UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Campus de São José do Rio Preto

André Furlan

Autenticação Biométrica de Locutores Drasticamente Disfônicos Aprimorada pela *Imagined Speech*

São José do Rio Preto
2022

André Furlan

Autenticação Biométrica de Locutores Drasticamente Disfônicos Aprimorada pela *Imagined Speech*

Tese apresentada como parte dos requisitos para obtenção do título de Doutor em Ciência da Computação, junto ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista 'Júlio de Mesquita Filho", Campus de São José do Rio Preto.

Orientador: Prof. Dr. Rodrigo Capobianco Guido

São José do Rio Preto

2022

André Furlan

Autenticação Biométrica de Locutores Drasticamente Disfônicos Aprimorada pela *Imagined Speech*

Tese apresentada como parte dos requisitos para obtenção do título de Doutor em Ciência da Computação, junto ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista 'Júlio de Mesquita Filho", Campus de São José do Rio Preto.

Comissão Examinadora

Prof. Dr. Rodrigo Capobianco Guido
UNESP – Câmpus de São José do Rio Preto
Orientador

Prof. Dr. Exemplo Jr
Universidade – Câmpus

Prof. Dr. Exempl2
Universidade – Câmpus

São José do Rio Preto

06 de Agosto de 2022

Agradecimentos

"Citação"

-O Autor.

Resumo

Abstract

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

1.1 Considerações Iniciais e Objetivos

1.2 Estrutura do trabalho

2 Revisão bibliográfica

2.1 Breve revisão dos conceitos utilizados neste trabalho

2.1.1 Redes neurais convolucionais

Um dos classificadores usados nesse trabalho é baseado na tecnologia de redes neural as convolucionais, Das redes tem como principal característica a convolução aplicada aos dados de entrada segundo um segundo um "kernel", Nesse caso entende-se "kernel"Como uma série de valores que deve ser aplicados valor a valor Afim de preparar os dados para classificação de uma rede neural densa, no caso específico das redes neurais convolucionais Essa série de valores que serviram como filtro da entrada da rede neural convolucional, Na verdade, É apenas a aplicação da função de ativação em cada valor, Ou seja, se a função de ativação escolhida for a sigmóide, Aplicar-se-á Esta função a cada valor de entrada da rede, caso a função de ativação seja a Relu O mesmo será feito (PALIWAL,).

A aplicação do *kernel* também pode ser feita da seguinte forma: soma se os valores vezes seus respectivos pesos e aplica-se a função de ativação.

Um aspecto importante da preparação dos dados para treinamento chama-se *padding* Que consiste no aumento do vetor de entrada para que o vetor de saída tenha sempre o mesmo tamanho que o vetor de entrada. Em redes convolucionais O resultado terá sempre um número menor de itens do que a entrada, assim, é preciso adicionar zeros no início ou fim do vetor de entrada ou apenas no início (recomendado no caso de realização de previsões).

O *stride* ou passo é outro fator que influencia no tamanho da saída de uma convolução. O Passo é a quantidade de deslocamentos que uma janela de convolução deve realizar quando necessita avançar no vetor fornecido, quanto maior o passo menos valores serão fornecidos como resultado aumentando assim a necessidade de *padding* caso se necessite manter o tamanho do vetor de entrada para a próxima camada.Dependendo da fase da convolução, esse encurtamento necessário e desejável pois pode revelar padrões cujos os ruídos impediam de serem vistos.

TODO: Ver *wavenets no notebook*

(AMIDI; AMIDI,)

2.1.2 Treinamento e particionamento da série de dados

Quando do treinamento de uma rede necessitas-se Que os dados usados para tal fim sejam particionados, Para tal, é necessário separar os dados em uma partição de treinamento,Outra de validação, e finalmente uma de teste.

2.1.3 Funções de ativação

As funções de ativação geralmente são aquelas cujo resultado varia entre -1 e 1 ou entre 0 e 1 caso sejam saturáveis como as *hiperbólicas* e *sigmoides* ou entre em zero e infinito como é o caso da *Relu* caso sejam não saturáveis.

2.1.4 Funções de erro

As funções de erro são usadas para computar a diferença entre os valores produzidos pelo modelo e os valores, de fato, desejados, dependendo dos resultados desejados se pode usar uma ou outra função de erro, na listagem abaixo estão descritas algumas funções de erro e seus usos recomendados.

- $erros = previsao - desejado$ → : Erro simples, não destaca nem suprime nenhuma característica.
- $mse = \frac{1}{n} * \sum_{n=1}^{tamanho} erros[n]^2$ → Média dos quadrados dos erros: Usada quando é necessário destacar os de maior valor. Então se o custo de erros grandes é importante para aplicação esta função é recomendada.
- $mae = \frac{1}{n} * \sum_{n=1}^{tamanho} |erros[n]|$ → Média dos valores absolutos: Usada quando os resultados são proporcionais ao erro.
- $mape = \frac{1}{n} * \sum_{n=1}^{tamanho} |erros[n]/desejado[n]|$ → Média da percentagem de erro absoluto: Informa o tamanho do erro comparado com o valor desejado.

2.1.5 Técnicas de treinamento de redes

2.1.5.1 Parada antecipada de treinamento

A técnica de parada antecipada de treinamento pode ser usada quando a rede neural por algum tempo pára de convergir, dessa forma é possível evitar o gasto desnecessário de processamento quando a rede pára de convergir. Esta técnica também ajuda a rede a evitar um *overffiting*.

2.1.6 Redes neurais residuais

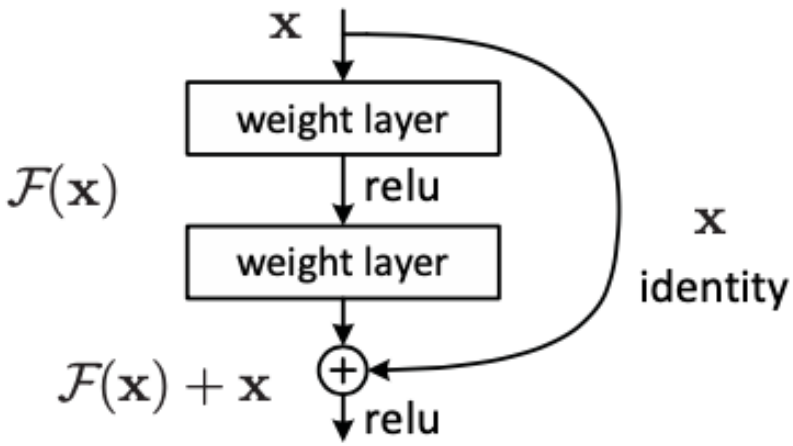


Figura 1 – Residual learning: a building block.

De acordo com (HE et al., 2015) Redes neurais residuais são aquelas que "pulam"algumas camadas, ou seja, a saída de uma camada vai para a próxima mas também vai para uma outra mais à frente.

2.1.7 Sinais digitais e sub-amostragem (*downsampling*)

2.1.8 Caracterização dos processos de produção da voz humana

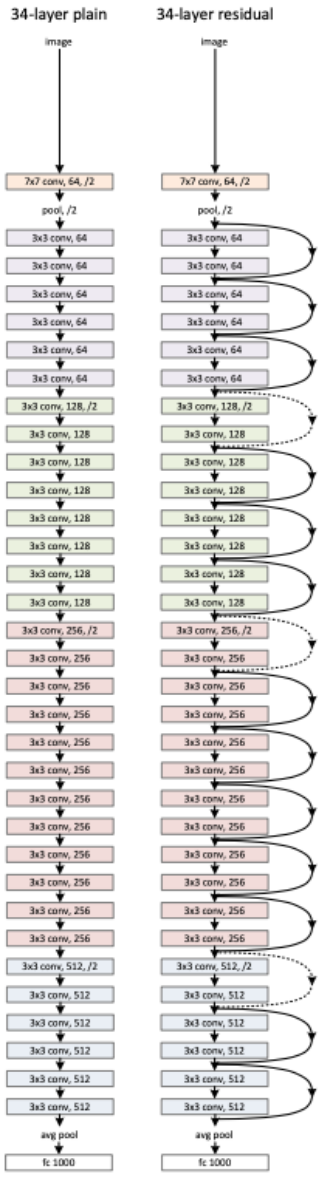


Figura 2 – Comparação entre uma rede neural regular e uma rede neural residual, a direita a rede neural regular percorre sequencialmente todas as suas camadas, a esquerda a rede neural residual "pula"algumas camadas reiteradamente.

2.2 Estado-da-arte em *Imagined Speech*

3 Abordagem proposta

3.1 A Base de sinais

3.1.1 Coleta dos sinais

3.1.2 Organização da base de sinais

3.2 Estrutura da estratégia proposta

3.3 Procedimentos

3.3.1 Procedimento 01

4 Testes e Resultados

4.1 Procedimento 01

5 Conclusões e Trabalhos Futuros

Referências

AMIDI, A.; AMIDI, S. *Convolutional Neural Networks cheatsheet*. Disponível em: ⟨https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks⟩.

HAYKIN, S. *Redes Neurais: Princípios e Prática*. [S.l.]: Bookman Editora, 2001. ISBN 9788577800865.

HE, K. et al. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. Disponível em: ⟨http://arxiv.org/abs/1512.03385⟩.

PALIWAL, A. *Understand your convolution network with visualizations*. Disponível em: ⟨https://towardsdatascience.com/understanding-your-convolution-network-with-visualizations-a4883441533b⟩.

RASHID, T. *Make Your Own Neural Network: A Gentle Journey Through the Mathematics of Neural Networks, and Making Your Own Using the Python Computer Language*. [S.l.]: CreateSpace Independent Publishing, 2016. ISBN 9781530826605.

APÊNDICE A – Como fazer uma rede neural

A.1 Entendo aproximadores de função

- Analisar o contexto do problema: O mesmo é linear, não linear (RASHID, 2016).
- A partir do item anterior, definir qual a função da ativação será usada (RASHID, 2016).

É importante também, dependendo do tipo de problema, saber escolher uma função de cálculo de erro, essa função, pode ser uma função quadrática, linear ou outra dependendo da importância que o erro tem na solução do problema. As funções de cálculo de erro mais usadas são mostradas abaixo:

- $erro = valorDesejado - valorCalculado$
-

Calculado o erro é necessário agora, de alguma forma, fazer com que esse erro seja corrigido utilizando uma abordagem iterativa que se aproxima cada vez mais do valor desejado usando pequenos passos. Considerando que Δ represente uma pequena variação em um valor *peso*, e sendo *peso* um dos parâmetros da função de ativação de um aproximador ($y = peso.x$ para uma função de ativação linear) então temos que a correção do valor deve ser dado como na equação abaixo:

$$valorDesejado = (peso + \Delta peso).entrada \quad , \tag{A.1}$$

(RASHID, 2016)

Então, sabendo que o $valorDesejado = peso + \Delta peso.entrada$ e que $erro = valorDesejado - valorCalculado$ se pode concluir que:

$$\Delta peso = \frac{erro}{entrada} \quad , \tag{A.2}$$

No tocante as funções de ativação se pode imaginar quantas forem necessárias, no entanto, algumas se destacam dentro do campo das redes neurais:

- *step function*:

$$y = \begin{cases} 0 & x \leq 0 \\ 1 & x > 0 \end{cases} \tag{A.3}$$

- *sigmoid function*:

$$y = \frac{1}{1 + e^{-x}} \tag{A.4}$$

No entanto, existe um problema quanto a esta abordagem, a função de ativação que recebe o parâmetro *peso* Se adaptará ao último exemplo mostrado a ela criando uma situação conhecida como "overfitting", invalidando assim todos os os outros exemplos anteriormente usados, portanto, como se pode notar, é necessário a criação de um elemento que impeça esse ajustamento extremo, chamado de taxa de aprendizado:

$$\Delta peso = taxaDeAprendizado.\left(\frac{erro}{entrada}\right) \qquad , \tag{A.5}$$

Usualmente a taxa de aprendizado é um valor suficientemente pequeno cujo o objetivo é garantir que o *ovefitting* não aconteça mas, suficientemente grande para que a rede aprenda em um tempo razoável. Sendo $0,1$ um dos valores normalmente usados.

A.2 Junção dos aproximadores: Redes neurais

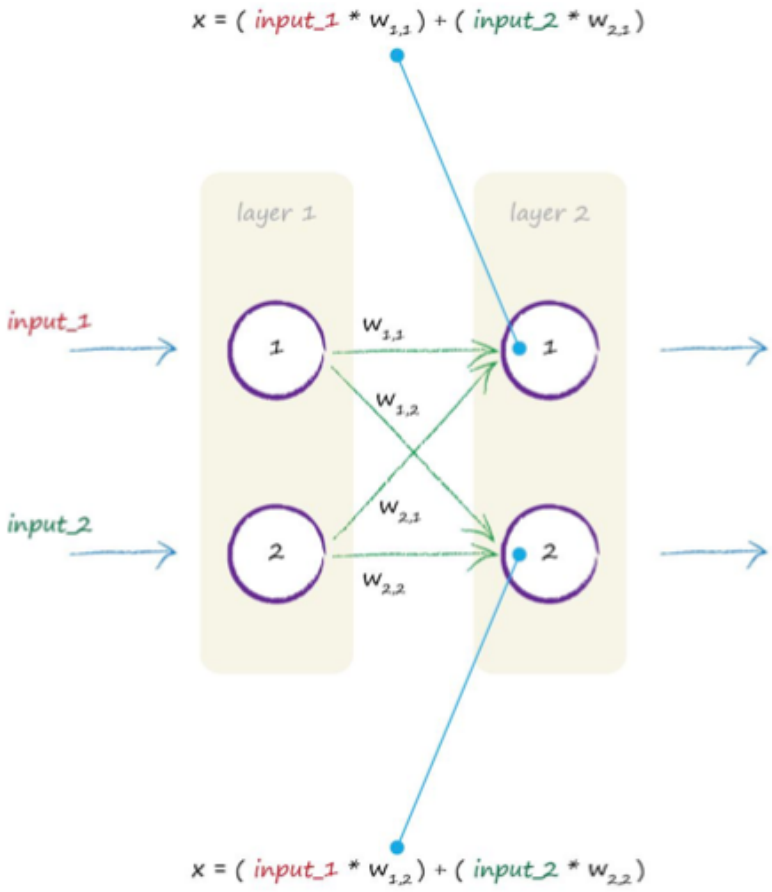
Já que a sessão anterior definiu o que é um aproximador de funções, nesta sessão, usando as definições já vistas, será apresentada a definição de uma rede neural que é a junção de vários aproximadores que, dessa forma, conseguem delimitar espaços de resultados mais complexos além daqueles que podem ser separados por apenas uma função de ativação.

Um dos exemplos clássicos que necessitam dessa abordagem mais complexa é o exemplo da porta lógica *xor*. Quando se tenta usar apenas um aproximador de funções para conseguir os resultados dessa porta é possível notar que o mesmo não é capaz de criar um espaço de resultados suficientemente complexo afim de separar os pontos fornecidos, sendo assim, Torna-se necessário o uso de múltiplos aproximadores (ou, nesse caso, 2).

Pode-se considerar tal junção como uma a rede neural. É possível perceber que a mesma passa ter múltiplas entradas tornando-se necessário, antes que a função de ativação seja aplicada, que a soma dessas entradas seja feita. A este procedimento se dá o nome de *feedforward* (HAYKIN, 2001):

Via de regra uma rede neural tem, no mínimo, duas camadas: A camada de entrada que é usada apenas para representar os valores que serão processados e a camada de saída, cujo papel é fazer a soma ponderada das entradas passando então o resultado dessa operação para uma função de ativação que finalmente gerará as saídas de nossa rede neural como ilustrado na figura 3.

Figura 3 – Uma rede neural simples Fonte: O autor



A.3 Uma rede neural simples

Como visto na figura 3 da sessão anterior uma rede neural conecta suas camadas usando pesos que servirão como moderadores das entradas da próxima camada com o fim desse evitar o *overfitting*. É possível representar uma rede neural de várias formas, no entanto, como ilustrado na figura 4, a forma mais eficiente computacionalmente será a matricial, usando esta abordagem será possível representar facilmente as camadas e os respectivos pesos que as ligam, melhorando também o desempenho computacional já que tais operações são altamente paralelizáveis.

Figura 4 – Representação de uma rede neural usando matrizes

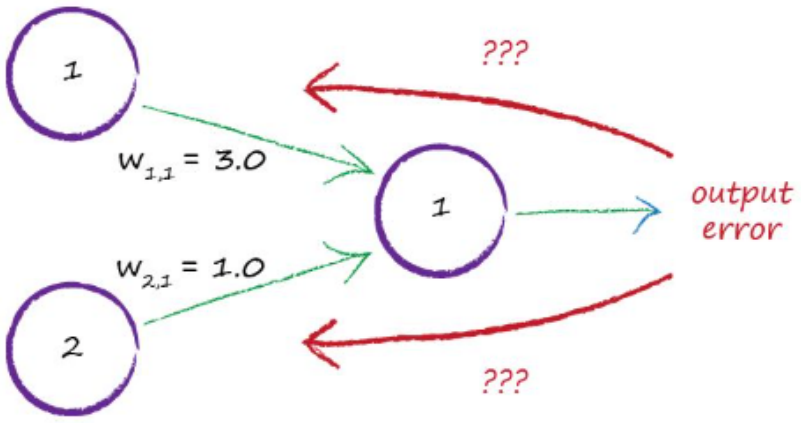
A.4 Redes neurais multicamadas

A depender da complexidade do problema apenas duas camadas não são suficientes para que este aproximados universal de funções, mais conhecido como rede neural, consiga obter resultados satisfatórios. Assim, se torna necessário adição de uma ou mais camadas que servirão para aumentar a complexidade do espaço das soluções de nossa rede.

Então, considerando o contexto, as tais camadas que devem ser adicionadas serão chamadas de ocultas ou *hidden layers*(RASHID, 2016).

Em redes neurais multicamadas surge um novo problema: Como atualizar os valores dos pesos entre as camadas já que o erro produzido depende das várias entradas da camada anterior? Como mostrado na figura 3 o valor da próxima camada depende da soma ponderada pelos pesos dos valores da anterior sendo que tal soma deve passar por uma função de ativação escolhida, portanto, após o cálculo do erro na resposta da rede é preciso recalcular os pesos de forma a distribuir esses erros usando um algoritmo ao qual daremos o nome de retro-propagação ou *backpropagation* (HAYKIN, 2001). É importante frisar que este é apenas um dos vários métodos de treinamento da rede neural, é possível, por exemplo, usar uma abordagem bio-inspirada e/ou evolutiva.

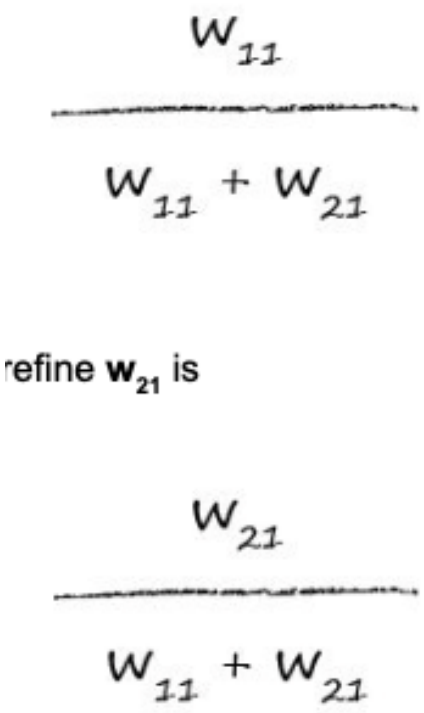
Figura 5 – Retro-propagação



A.5 Retro-propagação

Primeiramente é importante notar que os pesos associados a cada item da camada (neurônio) Contribuem com o erro resultante de forma proporcional aos seus respectivos valores, sendo assim, é necessário ao fazer a retro-propagação que esses erros sejam distribuídos proporcionalmente como indicado na figura 6.

Figura 6 – Parcela da distribuição do erro na retro-propagação para o $erro_1$, o mesmo deve ser feito para todos os outros erros.



Calculada a parcela da participação de cada peso nos erros produzidos, Se passar a fase do erro resultante na camada oculta para que, usando o mesmo raciocínio de cálculo da participação no erro, se possa continuar com algoritmo de retro-propagação. Assim, o cálculo do erro da camada oculta se dar como mostrado na equação A.6.

$$erro_{oculto_1} = \begin{cases} erro_{resultado_1} * \dfrac{peso_{1,1}}{peso_{1,1} + peso_{2,1} + ... + peso_{n,1}} + \\ erro_{resultado_2} * \dfrac{peso_{1,2}}{peso_{1,2} + peso_{2,2} + ... + peso_{n,2}} + \\ \vdots \\ + erro_{resultado_k} * \dfrac{peso_{1,k}}{peso_{1,k} + peso_{2,k} + ... + peso_{n,k}} \end{cases} \tag{A.6}$$

Onde, em relação ao neurônio atual da camada oculta($oculto_1$), n representa a quantidade de pesos vinculados e k o número de neurônios na camada de saída . As figuras 7 e 8 mostram em mais detalhes o funcionamento do algoritmo.

Figura 7 – Retro-propagação da camada de saída para a oculta

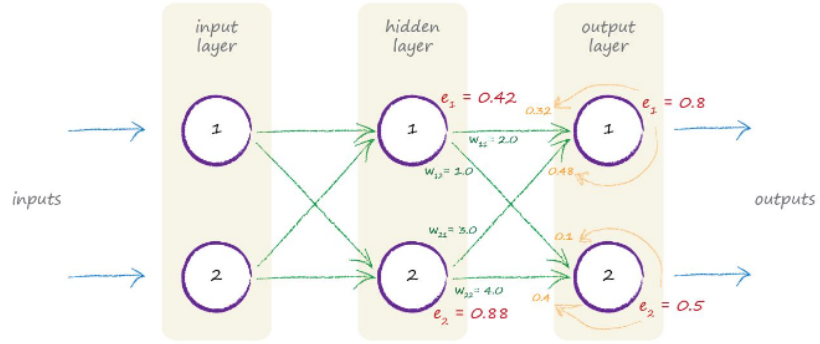
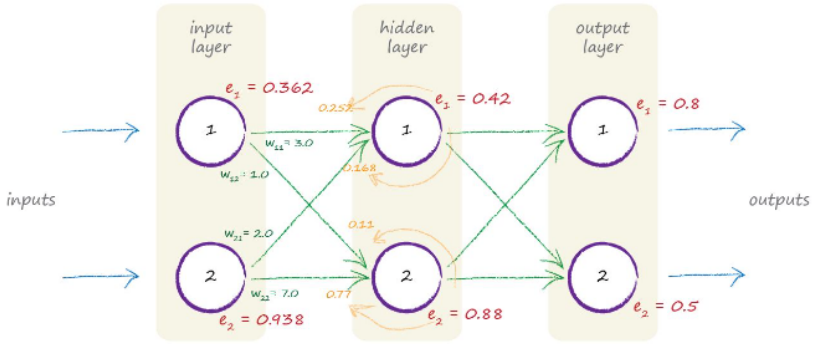


Figura 8 – Retro-propagação da camada oculta para a entrada



UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Campus de São José do Rio Preto

TERMO DE REPRODUÇÃO XEROGRÁFICA

Autorizo a reprodução xerográfica do presente Trabalho de Conclusão, na íntegra ou em partes, para fins de pesquisa.

São José do Rio Preto, 06 de Agosto de 2022

André Furlan