

Universidade Federal do Pampa

Marcos Vinícius Treviso

## **Modelo para Trabalho de Conclusão de Curso**

Alegrete

2015



Marcos Vinícius Treviso

## **Modelo para Trabalho de Conclusão de Curso**

Projeto de Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Fábio Natanael Kepler

Alegrete

2015



Marcos Vinícius Treviso

## **Modelo para Trabalho de Conclusão de Curso**

Projeto de Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Projeto de Trabalho de Conclusão de Cursodefendido e aprovado em ..... de ..... de .....

Banca examinadora:

---

**Fábio Natanael Kepler**  
Orientador

---

**Professor**  
sigla da instituição

---

**Professor**  
sigla da instituição



*Este trabalho é dedicado às crianças adultas que,  
quando pequenas, sonharam em se tornar cientistas.*



# Agradecimentos

O esforço para chegar até aqui é grande, e certamente não conseguiria ter feito isso sozinho, por isso gostaria de agradecer meu orientador Fábio por me ajudar em assuntos variados durante a faculdade, com o qual aprendi muito.

Não é só de estudo que concluímos uma faculdade, depende muito da nossa força de vontade, e nesse quesito gostaria de agradecer muito a minha irmã Thaís, e especialmente meu pai Gilmar e minha mãe Loeiri por todos os conselhos e apoio.

*Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.*

*Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.*



*"Existem muitas hipóteses em ciência que estão erradas. Isso é perfeitamente aceitável, eles são a abertura para achar as que estão certas."*  
*(Carl Sagan - O Mundo Assombrado Pelos Demônios)*



# Resumo

Segundo a ABNT (2003, 3.1-3.2), o resumo deve ressaltar o objetivo, o método, os resultados e as conclusões do documento. A ordem e a extensão destes itens dependem do tipo de resumo (informativo ou indicativo) e do tratamento que cada item recebe no documento original. O resumo deve ser precedido da referência do documento, com exceção do resumo inserido no próprio documento. (...) As palavras-chave devem figurar logo abaixo do resumo, antecedidas da expressão Palavras-chave:, separadas entre si por ponto e finalizadas também por ponto.

**Palavras-chave:** Aprendizado de máquina. Processamento de linguagem natural. Part-of-speech tagging. Redes neurais. Word embeddings.



# Abstract

This is the english abstract.

**Key-words:** Machine Learning. Natural language processing. Part-of-speech tagging.  
Neural networks. Word embeddings.



# Listas de ilustrações

Figura 1 – Exemplo de classificação gramatical . . . . .	27
Figura 2 – Diagrama para resultado final . . . . .	31
Figura 3 – Demonstração de regressão . . . . .	32
Figura 4 – Demonstração de classificação . . . . .	33
Figura 5 – Função sigmoide . . . . .	35
Figura 6 – Representação dos casos da função de custo . . . . .	37
Figura 7 – Função de custo - $J(\theta_0, \theta_1)$ . . . . .	38
Figura 8 – Funcionamento do Gradiente Descendente . . . . .	40
Figura 9 – Exemplo de <i>underfitting</i> e <i>overfitting</i> . . . . .	41
Figura 10 – Exemplo de vetores de <i>features</i> de cinco dimensões representando palavras	45
Figura 11 – t-SNE: Visualização para <i>word embeddings</i> . . . . .	46
Figura 12 – Exemplo de palavras similares . . . . .	46
Figura 13 – Exemplos de representação de sensores no cérebro . . . . .	47
Figura 14 – Representação de um neurônio . . . . .	48
Figura 15 – Abstração matemática de um neurônio . . . . .	49
Figura 16 – Rede neural com uma camada oculta . . . . .	49
Figura 17 – Modelo da rede neural . . . . .	52
Figura 18 – Passos do Backpropagation . . . . .	55
Figura 19 – Sentido do aprendizado de máquina . . . . .	57
Figura 20 – Rede neural com múltiplas camadas . . . . .	58
Figura 21 – Funcionamento de aprendizagem profunda em Part-of-speech (POS) Tagging . . . . .	58
Figura 22 – Árvore de recursão de Fibonacci . . . . .	66
Figura 23 – Gráfico produzido diretamente no arquivo fonte . . . . .	66
Figura 24 – Outro gráfico feito em L <sup>A</sup> T <sub>E</sub> X . . . . .	67
Figura 25 – Variação dos resultados utilizando seleção por Janela Deslizante . . . . .	68
Figura 26 – Gráfico produzido em Excel e salvo como PDF . . . . .	69
Figura 27 – Exemplo de subfiguras . . . . .	69
Figura 28 – Mesmo exemplo de subfiguras, agora em escala . . . . .	70
Figura 29 – Exemplo de tamanhos de fonte . . . . .	93
Figura 30 – Tela do JabRef para uma versão inicial do arquivo bib deste documento	95



# **Lista de tabelas**

Tabela 1 – Nomenclaturas . . . . .	34
Tabela 2 – Dados dos <i>córpus</i> . . . . .	43
Tabela 3 – Notação utilizada para representação redes neurais . . . . .	50
Tabela 4 – Notação utilizada para aprendizado em redes neurais . . . . .	53
Tabela 5 – Comparativo entre técnicas e resultados encontrados na literatura para o problema de POS Tagging. . . . .	62
Tabela 6 – Níveis de investigação . . . . .	65
Tabela 7 – Exemplo de tabela com dados de arquivo. . . . .	65
Tabela 8 – Cronograma de Atividades Restantes . . . . .	77
Tabela 9 – Linhas de código para inserir figura . . . . .	94



# Lista de abreviaturas

**Fig.** Figura



# **Lista de siglas**

**DDV** Dentro do Vocabulário

**FDV** Fora do Vocabulário

**GloVe** Global Vectors for Word Representation

**HAL** Hyperspace Analogue to Language

**NLM** Neural Language Model

**PLN** Processamento de Linguagem Natural

**POS** Part-of-speech

**SG** Skip-Gram

**TCC** Trabalho de Conclusão de Curso



# Listas de símbolos

$(x, y)$  Um exemplo de treinamento

$(X^{(i)}, Y^{(i)})$  O i-ésimo exemplo de treinamento

$\alpha$  Taxa de aprendizagem

$\lambda$  Parâmetro de regularização

$\mathcal{T}$  Conjunto de trabalhos de TCC

$\theta$  Parâmetros a serem aprendidos

$A_i$  Área do  $i^{\text{esimo}}$  componente

$m$  Número de exemplos de treinamento

$X$  Dados de entrada ou *features*

$X^{(i)}$  O vetor coluna de todas as *features* no i-ésimo exemplo de treinamento

$X_j^{(i)}$  O valor da *feature*  $j$  no i-ésimo exemplo de treinamento

$Y$  Dados de saída



# Sumário

<b>1</b>	<b>INTRODUÇÃO</b>	<b>27</b>
<b>1.1</b>	<b>Objetivo</b>	<b>27</b>
<b>1.2</b>	<b>Estrutura do trabalho</b>	<b>28</b>
<b>2</b>	<b>O PROBLEMA</b>	<b>29</b>
<b>3</b>	<b>FUNDAMENTOS</b>	<b>31</b>
<b>3.1</b>	<b>Aprendizado de máquina</b>	<b>31</b>
3.1.1	Classificação	33
3.1.2	Função de hipótese	34
3.1.3	Função de custo	36
3.1.4	Minimização da função de custo	38
3.1.5	Classificação multiclasse	40
3.1.6	Regularização	40
<b>3.2</b>	<b>Córpus e seu conjunto de classes gramaticais</b>	<b>42</b>
<b>3.3</b>	<b>Representação das palavras</b>	<b>43</b>
<b>3.4</b>	<b>Redes neurais</b>	<b>47</b>
3.4.1	Neurocomputação	47
3.4.2	Representação do modelo	47
3.4.3	Representação vetorizada do modelo	50
3.4.4	Classificação multiclasse	52
3.4.5	Função de custo	53
3.4.6	Algoritmo <i>Backpropagation</i>	53
3.4.7	Treinamento	56
<b>3.5</b>	<b>Aprendizagem profunda</b>	<b>56</b>
<b>4</b>	<b>TRABALHOS RELACIONADOS</b>	<b>61</b>
<b>5</b>	<b>METODOLOGIA</b>	<b>63</b>
<b>5.1</b>	<b>Formatação</b>	<b>63</b>
<b>5.2</b>	<b>Codificação dos arquivos: UTF8</b>	<b>63</b>
<b>5.3</b>	<b>Citações</b>	<b>63</b>
5.3.1	Referências internas	64
<b>5.4</b>	<b>Tabelas</b>	<b>64</b>
<b>5.5</b>	<b>Figuras</b>	<b>65</b>
5.5.1	Sobre a indicação da fonte de uma tabela ou figura	67

5.6	Expressões matemáticas . . . . .	69
5.7	Enumerações: alíneas e subalíneas . . . . .	70
5.8	Espaçamento entre parágrafos e linhas . . . . .	71
5.9	Inclusão de outros arquivos . . . . .	72
5.10	Compilar o documento L <sup>A</sup> T <sub>E</sub> X . . . . .	72
6	<b>CONCLUSÃO</b> . . . . .	75
7	<b>CRONOGRAMA DE ATIVIDADES</b> . . . . .	77
	<b>Referências</b> . . . . .	79
	<b>APÊNDICES</b>	83
	<b>APÊNDICE A – DERIVADAS PARCIAIS DO GRADIENTE DESCENDENTE</b> . . . . .	85
	<b>APÊNDICE B –</b> . . . . .	87
	<b>ANEXOS</b>	89
	<b>ANEXO A – FUNDAMENTOS DE ÁLGEBRA LINEAR</b> . . . . .	91
	<b>ANEXO B – LATEX PARA PRINCIPIANTES</b> . . . . .	93
B.1	<b>Outras Dicas</b> . . . . .	94
	<b>Índice</b> . . . . .	97

# 1 Introdução

O ato de classificar uma palavra pertencente a um conjunto de textos em uma classe gramatical depende de sua estrutura morfo-sintática, esse ato é conhecido no campo de Processamento de Linguagem Natural ([PLN](#)) como [POS Tagging](#). A [Figura 1](#) ilustra esse processo.

O conjunto de textos denominados *córpus* são amplamente utilizados para esse processo, e é sobre eles que é feito o treinamento do modelo de reconhecimento de padrões para que seja possível classificar uma palavra a uma certa classe gramatical.

Um dos problemas dessa classificação é justamente a eficiência com a qual cada classe gramatical é atribuída para certa palavra, nesse quesito, há vários métodos já idealizados que conseguiram uma eficiência de cerca de 97%, tais métodos são citados em ([SANTOS; ZADROZNY, 2014](#); [COLLOBERT, 2011](#); [FONSECA; ROSA; ALUÍSIO, 2015](#)). [Fonseca, Rosa e Aluísio \(2015\)](#) afirmam ter conseguido o estado-da-arte atual para o português com 97,57% de acurácia.

Apesar de muitos desses métodos já serem utilizados em larga escala, em [PLN](#) estamos sempre buscando ganhar mais performance, já que os [POS](#) Taggers podem ser aplicados em uma grande variedade de aplicações como tradução automática e extração de informações de textos ([MANNING; SCHÜTZ, 1999](#)), ferramentas de auxílio à leitura e escrita ([MARQUIAFÁVEL, 2010](#)), entre outras.

[Figura 1: Exemplo de classificação gramatical](#)



Nosso trabalho consiste em classificar palavras de acordo com seu contexto, ou seja, é feita a análise em termos das unidades primitivas que a compõem, e uma vez que ela é cumprida, podemos aplicar o resultado em outras análises.

## 1.1 Objetivo

Este trabalho tem, por fim, propor um novo método de classificação de palavras em classes gramaticais e analisar sua eficiência em relação a trabalhos já publicados que

utilizam métodos já consolidados. Primordialmente, isso será feito para o escopo da língua portuguesa brasileira.

Para buscar uma boa eficiência será proposto um método original, que se baseia em classificar primeiramente palavras mais fáceis (*e.g verbos*), desse modo, espera-se deixar palavras ambíguas por último. Por exemplo, para a [Figura 1](#) deixariámos a palavra *canto* para o final, uma vez que não sabemos se isso se trata de um verbo ou de um substantivo. Além disso, o método proposto também terá como base a utilização de valores para cada classe gramatical de uma palavra, seguida por outras duas classes gramaticais das palavras anteriores.

Como já mencionado, o estado-da-arte atual tem atualmente cerca de 97% de acurácia, tentaremos ultrapassar esse limite aplicando novas técnicas de classificação e utilizando características significantes das palavras. É importante salientar que a acurácia da classificação não será a única medida levada em consideração, o tempo de processamento gasto no treinamento para cada *córpus* também será.

Para exemplificar brevemente, no final pretende-se que o modelo seja capaz de classificar corretamente as palavras analisando o seu contexto, e com isso outros analisadores conseguirão identificar o uso das palavras de acordo com seu significado, como mostrado abaixo na [Equação 1.1](#). Essa equação será explicada na [seção 3.3](#).

$$\text{homem} + \text{coroa} = \text{rei} \tag{1.1}$$

## 1.2 Estrutura do trabalho

A fim de proporcionar uma boa interpretação, este trabalho foi dividido nos seguintes capítulos: No [Capítulo 2](#) será descrito o problema de etiquetagem. No [Capítulo 3](#) serão mostrados os principais fundamentos necessários para entender o método proposto e seus conceitos relacionados; após passar os fundamentos, será apresentado no [Capítulo 4](#) os trabalhos relacionados que procuram resolver o problema de [POS Tagging](#) utilizando diferentes técnicas e abordagens. Depois, no [Capítulo 5](#) será detalhado aspectos da metodologia a ser aplicada e a explicação do método proposto juntamente com as técnicas utilizadas. Então, no ?? será feito um comparativo entre os resultados preliminares alcançados e uma breve discussão a respeito. E por fim, no [Capítulo 6](#) será apresentado as considerações finais.

## 2 O problema

Primeiramente, devemos informar que o problema a ser resolvido não é fácil, pois uma palavra pode possuir diferentes categorias gramaticais, e para solucioná-lo é preciso, na maioria das vezes, analisar todo o contexto no qual a palavra está associada. Além disso, como já mostrado anteriormente, há muita ambiguidade no português do Brasil, e essa ambiguidade dificulta a análise morfo-sintática, porque não é possível determinar a priori qual classe gramatical a palavra pertence. No exemplo anterior, da [Figura 1](#), não fica claro qual é o tipo de canto que está sendo referenciado, onde pode ser um verbo ou um substantivo (*o ato de cantar na primeira pessoa do singular do presente; um substantivo que se refere a um ângulo saliente; ou um substantivo que se refere a uma composição poética*). A ambiguidade pode muitas vezes ser resolvida se o contexto for analisado, como por exemplo:

"O rio da cidade."

A palavra *rio* pode ser classificada como substantivo ou como verbo, mas levando em consideração o sentido da frase, fica evidente que a classificação correta é substantivo.

Uma estratégia trivial seria utilizar um longo dicionário com uma função de mapeamento de um para um, onde a *chave* seria a palavra e o *valor* seria a classe gramatical. Infelizmente essa técnica é inviável com os recursos computacionais que temos atualmente, visto que o número de entradas seria estupendamente grande por ter todas as palavras possíveis de português, caso contrário, haveria o problema de ter uma palavra fora do vocabulário, e portanto ela não teria uma classe gramatical associada. Outro problema dessa estratégia é a ambiguidade, que faz com que uma palavra tenha mais que uma classe gramatical associada, e portanto não é possível mapear com indubitabilidade de que a classe associada é a correta sem antes de analisar o contexto.

Dito isso, o problema deste trabalho consiste em desenvolver um método para classificar palavras em suas respectivas classes gramaticais de modo eficiente. Para solucionar esse problema uma abordagem que está sendo amplamente utilizada é aprendizado de máquina, pois ela permite treinar um modelo que aprenda a classificar as palavras de acordo com o contexto em que está associada.

Em ordem de conseguir solucionar esse problema com eficiência, é necessário escolher um bom método computacional. Este trabalho se baseará na utilização de um método de aprendizado de máquina conhecido como redes neurais profundas (*deep-learning*), para que seja possível treinar modelos capazes de realizar a classificação. Utilizamos redes neu-

rais pois elas oferecem um jeito alternativo de realizar aprendizado de máquina quando temos hipóteses complexas com muitas características diferentes.

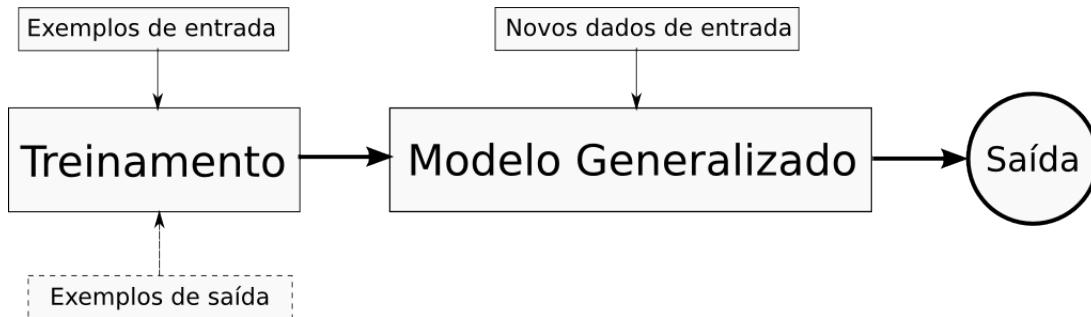
# 3 Fundamentos

Devido aos impasses demonstrados no [Capítulo 2](#), este trabalho é baseado em conceitos de aprendizado de máquina, que dedica-se na elaboração de algoritmos e técnicas que permitem um computador a aprender padrões.

## 3.1 Aprendizado de máquina

Para realizar a aprendizagem é necessário, a princípio, dados de entrada que servem como exemplo para nosso modelo, com esses dados é possível treinar o modelo para que ele possa então aprender com base nesses exemplos. Depois de realizar o treinamento, é possível generalizar sobre outros dados ainda não testados e gerar uma resposta apropriada como saída. O diagrama da [Figura 2](#) abaixo mostra os passos para obter o resultado final.

Figura 2: Diagrama para resultado final



O aprendizado de máquina pode ser dividido em duas abordagens: O **aprendizado supervisionado**, onde é dado um conjunto de dados de entrada e já se sabe como a saída deve parecer, tendo a ideia de que há uma relação entre a entrada e a saída; O **aprendizado não supervisionado**, que nos permite abordar problemas com pouca ou até nenhuma ideia de como os resultados devem parecer, nesse caso a caixa de exemplos de saída pode não existir.

Neste trabalho a abordagem utilizada vai ser o aprendizado supervisionado, pois já temos uma classe gramatical apropriada para cada palavra em seu contexto.

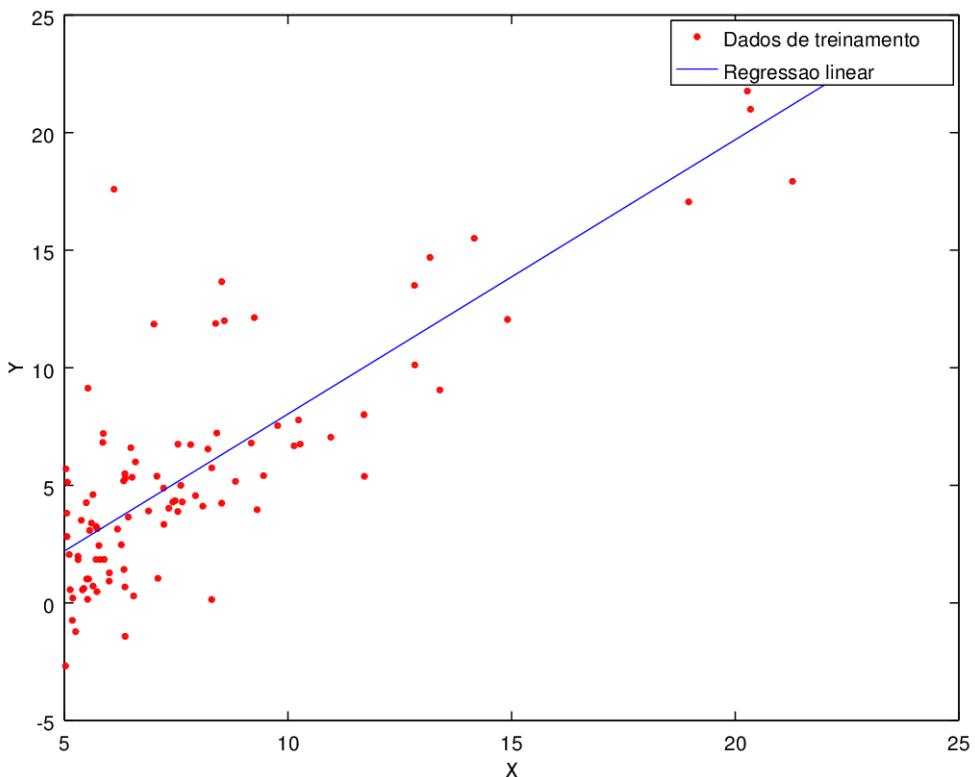
O aprendizado supervisionado permite dividir os problemas em duas categorias distintas:

- **Régressão:** Tenta-se prever resultados com uma saída contínua, significando que deseja-se mapear variáveis de entrada em alguma função contínua. A [Figura 3](#)

demonstra esse processo. Nessa demonstração, os círculos em vermelho são dados de exemplo, já a reta azul representa a predição feita pela regressão.

- **Classificação:** Tenta-se prever resultados em uma saída discreta, ou seja, deseja-se mapear variáveis de entrada em categoriais discretas. A [Figura 4](#) demonstra esse processo. Essa demonstração tem duas classes diferentes, as representadas pelos círculos amarelos e pelas cruzes vermelhas, já o contorno em verde representa o resultado da classificação sobre esses dados de exemplo.

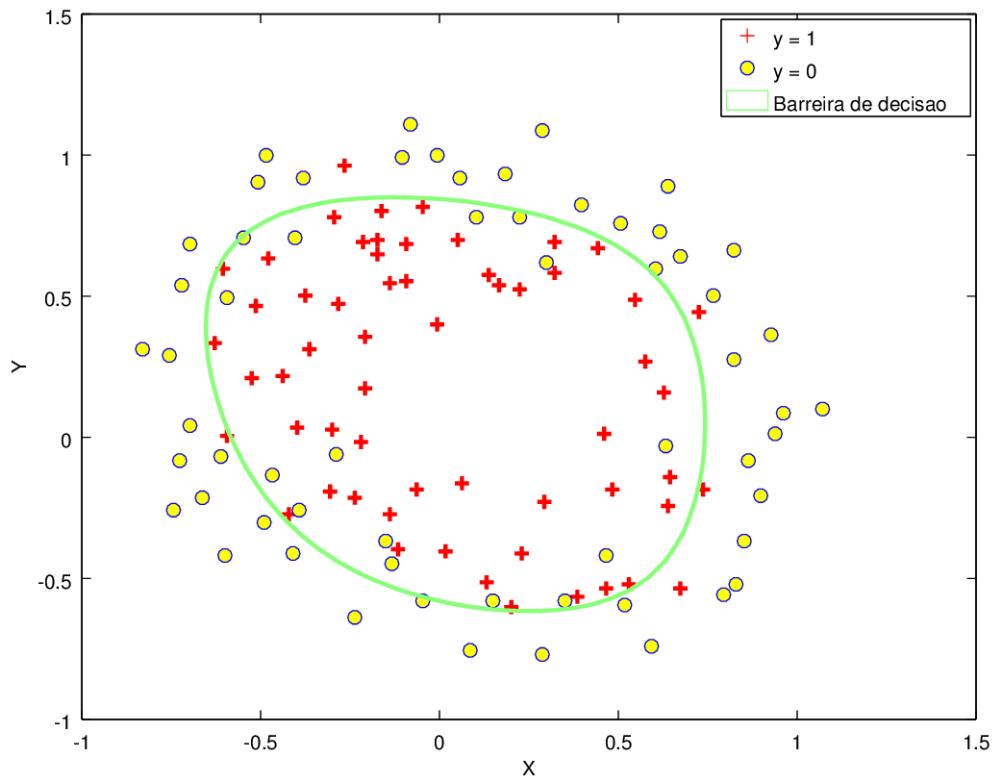
Figura 3: Demonstração de regressão



Fonte: [Ng \(2015\)](#)

Logo adiante será explicado os fundamentos de classificação e uma técnica para resolver esse problema chamada de regressão logística ([JR; LEMESHOW, 2004](#)). E em virtude de apresentar conceitos fundamentais há serem aplicados posteriormente nas redes neurais profundas, será falado um pouco mais sobre regressão logística, e também sobre propriedades importantes que serão referenciadas durante todo o trabalho.

Figura 4: Demonstração de classificação

Fonte: [Ng \(2015\)](#)

### 3.1.1 Classificação

Em um problema de classificação, nossa saída  $Y$  vai ser um vetor com valores sendo apenas zero ou um.

$$Y \in \{0, 1\} \quad (3.1)$$

A equação 3.1 está tratando apenas duas classes. Sendo assim, esse problema é chamado de classificação binária. Para resolver esse problema, um método que pode ser utilizado é regressão logística.

A fim de simplificar o uso das variáveis, faremos o uso de uma notação que é normalmente utilizada em textos de aprendizado de máquina, ela pode ser vista na [Tabela 1](#).

Tabela 1: Nomenclaturas

---

$X$	dados de entrada ou <i>features</i>
$Y$	dados de saída
$X_j^{(i)}$	o valor da <i>feature</i> $j$ no $i$ -ésimo exemplo de treinamento
$X^{(i)}$	o vetor coluna de todas as <i>features</i> no $i$ -ésimo exemplo de treinamento
$m$	número de exemplos de treinamento
$n$	$ X^{(i)} $ , o número de <i>features</i>
$(x, y)$	um exemplo de treinamento
$(X^{(i)}, Y^{(i)})$	o $i$ -ésimo exemplo de treinamento
$\theta$	parâmetros a serem aprendidos

---

Notação utilizada para classificação

### 3.1.2 Função de hipótese

A função de hipótese descreve a confiança dos dados de entrada para os dados de saída, para isso é atribuído valores para os parâmetros  $\theta$ s. Essa relação está definida na [Equação 3.2](#). Observe que ela tem apenas dois parâmetros. Na realidade uma hipótese pode ter vários parâmetros relacionados com os dados de entrada  $X$ , se tiver  $n$  dados de entrada, então haverá  $n+1$  parâmetros. A forma geral da função de hipótese está mostrada na [Equação 3.3](#).

$$h_{\theta}(x) = \theta_0 + \theta_1 x \quad (3.2)$$

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n \quad (3.3)$$

Uma maneira de simplificar a função de hipótese é trabalhar com a definição de multiplicação de matrizes. Então a função de hipótese pode ser representada de uma forma vetorializada, como mostrado na [Equação 3.4](#). Para maiores informações a respeito de matrizes e vetores, consultar o Anexo A, que traz informações básicas de álgebra linear.

$$h_{\theta}(x) = [\theta_0 \ \theta_1 \ \dots \ \theta_n] \times \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} = \theta^T X \quad (3.4)$$

Para que seja possível fazer operações de matrizes com  $\theta$  e  $X$ , será setado  $X_0^{(i)} = 1$  para todos os valores de  $i$ . Isso faz com que os vetores  $\theta$  e  $X^{(i)}$  combinem um com o outro elementarmente.

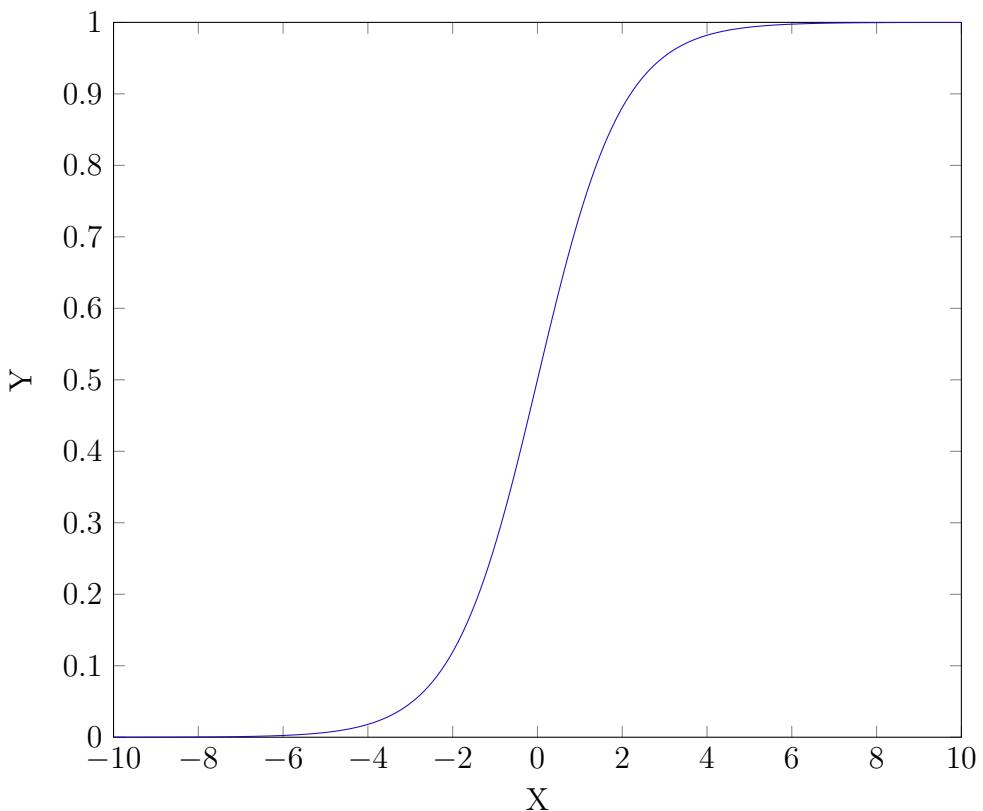
Uma função de hipótese tem o objetivo de mapear funções aleatórias dentro do intervalo de saída  $Y$ . Portanto, nossa função de hipótese deve satisfazer a [Equação 3.5](#).

$$0 \leq h_\theta(x) \leq 1 \quad (3.5)$$

Para fazer com que uma entrada contínua fique nesse intervalo, é necessário utilizar uma função linear que faça esse mapeamento de modo linear, e para isso mudamos a forma da função de hipótese. A nova forma usa a função sigmoide, também conhecida como função logística.

$$\begin{aligned} h_\theta(x) &= g(\theta^T X) \\ z &= \theta^T X \\ g(z) &= \frac{1}{1 + e^{-z}} \end{aligned} \quad (3.6)$$

Figura 5: Função sigmoide



A função da [Equação 3.6](#) representada na [Figura 5](#) mapeia qualquer número no intervalo  $[0, 1]$ , tornando isso útil para transformar qualquer valor arbitrário em uma função mais ideal para problemas de classificação.

Para calcular a função de hipótese, podemos colocar o  $\theta^T X$  na função logística. Isso nos dará a probabilidade de que a saída é 1.

$$h_\theta(x) = P(y = 1|X; \theta) = 1 - P(y = 0|X; \theta)$$

Ou seja, a probabilidade de nossa predição ser 1 é o oposto da probabilidade de ser 0. Sendo assim, a soma das probabilidades para  $y = 0$  e  $y = 1$  deve ser 1.

Para ter uma classificação discreta podemos traduzir a saída da função de hipótese de acordo com a [Equação 3.7](#):

$$\begin{aligned} h_\theta(x) \geq 0.5 &\Rightarrow y = 1 \\ h_\theta(x) < 0.5 &\Rightarrow y = 0 \end{aligned} \tag{3.7}$$

Então, se a entrada é  $\theta^T X$ , isso significa que:

$$h_\theta(x) = g(\theta^T X) \geq 0.5 \quad \text{quando} \quad \theta^T X \geq 0$$

A partir disso, podemos dizer que:

$$\begin{aligned} \theta^T X \geq 0 &\Rightarrow y = 1 \\ \theta^T X < 0 &\Rightarrow y = 0 \end{aligned}$$

O **limite de decisão** (ou barreira de decisão) é a linha que separa a área quando  $y = 0$  e quando  $y = 1$ . Isso é criado pela função de hipótese. Uma observação importante é que a função de hipótese não precisa ser linear, pode ser até mesmo um círculo ou ter qualquer forma para ajustar os dados, como mostrado na [Figura 4](#). Isto é, uma função de hipótese pode ter várias *features*. Novas *features* podem ser criadas ao combinar as *features* já existentes e modificando-as. Por exemplo, se houvesse dois dados de entrada  $x_1$  e  $x_2$ , seria possível criar a função de hipótese:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2 + \theta_6 x_1^2 x_2^2 + \dots$$

Todavia, essa abordagem pode gerar problemas de desempenho. Algumas *features* podem também acabar se tornando obsoletas e não ajudar em nada no resultado final, tornando seu cálculo inútil. Mas o principal problema dessa abordagem é o *overfitting*, que faz com que a função de hipótese interpole os dados, esse problema será explicado na [subseção 3.1.6](#).

### 3.1.3 Função de custo

Para medir a precisão da função de hipótese é necessário uma função que diga o quão precisa aquela hipótese é, tal função é denominada de função de custo. Com ela, é

feita uma média de todos os resultados das hipóteses com a entrada  $X$  comparada com o valor objetivo  $Y$ . Ela pode ser expressa como na Equação 3.8.

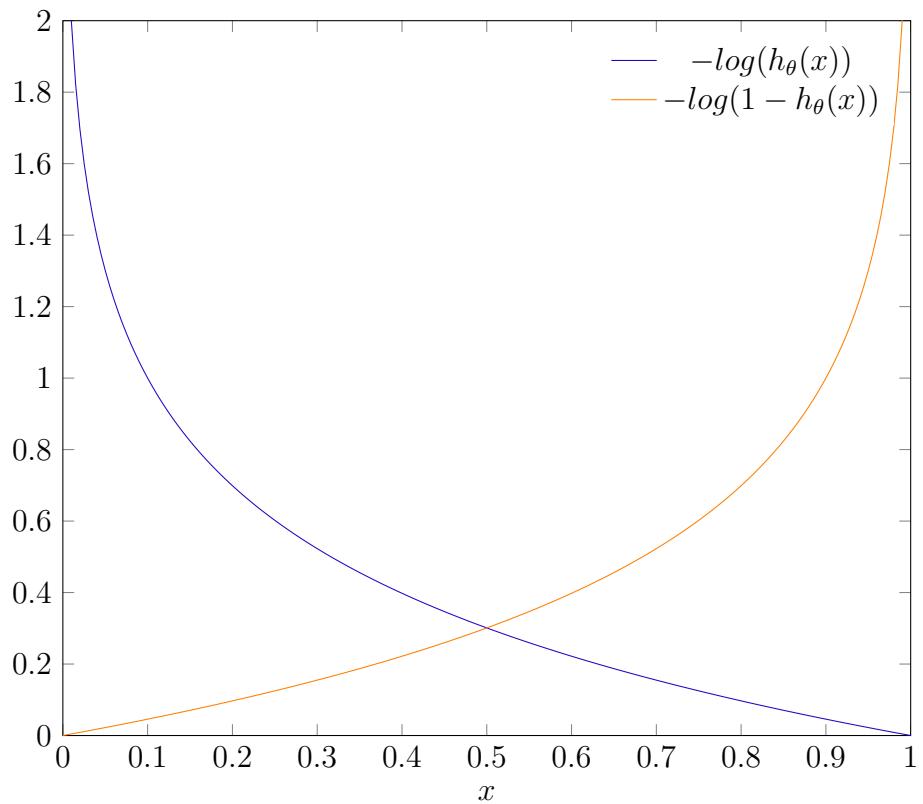
$$J(\theta) = \frac{1}{m} \sum_{i=1}^m Custo(h_\theta(x^{(i)}, y^{(i)})) \quad (3.8)$$

Onde,

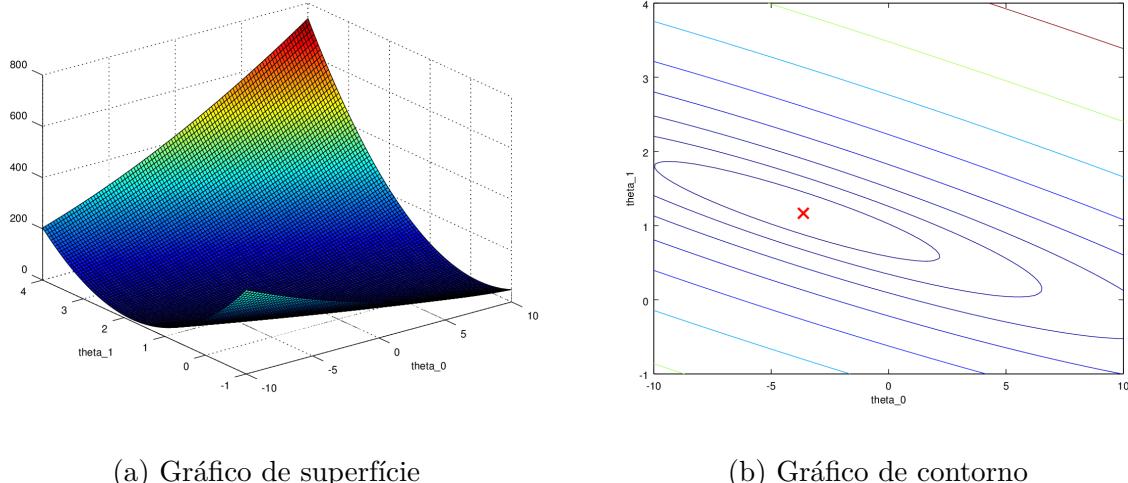
$$Custo(h_\theta(x^{(i)}, y^{(i)}) = \begin{cases} -\log(h_\theta(x)) & \text{se } y = 1 \\ -\log(1 - h_\theta(x)) & \text{se } y = 0 \end{cases} \quad (3.9)$$

A Equação 3.9 captura a intuição de que se  $h_\theta(x) = 0$ , mas  $y = 1$ , então o algoritmo de aprendizado será penalizado por um custo muito alto. A Figura 6 demonstra os dois respectivos casos dessa equação.

Figura 6: Representação dos casos da função de custo



O quanto mais a função de hipótese está longe de  $y$ , maior é a saída da função de custo. Se a função de hipótese é igual a  $y$ , então o custo é 0.

Figura 7: Função de custo -  $J(\theta_0, \theta_1)$ 

Fonte: Ng (2015)

Podemos simplificar a [Equação 3.9](#) com dois casos condicionais em apenas um caso:

$$Custo(h_\theta(x), y) = -y \log(h_\theta(x)) - (1 - y) \log(1 - h_\theta(x)) \quad (3.10)$$

Nota-se que quando  $y$  é igual a 1, o segundo termo será 0 e não afetará o resultado. E quando  $y$  é igual a 0, o primeiro termo será 0 e não afetará o resultado. Aplicando a [Equação 3.10](#) na função de custo da equação [Equação 3.8](#), podemos reescrevê-la como a equação [Equação 3.11](#) ou na versão vetorizada da [Equação 3.12](#).

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m \left( y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right) \quad (3.11)$$

$$J(\theta) = -\frac{1}{m} \left( \log(g(X\theta))^T Y + \log(1 - g(X\theta))^T (1 - Y) \right) \quad (3.12)$$

O objetivo da regressão logística é minimizar a função de custo em relação aos parâmetros  $\theta$ s. Usando dois parâmetros é possível visualizar uma função de custo através do número de iterações, isso pode ser observado no exemplo da [Figura 7\(a\)](#) e da [Figura 7\(b\)](#).

### 3.1.4 Minimização da função de custo

Para essa tarefa é necessário a aplicação de um algoritmo que pega a função de custo e tente minimizá-la, e por fim retorne os valores dos parâmetros  $\theta$ s aprendidos. Com isso, estaremos melhorando nossa função de hipótese.

Um dos algoritmos mais utilizados para essa tarefa é o **Gradiente Descendente** (MICHALSKI; CARBONELL; MITCHELL, 2013), seu funcionamento baseia-se em seguir a derivada da função de custo em relação aos parâmetros  $\theta$ s alternadamente, com isso a encosta da tangente dará a direção para seguir adiante. Ademais, é aplicada uma taxa de aprendizagem  $\alpha$  a cada iteração, para tentar controlar a convergência.

Seu funcionamento é descrito pelo algoritmo 3.13.

*repita até convergir {*

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \quad (3.13)$$

*}*

Ao trabalhar sobre a derivada parcial da função de custo, podemos chegar na Equação 3.14 e adiante na versão vetorializada na Equação 3.15. A explicação de como trabalhar sobre a derivada pode ser encontrado no Apêndice A.

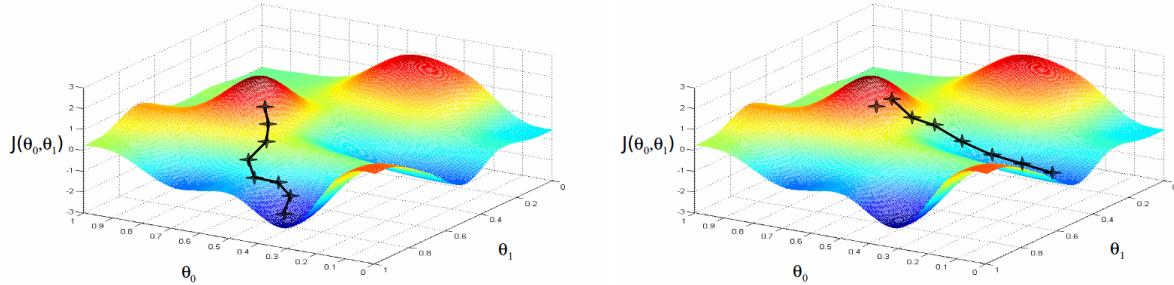
$$\theta_j := \theta_j - \alpha \sum_{i=1}^m \left( h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)} \quad (3.14)$$

$$\theta := \theta - \frac{\alpha}{m} X^T (g(X\theta) - Y) \quad (3.15)$$

Na Figura 8(a) é possível visualizar esse funcionamento sobre os parâmetros. Observe que nesse caso o algoritmo consegue convergir para o mínimo global, porém em alguns casos (dependendo de nossa taxa de aprendizagem  $\alpha$ ) pode-se convergir para um mínimo local como mostrado na Figura 8(b). Basicamente se  $\alpha$  for um valor muito alto, os passos para o mínimo ótimo serão grandes e portanto o risco de divergência é maior, caso seja muito baixo, os passos para o mínimo ótimo serão pequenos e portanto vai demorar muito para convergir. Em (NG, 2015), é dito para testar manualmente esse valor, começando com 0,001 e subindo multiplicando por 10 esse valor até atingir uma divergência no resultado. Sendo assim possível analisar o melhor valor da taxa de aprendizagem e selecioná-la para o treinamento definitivo.

O Gradiente Descendente é apenas um dos vários algoritmos que existem para minimizar a função de custo. De acordo com (NG, 2015), há alternativas mais sofisticadas como o Gradiente Conjugado, BFGS e L-BFGS, onde além de serem mais rápidos que o Gradiente Descendente, eles não precisam selecionar manualmente o valor de  $\alpha$ . Mas, também é sugerido que não devemos tentar codificar esses algoritmos, visto que são mais complexos e requerem um bom conhecimento de cálculo numérico.

Figura 8: Funcionamento do Gradiente Descendente



Fonte: Ng (2015)

### 3.1.5 Classificação multiclasse

O POS Tagging é um problema de classificação multiclasse, onde deve-se etiquetar uma palavra em uma de várias categorias gramaticais possíveis. É possível fazer isso ao expandir nossa definição para que a saída seja  $y = \{0, 1, \dots, n\}$ . Nesse caso, é dividido o problema em  $n + 1$  problemas de classificação binária e em cada um será feito a predizagem da probabilidade de que  $y$  é membro de uma dessas classes.

$$\begin{aligned} h_{\theta}^{(0)}(X) &= P(y = 0|X; \theta) \\ h_{\theta}^{(1)}(X) &= P(y = 1|X; \theta) \\ &\vdots \\ h_{\theta}^{(n)}(X) &= P(y = n|X; \theta) \end{aligned}$$

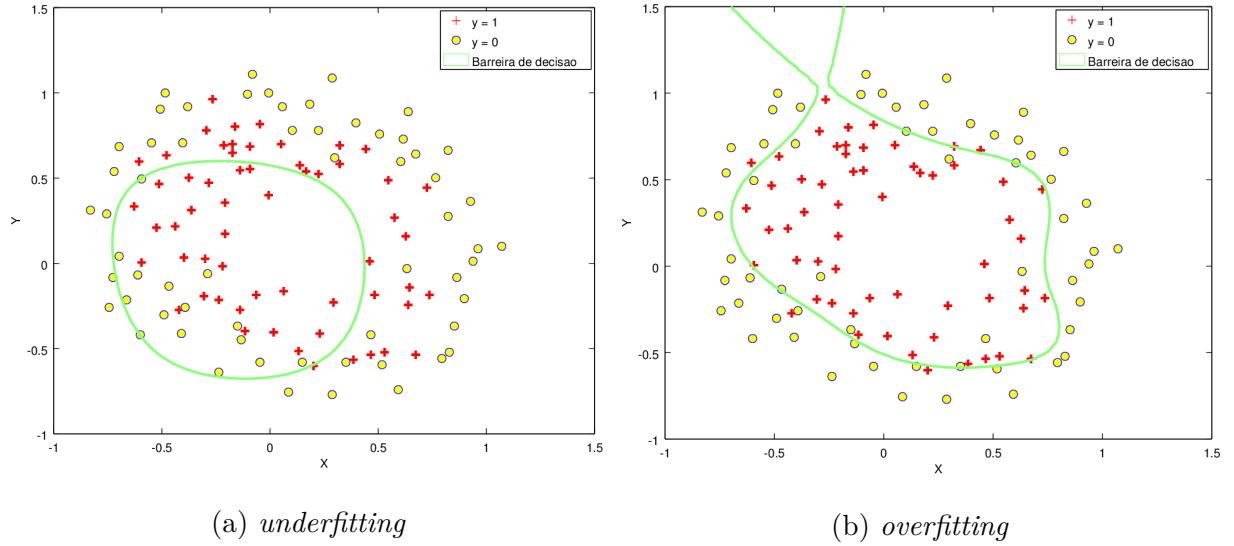
E então escolhe-se a classe com o valor de hipótese mais alto.

$$\text{prediction} = \max_i(h_{\theta}^{(i)}(X))$$

### 3.1.6 Regularização

Como já mencionado, regularização é um conceito importante designado para resolver o problema de *overfitting*.

Quando se trabalha com abordagens de aprendizagem supervisionada, tem dois problemas que podem ocorrer dependendo das *features* escolhidas. O ***underfitting*** ocorre quando a forma da função de hipótese mapeia mal à tendência dos dados. Isso é causado por uma função que é muito simples ou que usa pouquíssimas *features*. Em contrapartida, ***overfitting*** é causado por uma função de hipótese que encaixa os dados avaliados mas não generaliza bem para classificar novos dados. É usualmente causado por uma função complexa que cria muitas curvas e ângulos desnecessários não relacionados aos dados.

Figura 9: Exemplo de *underfitting* e *overfitting*

Fonte: Ng (2015)

Uma boa classificação para o conjunto de dados mostrados na Figura 9 é o exemplo mostrado na Figura 4.

Segundo (NG, 2015), há duas opções principais para resolver o problema de *overfitting*:

- a) Reduzir o número de *features*:
  - Manualmente selecionar quais *features* usar;
  - Usar um algoritmo de seleção de modelo.
- b) Regularização:
  - Manter todos as *features*, mas reduzir os parâmetros  $\theta_j$ .

Em geral, regularização funciona bem quando há bastante features levemente úteis. Quando há *overfitting* da função de hipótese, é possível reduzir a influência de alguns termos da função aumentando os seus custos. Por exemplo, caso quiséssemos eliminar a influência de  $\theta_3x^3$  e  $\theta_4x^4$  da Equação 3.16, poderíamos mudar a forma da função de custo. Com isso, a função de hipótese não é alterada e as *features* não são eliminadas, como mostrado na Equação 3.17.

$$h_{\theta}(x) = \theta_0 + \theta_1x^1 + \theta_2x^2 + \theta_3x^3 + \theta_4x^4 \quad (3.16)$$

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m \left( y^{(i)} \log(h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right) + \underbrace{1000\theta_3^2 + 1000\theta_4^2}_{\text{termo de regularização}} \quad (3.17)$$

Com isso, adicionamos dois termos extras no fim da função de custo para inflar o custo de  $\theta_3$  e  $\theta_4$ . Agora, para a função de custo chegar a zero, nós teremos que reduzir os valores de  $\theta_3$  e  $\theta_4$  para próximos a zero. Isso vai reduzir substancialmente o peso das *features* cuja influência queremos eliminar.

É possível regularizar todos os parâmetros em um único somatório conforme mostrado na [Equação 3.18](#). Isso pode ser feito com a inclusão do parâmetro de regularização  $\lambda$ , que determina o quanto os custos dos parâmetros  $\theta$ s serão inflados.

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m \left( y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right) + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2 \quad (3.18)$$

Observe que o índice  $j$  do segundo somatório começa em 1, isso significa que não estamos regularizando o  $\theta_0$ .

Como foi regularizado a função de custo, há de se regularizar o algoritmo que realiza sua minimização também. Quando é trabalhado com o Gradiente Descendente isso pode ser feito como no algoritmo [3.19](#). Como na função de custo, não queremos regularizar o  $\theta_0$  e então precisamos separá-lo em um outro passo.

*repita até convergir {*

$$\begin{aligned} \theta_j &:= \theta_j - \frac{\alpha}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)} && \text{se } j = 0 \\ \theta_j &:= \theta_j - \alpha \left[ \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)} + \frac{\lambda}{m} \theta_j \right] && \text{se } j > 0 \end{aligned} \quad (3.19)$$

*}*

Com isso a regressão logística pode ser aplicada normalmente, pois a regularização vai cuidar do *overfitting*. Entretanto, para casos mais complexos, que envolve a criação de muitas *features* como no caso de [POS Tagging](#), há uma melhor abordagem para realizar a etiquetagem. Essa abordagem é conhecida como redes neurais e ela será discutida na [seção 3.4](#).

## 3.2 Córpus e seu conjunto de classes gramaticais

Córpus são uma coleção de textos colocados como um só, eles são os exemplos de entrada para nosso treinamento. A qualidade e o tamanho desses córpus são importantes para que o etiquetador possa generalizar sentenças ainda não vistas. Eles contém também um conjunto de classes gramaticais associadas a cada palavra.

Esses conjuntos podem se diferenciar na sua granularidade, como por exemplo, podem ter diferentes classes gramaticais para nomes no plural e no singular, ou agrupar

eles em uma única classe (FONSECA; ROSA; ALUÍSIO, 2015). No inglês, há vários conjunto de classes gramaticais de qualidade que são amplamente utilizados, são o Penn Treebank tagset (THE... , 2014), CLAWS5 e CLAWS7.

Já no português, os córpuses estão evoluindo com o tempo, embora alguns erros são encontrados neles (FONSECA; ROSA, 2013), eles ainda são a melhor opção devido a seu tamanho e qualidade. Os principais e mais utilizados para essa tarefa são o Mac-Morpho (ALUÍSIO et al., 2003) com cerca de um milhão de palavras, que retrata artigos publicados na Folha de São Paulo em 1994. O Tycho Brahe (TEMPONI, 2004) que também tem cerca de um milhão de palavras e retrata assuntos históricos de 66 textos diferentes. Temos também o Bosque (AFONSO et al., 2002), que contém cerca de 185 mil palavras. Além desses, (FONSECA; ROSA; ALUÍSIO, 2015) apresenta uma versão revisada do Mac-Morpho, com classes gramaticais novas e junção de outras. Dados referentes a cada córpus podem ser encontrados na Tabela 2.

Tabela 2: Dados dos *córpus*

Córpus	Sentenças	Palavras	Classes gramaticais
Mac-Morpho original	53,374	1,221,465	41
Mac-Morpho revisado	49,932	945,958	26
Tycho Brahe	40,932	1,035,592	265

Infelizmente esses córpuses não podem ser combinados em um só, já que eles se diferenciam no conjunto de classes gramaticais e também no seu uso associado. Uma possível alternativa para essa combinação, seria o uso de uma aprendizagem não supervisionada, aplicando técnicas de clusterização.

A aprendizagem baseada em córpus tem mostrado uma estratégia atrativa, já que pode ser usado recursos criados com boa eficiência. Devido a isso, nesse trabalho vamos utilizar três córpuses: o Mac-Morpho original; Mac-Morpho revisado e o Tycho Brahe. O conjunto de etiquetas para esses córpuses podem ser encontrados, respectivamente, em (ALUÍSIO et al., 2003), (FONSECA; ROSA; ALUÍSIO, 2015) e (TEMPONI, 2004).

### 3.3 Representação das palavras

Palavras <sup>1</sup> podem ser representadas de várias maneiras em um modelo de aprendizagem, podendo ser até mesmo feito o uso real da palavra como uma *feature*.

No entanto, a melhor abordagem e a mais indicada atualmente é a utilização de *word embeddings*, elas são representação de palavras como vetores reais valorados em um espaço multidimensional (TURIAN; RATINOV; BENGIO, 2010). Elas podem ser geradas de maneiras diferentes dependendo da técnica utilizada, clássicas abordagens baseiam-se

<sup>1</sup> Uma palavra pode ser qualquer conjunto de caracteres, inclusive pontuações, números, etc.

na frequência e co-ocorrência das palavras. Mas normalmente o processo geração é feito através de redes neurais, onde é possível capturar informações sintáticas e semânticas sobre as palavras.

*Word embeddings* conseguem mapear palavras em um espaço relativamente pequeno (usando algumas centenas de dimensões ou até menos) e capturam similaridades entre as palavras (o tipo de similaridade varia com o método usado para gerá-las) de acordo com sua distância euclidiana, *cosine*, Jaccard, etc. Quando falamos que elas são similares, significa que elas tendem a serem usadas no mesmo contexto e usualmente pertencem a mesma classe gramatical. Além disso, palavras não vistas nos dados de treinamento não são completamente desconhecidas, pois elas tem uma representação vetorial. Por isso, espera-se que o impacto de palavras Fora do Vocabulário (**FDV**) seja menor (**FONSECA; ROSA; ALUÍSIO, 2015**).

As *word embeddings* podem então ser consideradas como novas *features* obtidas a partir das originais. Essa técnica tem interessado cada vez mais pesquisadores na área de **PLN**, e com isso novos processos de geração de *word embeddings* tem surgido.

(**COLLOBERT et al., 2011**) usa uma modelo de rede neural ( Neural Language Model (**NLM**), do inglês) para inicializar suas representações de palavras, com isso evita-se a tarefa de *engenharia de features*. Esse modelo é baseado na extração dos parâmetros aprendidos em uma rede neural treinada por *backpropagation*, onde apos o treinamento, será gerada uma tabela de busca com a palavra original com chave e o vetor de *features* como valor. Além disso, é mostrado a eficiência desse método em aplicações de análise sintática e semântica.

O número de dimensões dos vetores podem variar. Em geral, quanto mais dimensões há, melhores representações podem ser alcançadas, mas se a dimensão for muito grande, o processamento pode ser demorado.

(**COLLOBERT et al., 2011**) ainda nos mostra como extender a ideia de representação vetorial para incorporar *features* discretas. Um exemplo disse é a presença de capitalização, que traz informações importantes da palavra. Esse processo pode ser feito ao criar vetores correspondentes a *feature* criada. A [Figura 10](#) ilustra esse processo.

Outro método utilizado para a criação dos vetores é o Hyperspace Analogue to Language (**HAL**) (**LUND; BURGESS, 1996**). Ele é baseado na contagem de ocorrência de palavras próximas umas das outras para então obter uma grande matriz de contagem. Após finalizar a contagem, é analisada a distância euclidiana contabilizada entre as palavras do vocabulário e uma palavra alvo. Essas distâncias são então ordenadas, e os vizinhos com menores distâncias são examinados.

Além dessas, outra estratégia para gerar vetores é a modelação Skip-Gram (**SG**), que tem como objetivo minimizar a complexidade computacional na geração das *word*

Figura 10: Exemplo de vetores de *features* de cinco dimensões representando palavras

<b>Vetores de features para palavras no vocabulário</b>	
:	:
leal:	0.41   0.54   0.49
leão:	0.19   0.33   0.01
lenha:	0.90   0.57   0.16
:	:
<b>Vetores de features para capitalização de palavras</b>	
Todas minúsculas:	0.04   0.37
Todas maiúsculas:	0.98   0.20
Primeira maiúscula:	0.42   0.24
Outras combinações:	0.11   0.48
Nenhuma:	0.81   0.89
<b>Representação final das palavras no modelo</b>	
leal:	0.41   0.54   0.49   0.04   0.37
leão:	0.19   0.33   0.01   0.04   0.37
Leão:	0.19   0.33   0.01   0.42   0.24
LEÃO:	0.19   0.33   0.01   0.98   0.20

*embeddings*. Essa estratégia consiste em prever palavras próximas de uma dada palavra. Isso é feito usando uma palavra como entrada para o um classificador de complexidade logarítmica com uma camada de projeção contínua. A previsão é feita com um conjunto de palavras predecessoras e sucessoras, porém quando maior for esse conjunto, maior será a complexidade computacional (MIKOLOV et al., 2013). Já que o classificador não tem uma camada oculta na sua arquitetura, ele é consideravelmente mais rápido que um modelo de rede neural. Essa estratégia é utilizada pela ferramenta *word2vec*, que contém a contribuição do próprio Mikolov et al. (2013).

A estratégia mais recente e indicada por Socher (2015) é a utilização de Global Vectors for Word Representation (*GloVe*) idealizados em (LUO; XU, 2015). Essa estratégia consiste em modelar as palavras usando representações compartilhadas entre as palavras alvo e seu contexto. Essa estratégia é atualmente o estado da arte para a representação de palavras em vetores de *features* (PENNINGTON; SOCHER; MANNING, 2014).

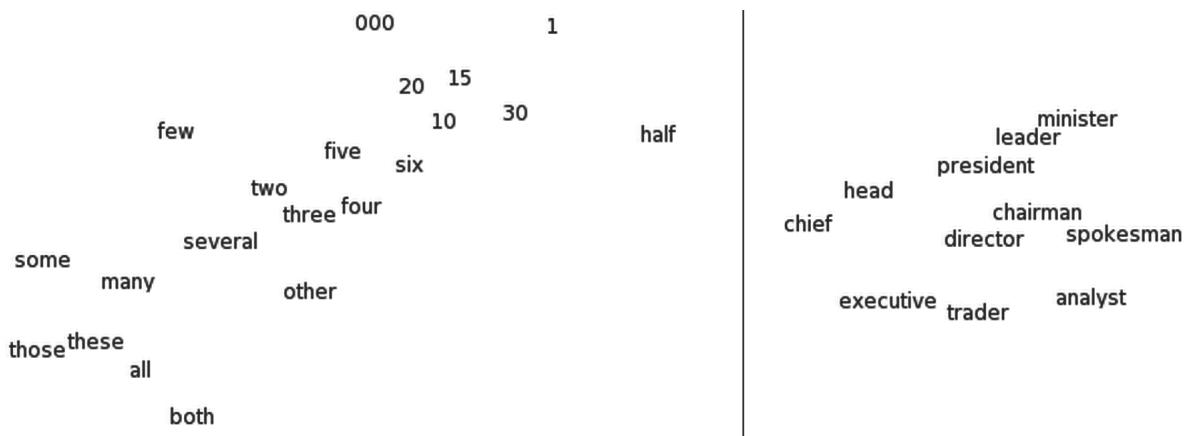
Portanto, para a criação de um vetor de *features*, temos a transformação  $W$  para uma palavra *word*:

$$W : \text{word} \rightarrow \mathbb{R}^n \quad (3.20)$$

Como já mencionado, a grande vantagem em se utilizar *word embeddings* é o fato

dessa representação conseguir capturar informações sintáticas e semânticas das palavras. Isso pode ser visualizado através do t-SNE (MAATEN; HINTON, 2008), uma técnica sofisticada para visualizar dados em altas dimensões.

Figura 11: t-SNE: Visualização para *word embeddings*



Fonte: Turian, Ratinov e Bengio (2010)

Na Figura 11 podemos visualizar um tipo de mapeamento entre os sentidos intuitivos das palavras, onde na esquerda estão palavras referentes a números e na direita palavras referentes a profissões. Ou seja, palavras similares estão pertos umas das outras.

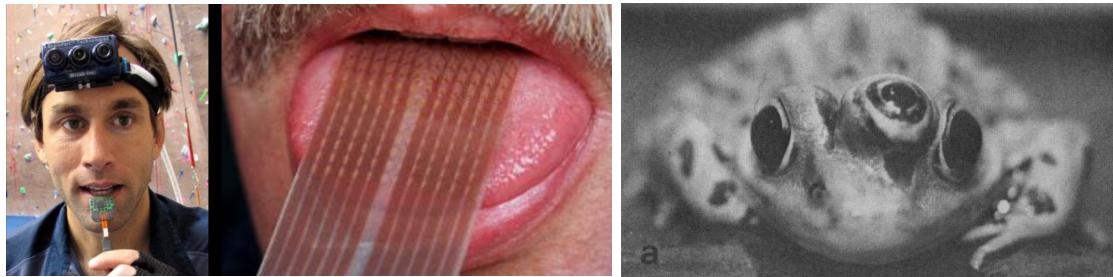
Figura 12: Exemplo de palavras similares

FRANCE	JESUS	XBOX	REDDISH	SCRATCHED	MEGABITS
AUSTRIA	GOD	AMIGA	GREENISH	NAILED	OCTETS
BELGIUM	SATI	PLAYSTATION	BLUISH	SMASHED	MB/S
GERMANY	CHRIST	MSX	PINKISH	PUNCHED	BIT/S
ITALY	SATAN	IPOD	PURPLISH	POPPED	BAUD
GREECE	KALI	SEGA	BROWNISH	CRIMPED	CARATS
SWEDEN	INDRA	PSNUMBER	GREYISH	SCRAPED	KBIT/S
NORWAY	VISHNU	HD	GRAYISH	SCREWED	MEGAHERTZ
EUROPE	ANANDA	DREAMCAST	WHITISH	SECTIONED	MEGAPIXELS
HUNGARY	PARVATI	GEFORCE	SILVERY	SLASHED	GBIT/S
SWITZERLAND	GRACE	CAPCOM	YELLOWISH	RIPPED	AMPERES

Fonte: Collobert et al. (2011)

Na Figura 12 temos mais alguns exemplos desse conceito, com isso podemos realizar tarefas semânticas mais facilmente, como o exemplo 1.1, que é uma análise que pode ser feita ao trabalhar com essa representação.

Figura 13: Exemplos de representação de sensores no cérebro



(a) Visualizando com a língua

(b) Implantando um terceiro olho

Fonte: [Ng et al. \(2013\)](#)

## 3.4 Redes neurais

Redes neurais tem como origem algoritmos que tentam imitar o cérebro humano. Elas eram amplamente utilizadas na década de 80 e no começo da década de 90, porém sua popularidade diminuiu no fim dessa mesma década. Resurgiram recentemente e são o estado-da-arte para varias aplicações de inteligência artificial ([NG, 2015](#)).

### 3.4.1 Neurocomputação

Há evidências de que o cérebro usa apenas um “algoritmo de aprendizagem” para todas diferentes funções. Cientistas já tentaram cortar (no cérebro de animais) a conexão entre as orelhas e o córtex auditivo e reconectar com o nervo óptico, o resultado foi que o córtex auditivo literalmente aprendeu a ver.

Na [Figura 13\(a\)](#) os cientistas conseguiram fazer com que o ser humano possa enxergar através de sensores da língua conectado a uma câmera fotográfica. Já na [Figura 13\(b\)](#), foi implantando um terceiro olho em um sapo e após um tempo comprovou-se que o sapo definitivamente aprender a ver com aquele olho.

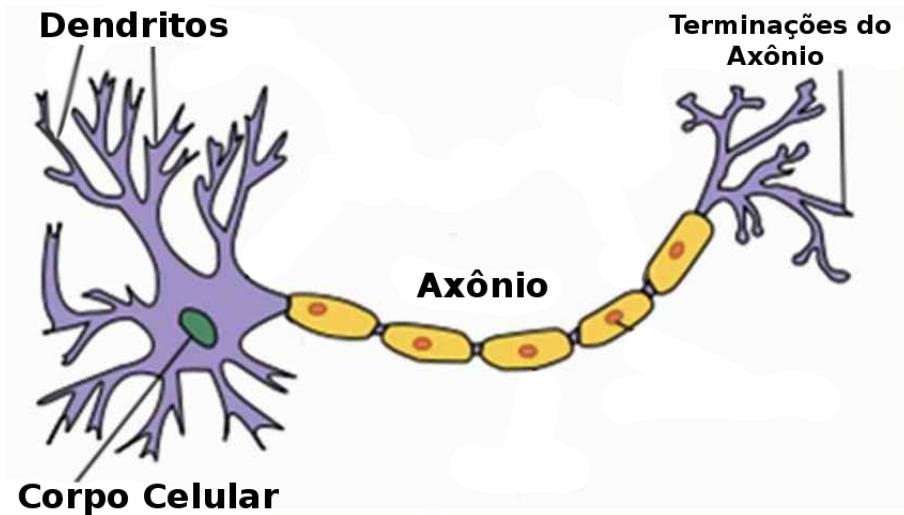
Esse princípio é conhecido como “neuroplasticidade” e tem vários exemplos e evidências de experimentos.

### 3.4.2 Representação do modelo

Em um nível bem simples de abstração, neurônios são basicamente unidades computacionais que recebem entradas (dendritos) como impulso elétrico que são canalizados para a saída (axônio), já o corpo celular é responsável por realizar os cálculos. Isso é ilustrado na [Figura 14](#).

A função de hipótese pode ser representada através de um modelo de redes neurais. Nesse modelo, o  $x_0$  é usualmente chamado de viés de entrada ou unidade viés (do inglês

Figura 14: Representação de um neurônio



Fonte: [Ng \(2015\)](#)

*bias unit) e ele sempre será igual a 1.*

Em redes neurais, é usado a mesma função logística como em classificação:

$$\frac{1}{1 + e^{\theta^T X}}$$

A diferença é que em redes neurais ela é usualmente conhecida como função de ativação sigmoide. Os parâmetros  $\theta$ s a serem aprendidos são chamados de pesos. Visualmente, uma representação simples desse modelo se parece como a [Figura 15](#). Os dados dos nodos de entrada (camada 1) vão para outro nodo (camada 2) que aplica a função de ativação, e então vão para a saída sendo a função de hipótese.

A primeira camada é chamada de “camada de entrada” e a camada final de “camada de saída”, que nos dá o valor final computado sobre a hipótese. É possível ter camadas intermediárias entre as camadas de entrada e saída chamadas de “camadas ocultas”.

Nomeamos os nodos das camadas intermediárias de  $a_0^2, a_1^2, \dots, a_n^2$  e chamamo-os de unidades de ativação. A tabela [Tabela 3](#) reúne as notações geralmente usadas no contexto de redes neurais.

A [Figura 16](#) mostra uma rede neural com uma camada oculta. Apesar do nodo viés não aparecer na camada oculta ele está presente nela, aliás, ele está presente e alimenta todos os nodos de todas camadas da rede (exceto a camada de saída), porém ele não recebe entradas.

Figura 15: Abstração matemática de um neurônio

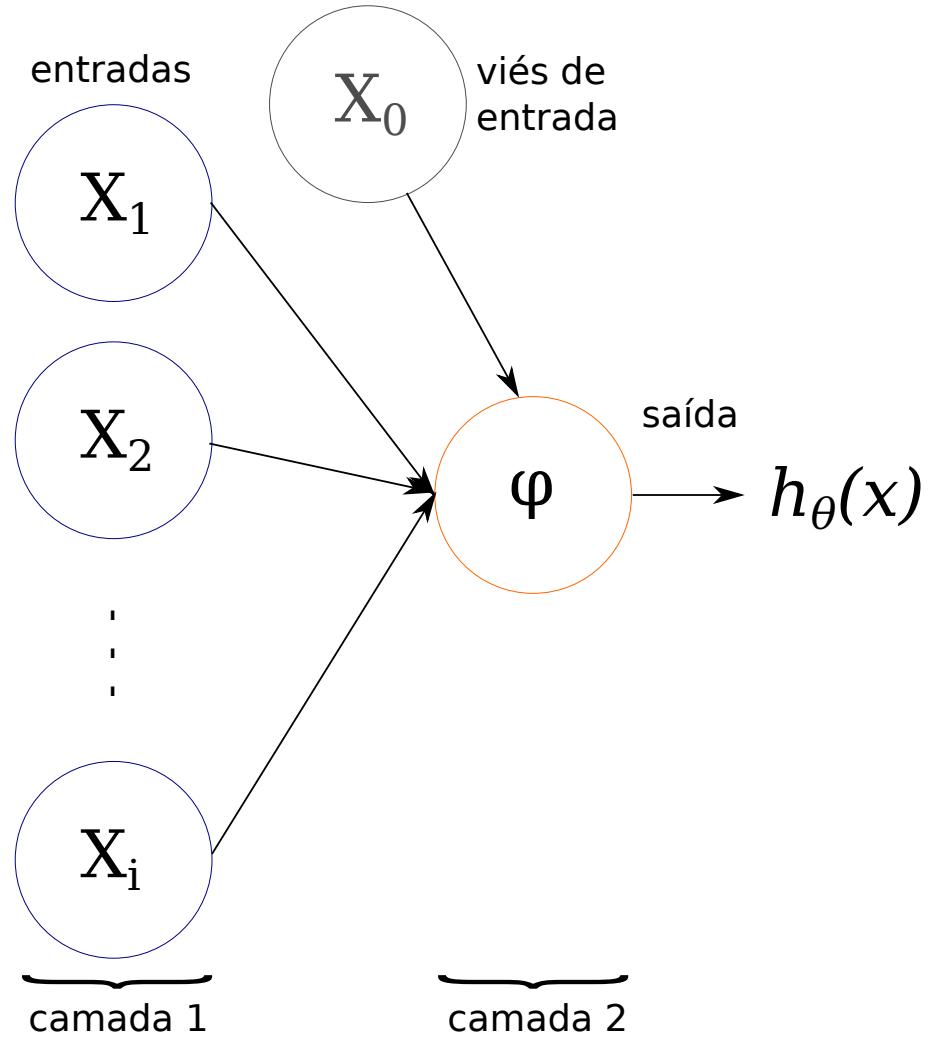


Figura 16: Rede neural com uma camada oculta

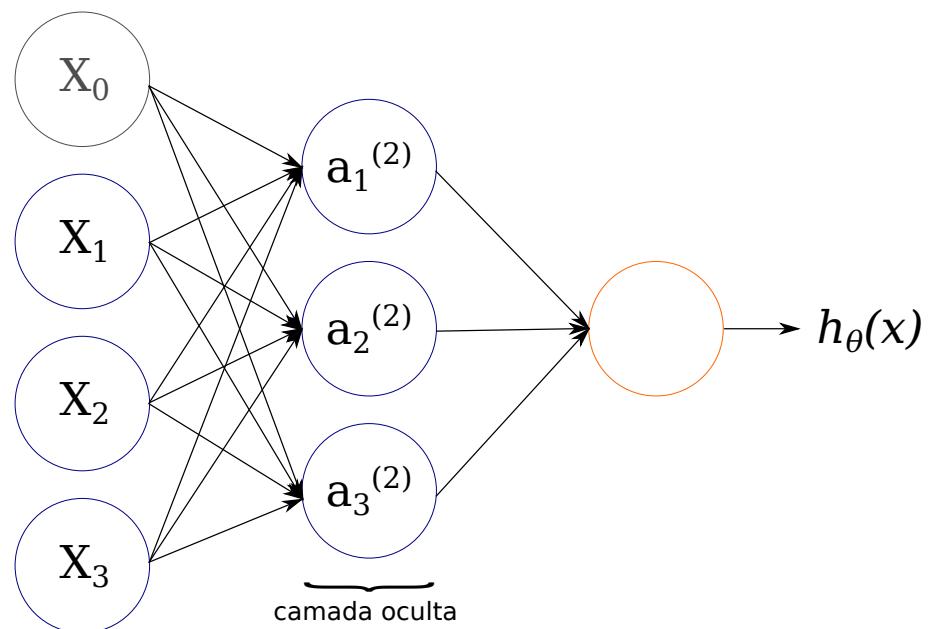


Tabela 3: Notação utilizada para representação redes neurais

---

$\varphi$ ou $g(z)$	função de ativação sigmoide
$a_i^{(j)}$	unidade de ativação $i$ na camada $j$
$\theta^{(j)}$	matriz de pesos mapeando funções da camada $j$ para camada $j + 1$
$z_k^{(j)}$	parâmetros da função $g$ para a unidade de ativação $k$ na camada $j$ .

---

O valor de cada nodo de ativação pode ser obtido pelo conjunto de equações 3.21:

$$\begin{aligned} a_1^{(2)} &= g(\theta_{1,0}^{(1)}x_0 + \theta_{1,1}^{(1)}x_1 + \theta_{1,2}^{(1)}x_2 + \theta_{1,3}^{(1)}x_3) \\ a_2^{(2)} &= g(\theta_{2,0}^{(1)}x_0 + \theta_{2,1}^{(1)}x_1 + \theta_{2,2}^{(1)}x_2 + \theta_{2,3}^{(1)}x_3) \\ a_3^{(2)} &= g(\theta_{3,0}^{(1)}x_0 + \theta_{3,1}^{(1)}x_1 + \theta_{3,2}^{(1)}x_2 + \theta_{3,3}^{(1)}x_3) \end{aligned} \quad (3.21)$$

E a função de hipótese é justamente a saída do único nodo de ativação na camada 3 (saída):

$$h_\theta(x) = g(\theta_{1,0}^{(2)}a_0^{(2)} + \theta_{1,1}^{(2)}a_1^{(2)} + \theta_{1,2}^{(2)}a_2^{(2)} + \theta_{1,3}^{(2)}a_3^{(2)})$$

Isso significa que estamos computando os nodos de ativação fazendo uma matriz de parâmetros com dimensão  $3 \times 4$ . Aplicamos cada linha dos parâmetros para as entradas, obtendo assim o valor para um nodo de ativação. A saída da hipótese é a função logística aplicada a soma dos valores dos nodos de ativação da camada oculta, os quais foram multiplicados por outra matriz de parâmetros ( $\theta^{(2)}$ ) contendo os pesos para a segunda camada de nodos.

Cada camada  $j$  tem sua própria matriz de pesos  $\theta^{(j)}$ . As dimensões dessas matrizes de pesos são determinadas pela regra 3.22.

*Se a rede tem  $S_j$  unidades na camada  $j$  e  $S_{j+1}$  na camada  $j + 1$ , então  $\theta^{(j)}$  terá a dimensão de  $S_{j+1} \times (S_j + 1)$ .* (3.22)

Isso significa que os nodos de saída não incluirão o viés de entrada, enquanto os de entrada sim. Essa regra é importante para ter uma versão vetorizada das redes neurais, pois facilita a codificação da mesma e também no entendimento de definições mais complexas.

### 3.4.3 Representação vetorizada do modelo

Em ordem de fazer uma implementação vetorizada das funções anteriores, será definido uma nova variável  $z_k^{(j)}$  que engloba os parâmetros dentro da função  $g$ . Para o conjunto de equações 3.21 é possível simplificar a definição ao substituir os parâmetros pela variável  $z$ , como mostrado no conjunto de equações 3.23.

$$\begin{aligned} a_1^{(2)} &= g(z_1^{(2)}) \\ a_2^{(2)} &= g(z_2^{(2)}) \\ a_3^{(2)} &= g(z_3^{(2)}) \end{aligned} \quad (3.23)$$

Onde a definição de  $z$  é dada pela [Equação 3.24](#).

$$z_k^{(j)} = \theta_{k,0}^{(j-1)}x_0 + \theta_{k,1}^{(j-1)}x_1 + \dots + \theta_{k,n}^{(j-1)}x_n \quad (3.24)$$

Levando em consideração que  $x$  é um vetor coluna com dimensão  $(n+1)$ , podemos calcular  $z^{(j)}$  ao fazer a multiplicação matriz-vetor:

$$z^{(j)} = \theta^{(j-1)}x$$

Como sabemos que  $x$  é nosso vetor de entrada ( $x = a^{(1)}$ ), podemos reescrever a equação como  $z^{(j)} = \theta^{(j-1)}a^{(1)}$ . Lembrando da regra [3.22](#), é possível analisar que a matriz  $\theta^{(j-1)}$  tem dimensões  $S_j \times (n+1)$ , e portanto a multiplicação dessa matriz com o vetor  $x$  com altura  $(n+1)$ , irá resultar no vetor  $z$  com altura  $S_j$ . E com isso, podemos obter um vetor de nodos de ativação para a camada  $j$  através da [Equação 3.25](#).

$$a^{(j)} = g(z^{(j)}) \quad (3.25)$$

Onde a função  $g$  pode ser aplicada elementarmente no vetor  $z^{(j)}$ . Com isso, é adicionado a unidade viés para a camada  $j$  após o término da computação de  $a^{(j)}$ . Onde essa unidade será o elemento  $a_0^{(j)}$ , que é por definição igual a 1.

Para computar as hipóteses finais, é necessário primeiro computar os outros vetores  $z$  das próximas camadas. Podemos fazer isso de acordo com a [Equação 3.26](#).

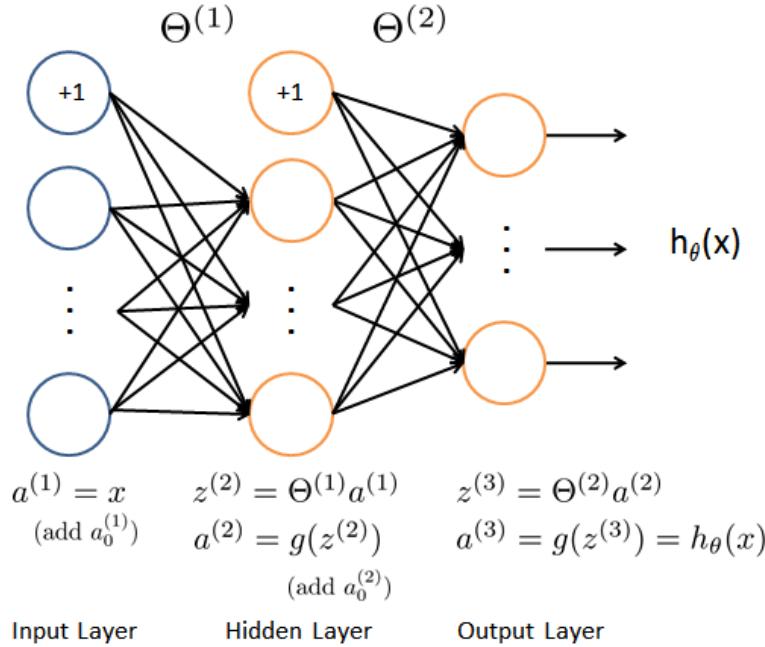
$$z^{(j+1)} = \theta^{(j)}a^{(j)} \quad (3.26)$$

A última matriz de pesos  $\theta^{(j)}$  terá apenas uma linha, que fará com que o resultado seja apenas um escalar. Para que então seja possível obter o resultado final com a [Equação 3.27](#).

$$h_\theta(x) = a^{(j+1)} = g(z^{(j+1)}) \quad (3.27)$$

Ao adicionar todas as camadas intermediárias nas redes neurais, permite-se a produção de hipóteses não lineares mais elegantes e complexas. Ao chegar na hipótese final, conclui-se os passos que são conhecidos como *forward propagation*.

Figura 17: Modelo da rede neural

Fonte: [Ng \(2015\)](#)

### 3.4.4 Classificação multiclasse

Para classificar dados em múltiplas classes, a função de hipótese deve retornar um vetor de valores. Quando a camada final for multiplicada por sua matriz  $\theta$ , vai resultar em outro vetor, no qual será aplicado a função de ativação  $g$  para obter o vetor de hipóteses finais. Essa configuração pode ser vista na [Figura 17](#).

O vetor de hipóteses resultante para um conjunto de entradas pode parecer como:

$$h_{\theta}(x) = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

No qual a classe resultante é a terceira linha, ou  $h_{\theta}(x)_3$ . É possível extender essa definição para definir todo o conjunto de classes resultados como  $y'$ :

$$y^{(i)} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

E o valor final da hipótese para um conjunto de entrada será um dos elementos em  $y$ .

### 3.4.5 Função de custo

As subseções anteriores mostraram como representar uma rede neural, agora estamos interessados em como realizar a aprendizagem para resolver o problema de classificação.

Antes de apresentar a função de custo para redes neurais, será definido algumas variáveis úteis nesse escopo. Elas estão na [Tabela 4](#).

Tabela 4: Notação utilizada para aprendizado em redes neurais

---

$L$	número total de camadas na rede
$S_l$	número de unidades (sem o viés de entrada) na camada $l$
$K$	número de unidades de saída ou número de classes

---

Na [subseção 3.4.4](#) foi visto que é possível ter vários nodos de saída. É denotado  $h_\theta(x)_k$  como sendo uma hipótese que resulta na  $k$ -ésima saída.

A função de custo para redes neurais é um pouco mais complicada que a utilizada na regressão logística, na verdade, ela é uma generalização da [Equação 3.18](#).

Na [Equação 3.28](#), foi adicionado alguns somatórios aninhados para tratar os múltiplos nodos de saída. Na primeira parte da equação, entre os colchetes, foi adicionado um somatório aninhado que itera sobre o número de nodos de saída.

Na parte da regularização (após os colchetes), foi lido com as múltiplas matrizes  $\theta$ s. Conforme a regra [3.22](#), o número de colunas de uma dada matriz  $\theta$  é igual ao número de nodos na camada atual (incluindo o viés). Já o número de linhas da matriz  $\theta$  é igual ao número de nodos na próxima camada (sem o viés).

$$J(\theta) = \frac{-1}{m} \left[ \sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \log(h_\theta(x^{(i)})_k) + (1 - y_k^{(i)}) \log(1 - h_\theta(x^{(i)})_k) \right] + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{S_l} \sum_{j=1}^{S_{l+1}} (\theta_{j,i}^l)^2 \quad (3.28)$$

### 3.4.6 Algoritmo *Backpropagation*

Em redes neurais, para realizar a minimização de custo um dos algoritmos amplamente utilizado é o *Backpropagation*. Seu objetivo é calcular as derivadas parciais da função de custo, para que depois seja possível realizar a operação:  $\min_\theta J(\theta)$ . Ou seja, o objetivo é minimizar a função de custo  $J$  usando um conjunto ótimo de parâmetros  $\theta$ . E para isso é necessário o cálculo da derivada parcial da função de custo em relação

aos parâmetros  $\theta$  para cada nodo de ativação em uma camada  $l$ , conforme mostrado na [Equação 3.29](#).

$$\frac{\partial}{\partial \theta_{i,j}^{(l)}} J(\theta) \quad (3.29)$$

Em *backpropagation* é calculado o **erro** para cada nodo. Extendemos a notação usada pela [Tabela 4](#) ao adicionar uma nova variável  $\delta_j^{(l)}$  que é igual ao “erro” do nodo  $j$  na camada  $l$ . Sendo assim, para a última camada, é possível calcular um vetor de valores  $\delta$  com:  $\delta^{(L)} = a^{(L)} - y$ . Ou seja, os “valores de erro” para a última camada são simplesmente a diferença entre os resultados atuais na última camada e as saídas corretas em  $y$ .

A questão chave desse algoritmo é como obter os valores de  $\delta$  nas camadas anteriores à última. Para isso é usado a [Equação 3.30](#) que nos leva para trás, da direita para esquerda.

$$\delta^{(l)} = ((\theta^{(l)})^T \delta^{(l+1)}). * g'(z^{(l)}) \quad (3.30)$$

Os valores de  $\delta$  da camada  $l$  são calculados ao multiplicar os valores de  $\delta$  na próxima camada com a matriz  $\theta$  na camada  $l$ . Isso é multiplicado elementarmente com uma função  $g'$ , a qual é a derivada da função de ativação  $g$  avaliada com os valores de entrada dados por  $z^{(l)}$ . A função  $g'$  pode ser descrita como na [Equação 3.31](#).

$$g'(z^{(l)}) = a^{(l)}. * (1 - a^{(l)}) \quad (3.31)$$

Fonte: [Ng \(2015\)](#)

Portanto, a equação completa de *backpropagation* para os nodos interno é mostrada na [Equação 3.32](#).

$$\delta^{(l)} = ((\theta^{(l)})^T \delta^{(l+1)}). * a^{(l)}. * (1 - a^{(l)}) \quad (3.32)$$

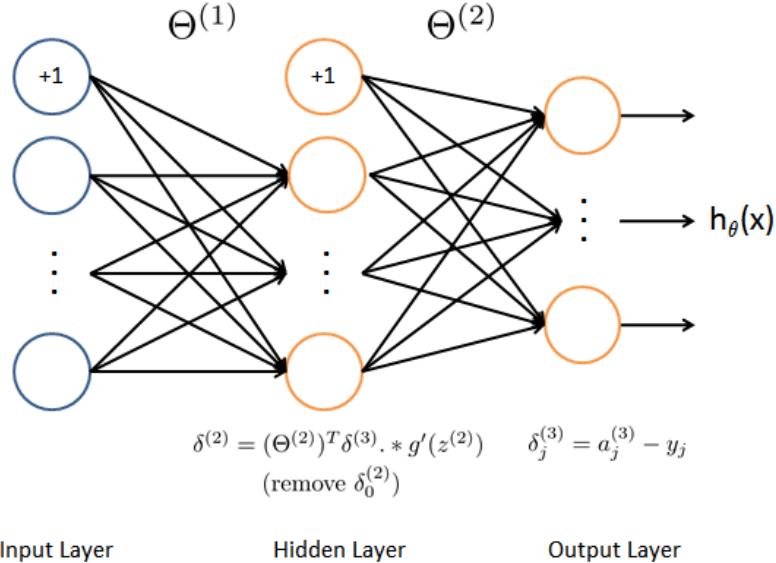
Agora é possível computar a derivada parcial ao multiplicar os valores de ativação e os valores de erro para cada exemplo de treinamento  $t$ , conforme mostrado na [Equação 3.33](#).

$$\frac{\partial}{\partial \theta_{i,j}^{(l)}} J(\theta) = \frac{1}{m} \sum_{t=1}^m a_j^{(t)(l)} \delta_i^{(t)(l+1)} \quad (3.33)$$

Isso porém ignora a regularização, que é feita depois.

Uma observação importante é que  $\delta^{(l+1)}$  é um vetor com  $S_{l+1}$  elementos, e  $a^{(l)}$  é um vetor com  $S_l$  elementos. Portanto a multiplicação deles vai gerar uma matriz com

Figura 18: Passos do Backpropagation



Fonte: Ng (2015)

dimensão  $S_{l+1} \times S_l$ , que é a mesma dimensão que  $\theta^{(l)}$ . Ou seja, o processo irá produzir um termo gradiente para todos elementos em  $\theta^{(l)}$ . O algoritmo 1 junta todas essas equações.

---

**Algorithm 1** Algoritmo de Backpropagation
 

---

```

procedure BACKPROPAGATION
    dado um conjunto de treinamento  $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$ 
     $\Delta_{i,j}^{(l)} = 0$  para todo  $(l, i, j)$ 
    for cada exemplo de treinamento  $t = 1$  até  $m$  do
         $a^{(i)} = x^{(t)}$ 
        Forward-Propagation( $a^{(l)}$ ) para  $l = 2$  até  $L$ 
         $\delta^{(L)} = a^{(L)} - y^{(t)}$ 
         $\delta^{(l)} = ((\theta^{(l)})^T \delta^{(l+1)}) * a^{(l)} * (1 - a^{(l)})$  para  $l = L - 1$  até 2
         $\Delta^{(l)} = \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$                                 ▷ usando vetorização
    end for
    if  $j = 0$  then
         $D_{i,j}^{(l)} = \frac{1}{m} \Delta^{(l)}$ 
    else
         $D_{i,j}^{(l)} = \frac{1}{m} (\Delta^{(l)} + \lambda \theta^{(l)})$ 
    end if
end procedure
  
```

---

A matriz  $\Delta$  é usada apenas como um acumulador, com o intuito de passar adiante os valores enquanto é computado a derivada parcial. Os termos  $D_{i,j}^{(l)}$  são as derivadas parciais finais, ou seja, os resultados esperados. A Figura 18 ilustra os passos do algoritmo de *backpropagation* para uma rede neural com uma camada oculta.

Segundo (NG, 2015), uma observação importante a ser feita é que os pesos  $\theta$  não

podem ser inicializados com zero, pois isso não funciona em redes neurais. Uma solução para esse problema é realizar uma **inicialização aleatória** dos pesos.

### 3.4.7 Treinamento

Para treinar a rede, é necessário primeiramente escolher uma arquitetura (*layout*) da rede neural, incluindo quantas unidades ocultas em cada camada e quantas camadas no total.

- Número de unidades de entrada: igual ao número de dimensões das *features*  $x^{(i)}$ ;
- Número de unidades de saídas: igual ao número de classes  $|y|$ ;
- Número de camadas ocultas: por padrão é 1, caso seja mais que isso, então deve ter a mesma quantidade de unidades de ativação em cada camada oculta.

Para treinar a rede neural, é possível seguir os seguintes passos:

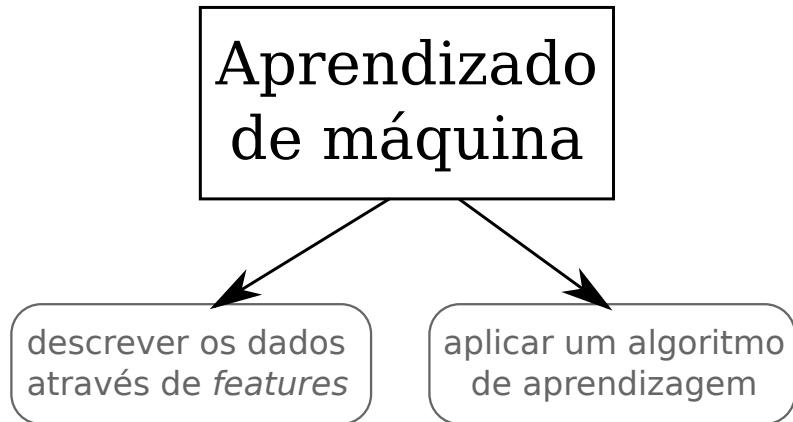
1. Inicializar aleatoriamente os pesos  $\theta$ ;
2. Implementar o *Forward-propagation* para obter  $h_\theta(x^{(i)})$ ;
3. Implementar a função de custo;
4. Implementar o *Backpropagation* para computar as derivadas parciais;
5. Usar o Gradiente Descendente ou outra função de otimização para minimizar a função de custo com os pesos  $\theta$ .

## 3.5 Aprendizagem profunda

A aprendizagem profunda é uma ramificação da aprendizagem de máquina baseado em um conjunto de algoritmos que procuram modelar abstrações de alto nível usando modelos arquitetoriais, com estruturas complexas ou outrora, compostas de múltiplas transformações não lineares (DENG; YU, 2014).

A Figura 19 mostra que o aprendizado de máquina se resumiu em otimizar pesos para fazer uma ótima predição final. E para isso, é necessário dois itens: Descrever os dados através de *features* de modo que o computador consiga entender; Aplicar um algoritmo de aprendizagem. Para o primeiro item, é necessário conhecimento do domínio específico, o que acaba gerando o chamado *engenharia de features*, que ultimamente não é bem visto pela comunidade de PLN. O segundo item consiste em otimizar os pesos

Figura 19: Sentido do aprendizado de máquina



de acordo com as *features*, e qualquer bom algoritmo de minimização da função de custo pode ser utilizado.

Já a aprendizagem profunda tenta automaticamente aprender boas *features* ou representações. Na [seção 3.3](#) foi mostrado o conceito de *word embeddings* ou vetor de *features*, na verdade esse conceito está fortemente associado ao aprendizado profundo, onde tenta-se aprender múltiplos níveis da representação.

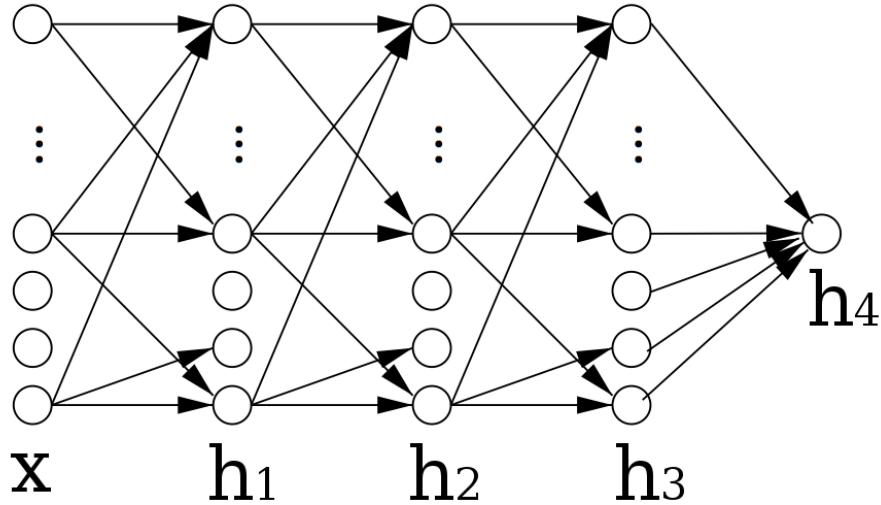
O modelo dominante em aprendizado profundo é redes neurais. Nesse modelo, as *features* são aprendidas rapidamente e automaticamente, adaptando-se muito bem as tarefas de [PLN](#). Com isso, é possível aprender representações das informações linguísticas de modo não supervisionado (a partir do texto fonte) e também de modo supervisionado (usando classes específicas). Ela começou a ser usada recentemente devido a criação de novos modelos, algoritmos e ideias, e também devido ao aumento do desempenho computacional ([SOCHER, 2015](#)).

Como o processo de criação automatizada de *features* acaba gerando muitos vetores de representações, temos que criar um modelo capaz de trabalhar com essas informações com o objetivo de aprender. Por isso, geralmente se trabalha com redes neurais com múltiplas camadas, como mostrado na imagem [Figura 20](#).

O problema dessa abordagem é que o algoritmo de *backpropagation* não funciona muito bem, pois acaba perdendo precisão aritmética nas camadas intermediárias, e quando chega no fim o resultado da hipótese é muito pior do que se tivesse apenas uma arquitetura básica com apenas uma camada oculta.

A técnica utilizada ultimamente em [POS Tagging](#) está ilustrada na [Figura 21](#). Que consiste em primeiramente obter a representação das palavras em vetores reais com dimensões definidas pelo usuário, e após isso mesclar com *features* de formato das palavras (e.g capitalização), para então jogar esses vetores como entrada numa rede neural. O que muda em cada abordagem geralmente é a questão do treinamento da representação das

Figura 20: Rede neural com múltiplas camadas

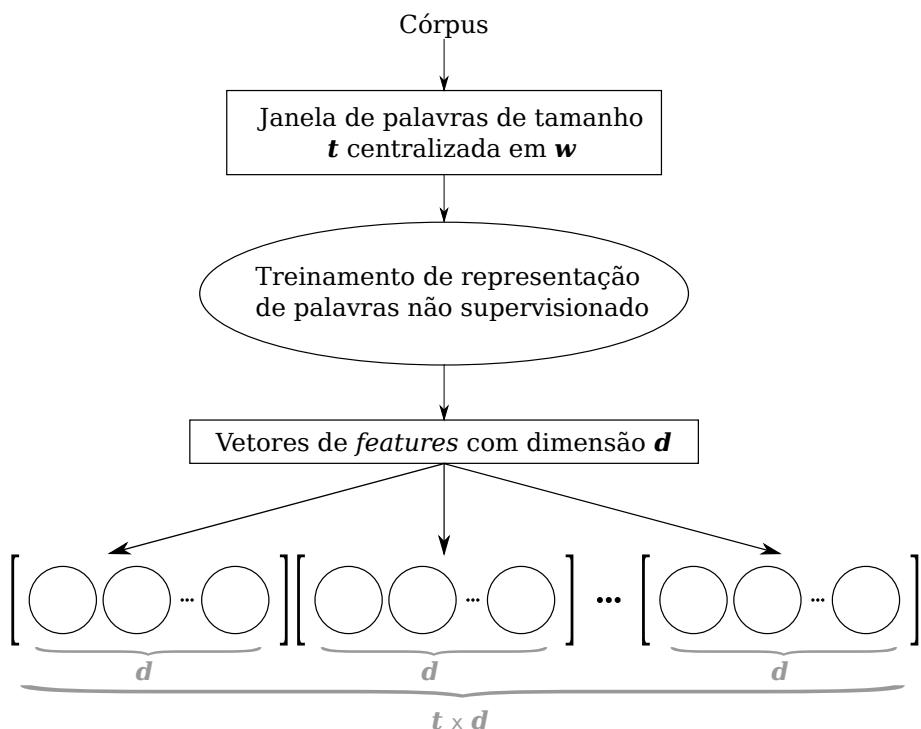


Fonte: Socher (2015)

palavras, as novas *features* criadas e como é organizado o modelo da rede neural.

- *secao a ser melhorada*
- *incluir de <<http://www.iro.umontreal.ca/~bengioy/dlbook/>>*

Figura 21: Funcionamento de aprendizagem profunda em POS Tagging



- incluir de <[http://cs224d.stanford.edu/lecture\\_notes/LectureNotes1.pdf](http://cs224d.stanford.edu/lecture_notes/LectureNotes1.pdf)>
- talvez incluir de <<http://colah.github.io/posts/2014-07-NLP-RNNs-Representations/>> (dar mais uma lida para formalizar as ideias)



## 4 Trabalhos relacionados

Vários métodos já foram propostos para resolver esse mesmo problema em português brasileiro, apesar de nenhum deles ter um aproveitamento de 100%, eles conseguiram ótimos resultados e utilizaram técnicas que são consideradas como o melhor do que se há atualmente.

É apresentado em ([KEPLER, 2005](#)) um etiquetador morfo-sintático baseado em cadeias de Markov. É realizado testes com dois etiquetadores diferentes, um baseado em Cadeias de Markov Ocultas (HMM, do inglês *Hidden Markov Models*), e outro baseado em Cadeias de Markov de Tamanho variável (VLMC, do inglês *Variable Length Markov Chains*). A representação das palavras é feita de forma empírica, pois a etiquetagem baseia-se em modelos probabilísticos, onde a etiqueta de uma palavra depende da própria palavra e de etiquetas anteriores. Ele é testado sobre o *córpus* Tycho Brahe, e apresenta uma precisão de 95,51% com o etiquetador VLMC, essa precisão é alcançado com um tempo de aprendizagem + etiqueatem de 157 segundos.

- falar de mais algum aqui? qual?

Em ([SANTOS; ZADROZNY, 2014](#)) é apresentado um etiquetador que aprende automaticamente as features a serem usadas através de uma rede neural profunda que emprega uma camada evolutiva capaz de aprender *embeddings* em nível de caractere e em nível de palavra. Éssa rede neural profunda é conhecida como CharWNN e foi proposta originalmente por [Collobert et al. \(2011\)](#). Além disso, é usado um modelo em janela utilizado em ([COLLOBERT et al., 2011](#)) para atribuir classes gramaticais para cada palavra em uma sentença. Essa estratégia assume que a classe de uma palavra depende geralmente das palavras vizinhas. No fim é utilizado o algoritmo de Viterbi ([VITERBI, 1967](#)) para prever qual a sequência de classes gramaticais é a mais provável para aquela sentença, para então ser treinada em um modelo de rede neural com algumas *features* que obtém informações dos formatos da palavra. Esse trabalho usa três diferentes para o treinamento: *córpus*; o Mac-Morpho original; o Mac-Morpho revisado em ([FONSECA; ROSA, 2013](#)), e o Tycho Brahe. Eles avaliam seu modelo sobre palavras fora do vocabulário e sobre palavras presente no vocabulário. Com o Mac-Morpho foi obtido o melhor desempenho do trabalho, uma acurácia de 97,47% sobre palavras Dentro do Vocabulário ([DDV](#)). Nesse trabalho não é mostrado estatísticas de tempo de treinamento e etiquetagem.

O mais recente etiquetador para o português brasileiro é mostrado em ([FONSECA; ROSA; ALUÍSIO, 2015](#)), onde é utilizado diferentes técnicas de representação das palavras: vetores gerados de forma aleatória, [NLM](#), [HAL](#), [SG](#). É feita então uma comparação entre elas. Nele é implementado um modelo de rede neural profunda idealizado em

(COLLOBERT; WESTON, 2008), que se baseria em uma rede neural com simples com múltiplas camadas que recebe as *word embeddings* como entrada e aprende a sua classe grammatical. Para isso, eles realizam o treinamento utilizando o Tycho Brahe, a versão original do Mac-Morpho (versão 1) e a revisada pelos mesmos autores em um trabalho anterior (FONSECA; ROSA, 2013) (versão 2), e também sobre mais uma versão do Mac-Morpho revisado (versão 3) por eles nesse mesmo trabalho. Atualmente, Fonseca, Rosa e Aluísio (2015) dizem ter alcançado o estado-da-arte do POS Tagging para o português brasileiro, com uma acurácia de 97,57% sobre o Mac-Morpho original e no escopo de palavras DDV. Esse trabalho não apresenta estatísticas de tempo de treinamento e etiquetagem.

A Tabela 5 sumarizada as técnicas e resultados encontrados pelos trabalhos relacionados que tentam resolver o mesmo problema que este trabalho.

Tabela 5: Comparativo entre técnicas e resultados encontrados na literatura para o problema de POS Tagging.

Autores	Modelo	Representação das palavras	Córpus	Acurácia	Tempo
Kepler (2005)	VLMM	Sequência de caracteres	Tycho Brahe	95,51%	157 segundos
Santos e Zadrozny (2014)	Redes neurais profundas	Representação vetorial (CharWNN)	Tycho Brahe; Mac-Morpho v1; Mac-Morpho v2	97,47%	-
Fonseca, Rosa e Aluísio (2015)	Redes neurais	Representação vetorial (NLM, HAL, SG)	Tycho Brahe; Mac-Morpho v1; Mac-Morpho v2; Mac-Morpho v3	97,57%	-
Este trabalho	Redes neurais profundas	Representação vetorial (NLM, SG, GloVe)	Tycho Brahe; Mac-Morpho v1; Mac-Morpho v3	-	-

- Ajeitar a tabela !!!!

# 5 Metodologia

Alguns cuidados devem ser tomados no uso deste pacote. Leia as orientações a seguir e contate o responsável em caso de dúvidas.

## 5.1 Formatação

Embora não faça diferença no resultado final, é importante formatar adequadamente o seu código L<sup>A</sup>T<sub>E</sub>X. Da mesma forma que para outras linguagens de programação, isso aumenta a legibilidade do código e ajuda a encontrar partes específicas mais rapidamente. As principais dicas para arquivos T<sub>E</sub>Xsão:

- Indente seu código. Não só os ambientes (begin, end) mas também os parágrafos! Coloque cada sentença em uma linha, indentando a partir da segunda;
- Coloque marcações comentadas para delimitar o início de capítulos, seções, etc. Isso facilita buscar partes específicas em um arquivo.

Cuidado com abreviaturas e acrônimos. É fácil esquecer de os definir ou definir de maneira diferente em capítulos diferentes. Use os comandos do pacote `acro` para abreviaturas e acrônimos. Por exemplo, Figura ([Fig.](#)) é uma abreviação, então Trabalho de Conclusão de Curso ([TCC](#)) é um acrônimo/sigla. Eles são definidos no preâmbulo do documento.

Também vale a pena usar uma tabela de nomenclatura caso você use muitos símbolos, em especial símbolos matemáticos. Veja os comandos do pacote `nomencl`. As definições também ficam no preâmbulo do documento.

## 5.2 Codificação dos arquivos: UTF8

A codificação de todos os arquivos deste pacote é UTF8. É necessário que você utilize a mesma codificação nos documentos que escrever, inclusive nos arquivos de bases bibliográficas `|.bib|`.

## 5.3 Citações

Utilize o ambiente `citacao` para incluir citações diretas com mais de três linhas:

As citações diretas, no texto, com mais de três linhas, devem ser destacadadas com recuo de 4 cm da margem esquerda, com letra menor que a do texto utilizado e sem as aspas. No caso de documentos datilografados, deve-se observar apenas o recuo (ABNT, 2002, 5.3)

Citações simples, com até três linhas, devem ser incluídas com aspas. Observe que em L<sup>A</sup>T<sub>E</sub>X as aspas iniciais são diferentes das finais: “Amor é fogo que arde sem se ver”.

Para as citações indiretas, o comando padrão, \cite, realiza a forma mais comum de citação (??). A outra das formas mais usadas, para citar em texto corrido, é conseguida com o comando \citeonline: segundo ??), na citação indireta, o número da página é opcional.

### 5.3.1 Referências internas

Usa-se o comando \ref{} para referenciar uma Tabela ou Figura. Por exemplo, esta é uma referência para a Tabela 6. Mas também pode-se usar o comando \autoref{}, que insere o tipo também. Por exemplo, esta é outra referência para a Tabela 6.

Há vários outros comandos interessantes. Eles estão no fonte do Capítulo 1, na subseção 5.3.1 <sup>1</sup> (Introdução, página 27).

## 5.4 Tabelas

A Tabela 6 é um exemplo de tabela construída em L<sup>A</sup>T<sub>E</sub>X. Como sugestão de formatação, evite ao máximo o uso de linhas verticais. As colunas de uma tabela devem ser separadas visivelmente. O contrário indica que a tabela está mal formatada ou que certas informações não deveriam estar nela.

Da mesma forma, evite o uso de linhas horizontais para separar linhas da tabela. Use-as apenas para separar o cabeçalho e eventuais partes importantes. Para obter um resultado ainda mais elegante, use os comandos do pacote booktabs.

Veja essas sugestões aplicas na Tabela 6.

Uma opção avançada para a criação de tabelas é usar o pacote pgfplotstable. Ele permite que os dados de um arquivo sejam lidos e colocados em uma tabela, formatando-os da maneira que se quiser. A Tabela 7 é um exemplo. Veja o arquivo desenvolvimento.tex para os comandos necessários.

---

<sup>1</sup> O número do capítulo indicado é 1, que se inicia à página 27.

Tabela 6: Níveis de investigação.

Nível de Investigação	Insumos	Sistemas de Investigação	Produtos
Meta-nível	Filosofia da Ciência	Epistemologia	Paradigma
Nível do objeto	Paradigmas do metanível e evidências do nível inferior	Ciência	Teorias e modelos
Nível inferior	Modelos e métodos do nível do objeto e problemas do nível inferior	Prática	Solução de problemas

Fonte: [van Gigch e Pipino \(1986\)](#)

Tabela 7: Exemplo de tabela com dados de arquivo.

TAMANHO	MÉTODO 1 (%)	MÉTODO 2 (%)	MÉTODO 3 (%)	MÉDIA (%)
10	30.0	36.2	28.3	31.5
20	54.8	52.5	56.8	54.7
30	65.0	59.6	74.1	66.2
40	64.5	59.6	76.7	66.9
50	64.6	59.6	76.5	66.9

## 5.5 Figuras

Figuras podem ser criadas diretamente em L<sup>A</sup>T<sub>E</sub>X. Uma das melhores formas, por ser relativamente simples, bem documentada e gerar ótimos resultados, é com o uso do pacote tikz<sup>2</sup>. Ele permite gerar diagramas, árvores, fluxogramas etc. A [Figura 22](#) mostra um exemplo simples de árvore.

Junto com o pacote pgfplots também é possível gerar gráficos de funções ou a partir de dados em um arquivo (como no caso da [Tabela 7](#)). As [Figuras 23](#) e [24](#) mostram exemplos de gráficos de função, e a [Figura 25](#) um exemplo de gráfico a partir dos mesmos dados que os da [Tabela 7](#).

Figuras também podem ser incorporadas de arquivos externos, como é o caso da [Figura 26](#). Se a figura que ser incluída se tratar de um diagrama, um gráfico ou uma ilustração que você mesmo produza, priorize o uso de imagens vetoriais no formato PDF. Com isso, o tamanho do arquivo final do trabalho será menor, e as imagens terão uma apresentação melhor, principalmente quando impressas, uma vez que imagens vetoriais são perfeitamente escaláveis para qualquer dimensão. Nesse caso, se for utilizar o Microsoft Excel para produzir gráficos, ou o Microsoft Word para produzir ilustrações, exporte-os como PDF e os incorpore ao documento conforme o exemplo abaixo. No entanto, para manter a coerência no uso de software livre (já que você está usando L<sup>A</sup>T<sub>E</sub>Xe abnT<sub>E</sub>X2),

<sup>2</sup> Há vários exemplos em <<http://www.texample.net/>>.

Figura 22: Árvore de recursão de Fibonacci.

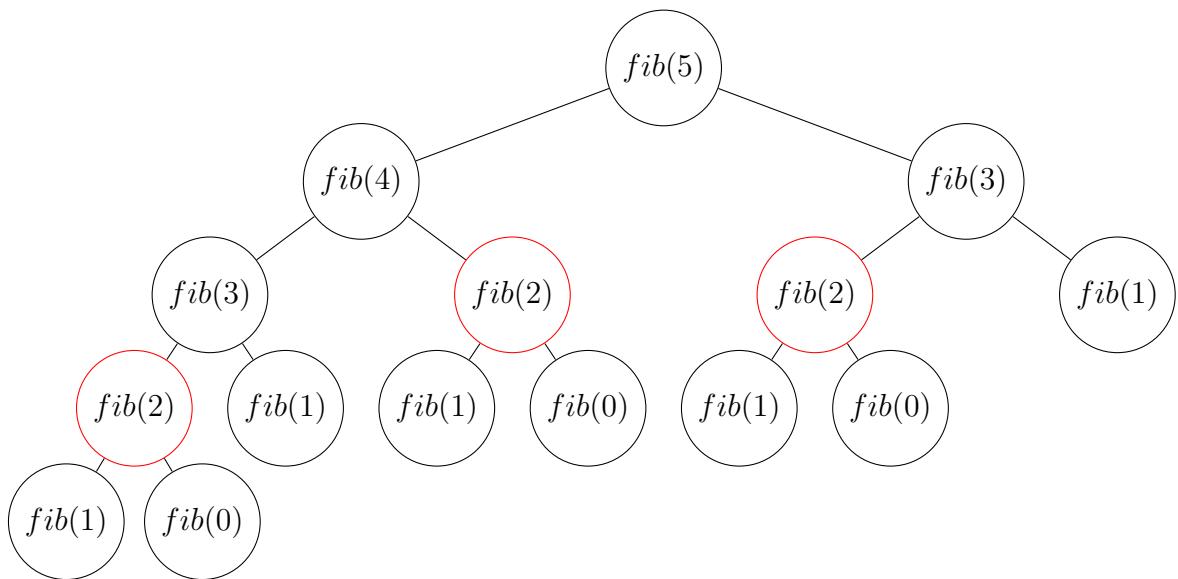
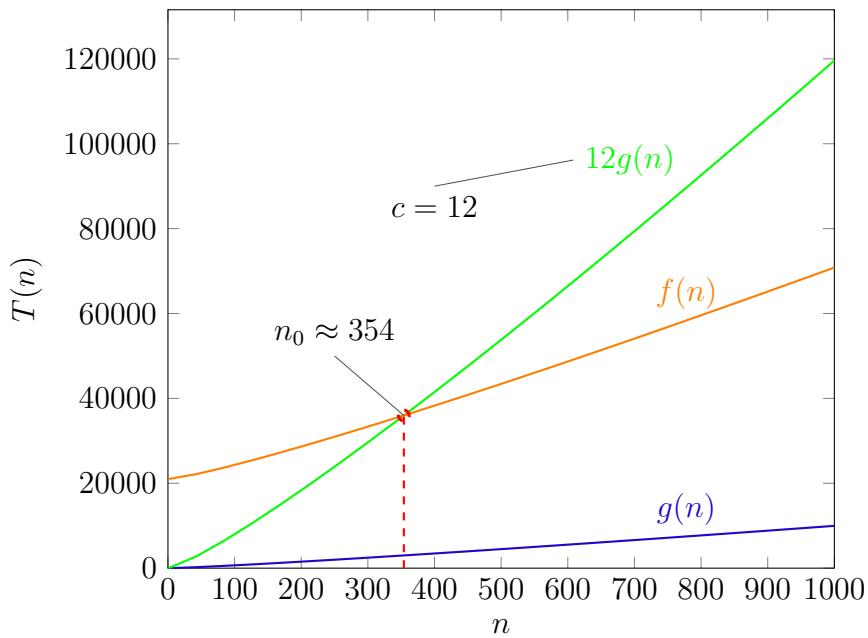


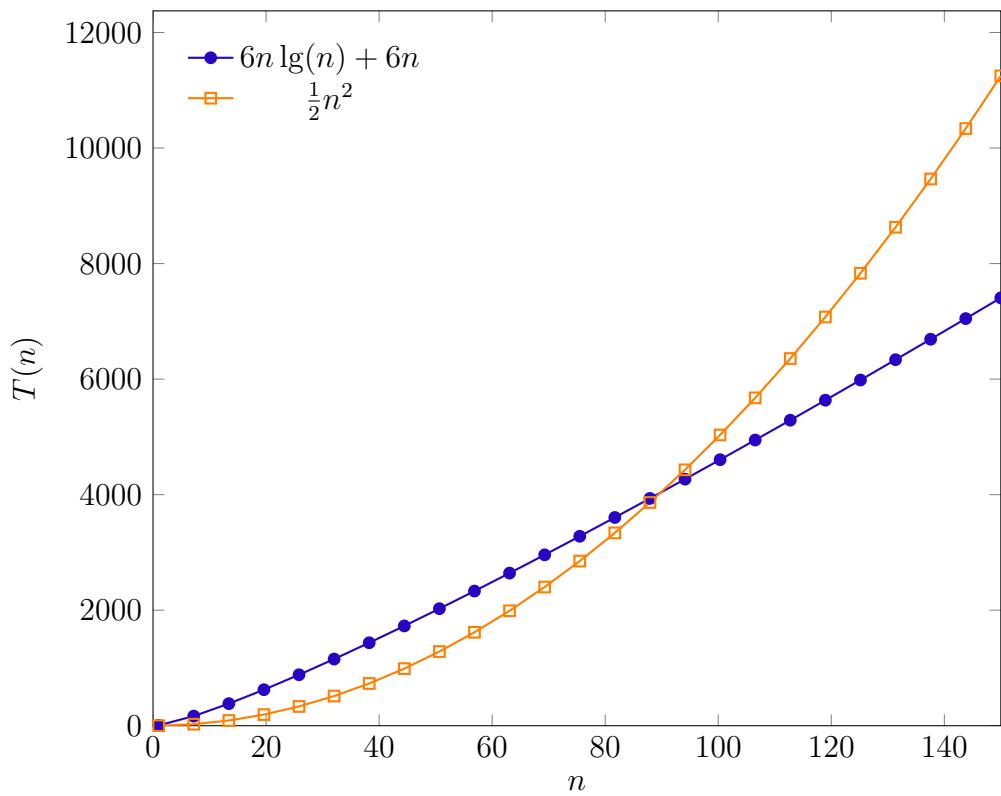
Figura 23: Gráfico produzido diretamente no arquivo fonte.



teste a ferramenta **InkScape**<sup>3</sup>. Ela é uma excelente opção de código-livre para produzir ilustrações vetoriais, similar ao CorelDraw ou ao Adobe Illustrator.

De todo modo, caso não seja possível utilizar arquivos de imagens como PDF, utilize qualquer outro formato, como PNG, JPEG, etc. Nesse caso, você pode tentar

<sup>3</sup> <http://inkscape.org/>

Figura 24: Outro gráfico feito em L<sup>A</sup>T<sub>E</sub>X.

aprimorar as imagens incorporadas com o software livre Gimp<sup>4</sup>. Ele é uma alternativa livre ao Adobe Photoshop.

A Figura 27 na página 69 contém duas subfiguras, Figura 27(a) e (b). A Figura 27(a) foi inserida de um arquivo externo, enquanto a Figura 27(b) foi escrita dentro do próprio código T<sub>E</sub>X. A Figura 28 contém o mesmo exemplo, mas usando comandos diferentes para inserir as Subfiguras 28(a) e (b).

Na Figura 27, as legendas (que indicam a fonte) para cada subfigura só funcionaram porque as figuras ficaram uma embaixo da outra. Se elas estivessem lado a lado, a inserção do comando `\legend` em cada uma faria com que elas ficassem organizadas na vertical. Uma legenda geral funcionaria, entretanto.

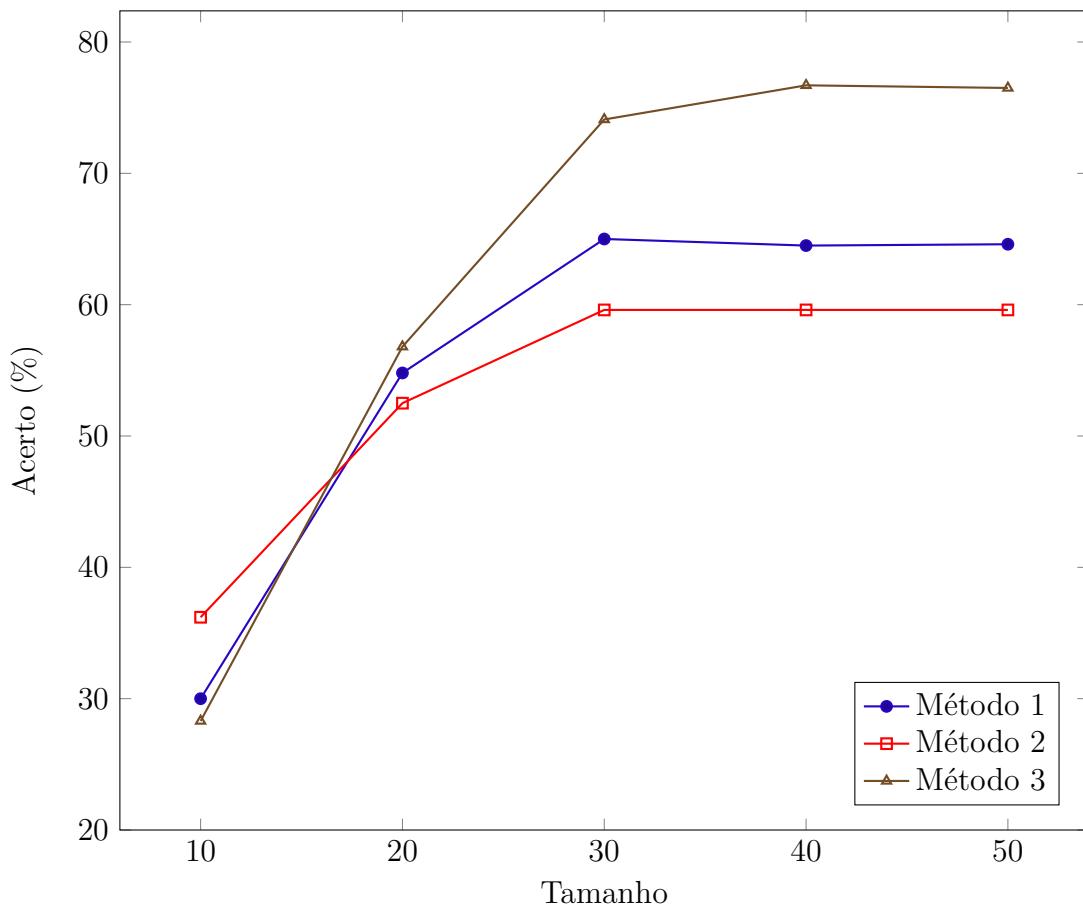
Na Figura 28, tanto legendas para subfiguras quanto uma legenda geral funcionam.

### 5.5.1 Sobre a indicação da fonte de uma tabela ou figura

As normas ABNT (2011, 5.8) e o Manual de Normatização da UNIPAMPA (?) dizem para, “Após a ilustração, na parte inferior, indicar a fonte consultada (elemento obrigatório, mesmo que seja produção do próprio autor), legenda, notas e outras infor-

<sup>4</sup> <http://www.gimp.org/>

Figura 25: Variação dos resultados utilizando seleção por Janela Deslizante.

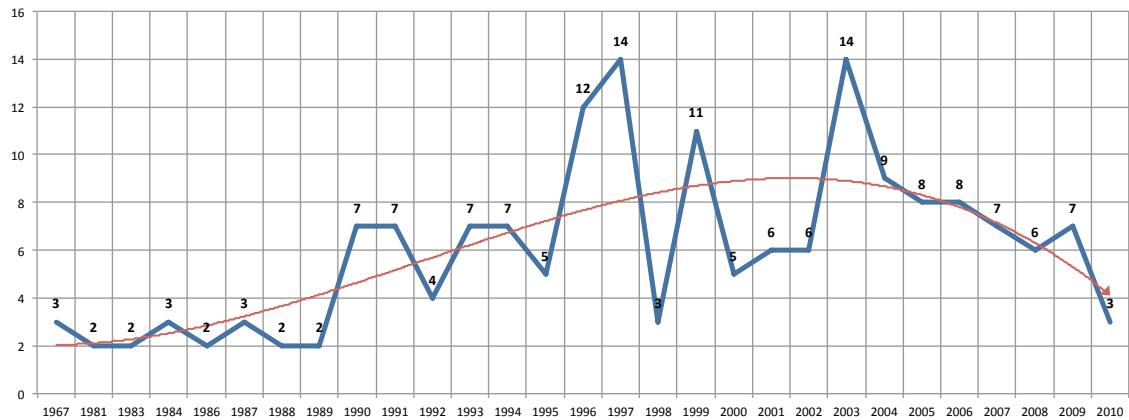


mações necessárias à sua compreensão (se houver).” A primeira interpretação é a de que, mesmo que o autor tenha criado a figura, a fonte deverá ser indicada. Com efeito, várias outras normas, manuais e inclusive o exemplo do pacote abnT<sub>E</sub>X22 usam “Fonte: os autores” em alguns lugares.

Entretanto, isso não está correto. Veja o trecho em destaque: “Após a ilustração, na parte inferior, indicar a fonte **consultada** (elemento obrigatório, mesmo que seja produção do próprio autor) (...).” A interpretação correta é a de que, caso a ilustração tenha sido **extraída** de um documento, a fonte deve ser indicada, ainda que esse documento pertença ao próprio autor. A sentença original das normas deveria ter sido melhor escrita para evitar a interpretação incorreta.

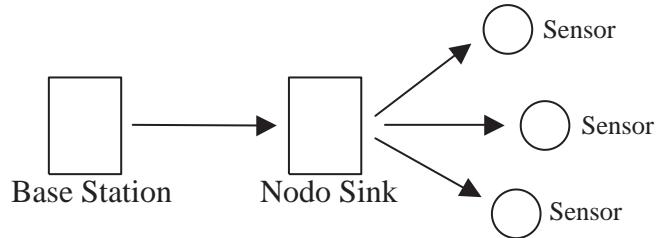
Assim, não indique a fonte se a figura ou tabela foi criada para o trabalho, ou seja, se é inédita. Caso contrário, indique a fonte. Mas cuidado: caso a figura ou tabela tenha sido adaptada de outra já publicada, então é obrigatório indicar “adaptado de” ou “acrescida de” seguido da referência da fonte de onde ela foi extraída.

Figura 26: Gráfico produzido em Excel e salvo como PDF.



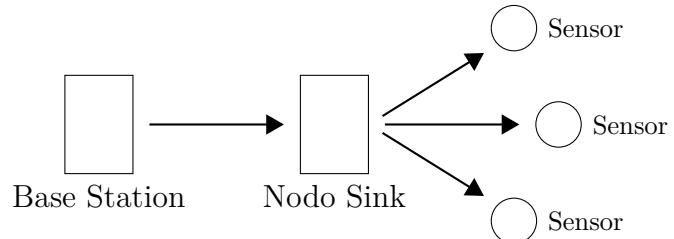
Fonte: Araujo (2012, p. 24)

Figura 27: Exemplo de subfiguras.



(a) Uma figura de um arquivo.

Fonte: ??)



(b) Uma figura em puro código TikZ.

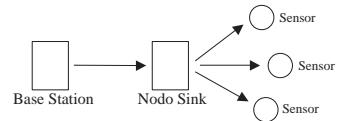
Alterado: de ??)

## 5.6 Expressões matemáticas

Use o ambiente `equation` para escrever expressões matemáticas numeradas:

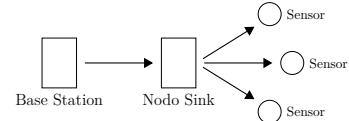
$$\forall x \in X, \quad \exists y \leq \epsilon \quad (5.1)$$

Figura 28: Mesmo exemplo de subfiguras, agora em escala.



(a) Uma figura de um arquivo.

Fonte: ??)



(b) Uma figura em puro código TikZ.

Alterado: de ??)

Fonte geral

Escreva expressões matemáticas entre \$ e \$, como em  $\lim_{x \rightarrow \infty} \exp(-x) = 0$ , para que fiquem na mesma linha.

Também é possível usar colchetes para indicar o início de uma expressão matemática que não é numerada.

$$\left| \sum_{i=1}^n a_i b_i \right| \leq \left( \sum_{i=1}^n a_i^2 \right)^{1/2} \left( \sum_{i=1}^n b_i^2 \right)^{1/2}$$

Consulte mais informações sobre expressões matemáticas em <<http://code.google.com/p/abntex2/w/edit/Referencias>>.

## 5.7 Enumerações: alíneas e subalíneas

Quando for necessário enumerar os diversos assuntos de uma seção que não possua título, esta deve ser subdividida em alíneas (ABNT, 2012, 4.2):

- a) os diversos assuntos que não possuem título próprio, dentro de uma mesma seção, devem ser subdivididos em alíneas<sup>5</sup>;
- b) o texto que antecede as alíneas termina em dois pontos;
- c) as alíneas devem ser indicadas alfabeticamente, em letra minúscula, seguida de parêntese. Utilizam-se letras dobradas, quando esgotadas as letras do alfabeto;
- d) as letras indicativas das alíneas devem apresentar recuo em relação à margem esquerda;
- e) o texto da alínea deve começar por letra minúscula e terminar em ponto-e-vírgula, exceto a última alínea que termina em ponto final;

<sup>5</sup> As notas devem ser digitadas ou datilografadas dentro das margens, ficando separadas do texto por um espaço simples de entre as linhas e por filete de 5 cm, a partir da margem esquerda. Devem ser alinhadas, a partir da segunda linha da mesma nota, abaixo da primeira letra da primeira palavra, de forma a destacar o expoente, sem espaço entre elas e com fonte menor. ABNT (2011, 5.2.1)

- f) o texto da alínea deve terminar em dois pontos, se houver subalínea;
- g) a segunda e as seguintes linhas do texto da alínea começa sob a primeira letra do texto da própria alínea;
- h) subalíneas (ABNT, 2012, 4.3) devem ser conforme as alíneas a seguir:
  - as subalíneas devem começar por travessão seguido de espaço;
  - as subalíneas devem apresentar recuo em relação à alínea;
  - o texto da subalínea deve começar por letra minúscula e terminar em ponto-e-vírgula. A última subalínea deve terminar em ponto final, se não houver alínea subsequente;
  - a segunda e as seguintes linhas do texto da subalínea começam sob a primeira letra do texto da própria subalínea.
- i) no abnTeX2 estão disponíveis os ambientes `incisos` e `subalineas`, que em suma são o mesmo que se criar outro nível de `alineas`, como nos exemplos à seguir:
  - *Um novo inciso em itálico;*
- j) Alínea em **negrito**:
  - *Uma subalínea em itálico;*
  - *Uma subalínea em itálico e sublinhado;*
- k) Última alínea com *ênfase*.

## 5.8 Espaçamento entre parágrafos e linhas

O tamanho do parágrafo, espaço entre a margem e o início da frase do parágrafo, é definido por:

```
\setlength{\parindent}{1.3cm}
```

Por padrão, não há espaçamento no primeiro parágrafo de cada início de divisão do documento (??). Porém, você pode definir que o primeiro parágrafo também seja indentado, como é o caso deste documento. Para isso, apenas inclua o pacote `indentfirst` no preâmbulo do documento:

```
\usepackage{indentfirst}      % Indenta o primeiro parágrafo de cada seção.
```

O espaçamento entre um parágrafo e outro pode ser controlado por meio do comando:

---

```
\setlength{\parskip}{0.2cm} % tente também \onelineskip
```

O controle do espaçamento entre linhas é definido por:

```
\OneshiftSpacing      % espaçamento um e meio (padrão);
\DoubleSpacing        % espaçamento duplo
\SingleSpacing        % espaçamento simples
```

Para isso, também estão disponíveis os ambientes:

```
\begin{SingleSpace} ... \end{SingleSpace}
\begin{Spacing}{hfactori} ... \end{Spacing}
\begin{OneshiftSpace} ... \end{OneshiftSpace}
\begin{OneshiftSpace*} ... \end{OneshiftSpace*}
\begin{DoubleSpace} ... \end{DoubleSpace}
\begin{DoubleSpace*} ... \end{DoubleSpace*}
```

Para mais informações, consulte [Wilson e Madsen \(2010\)](#), p. 47-52 e 135).

## 5.9 Inclusão de outros arquivos

É uma boa prática dividir o seu documento em diversos arquivos, e não apenas escrever tudo em um único. Esse recurso foi utilizado neste documento. Para incluir diferentes arquivos em um arquivo principal, de modo que cada arquivo incluído fique em uma página diferente, utilize o comando:

```
\include{documento-a-ser-incluido}      % sem a extensão .tex
```

Para incluir documentos sem quebra de páginas, utilize:

```
\input{documento-a-ser-incluido}      % sem a extensão .tex
```

## 5.10 Compilar o documento L<sup>A</sup>T<sub>E</sub>X

Geralmente os editores L<sup>A</sup>T<sub>E</sub>X, como o TeXlipse<sup>6</sup>, o Texmaker<sup>7</sup>, entre outros, compilam os documentos automaticamente, de modo que você não precisa se preocupar com isso.

---

<sup>6</sup> <<http://texlipse.sourceforge.net/>>

<sup>7</sup> <<http://www.xm1math.net/texmaker/>>

No entanto, você pode compilar os documentos L<sup>A</sup>T<sub>E</sub>X usando os seguintes comandos, que devem ser digitados no *Prompt de Comandos* do Windows ou no *Terminal* do Mac ou do Linux:

```
pdflatex ARQUIVO_PRINCIPAL.tex
bibtex ARQUIVO_PRINCIPAL.aux
makeindex ARQUIVO_PRINCIPAL.idx
makeindex ARQUIVO_PRINCIPAL.nlo -s nomencl.ist -o ARQUIVO_PRINCIPAL.nls
pdflatex ARQUIVO_PRINCIPAL.tex
pdflatex ARQUIVO_PRINCIPAL.tex
```



## 6 Conclusão

Bom trabalho!



## 7 Cronograma de atividades

Nesse capítulo, é apresentado o quadro de tarefas previstas até a conclusão deste trabalho, juntamente com o período de tempo previsto para a realização das atividades.

A1 - Implementação do modelo neural profundo

A2 - Treinamento do modelo

A3 - Avaliação dos resultados obtidos;

A4 - Escrita da monografia.

Tabela 8: Cronograma de Atividades Restantes

	<b>Agosto</b>	<b>Setembro</b>	<b>Outubro</b>	<b>Novembro</b>	<b>Dezembro</b>
<b>A1</b>	X	X	X		
<b>A2</b>			X	X	
<b>A3</b>			X	X	X
<b>A4</b>				X	X



## Referências

- AFONSO, S. et al. Floresta sintá (c) tica: A treebank for portuguese. In: *LREC*. [S.l.: s.n.], 2002. Citado na página [43](#).
- ALUÍSIO, S. et al. An account of the challenge of tagging a reference corpus for brazilian portuguese. In: *Computational Processing of the Portuguese Language*. [S.l.]: Springer, 2003. p. 110–117. Citado na página [43](#).
- ARAUJO, L. C. *Configuração: uma perspectiva de Arquitetura da Informação da Escola de Brasília*. Dissertação (Mestrado) — Universidade de Brasília, Brasília, Março 2012. Citado na página [69](#).
- ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. *NBR 10520*: Informação e documentação — apresentação de citações em documentos. Rio de Janeiro, 2002. 7 p. Citado na página [64](#).
- ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. *NBR 6028*: Resumo - apresentação. Rio de Janeiro, 2003. 2 p. Citado na página [11](#).
- ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. *NBR 14724*: Informação e documentação — trabalhos acadêmicos — apresentação. Rio de Janeiro, 2011. 15 p. Citado 2 vezes nas páginas [67](#) e [70](#).
- ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. *NBR 6024*: Numeração progressiva das seções de um documento. Rio de Janeiro, 2012. 4 p. Citado 2 vezes nas páginas [70](#) e [71](#).
- COLLOBERT, R. Deep learning for efficient discriminative parsing. In: *International Conference on Artificial Intelligence and Statistics*. [S.l.: s.n.], 2011. Citado na página [27](#).
- COLLOBERT, R.; WESTON, J. A unified architecture for natural language processing: Deep neural networks with multitask learning. In: ACM. *Proceedings of the 25th international conference on Machine learning*. [S.l.], 2008. p. 160–167. Citado na página [62](#).
- COLLOBERT, R. et al. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, JMLR. org, v. 12, p. 2493–2537, 2011. Citado 3 vezes nas páginas [44](#), [46](#) e [61](#).
- DENG, L.; YU, D. Deep learning: methods and applications. *Foundations and Trends in Signal Processing*, Now Publishers Inc., v. 7, n. 3–4, p. 197–387, 2014. Citado na página [56](#).
- FONSECA, E. R.; ROSA, J. L. G. Mac-morpho revisited: Towards robust part-of-speech tagging. In: *Proceedings of the 9th Brazilian Symposium in Information and Human Language Technology*. [S.l.: s.n.], 2013. p. 98–107. Citado 3 vezes nas páginas [43](#), [61](#) e [62](#).

FONSECA, E. R.; ROSA, J. L. G.; ALUÍSIO, S. M. Evaluating word embeddings and a revised corpus for part-of-speech tagging in portuguese. *Journal of the Brazilian Computer Society*, Springer, v. 21, n. 1, p. 1–14, 2015. Citado 5 vezes nas páginas 27, 43, 44, 61 e 62.

JR, D. W. H.; LEMESHOW, S. *Applied logistic regression*. [S.l.]: John Wiley & Sons, 2004. Citado na página 32.

KEPLER, F. N. *Um etiquetador morfo-sintático baseado em cadeias de Markov de tamanho variável*. Tese (Doutorado) — Instituto de Matemática e Estatística da Universidade de São Paulo, 12/04/2005., 2005. Citado 2 vezes nas páginas 61 e 62.

LUND, K.; BURGESS, C. Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior Research Methods, Instruments, & Computers*, Springer, v. 28, n. 2, p. 203–208, 1996. Citado na página 44.

LUO, Q.; XU, W. Learning word vectors efficiently using shared representations and document representations. In: *Twenty-Ninth AAAI Conference on Artificial Intelligence*. [S.l.: s.n.], 2015. Citado na página 45.

MAATEN, L. Van der; HINTON, G. Visualizing data using t-sne. *Journal of Machine Learning Research*, v. 9, n. 2579-2605, p. 85, 2008. Citado na página 46.

MANNING, C. D.; SCHÜTZE, H. *Foundations of statistical natural language processing*. [S.l.]: MIT press, 1999. Citado na página 27.

MARQUIAFÁVEL, V. S. Um processo para a geração de recursos lingüísticos aplicáveis em ferramentas de auxílio à escrita científica. Biblioteca Digital de Teses e Dissertações da Universidade Federal de São Carlos, 2010. Citado na página 27.

MICHALSKI, R. S.; CARBONELL, J. G.; MITCHELL, T. M. *Machine learning: An artificial intelligence approach*. [S.l.]: Springer Science & Business Media, 2013. Citado na página 39.

MIKOLOV, T. et al. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013. Citado na página 45.

NG, A. *Course of Machine Learning*. [S.l.], 2015. Disponível em: <<https://www.coursera.org/learn/machine-learning/>>. Citado 11 vezes nas páginas 32, 33, 38, 39, 40, 41, 47, 48, 52, 54 e 55.

NG, A. et al. *Unsupervised feature learning and deep learning*. [S.l.]: Stanford, Tutorial, 2013. Citado na página 47.

PENNINGTON, J.; SOCHER, R.; MANNING, C. D. Glove: Global vectors for word representation. *Proceedings of the Empirical Methods in Natural Language Processing (EMNLP 2014)*, v. 12, p. 1532–1543, 2014. Citado na página 45.

SANTOS, C. N. dos; ZADROZNY, B. Training state-of-the-art portuguese pos taggers without handcrafted features. In: *Computational Processing of the Portuguese Language*. [S.l.]: Springer, 2014. p. 82–93. Citado 3 vezes nas páginas 27, 61 e 62.

SOCHER, R. *Deep Learning for Natural Language Processing*. 2015. Disponível em: <<http://cs224d.stanford.edu/>>. Citado 3 vezes nas páginas 45, 57 e 58.

- TEMPONI, C. N. O corpus anotado do português histórico: um avanço para as pesquisas em lingüística histórica do português. *Revista Virtual de Estudos da Linguagem: ReVEL*, v. 2, n. 3, p. 1, 2004. Citado na página 43.
- THE Penn Treebank Project (2014). [S.l.], 2014. Disponível em: <<http://www.cis.upenn.edu/~treebank/>>. Citado na página 43.
- TURIAN, J.; RATINOV, L.; BENGIO, Y. Word representations: a simple and general method for semi-supervised learning. In: ASSOCIATION FOR COMPUTATIONAL LINGUISTICS. *Proceedings of the 48th annual meeting of the association for computational linguistics*. [S.l.], 2010. p. 384–394. Citado 2 vezes nas páginas 43 e 46.
- van GIGCH, J. P.; PIPINO, L. L. In search for a paradigm for the discipline of information systems. *Future Computing Systems*, v. 1, n. 1, p. 71–97, 1986. Citado na página 65.
- VITERBI, A. J. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *Information Theory, IEEE Transactions on*, IEEE, v. 13, n. 2, p. 260–269, 1967. Citado na página 61.
- WILSON, P.; MADSEN, L. *The Memoir Class for Configurable Typesetting - User Guide*. Normandy Park, WA, 2010. Disponível em: <<http://ctan.tche.br/macros/latex/contrib/memoir/memman.pdf>>. Acesso em: 19.12.2012. Citado na página 72.



# Apêndices



# APÊNDICE A – Derivadas parciais do Gradiente Descendente

De acordo com a ABNT:

Apêndice (opcional): texto utilizado quando o autor pretende complementar sua argumentação. São identificados por letras maiúsculas e travessão, seguido do título. Ex.: APÊNDICE A - Avaliação de células totais aos quatro dias de evolução

Anexo (opcional): texto ou documento **não elaborado pelo autor** para comprovar ou ilustrar. São identificados por letras maiúsculas e travessão, seguido do título. Ex.: ANEXO A - Representação gráfica de contagem de células

Tais definições (e outras) podem ser encontradas na NBR 14724-2001 Informação e documentação - trabalhos acadêmicos<sup>1</sup>.

---

<sup>1</sup> <http://www.firb.br/abntmonograf.htm>



## APÊNDICE B -

Pode ser que tenha outro...



## Anexos



# ANEXO A – Fundamentos de Álgebra Linear

Sendo anexo, a formatação dessa seção é livre. Ou seja: aceita-se fonte diferente e menor



## ANEXO B – LaTex para Principiantes

TEste<sup>1</sup> Dentro dos arquivos .tex o texto pode estar organizado em partes, capítulos, seções, etc. conforme os seguintes comandos:

- `\part{Nome da Parte}`, partes do documento
- `\chapter{Nome}`, capítulos somente para arquivos do tipo *book* e *report*
- `\section{Nome}`, seções
- `\subsection{Nome}`, subseções
- `\subsubsection{Nome}`, seções dentro de subseções
- `\paragraph{Texto}`, parágrafos formatados
- `\ subparagraph{Texto}`, subparágrafos

**Parágrafos** Parágrafos são definidos deixando uma linha em branco entre os mesmos. Pode-se também forçar usando `\ \` bem como deixar uma linha em branco com um ~ sozinho na linha.

**Formato de texto** O tamanho do texto pode ser definido pelos comandos específicos: `tiny`, `scriptsize`, `footnotesize`, `small`, `normalsize`, `large`, `Large`, `huge` e `Huge`, conforme ilustra a Figura 29.

Command	Sample	Command	Sample
<code>\tiny</code>	<code>tiny</code>	<code>\scriptsize</code>	<code>scriptsize</code>
<code>\footnotesize</code>	<code>footnotesize</code>	<code>\small</code>	<code>small</code>
<code>\normalsize</code>	<code>normalsize</code>	<code>\large</code>	<code>large</code>
<code>\Large</code>	<code>larger</code>	<code>\LARGE</code>	<code>even larger</code>
<code>\huge</code>	<code>huge</code>	<code>\Huge</code>	<code>largest</code>

Figura 29: Exemplo de tamanhos de fonte

Perceba que a figura tem resolução ruim; deveria ser uma tabela

**Referências dentro do Texto** Partes do texto podem ser referenciadas através do par de comandos `\label` e `\ref`. Por exemplo, podemos inserir uma seção no artigo utilizando o seguinte comando:

```
\section{Seção principal}\label{sec:prcpal}
```

Vejam que o título da seção é seguido do comando `\label{nome}`. Esta seção pode ser referenciada em qualquer parte do texto, como o exemplo a seguir.

Conforme explicado na Seção `\ref{sec:prcpal}`, nosso método utiliza...

<sup>1</sup> (??)

## B.1 Outras Dicas

**Caracteres Especiais** Esses não podem ser usados no texto sem a barra à frente:  
`# $ % ^ & _ { } ~ e / .`

**Comentários** Comentários são precedidos de `%` e podem estar em qualquer parte do texto. Lembrando que tudo que estiver após `%` será considerado como comentário e ignorado pelo processador.

**Incluir Figuras** Incluir figuras no LaTeX é relativamente fácil quando se tem um formato de arquivo pré-definido. Por exemplo, neste documento, usa-se apenas figuras do tipo `pdf`, mas também poderia-se usar do tipo `png` (e `jpeg`, mas este tipo não é recomendado). A Tabela 9 ilustra as linhas que inserem uma figura no texto.

Tabela 9: Linhas de código para inserir figura

Linha de Código	Explicação
<code>\usepackage{graphicx}</code>	inclui pacote gráfico no início do documento
<code>\begin{figure}[tb]</code>	inicia figura, define sua posição no texto
<code>\centering</code>	centraliza a figura na página
<code>\includegraphics[scale=.7]</code>	define escala da figura
<code>{img/figura}</code>	inclui o arquivo da figura no texto
<code>\caption{Legenda}</code>	inclui a legenda da figura
<code>\label{fig:ap}</code>	inclui o apelido da figura
<code>\end{figure}</code>	termina figura

**Hifenização** Às vezes aparece uma palavra cuja hifenização, divisão silábica, está errada. Para resolver esse tipo de problema, pode-se recorrer à divisão manual da palavra, acrescentando `\-` entre cada sílaba: `Mi\-\re\-\lla`. Se, ao invés desta solução, você quiser evitar completamente que suas palavras sejam divididas, acrescente os dois comandos no início do seu documento (ou seja, antes do `\begin{document}`).

```
\hyphenpenalty=5000
\tolerance=1000
```

**BibTeX** Para editar facilmente o BibTeX, pode-se utilizar uma ferramenta própria<sup>2</sup>. A minha favorita é o JabRef<sup>3</sup>, ilustrado na Figure 30, porque:

- É de graça;
- Possui interface gráfica super intuitiva;
- Permite importar referências de bases clássicas, como ISI, Medline e RIS;

<sup>2</sup> Ferramentas para BibTeX: <http://dmoz.org/Computers/Software/Typesetting/TeX/BibTeX>

<sup>3</sup> JabRef Editor: <http://jabref.sourceforge.net/>

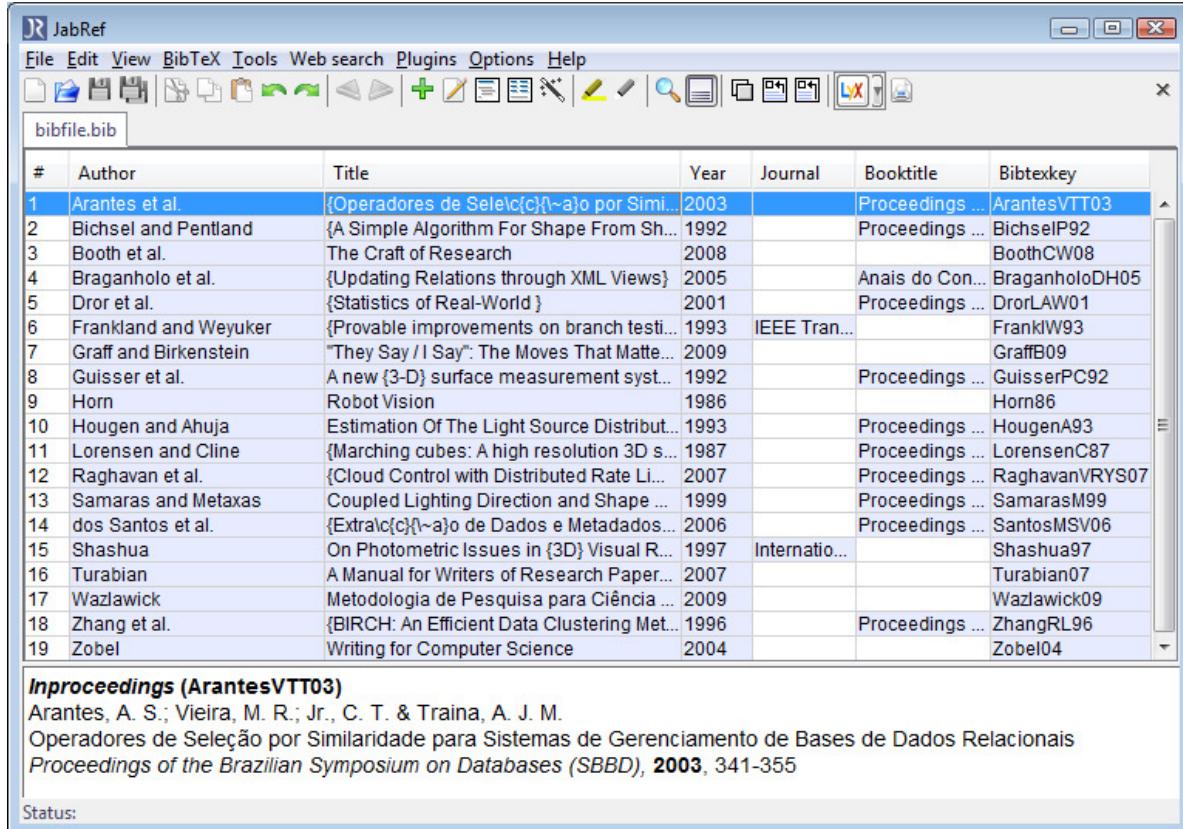


Figura 30: Tela do JabRef para uma versão inicial do arquivo bib deste documento

- Permite exportar para diferentes formatos, inclusive para um banco de dados utilizando SQL;
- Tem botão para procurar o artigo da respectiva referência e fazer o seu download;
- Permite adicionar comentários próprios para cada entrada;
- Pode-ser classificar as referências e criar grupos para as mesmas, e muito muito mais.

**Listas** Listas podem ser definidas com *bullets* ou com números, conforme os exemplos a seguir.

```
\begin{itemize}
\item Item 1 com bullet
\item Item 2 com bullet
\end{itemize}
```

```
\begin{enumerate}
\item Item 1 numerado
\item Item 2 numerado
\end{enumerate}
```

**Fontes Coloridas** Para adicionar texto em cores (muito útil para marcar trechos do texto que estão *em trabalho*, deve-se adicionar os pacotes *graphicx* e *color* (usando o comando `\usepackage` e depois utilizar o comando `\textcolor{cor}{texto}` para colorir o *texto* com a *cor* especificada. Por exemplo `\textcolor{blue}{texto em azul}`. Outras cores comuns são *red* e *green*.

**Para Economizar Espaço** Existem alguns *dirty tricks*<sup>4</sup> pra economizar espaço, como por exemplo:

- `\usepackage{times}` Usa fonte *Times* no lugar da default.
- `\usepackage[small,compact]{titlesec}` Modifica o título e os espaços antes/depois dos mesmos.
- `\usepackage[small,it]{caption}` Reduz o tamanho das legendas de tabelas e figuras.

**WEB** A Web é repleta de páginas e documentos sobre LaTe<sub>X</sub>. Alguns exemplos incluem:

- Favorito inglês: <<http://en.wikibooks.org/wiki/LaTeX/>>
- Favorito português: <<http://linorg.usp.br/CTAN/info/lshort/portuguese/pt-lshort.pdf>>
- <<http://www.mat.ufmg.br/~regi/topicos/intlat.pdf>>
- <<http://www.duke.edu/~hg9/ctex/LaTeXManual.pdf>>
- <<http://minerva.ufpel.tche.br/~campani/cursolatex.pdf>>
- <<http://www.personal.ceu.hu/tex/words.htm>>

---

<sup>4</sup> Ou seja, eles irão alterar a formatação dada pelo estilo default do texto.

# Índice

- Adobe Illustrator, 66
- Adobe Photoshop, 67
- alíneas, 70
- citações
  - diretas, 63
  - simples, 64
- CorelDraw, 66
- DDV, 61, 62
- espaçamento
  - do primeiro parágrafo, 71
  - dos parágrafos, 71
  - entre as linhas, 72
  - entre os parágrafos, 71
- expressões matemáticas, 69
- FDV, 44
- Fig., 63
- figuras, 65
- filosofia, 65
- Gimp, 67
- GloVe, 45, 62
- HAL, 44, 61, 62
- incisos, 70
- InkScape, 66
- NLM, 44, 61, 62
- PLN, 27, 44, 56, 57
- POS, 15, 17, 27, 28, 40, 42, 57, 58, 62
- SG, 44, 61, 62
- subalíneas, 70
- tabelas, 64
- TCC, 63