



UNIVERSIDADE FEDERAL DO AMAZONAS
FACULDADE DE TECNOLOGIA
ENGENHARIA DA COMPUTAÇÃO

CLASSIFICAÇÃO DE SÉRIES TEMPORAIS COM APRENDIZAGEM PROFUNDA

CLEMILTON VASCONCELOS PEREIRA

MANAUS-AM

2018

CLEMILTON VASCONCELOS PEREIRA

CLASSIFICAÇÃO DE SÉRIES TEMPORAIS COM APRENDIZAGEM PROFUNDA

Monografia apresentada à Coordenação do
Curso de Engenharia da Computação da
Universidade Federal do Amazonas, como
parte dos requisitos necessários à obtenção
do título de Engenheiro de Computação.

Orientador: RAFAEL GIUSTI

MANAUS-AM

2018

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

C957s Cruz Junior, António César Vieira da
Sistema de detecção de frequência fundamental para música /
António César Vieira da Cruz Junior. 2018
66 f.: il. color; 31 cm.

Orientador: Waldir Sabino da Silva Júnior
TCC de Graduação (Engenharia da Computação) - Universidade
Federal do Amazonas.

1. Frequência fundamental. 2. Processamento digital de áudio. 3.
Transcrição musical. 4. Detecção automática. I. Silva Júnior, Waldir
Sabino da II. Universidade Federal do Amazonas III. Título

CLASSIFICAÇÃO DE SÉRIES TEMPORAIS COM APRENDIZAGEM PROFUNDA

Clemilton Vasconcelos Pereira

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO CURSO DE ENGENHARIA DA COMPUTAÇÃO DA UNIVERSIDADE FEDERAL DO AMAZONAS COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO.

Aprovada por:

Prof. Rafael Giusti, D. Sc.

Prof., M. Sc.

Prof. , D. Sc.

Manaus
Julho de 2018

*Dedico este trabalho à minha mãe
Lúcia, "In memoriam", também aos meus
pais, Antonio e Elenir, e às minhas avós
Jacyra e Maria, meus exemplos de vida.*

Agradecimentos

Antes de tudo, a Deus, autor da vida, cujo a bondade e a misericórdia me seguem a cada dia. Tu és fiel, Senhor!

Aos meus pais, Antonio e Elenir, que sempre me apoiaram de todas as maneiras possíveis, à minha amada esposa Yasmin, sempre ao meu lado, e aos meus familiares, que sempre me incentivaram a persistir.

Ao meu orientador, prof. Dr. Waldir Sabino, pela paciência, disposição e direcionamento durante a execução deste projeto, e também pelos conselhos que levo para minha vida profissional.

A todos os professores da UFAM, em especial o prof. Dr. César Melo, aos amigos de turma Rosmael, Helton, João Victor, Gabriel, Jordan, Kabashima, Eduardo, Luiz Henrique, Micael e Clemilton, e aos demais amigos e colegas da UFAM pelo convívio, aprendizado e cooperação durante todos esses anos. Vocês foram essenciais nessa caminhada.

À família Machado, por todo o cuidado no momento em que mais precisei, e aos irmãos da Segunda Igreja Batista de Manaus, que me adotaram assim que cheguei nesta cidade. Também à Keyseane, que me ajudou em momentos importantes na escrita deste trabalho, e aos meus amigos José Eduardo, Vanessa e Débora.

A todos que, de algum modo, colaboraram para que este momento se concretizasse:

Muito obrigado!

“Se algum de vocês tem falta de sabedoria, peça-a a Deus, que a todos dá livremente, de boa vontade; e lhe será concedida.”

(Tiago 1.5)

Resumo

Existem diferentes caminhos para o reconhecimento automático de uma notas musical, e um deles é por meio da frequência fundamental (f_0). Muitas soluções e métodos de detecção automática de f_0 já foram desenvolvidas e apresentadas, obtendo resultados positivos. Todavia, é difícil que uma única solução de detecção seja de fato eficaz em ambientes muito diferentes daqueles para o qual a solução foi proposta. Diante disso, o presente projeto aborda o desenvolvimento de um sistema de detecção de f_0 para áudios musicais monofônicos, através de processamento digital de sinais, com o intuito de mapear os áudios, fornecendo as frequências fundamentais soadas em cada instante de tempo. Este trabalho também avaliou o sistema desenvolvido, por meio de experimentação a partir de uma base de dados construída para este fim. Percebeu-se que o sistema proposto apresenta boas respostas, sendo necessário melhorias em relação ao tratamento do efeito de ressonância.

Palavras-chave: Detecção automática de frequência fundamental, Processamento digital de áudio, Transcrição musical.

Abstract

There are different ways for a automatic recognition of musical note, one of them is through fundamental frequency (f_0). Many solutions and methods of automatic detection of f_0 already developed and presented, getting positive results. However, it's hard that one only solution has been effective in environments much different from these that it was proposed. Therefore, the present project approaches the development of a f_0 detector system for monophonic musical audios through digital signal processing, with the intention of mapping it, recording the fundamental frequencies sounded at every instant time. This work also evaluated the system developed, through experiments using a database built for it. It was perceived that the proposed system presents good answers, being necessary improvements in the treatment of the resonance effect.

Keywords: Automatic Detection of fundamental frequency, Digital Audio Processing, Musical transcription.

Lista de Figuras

2.1	Codificação One-hot	4
2.2	Princípio dos k-vizinhos mais próximos	5
2.3	Representação simplificada de um neurônio	6
2.4	Modelo matemático de um neurônio	7
2.5	Multilayer Perceptron. Cada círculo representa um neurônio mostrado anteriormente	8
2.6	Softmax vs ReLU. Imagem tirada do site <i>Towards Data Science</i> [1]	11
2.7	Imagem ilustrando o efeito do Dropout em uma rede neural onde os neurônios marcados foram desativados, forçando a rede não depender deles, evitando o <i>overfitting</i> [2].	13
2.8	Campo receptivo de um neurônio. Fonte: [3]	13
2.9	Disposição dos neurônios na camada de convolução e seu campo receptivo. Fonte: [4]	15
2.10	Neurônios espaçados por stride 1 e 2 das esquerda para a direita respectivamente. Separados à direita estão os pesos compartilhados entre todos os neurônios. Fonte: [5]	16
2.11	Representação de max-pooling e average-pooling [5]	16
2.12	Arquitetura LeNet-5. [6]	17
2.13	Bloco Residual. Fonte: [7]	18

Lista de Tabelas

2.1	Representação da base de dados	3
-----	--	---

Lista de Abreviaturas e Siglas

PDS	Processamento Digital de Sinais
MIDI	Interface Digital de Instrumentos Musicais – do inglês <i>Musical Instrument Digital Interface</i>
DTFT	Transformada de Fourier em Tempo Discreto – do inglês <i>Discret-Time Fourier Transform</i>
DFT	Transformada Discreta de Fourier – do inglês <i>Discret Fourier Transform</i>
FFT	Transformada Rápida de Fourier – do inglês <i>Fast Fourier Transform</i>
STFT	Transformada de Fourier de Tempo Curto – do inglês <i>Short-Time Fourier Transform</i>

Lista de Símbolos

Símbolos Matemáticos

f_s	Frequência de amostragem
f_0	Frequência fundamental
Hz	Hertz - Unidade do Sistema Internacional (SI) para frequência
bpm	Batidas por minutos

Sumário

1	Introdução	1
1.1	Organização do Trabalho	1
1.2	Objetivo	1
2	Fundamentação Teórica	2
2.1	Aprendizado de Máquina	2
2.1.1	Aprendizado Supervisionado	3
2.1.2	Normalização e One-Hot Enconding	4
2.1.3	K-Nearest Neighbors (KNN)	5
2.2	Redes Neurais Artificiais	6
2.2.1	Inspiração Biológica e Perceptron	6
2.2.2	Perceptron multicamadas	7
2.2.3	Camada Softmax	9
2.2.4	Função de Perda(Loss Function)	9
2.2.5	Otimizadores	9
2.2.6	Hiperparâmetros de uma Rede Neural	10
2.2.6.1	Número de camadas e neurônios em cada camada	11
2.2.6.2	Inicialização dos pesos	11
2.2.6.3	Função de Ativação	11
2.2.6.4	Mini-batches	11
2.2.6.5	Taxa de aprendizado	12
2.2.6.6	Número de épocas	12
2.2.7	Regularização	12
2.2.8	Redes Convolutivas	13
2.2.8.1	Camada de Convolução	14

2.2.8.2	Camada Pooling	16
2.2.8.3	Normalização em lote	17
2.2.8.4	Arquitetura de uma Rede Convolutiva	17
2.2.9	Redes Residuais	18
2.3	Séries Temporais	18
2.3.1	Aplicações de Séries Temporais	19
2.4	Classificação de Séries Temporais	19
3	Sistema Proposto	20
3.1	Visão geral	20
4	Experimentos	21
4.1	Métricas	21
4.2	Objetivos	21
4.3	Resultados Obtidos	21
5	Conclusão	22
	Referências Bibliográficas	23

Capítulo 1

Introdução

1.1 Organização do Trabalho

1.2 Objetivo

Capítulo 2

Fundamentação Teórica

Este capítulo aborda, de forma sucinta, os fundamentos teóricos que foram aplicados no planejamento e implementação deste projeto.

2.1 Aprendizado de Máquina

Aprendizado de máquina(AM) é uma área da inteligência artificial cujo objetivo é o desenvolvimento de técnicas computacionais sobre o aprendizado, bem como a construção de sistemas capazes de adquirir conhecimento de forma automática. Um sistema baseado em aprendizado trabalha por meio de experiências acumuladas e de soluções bem-sucedidas de problemas anteriores [8]. Normalmente, algoritmos de aprendizado utilizam experiências anteriores, denominadas conjunto de treino, para auxiliar no processo de tomada de decisão.

Existem três diferentes tipos de aprendizado: supervisionado, não-supervisionado e semi-supervisionado. A diferença entre esses tipos de aprendizado é se o método utiliza ou não utiliza o rótulo de treino. No aprendizado supervisionado, esse rótulo é conhecido, enquanto que no aprendizado não-supervisionado os exemplos não vistos. Já no aprendizado semi-supervisionado, o conjunto de treinamento consiste de uns poucos exemplos rotulados e muitos não rotulados[9]. O escopo deste trabalho se encontra no aprendizado supervisionado.

	A_1	A_2	\dots	A_M	Classe(Y)
E_1	x_{11}	x_{12}	\vdots	x_{1M}	y_1
E_2	x_{21}	x_{22}	\vdots	x_{2M}	y_2
\vdots	\vdots	\vdots	\ddots	\vdots	\vdots
E_N	x_{N1}	x_{N2}	\vdots	x_{NM}	y_N

Tabela 2.1: Representação da base de dados

2.1.1 Aprendizado Supervisionado

O objetivo do aprendizado supervisionado é construir um modelo que consegue fazer predição através de instâncias de uma base de dados rotuladas. Cada instância é representada por um conjunto de características. Na Tabela 2.1 é mostrada a estrutura de uma base. Neste exemplo cada vetor $E_i = [x_{i1}, \dots, x_{iM}]$ se refere a classe y_i .

A idéia da aprendizagem supervisionada é o conseguir encontrar uma função desconhecida f (função conceito) tal que $y = f(\mathbf{x})$, onde o vetor \mathbf{x} são os atributos de uma instância específica. Na prática, a função f deve conseguir prever o valor correto y_i de uma instância E_i não vista. Normalmente, o número de exemplos da base de dados não é suficiente para descrever a função conceito. Nesse caso, o classificador é visto como uma hipótese h que aproxima f , ou seja, $h(x) \approx f(x)$. Caso os valores dos rótulos y_1, y_2, \dots, y_N sejam numéricos o problema é denominado de *regressão*, caso sejam valores categóricos o problema é denominado de *classificação*.

De maneira geral, a base de dados é dividida em dois conjuntos: conjunto de *treino* e conjunto de *teste*. O conjunto de treinamento é utilizado para ajustar o classificador. Como dito anteriormente, o classificador é uma hipótese da função conceito f , logo, é fundamental que o conjunto de treinamento tenha uma distribuição o mais semelhante possível do conjunto original. O conjunto de teste é utilizado para avaliar o modelo construído. Idealmente, esse conjunto não deve ter exemplos em comum com o conjunto de treinamento.

Em alguns casos, pode ser necessário utilizar um conjunto de *validação*, extraído do conjunto de treinamento, para realizar ajustes no modelo construído pelo algoritmo de aprendizado. Logo tem-se três conjuntos: *treino*, *validação* e *teste*. O treino é utilizado para aprendizagem do algoritmo. O modelo é avaliado através do conjunto de validação. É feita uma alteração dos parâmetros do classificador e outro treinamento é realizado.

O intuito é melhorar o desempenho do modelo através desses "ajustes". Dessa maneira os exemplos de validação são indiretamente "vistos" durante o processo de aprendizado, o que obriga que esses exemplos sejam diferentes dos exemplos de testes.

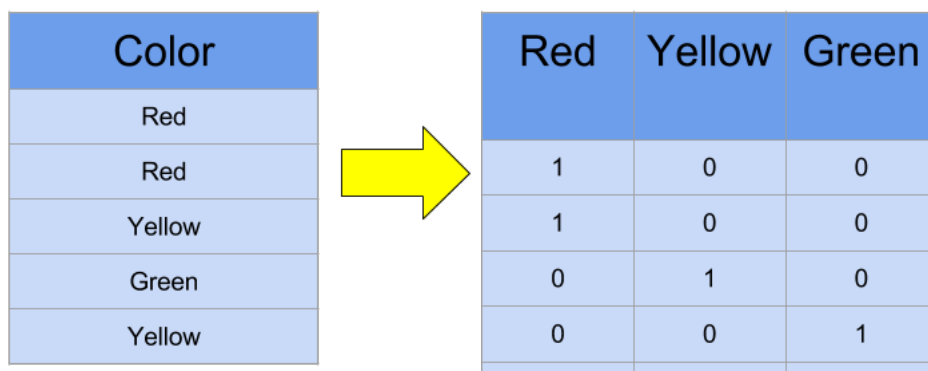
2.1.2 Normalização e One-Hot Encoding

Os algoritmos de aprendizagem de máquina aprendem através dos dados. Dados do mundo real apresentam valores que estão em distintas faixas. A fim de evitar que algum atributo predomine sobre outro ou que inclua alguma ponderação indesejada ao induzir um modelo de AM, é comum fazer uma normalização dos valores de cada atributo. Um forma de normalizar os dados é mostrada na Equação 2.1:

$$x_{ij} = \frac{x_{ij} - \bar{x}}{\sigma_j} \quad (2.1)$$

onde \bar{x} representa a média do atributo e σ_j representa o desvio padrão.

Os algoritmos de AM geralmente possuem como entrada e saída valores numéricos, portanto é necessário converter os valores categóricos da base de dados para valores numéricos. A codificação one-hot é um processo que converte rótulos em vetores binários como mostrado na Figura 2.1. Uma vantagem dessa codificação é que não cria uma "ordem" numérica nos dados. Essa ordem poderia interferir na indução do classificador, podendo dar maior importância para valores maiores, o que não faz sentido para variáveis categóricas.



Color
Red
Red
Yellow
Green
Yellow

Red	Yellow	Green
1	0	0
1	0	0
0	1	0
0	0	1

Figura 2.1: Codificação One-hot

2.1.3 K-Nearest Neighbors (KNN)

Um classificador bastante popular é o K-Nearest Neighbors (K-Vizinhos mais próximos). O KNN utiliza os próprios dados de treinamento como modelo de classificação. Para classificar uma instância de teste, procura-se entre os dados de treinamento os K mais próximos da instância de teste. Por fim, verifica-se qual a classe predominante desses K dados de treinamento, e instância de teste é classificada com essa mesma classe. A cada nova exemplo a ser classificado faz-se uma varredura nos dados de treinamento, o que provoca um grande esforço computacional

O princípio do classificador k-NN é a “regra dos vizinhos mais próximos”. A hipótese é que, dado um conjunto de exemplos distribuídos sobre o espaço de dados X , a “vizinhança” de um exemplo $x \in X$ estabelecida por uma função de distância apropriada tende a ser ocupada por exemplos que pertencem à mesma classes que x [10], como ilustrado na Figura 2.2. Desse modo a informação fornecida pelos exemplos conhecidos que são mais similares a x .

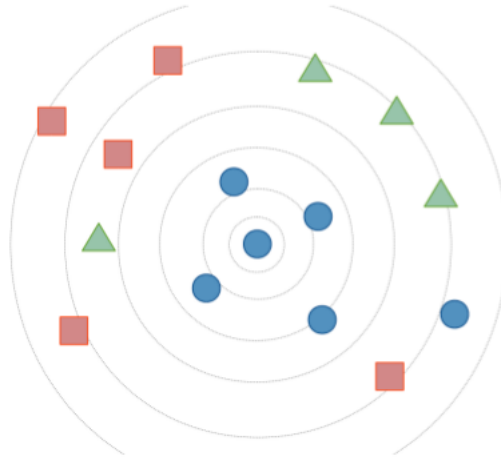


Figura 2.2: Princípio dos k-vizinhos mais próximos

Para encontrar os vizinhos mais próximos é necessário definir uma medida de similaridade entre dois exemplos. Uma medida de similaridade bastante popular é a *distância euclidiana*. Tal medida calcula a raiz quadrada da norma do vetor diferença entre os vetores x e y :

$$d(x, y) = \sqrt{\sum_{i=1}^K (x_i - y_i)^2} \quad (2.2)$$

2.2 Redes Neurais Artificiais

Redes Neurais Artificiais são modelos computacionais que buscam simular o processamento de informação pelo cérebro humano. Elas são compostas por unidades simples de processamento, os neurônios, que se unem por meio de conexões sinápticas [11]. Cada conexão, além de ser altamente especializada, é responsável pelo envio de sinais de um neurônio para outro. Segundo (Haykin 2009[12]), os neurônios e suas conexões podem ser implementados utilizando-se de componentes eletrônicos ou via simulação programada em computador.

2.2.1 Inspiração Biológica e Perceptron

Um bloco básico de uma rede neural tem algumas semelhanças com um neurônio biológico. O neurônio biológico é uma célula formada por três seções com funções específicas e complementares: *corpo*, *dendritos* e *axônio*. Os dendritos captam os estímulos recebidos em um determinado período de tempo e os transmitem ao corpo do neurônio, onde são processados. Quando tais estímulos atingirem determinado limite, o corpo da célula envia um novo impulso que se propaga pelo axônio e é transmitido às células vizinhas por meio de sinapses. Este processo pode se repetir em várias camadas de neurônios. Como resultado, a informação de entrada é processada, podendo levar o cérebro a comandar reações físicas. [13]. A figura 2.3 ilustra de forma simplificada as partes de um neurônio.

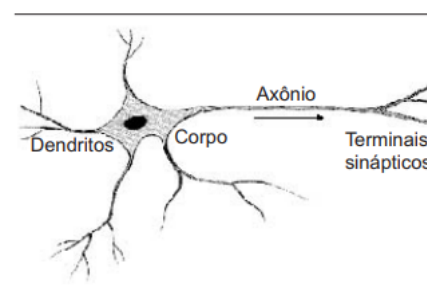


Figura 2.3: Representação simplificada de um neurônio

O modelo de um neurônio artificial é apresentado na Figura ???. Este modelo é composto por três elementos:

- Um conjunto de entradas (x_1, x_2, \dots, x_n) que são multiplicadas por um conjunto de pesos (p_1, p_2, \dots, p_n) ;

- Um somador (Σ) para acumular o sinais de entrada;
- Uma função de ativação que (φ) limita o intervalo permissível de amplitude do sinal de saída (y) a um valor fixo.

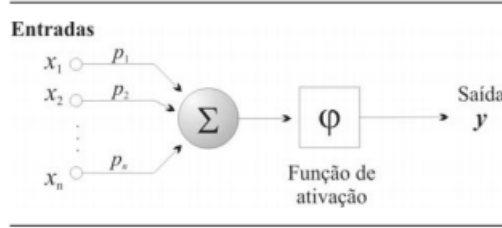


Figura 2.4: Modelo matemático de um neurônio

Esse modelo foi proposto por *McCulloch and Pitts* em 1943 [14] e é conhecido como *perceptron*. A função de ativação (φ) tem a seguinte definição:

$$\varphi = \begin{cases} 1 & \text{if } \sum_{i=1}^n p_i x_i^k \geq T, \\ 0 & \text{if } \sum_{i=1}^n p_i x_i^k < T \end{cases} \quad (2.3)$$

Os valores dos pesos podem ser positivos ou negativos e eles refletem se aquela conexão é inibitória ou excitatória. Um valor positivo ou negativo reflete a importância da respectiva entrada para o processamento. Frequentemente é adicionado um *viés* b na entrada da função de ativação. A forma geral do modelo é descrito como:

$$y(k) = \varphi\left(\sum_{i=1}^n p_i(k)x_i(k) + b(k)\right) \quad (2.4)$$

Os pesos sinápticos do Perceptron podem ser adaptados empregando um processo de aprendizado com um número finito de iterações. A aprendizagem é conduzida pela regra de correção de erro conhecida como algoritmo de convergência do Perceptron. Esse algoritmo visa encontrar um vetor de pesos w tal que as duas igualdades da função degrau sejam satisfeitas (*Lippmann*, 1987 [15])

2.2.2 Perceptron multicamadas

O Perceptron multicamadas (*Multi-Layer Perceptron - MLP*) é uma generalização da rede perceptron. Novas camadas são adicionadas o que possibilita a solução de problemas

que não sejam linearmente separáveis. O vetor de entradas \mathbf{x} passa pela camada inicial, cujos valores de saída são ligados a camada seguinte. Esse processo se repete até chegar na última camada. (Figura 2.5) Pode-se arranjar a rede em várias camadas, tornando-a profunda e capaz de aprender relações cada vez mais complexas.

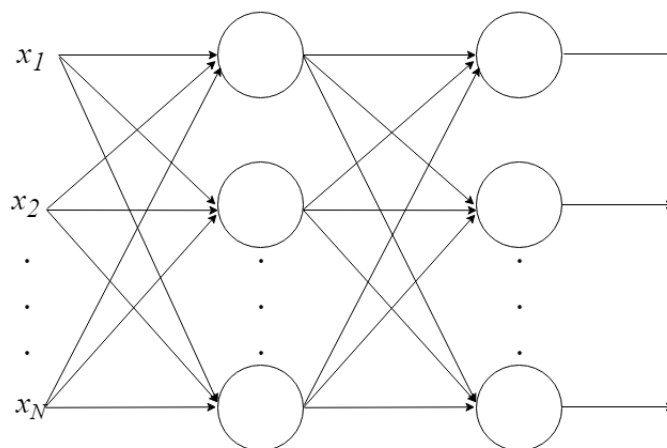


Figura 2.5: Multilayer Perceptron. Cada círculo representa um neurônio mostrado anteriormente

Em 1986, *Rumelhart, Hinton e Williams* [16] desenvolveram o algoritmo *backpropagation*, que utiliza o gradiente descendente para treinar uma MLP. Este método é composto pelas fases *forward* e *backward*. O objetivo do *backpropagation* é otimizar os pesos para que a rede neural possa aprender a mapear corretamente as entradas para as saídas. A primeira fase é a “propagação adiante” (*forward*), onde as entradas inseridas na rede se propagam entre as camadas, uma a uma, até a produção das respectivas saídas, portanto a função dessa fase é gerar uma resposta considerando as entradas e os respectivos pesos sinápticos, os quais permanecem inalterados.

Na fase *backward* é onde a aprendizagem dos pesos é realizada. Esse aprendizado se dá através da otimização (minimização) da função *loss*, a qual determina a qualidade da classificação do dado de entrada. Essa otimização é realizada através de um método chamado *Gradiente Descendente* que busca a minimização da função *loss* ao alterar os pesos na direção de maior declive do gradiente. O gradiente é calculado na última camada e então é retro-propagado para as camadas intermediárias anteriores que contribuem diretamente para a formação da saída. Cada elemento da camada intermediária recebe é responsável apenas por uma porção do gradiente total. Este processo se repete, camada por camada, até que cada elemento tenha a sua parcela de gradiente para o gradiente total. Baseado no gradiente, é feita uma alteração dos valores dos pesos e bias de modo

que a rede aprenda os padrões do conjunto de treinamento.

2.2.3 Camada Softmax

Como explicado em *GoodFellow, Bengio, e Courville*(2016 [17]), a camada softmax é utilizada como um classificador na camada de saída e tem como objetivo representar a probabilidade de cada classe para cada valor de entrada.

Pela Equação 3.5 nota-se que os valores de saída da camada softmax estão entre 0 e 1, e que a soma de todas as saídas é igual a 1. Desta forma, cada neurônio de saída representa a probabilidade da entrada pertencer a uma determinada classe.

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)} \quad (2.5)$$

2.2.4 Função de Perda(Loss Function)

A função de perda compara a saída da rede para um exemplo de treinamento com o rótulo verdadeiro. Uma função de perda comum é o erro médio quadrático dado pela Equação 2.6

$$MSE = \frac{\sum_{i=1}^N (y_i - \hat{y}_i)^2}{N} \quad (2.6)$$

Quando a saída da rede neural está sendo tratada como uma distribuição de probabilidade é comum utilizar entropia cruzada *cross-entropy*. Ela é usada para quantificar as distâncias entre duas distribuições de probabilidade. Em redes neurais a entropia cruzada é comparada a distribuição do que o modelo prediz com a previsão. Ela é definida como:

$$L = - \sum_i y_i \log(z_i) \quad (2.7)$$

Para problemas de classificação a entropia cruzada é mais adequada.

2.2.5 Otimizadores

A finalidade dos otimizadores é minimizar o valor da função de perda alterando os pesos da rede. Existem atualmente diversos métodos para alcançar esse fim, esses métodos utilizam de alguns hiperparâmetros para calibrar a forma com que os pesos devem ser

atualizados. Dentre os mais utilizados se destacam:

- **Stochastic Gradient Descent (SGD) com Nesterov Momentum** é um método onde, em uma alusão à mecânica clássica, a velocidade de descida do resultado da função de perda na superfície multidimensional determinada pelo gradiente é calculada e usada para escalar o tamanho do passo que será dado em direção ao ponto mínimo. Utilizando essa velocidade e o valor do gradiente na posição atual é possível prever a próxima posição dessa partícula virtual onde então é calculado o gradiente da nova posição. Os parâmetros são então atualizados baseados nesse segundo gradiente e em um hiperparâmetro denominado taxa de aprendizado (learning rate). [18]
- **Adagrad** é um método adaptativo de taxa de aprendizado proposto originalmente por [19] onde a taxa de aprendizado é normalizada pela raiz da soma dos gradientes de cada parâmetro ao quadrado. Isso tem o efeito de reduzir a taxa de aprendizado dos pesos que recebem gradientes altos e ao mesmo tempo aumentar a taxa de aprendizado de pesos que recebem atualizações infrequentes. Um hiperparâmetro de suavização também é usado para inibir a divisão por zero.
- Adam é um método derivado de um outro método chamado RMSprop que basicamente ajusta o método Adagrad para que a taxa de aprendizado não diminua agressivamente. A contribuição do método Adam é adicionar um momento na atualização (semelhante ao Nesterov Momentum) e suavizar os ruídos do gradiente antes de fazer essa operação. Ele herda do RMSprop adição de uma taxa de decaimento na soma dos gradientes de cada parâmetro que reduz a agressividade de quanto a taxa de aprendizado é reduzida a cada passo. [20]

2.2.6 Hiperparâmetros de uma Rede Neural

A maioria dos algoritmos de AM envolvem "hiperparâmetros", que são variáveis definidas para o algoritmo antes do treinamento com o objetivo de otimizá-lo. Definir os valores de hiperparâmetros pode ser visto como uma seleção de modelos, ou seja, escolher qual modelo usar do conjunto hipotético de modelos possíveis. Em redes neurais os hiperparâmetros determinam a estrutura da rede (Ex: Número de neurônios em uma camada) ou como a rede será treinada (Ex: Taxa de aprendizado)

2.2.6.1 Número de camadas e neurônios em cada camada

Em problemas de classificação geralmente as redes neurais tem: camada de entrada tem o mesmo número de neurônios do tamanho do vetor de características, camadas escondidas e a camada de saída que tem o mesmo número de classes do problema. Conforme o teorema da aproximação universal [21], uma rede neural com uma única camada oculta com um número finito de neurônios é capaz de aproximar qualquer função contínua de \mathbb{R}^n . Isto quer dizer que teoricamente é possível aproximar qualquer

2.2.6.2 Inicialização dos pesos

Idealmente, pode ser melhor usar diferentes estratégias de inicialização dos pesos conforme a função de ativação aplicada em cada camada. A distribuição uniforme é comumente utilizada. (*)

2.2.6.3 Função de Ativação

As funções de ativações são usadas para introduzir não-lineariedade nos modelos, que permite que as redes aprendam superfícies de decisões mais complexas. Dentre as funções de ativações mais utilizadas estão a Sigmoid e ReLU (*Rectified Linear Units*). A vantagem da ReLU é que ela evita o problema de dissipação do gradiente.

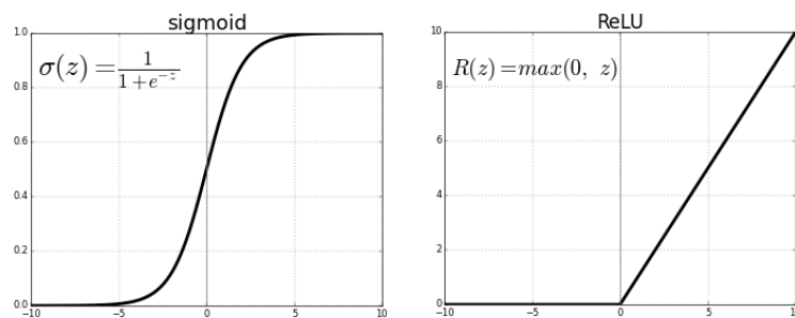


Figura 2.6: Softmax vs ReLU. Imagem tirada do site *Towards Data Science* [1]

2.2.6.4 Mini-batches

O treinamento geralmente é realizado em *batches* também conhecidos como *mini-batches* que são subconjuntos do treino. As razões de utilizar batches e não o treino inteiro são: diminuição do tempo de treinamento; caso o conjunto de treino seja suficientemente grande, uma amostra desse conjunto pode representar de forma *boa* o conjunto original.

A tamanho dessa amostra é um hiperparâmetro da rede, sendo geralmente aceito que o treinamento com *batches* maiores resulta numa melhor performance.

2.2.6.5 Taxa de aprendizado

A taxa de aprendizado é um parâmetro constante no intervalo de $[0,1]$ que interfere na convergência do aprendizado. Ela determina o quão “rápido” as atualizações dos pesos irão em direção do gradiente. Se a taxa de aprendizado for muito pequena, o modelo convergirá muito lentamente; Se a taxa de aprendizado for muito grande, o modelo irá divergir.

2.2.6.6 Número de épocas

Número de épocas é a quantidade de vezes que o conjunto de treino é passado pela rede. O recomendável é aumentar o número de épocas até que a acurácia do conjunto de validação diminuir, mesmo quando a acurácia do treino aumentar (*overfitting*)

2.2.7 Regularização

Além do *tuning* dos hiperparâmetros existe outras maneiras para aumentar a performance da rede e torná-la mais robusta em sua capacidade de generalização. Um exemplo é a adição de um termo de regularização na função de perda. Esse termo é introduzido para regularizar os valores dos pesos buscando distribuir melhor o aprendizado por todos os neurônios. A função provavelmente mais usada para este fim é a da regularização L2, onde para cada parâmetro da rede o termo $\frac{1}{2}\lambda\omega^2$ é adicionado ao resultado da função de perda, ω representando o peso em questão, λ sendo a força da regularização e o fator $\frac{1}{2}$ é multiplicado para simplificar o gradiente da função.

Essa regularização penaliza neurônios com pesos muito altos levando a uma distribuição das atualizações para os neurônios com pesos menores. Ao introduzir essa função é necessário lembrar que os pesos também devem ser atualizados pelo gradiente da função de regularização, o que é um passo geralmente realizado após a atualização quanto ao gradiente da função loss.

A introdução da função de regularização ajuda a combater o overfitting e juntamente com ela uma outra técnica que é muito utilizada para esse fim é o *Dropout* [2]. O Dropout é uma técnica que condiciona à existência de um neurônio na fase de treinamento em função

de uma probabilidade predefinida por um hiperparâmetro. Esse método essencialmente desativa neurônios com uma probabilidade p para que a rede seja capaz de otimizar a função loss mesmo sem a presença de todos os seus neurônios.

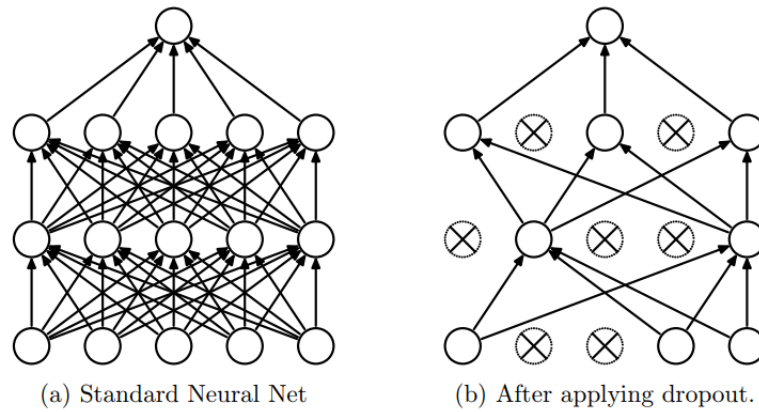


Figura 2.7: Imagem ilustrando o efeito do Dropout em uma rede neural onde os neurônios marcados foram desativados, forçando a rede não depender deles, evitando o *overfitting* [2].

2.2.8 Redes Convolutivas

As redes convolutivas foram desenvolvidas tomando como inspiração o córtex visual de animais, que contém agrupamentos de células que detectam luz em pequena escala que são chamadas de campo receptivos. O campo receptivo de um único neurônio sensorial é a região específica da retina em que a presença de um estímulo causa a ativação dele. Os neurônios sensoriais que tem campos receptivos semelhantes se sobrepõem. Na figura 2.8 é mostrada uma representação de um neurônio e o seu campo receptivo.

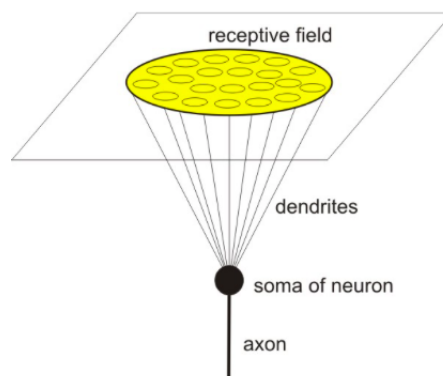


Figura 2.8: Campo receptivo de um neurônio. Fonte: [3]

Tais células atuam como filtros locais sobre o espaço de entrada e quanto mais com-

plexa a célula, maior o seu campo receptivo. As redes convolutivas seguem esse mesmo princípio de campos receptivos, o que ajuda a diminuir o custo computacional de treinamento. Esse tipo de rede recebe esse nome devido a sua camada mais predominante, a camada de convolução.

Em [22] Haykin divide a estrutura das redes convolutivas em três objetivos principais:

- Extração de características: Cada neurônio recebe sinais de entrada de uma determinada janela da camada anterior, possibilitando a extração de características locais. Isto faz com que a posição exata de cada característica (ou pixel, no caso de uma imagem), seja irrelevante, desde que sua posição em relação às características vizinhas seja mantida.
- Mapa de características: As camadas são compostas por diversos mapas de características (*feature maps*) que são regiões onde os neurônios compartilham os mesmos pesos. Esses pesos são chamados de filtros ou kernels, e dão robustez ao modelo, fazendo com que ele seja capaz de lidar com variações de distorção, rotação e translação na imagem. O compartilhamento dos pesos também possibilita uma redução drástica no número de parâmetros a serem otimizados.
- Subamostragem: Após cada camada de convolução é aplicada uma camada de subamostragem (*subsampling*), que é uma coleta de amostras de cada mapa de característica. Essas amostragens podem ser realizadas através da média, maior ou menor valor das amostras. O que produz uma sumarização do mapa de características

A principal aplicação das CNNs é para processamento de informações visuais em particular imagens, pois a convolução permite filtrar as imagens considerando a sua informação visual. Diferente da MLP que vetoriza as imagens a CNN leva em consideração a estrutura espacial entre os pixels vizinhos em uma determinada imagem através da operação de convolução.

A seguir é apresentado os tipos de camadas que geralmente são usadas para redes convolutivas

2.2.8.1 Camada de Convolução

É na camada de convolução que cada característica específica em alguma posição espacial da entrada é detectada. Os parâmetros dessa camada são distribuídos entre os

filtros que tem a mesma ideia dos campos receptivos comentados anteriormente. Os filtros são parâmetros ajustáveis do modelo que devem aprender uma característica específica. A convolução ocorre durante a fase *forward*, onde cada um desses filtros são convoluídos com a entrada. Em aprendizagem de máquina os dados são discretos o que acaba simplificando a operação de convolução. O cálculo se torna em um produto entre os elementos de entrada sobrepostos pelos filtros deslocados em toda entrada.

Alguns hiperparâmetros são importantes na configuração de tal camada[23]:

- Tamanho do campo receptivo(*kernel size*): Para o caso de imagens é o tamanho bidimensional do campo receptivo que será convoluído com a entrada, e pode ter valores como 3x3, 5x5, 16x16. É o mesmo para todos os neurônios da camada. Para o caso em 1-D esse hiperparâmetro é apenas um inteiro.
- Profundidade: É o número de kernels que serão estimulados pela mesma parte do campo receptivo. Cada um desses kernels é responsável por detectar uma característica por toda entrada, constituindo então um conjunto de mapas de características. A Figura 2.9 ilustra uma camada de convolução com profundidade cinco.

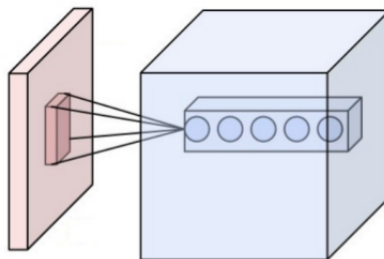


Figura 2.9: Disposição dos neurônios na camada de convolução e seu campo receptivo. Fonte: [4]

- Stride: É o parâmetro que determina o espaçamento da sobreposição dos campos receptivos. A Figura 2.10 mostra que valores diferentes de stride podem alterar o campo receptivo de tais neurônios e o número total dos neurônios de cada fatia da profundidade. Esse parâmetro é decisivo na capacidade de identificar os padrões visuais da entrada, sendo responsável por definir quão refinada será a identificação do padrão mesmo que trasladado, onde o aumento desse refinamento é inversamente proporcional ao desempenho da camada, por conta do maior número de neurônios da mesma.

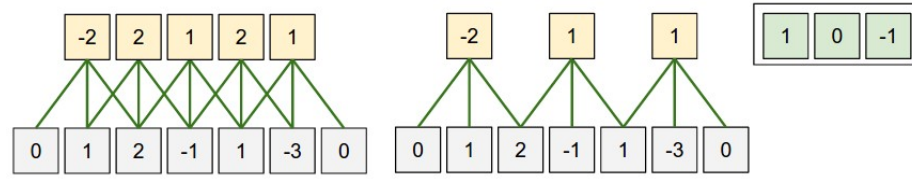


Figura 2.10: Neurônios espaçados por stride 1 e 2 das esquerda para a direita respectivamente. Separados à direita estão os pesos compartilhados entre todos os neurônios. Fonte: [5]

- Zero-padding: Determina qual tamanho do preenchimento com zeros que será adicionado às bordas da entrada. Isso muitas vezes é utilizado para adequar o tamanho da entrada ao stride escolhido, como pode ser visto na Figura 2.10 onde o tamanho da entrada (representada unidimensionalmente) é cinco e o zero-padding é um.

2.2.8.2 Camada Pooling

A camada de pooling faz uma subamostragem da entrada, onde cada valor de saída é encontrado pela a média ou valor máximo de uma janela definida. Na figura 2.11 é mostrado o funcionamento de cada pooling. A entrada é uma imagem 4x4 e a saída é obtida através de uma janela 2x2. O valor de saída é calculado conforme o tipo de pooling. Para o pooling médio é calculada médio da janela. Para o caso máximo é encontrado o maior valor da janela.

Estas pequenas amostras de acordo com [17], melhora o poder computacional da rede pois as camadas posteriores vão ficando cada vez menores o que resulta em uma diminuição do uso de memória. Outra vantagem do pooling é que ele ajuda na invariância a translação da entrada.

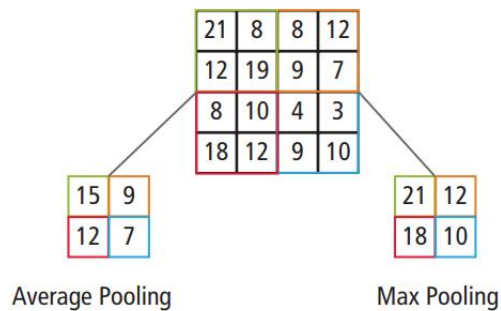


Figura 2.11: Representação de max-pooling e average-pooling [5]

2.2.8.3 Normalização em lote

Conforme explicado na Seção 2.1.2 o conjunto de dados da entrada é normalizado para evitar problemas de comparação devido a diferentes escalas dos dados. Ocorre que durante a passagem dos exemplos normalizados através das camadas da rede, esses valores são novamente transformados (por meio das funções de ativações), podendo fazer com que os dados de entrada de algumas camadas ocultas fiquem desnormalizados novamente. Esse problema é conhecido como *internal covariate shift* e apresenta maior gravidade quanto mais profunda for a rede a ser treinada. Esse problema aumenta o tempo de treinamento porque implica na definição de uma taxa de aprendizagem pequena, além de propiciar a dissipação dos gradientes.

A normalização em lote (do inglês *batch normalization*) é um mecanismo proposto recentemente para resolver o problema acima, e que consiste em normalizar os dados fornecidos a cada camada oculta. A normalização é aplicada em cada mini-lote, para aumentar a eficiência durante a aplicação da transformação. Esse método tem um efeito regularizador, tendo substituído o uso de regularização L1 e L2 e em alguns casos tornando desnecessário o uso de dropout.

2.2.8.4 Arquitetura de uma Rede Convolutiva

Após todo o processo de convolução realizado nas camadas anteriores, em muitas estruturas de CNNs é usado antes de sua saída, pelo menos uma camada completamente preenchida com as ativações anteriores, como mostrado na Figura 2.12, nomeada de "Fully Connected". Nesta imagem é mostrada uma arquitetura de rede convolutiva para reconhecimento de dígitos de 0 a 9.

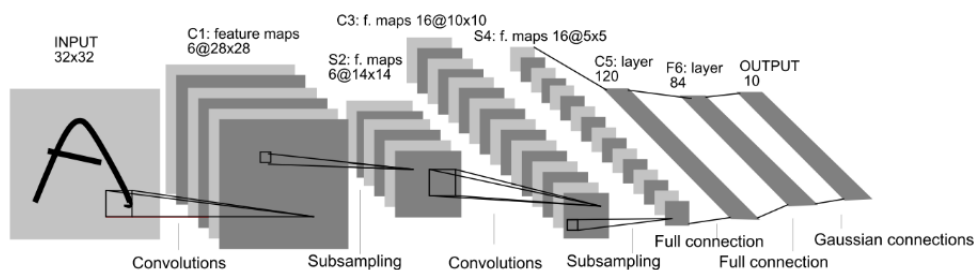


Figura 2.12: Arquitetura LeNet-5. [6]

As redes convolutivas são treinadas utilizando *backpropagation* através de um método de otimização conforme explicado na Seção 2.2.5. É necessário definir os hiperparâmetros

como número de filtros, tamanho do kernel, função de ativação, número de camadas entre outros. Isso pode ser feito através de métodos de validação cruzada como *10-fold* ou *leave-one-out*.

2.2.9 Redes Residuais

As Redes Residuais Também conhecidas como ResNets [7], essas redes se diferenciam das CNNs convencionais pois incorporam uma técnica para evitar a degradação de dados nas camadas mais profundas da rede e o desaparecimento do gradiente durante o Backpropagation. Uma ResNet possui camadas residuais que são compostas, cada uma, de duas camadas convolutivas e duas funções de ativação ReLU, conforme demonstra a Figura 2.13. A camada residual difere das demais camadas convolutivas pois sua saída é a soma da saída da segunda camada convolutiva com a entrada da camada residual. Essa estrutura possibilita que camadas mais profundas recebam diretamente dados das camadas mais superficiais, sem a necessidade de passar por camadas convolutivas intermediárias.

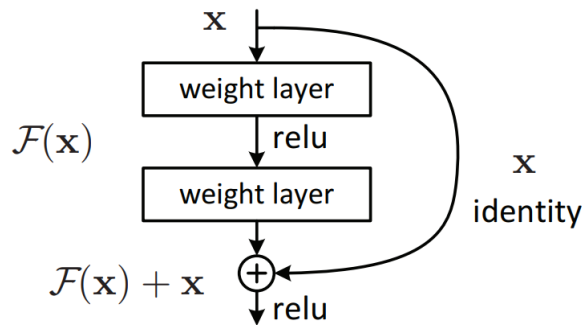


Figura 2.13: Bloco Residual. Fonte: [7]

2.3 Séries Temporais

Série temporal é uma sequência de observações de um fenômeno ao longo do tempo. Geralmente essas medições são feitas em um intervalo de tempo regular. A ordem das amostras é crucial, pois há uma dependência entre os dados e uma alteração da ordem pode modificar o significado dos dados. Uma série temporal pode ser definida como:

$$X(t) = (x_1, x_2, \dots, x_n) \quad (2.8)$$

onde x_n representa uma observação no instante t , n o número de observações e $X(t)$ a função que descreve a série temporal em termos de t . Caso a série seja constituída de uma observação em cada instante de tempo ela é chamada de *univariada*. Caso a série foi obtida por uma coleta simultânea de dois ou mais fenômenos ela é chamada de *multivariada*.

As séries temporais estão em diversas áreas do conhecimento como Economia (preços diários de ações, taxa mensal de desemprego, produção industrial), Medicina (eletrocardiograma, eletroencefalograma), Epidemiologia (número mensal de novos casos de meningite), Meteorologia (precipitação pluviométrica, temperatura diária, velocidade do vento).

2.3.1 Aplicações de Séries Temporais

A análise de séries temporais tem atraído muitos pesquisadores em aprendizado de máquina ao redor do mundo. As principais tarefas envolvendo séries temporais na qual se utiliza aprendizagem de máquina são as seguintes:

- *Classificação*: cada série temporal representa uma classe distinta de objetos. Dada uma série temporal, o objetivo é descobrir qual é a classe de objetos ela representa;
- *Agrupamento*: Dado um conjunto de séries temporais, o objetivo é encontrar uma estrutura natural que permita distribuir as séries em grupos;
- *Detecção de Motivos*: também conhecido como detecção de *motifs*; o objetivo é encontrar uma ou mais subsequências que aparecem frequentemente na séries;
- *Detecção de Anomalias*: encontrar subsequências ou séries que são inesperadas em algum contexto.

Este trabalho está inserido na tarefa de classificações de séries temporais.

2.4 Classificação de Séries Temporais

Capítulo 3

Sistema Proposto

Neste capítulo é abordado o sistema de detecção de f_0 proposto neste projeto, com a descrição da metodologia e dos materiais utilizados no desenvolvimento.

3.1 Visão geral

Capítulo 4

Experimentos

4.1 Métricas

4.2 Objetivos

4.3 Resultados Obtidos

Capítulo 5

Conclusão

Referências Bibliográficas

- [1] Towards Data Science. *Activations Functions: Neural Networks*. 2017. [Online; acessado em 18 de Dezembro, 2013]. Disponível em: <<https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>>.
- [2] SRIVASTAVA, N. et al. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, v. 15, p. 1929–1958, 2014. Disponível em: <<http://jmlr.org/papers/v15/srivastava14a.html>>.
- [3] neuroclusterbrain.com. Neuron model rf-psth. 2017. [Online; acessado em 08 de Janeiro, 2019]. Disponível em: <<http://neuroclusterbrain.com>>.
- [4] APHEX34. Input volume connected to a convolutional layer. 2015. [Online; acessado em 08 de Janeiro, 2019]. Disponível em: <https://commons.wikimedia.org/wiki/File:Conv_layer.png>.
- [5] KARPATY, A. *Illustration of spatial arrangement*. 2015. [Online; acessado em 17 de Janeiro, 2019]. Disponível em: <<https://github.com/cs231n/cs231n.github.io/blob/master/assets/cnn/stride.jpeg>>.
- [6] LECUN, Y. et al. Gradient-based learning applied to document recognition. In: *Proceedings of the IEEE*. [S.l.: s.n.], 1998. p. 2278–2324.
- [7] HE, K. et al. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. Disponível em: <<http://arxiv.org/abs/1512.03385>>.
- [8] MONARD, M.; BARANAUSKAS, J. *Conceitos sobre aprendizado de máquinas. Em Sistemas Inteligentes: Fundamentos e Aplicações*. 1^o. ed. [S.l.]: Editora Manole, 2003.

- [9] CHAPELLE, O.; SCHÖLKOPF, B.; ZIEN, A. *Semi-Supervised Learning*. 1^o. ed. [S.l.]: MIT Press, Cambridge, 2006. 12 – 13 p.
- [10] COVER, T.; HART, P. Nearest neighbor pattern classification. *Information Theory, IEEE Transaction on*, 1967.
- [11] ZHANG, G.; PATUWO, B. E.; HU, M. Y. Forecasting with artificial neural networks: The state of art. *International Journal of Forecasting*, 1998.
- [12] HAYKIN, S. S. *Neural networks and learning machines*. 3^o. ed. Upper Saddle River, United States of America: Prentice Hall.
- [13] FERNEDA, E. Redes neurais e sua aplicação em sistemas de recuperação de informação. *Ciência da Informação*, v. 35, n. 1, 2006. ISSN 1518-8353. Disponível em: <<http://revista.ibict.br/ciinf/article/view/1149>>.
- [14] MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 1943. ISSN 1522-9602. Disponível em: <<https://doi.org/10.1007/BF02478259>>.
- [15] LIPPMANN, R. An introduction to computing with neural nets. *IEEE ASSP Magazine*, v. 4, n. 2, 1987.
- [16] RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. *Nature*, Nature Publishing Group, v. 323, p. 533–, out. 1986. Disponível em: <<http://dx.doi.org/10.1038/323533a0>>.
- [17] GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>.
- [18] BENGIO, Y.; BOULANGER-LEWANDOWSKI, N.; PASCANU, R. Advances in optimizing recurrent networks. *CoRR*, abs/1212.0901, 2012. Disponível em: <<http://arxiv.org/abs/1212.0901>>.
- [19] DUCHI, J. C.; HAZAN, E.; SINGER, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, v. 12, p. 2121–2159, 07 2011.

-
- [20] KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. Disponível em: <<http://arxiv.org/abs/1412.6980>>.
- [21] CYBENKO, G. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems. 1989 Springer-Verlag New York Inc*, p. 303–314, 1989.
- [22] HAYKIN, S. S. *Neural networks and learning machines*. Third. Upper Saddle River, NJ: Pearson Education, 2009.
- [23] GUEDES, A. B. *Reconhecimento de Gestos usando Redes Convolucionadas*. 49 p. — UnB - Universidade de Brasília, Brasília, 2017.