytes)
  (14) Multiprocessors, (192) CUDA Cores/MP: 2688 CUDA Cores
  GPU Max Clock rate: 876 MHz (0.88 GHz)
  Memory Clock rate: 3004 Mhz
  Memory Bus Width: 384-bit
  L2 Cache Size: 1572864 bytes
  Maximum Texture Dimension Size (x,y,z)   1D=(65536), 2D=(65536, 65536), 3D=(4096, 4096, 4096)
  Maximum Layered 1D Texture Size, (num) layers 1D=(16384), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers 2D=(16384, 16384), 2048 layers
  Total amount of constant memory: 65536 bytes
  Total amount of shared memory per block: 49152 bytes
  Total number of registers available per block: 65536
  Warp size: 32
  Maximum number of threads per multiprocessor: 2048
  Maximum number of threads per block: 1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size   (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch: 2147483647 bytes
  Texture alignment: 512 bytes
  Concurrent copy and kernel execution: Yes with 1 copy engine(s)
  Run time limit on kernels: Yes
  Integrated GPU sharing Host Memory: No
  Support host page-locked memory mapping: Yes
  Alignment requirement for Surfaces: Yes
  Device has ECC support: Disabled
  Device supports Unified Addressing (UVA):   Yes
  Device PCI Domain ID / Bus ID / location ID: 0 / 3 / 0
  Compute Mode:
    < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 8.0, CUDA Runtime Version = 7.5, NumDevs = 1, Device0 = GeForce GTX TITAN
Result = PASS