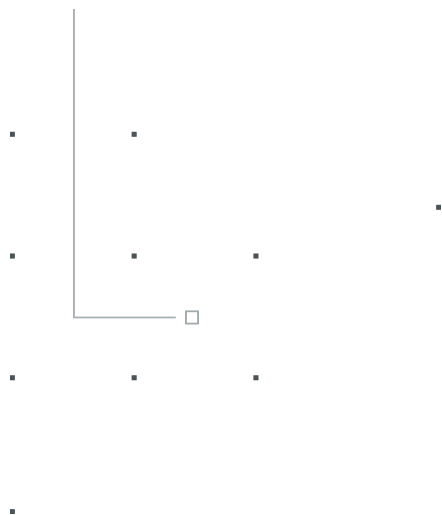


FIAP

NBA



# Embeddings

Dheny R. Fernandes

## **1. Embeddings**

1. Definição
2. Motivação

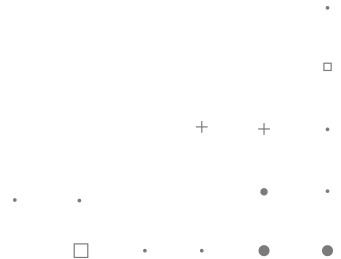
## **2. Word2Vec**

1. Abstração
2. Motivação
3. Conceituação
4. Negative Sampling

## **3. Demo Word2Vec**

## **4. Case End-to-end**

# Embeddings



Nas representações que vimos até aqui, descobrimos várias fraquezas que fazem com que esses modelos se tornem impraticáveis de serem usados em grandes conjuntos de dados. Podemos elencar os principais problemas que discutimos:

- Alta dimensionalidade dos dados
- Falta de representação semântica
- Perda de significado
- Entre outras..

Para resolver esse problema, métodos para **aprender uma representação de baixa dimensão** foram sugeridos.

Para entendermos o conceito dessas representações, também chamadas de Representações Distribuídas, precisamos entender alguns pontos antes.

O primeiro deles é o de **Similaridade Distribucional**, que representa a ideia de que o significado de uma palavra pode ser entendido a partir do contexto em que aparece.

Isto é conhecido também como **conotação**, ou seja, o significado é definido pelo contexto. Diferente de **denotação**, que é o significado literal de uma palavra.

O segundo ponto é a **Hipótese Distribucional**, que diz que **palavras que aparecem em contextos similares possuem significados similares**. Assim, se palavras podem ser representadas por vetores, então duas palavras que aparecem em contextos similares devem possuir vetores similares.

O terceiro refere-se à **Representação Distribucional**, que são os vetores de representação que vimos até o presente momento, **caracterizados pela alta dimensão e grande esparsidade**.

Por fim, todos esses conceitos nos levam à **Representação Distribuída**, que são vetores compactos (baixa dimensão) e densos (não-esparcos). Daqui, surgiu o conceito de *Word Embeddings*.

- Como representamos o significado de uma palavra?
- Definição: significado ([dicionário](#))
  - Definição atribuída a um termo, palavra, frase, texto; acepção.
  - Aquilo que alguma coisa quer dizer; sentido.
  - Uma ideia que é representada por uma palavra, frase, etc.



- A maioria das representações de texto são discretas e possuem alguns problemas:
  - Perda de **nuances**: sinônimos – *apto, bom, expert, proficiente*.
  - Perde **novas palavras** (impossível de manter atualizado): *fodão, ninja*
  - **Subjetivo** (não leva em consideração contexto)
  - Necessita **trabalho humano** para criar e adaptar
  - Difícil calcular **similaridade** de palavras

- A maioria dos algoritmos estatísticos e baseados em regras de NLP tratam palavras como símbolos atômicos: *hotel, conferência, andar*
- Em termos de vetores espaciais, é um vetor com um 1 e muitos 0's.
- Chamamos essa representação de *one-hot vector*.
- Problema:
  - Hotel: [0 0 0 0 0 0 1 0 0 0 0] AND
  - Motel: [0 0 0 0 1 0 0 0 0 0 0] = 0

- Assim, precisamos de uma representação que consiga capturar tais relações entre diferentes palavras levando em consideração o contexto.
- Para resolver esse problema, usaremos **representações baseadas em similaridade distribucional**.
- “You shall know a word by the company it keeps” (J.R.Firth, 1957)

- Podemos obter muito valor representando uma palavra por meio de seus vizinhos.

government debt problems turning into banking crises as has happened in  
saying that Europe needs unified banking regulation to replace the hodgepodge



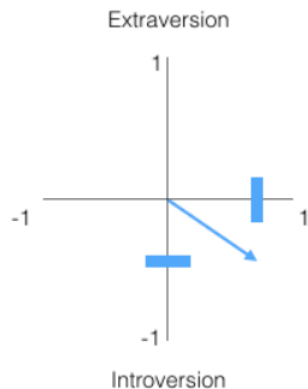
Estas palavras irão representar 'banking'



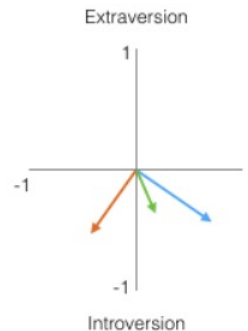
- Mas como representar uma palavra por meio de seus vizinhos?
- Vejamos um exemplo:
  - Numa escala de 0 a 100, quão extrovertido uma pessoa é?
  - Suponha que o escore tenha sido 38/100. Então, podemos representar isso graficamente:



- Vamos adicionar mais um traço de personalidade:



De certa forma,  
esse vetor  
representa a  
personalidade de  
alguém



Com essa representação,  
fica fácil realizar  
comparação entre  
pessoas

- Para obter essa similaridade, posso utilizar a similaridade do cosseno:

$$\text{cosine\_similarity}\left(\begin{array}{|c|c|} \hline \text{Jay} \\ \hline -0.4 & 0.8 \\ \hline \end{array}, \begin{array}{|c|c|} \hline \text{Person \#1} \\ \hline -0.3 & 0.2 \\ \hline \end{array}\right) = 0.87$$

$$\text{cosine\_similarity}\left(\begin{array}{|c|c|} \hline \text{Jay} \\ \hline -0.4 & 0.8 \\ \hline \end{array}, \begin{array}{|c|c|} \hline \text{Person \#2} \\ \hline -0.5 & -0.4 \\ \hline \end{array}\right) = -0.20$$

- Diante disso, duas ideias principais surgem:
  1. Podemos representar pessoas e coisas como um vetor de números
  2. Podemos facilmente calcular quão similar dois vetores são.

# Word2Vec





- Como vimos, **qualquer objeto pode ser representado através de vetores**. Por hora, vamos olhar para vetores de palavras já treinados (depois vamos entender como chegar nesses vetores e o que ele representa) para entender sua principais propriedades

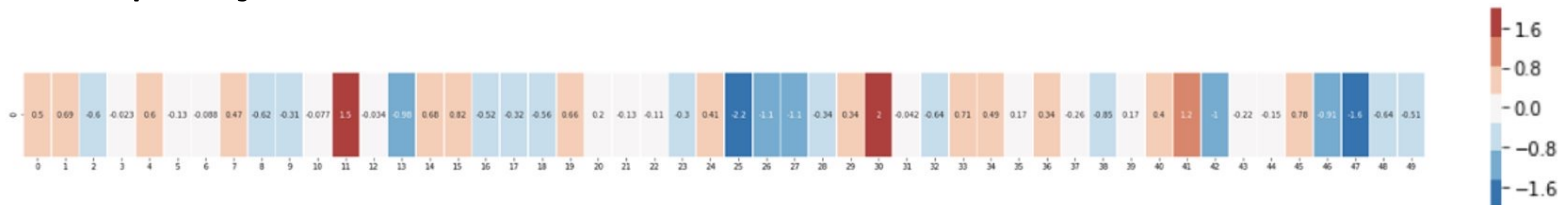
- Este vetor representa a palavra “King”:

```
[ 0.50451 , 0.68607 , -0.59517 , -0.022801, 0.60046 , -0.13498 , -0.08813 , 0.47377 , -0.61798 , -0.31012 ,  
-0.076666, 1.493 , -0.034189, -0.98173 , 0.68229 , 0.81722 , -0.51874 , -0.31503 , -0.55809 , 0.66421 , 0.1961  
, -0.13495 , -0.11476 , -0.30344 , 0.41177 , -2.223 , -1.0756 , -1.0783 , -0.34354 , 0.33505 , 1.9927 ,  
-0.04234 , -0.64319 , 0.71125 , 0.49159 , 0.16754 , 0.34344 , -0.25663 , -0.8523 , 0.1661 , 0.40102 , 1.1685 ,  
-1.0137 , -0.21585 , -0.15155 , 0.78321 , -0.91241 , -1.6106 , -0.64426 , -0.51042 ]
```

- É uma lista de 50 números. Vamos coloca-los numa única linha para poder compará-la com vetores de outras palavras:



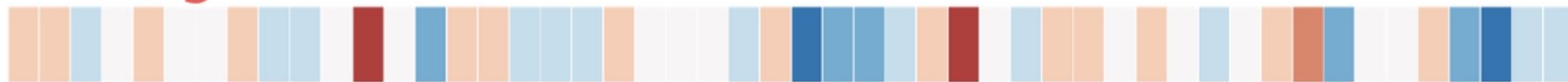
- Podemos colorir cada uma das células para melhor visualização e comparação:



- Vamos deixar os números de lado e focar somente nas cores.

- Comparando com outros vetores de palavras:

“king”



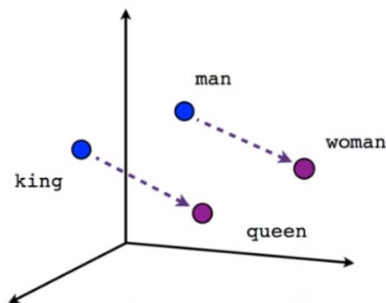
“Man”



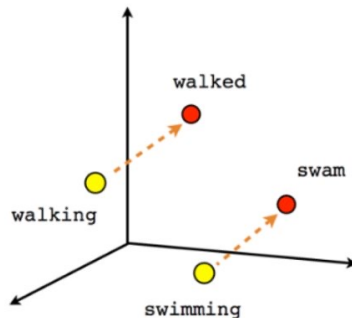
“Woman”



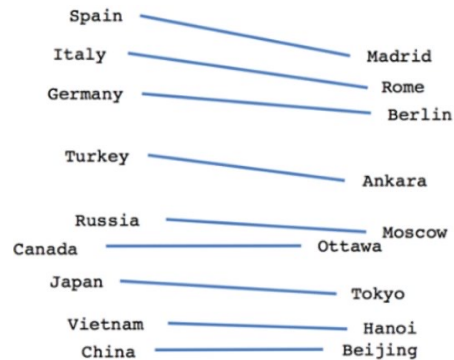
- Esse tipo de representação vetorial é que nos permite estabelecer relações como as seguintes:



Male-Female



Verb tense



Country-Capital

- E mais, eu posso realizar operações algébricas do tipo:  $v[\text{'king'}] - v[\text{'man'}] + v[\text{'woman'}] = v[\text{'queen'}]$

king - man + woman  $\approx$  queen



- Dissemos anteriormente que usando *one-hot vectors* não conseguimos capturar as relações semânticas entre palavras.
- Uma abordagem é usar **matriz de co-ocorrência**.
- Considere esses exemplos:
  - I like deep learning
  - I like NLP
  - I enjoy flying

- A matriz de co-ocorrência para esse exemplo é essa:

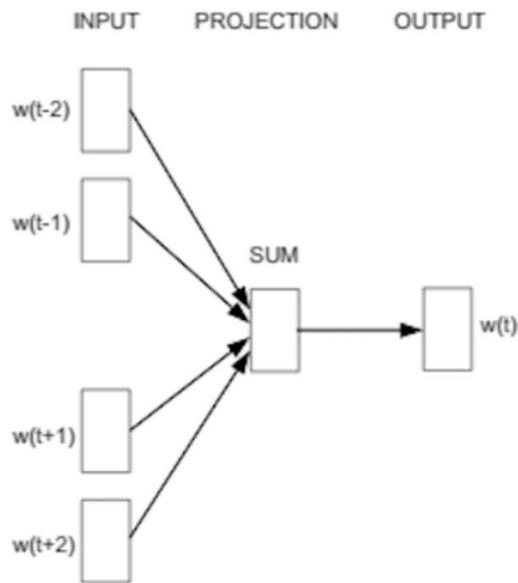
counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0



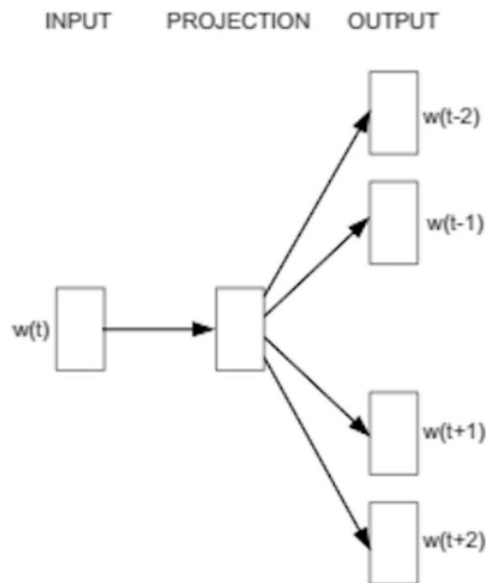
- Entretanto, há vários problemas relacionados à essa representação:
  - **Aumenta** em tamanho de acordo com o vocabulário
  - **Alta dimensionalidade**: requer muito armazenamento
  - Classificadores possuem problemas com **esparsidade**
- Solução?

- Armazenar apenas as características mais importantes – menor número dimensões = vetor denso.
- Geralmente entre 25 – 1000 dimensões
- Ao invés de tentar reduzir a dimensionalidade de matrizes de co-ocorrência (usando SVD, por exemplo), vamos diretamente aprender vetores de palavras de menores dimensões.
- Essa é a ideia do Word2Vec

- Word2Vec pode criar vetores densos a partir de duas abordagens: *Continuous Bag-of-word (CBOW)* e *Skip-gram*:



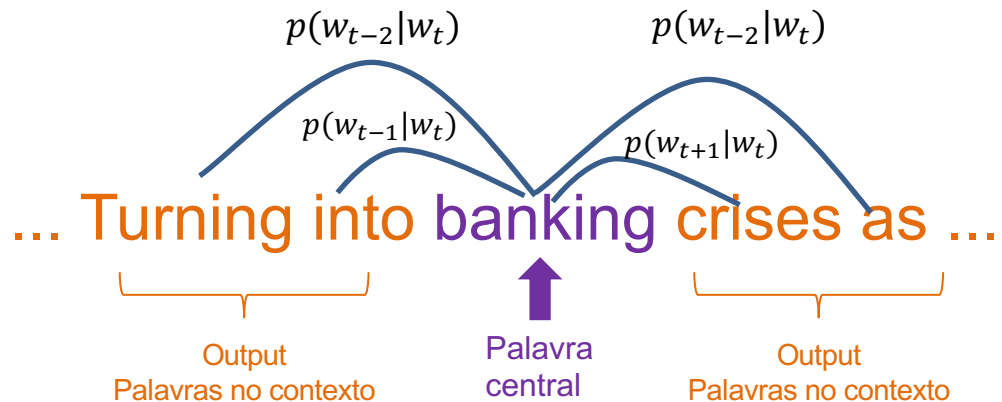
**CBOW**



**Skip-gram**

- Aqui, vamos explorar a abordagem Skip-gram.
- Ideia básica:
  - Definimos um modelo cujo objetivo é **predizer o contexto (outras palavras) a partir de uma palavra  $w_t$**  em termos de vetores de palavras.
  - $p(\text{contexto}|w_t)$
  - Que possui uma determinada loss function, e.g:
    - $J = 1 - p(w_{-t}|w_t)$
  - Continue ajustando esse vetor de palavras para minimizar a loss.

## Esquema de predição usando Skip-gram:



- Para cada palavra  $t = 1 \dots T$  prediga as palavras ao redor de  $t$ .
  - ‘ao redor’ é definido por uma janela de tamanho  $m$
- Loss Function:
  - Maximize a probabilidade de qualquer palavra de contexto dada a palavra central atual
  - $J(\theta) = \prod_{t=1}^T \prod_{-m \leq j \leq m} p(w_{t+j} | w_t; \theta)$
  - $\theta$  será o vetor de palavras

- As pessoas de ML gostam de minimizar funções e, para facilitar os cálculos, utilizamos o log da probabilidade negativa (*negative log likelihood*):

$$-j(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{-m \leq j \leq m} \log P(w_{t+j} | w_t)$$

- Entretanto, como prever palavras ao redor de uma palavra central com uma janela  $m$ ?

$$- p(o|c) = \frac{\exp(u'_o v_c)}{\sum_{w=1}^v \exp(u'_w v_c)}$$

- Em que  $o$  é o índice das palavras a serem preditas,  $c$  é o índice da palavra central,  $v_c$  e  $u_o$  são vetores de índices  $c$  e  $o$

- $$p(o|c) = \frac{\exp(u'_o v_c)}{\sum_{w=1}^v \exp(u'_w v_c)}$$
- Produto escalar:  
$$- u'_o v_c = \sum_{i=1}^n u_i v_i$$
- A forma é de  $p(o|c)$  é chamada de forma de Softmax e é o jeito padrão de converter números em distribuição de probabilidades

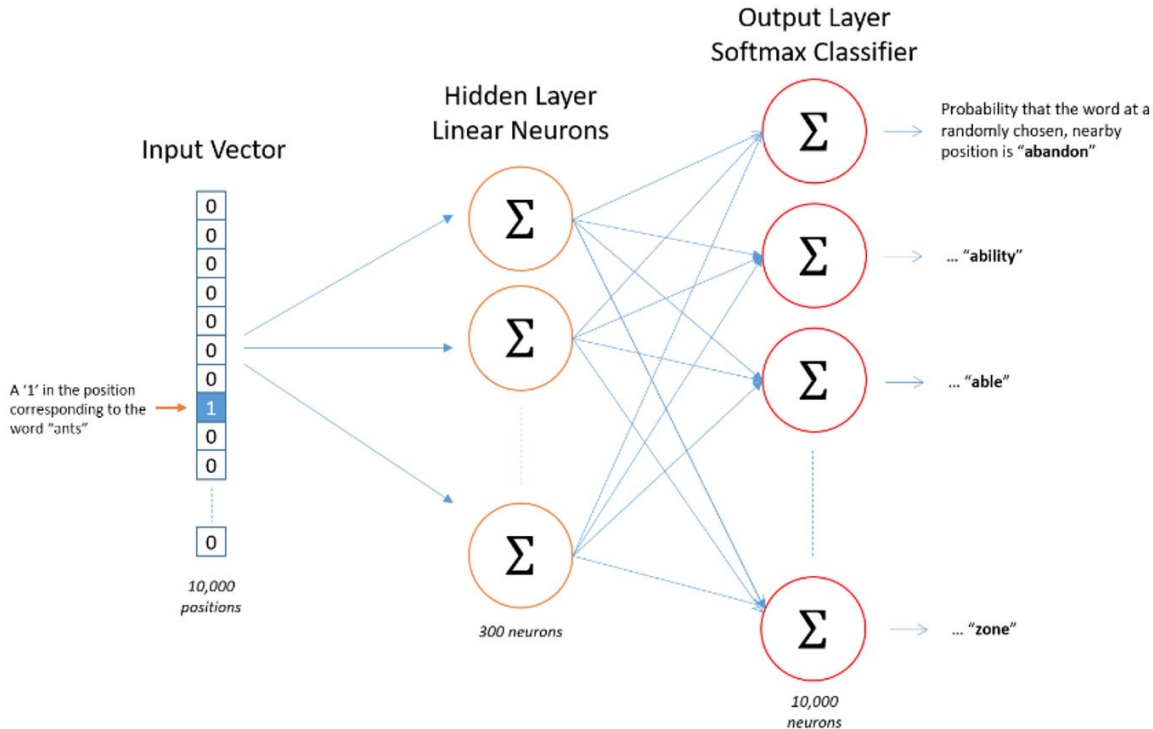


$$\bullet \quad p(o|c) = \frac{\exp(u'_o v_c)}{\sum_{w=1}^v \exp(u'_w v_c)}$$

Eleva ao  
expoente  
para tornar  
positivo

Normaliza  
para retornar  
probabilidade

- Visualmente, temos o seguinte:



- Pela imagem, estamos usando um vetor de pesos de 300 dimensões (neurônios). Assim, considerando um vocabulário de 10000 palavras, a camada oculta da rede será representada por uma matriz de 10000 x 300.
- Os pesos (300 neurônios) da camada oculta serão, iterativamente, ajustados para projetar o resultado final, ou seja, a probabilidade de cada palavra estar no contexto m da palavra central

- Então, o que queremos, de fato, ou seja, o vetor de representação de uma palavra, é esse **vetor de 300 dimensões com pesos ajustados segundo uma função de distribuição de probabilidade**.
- Esse vetor representa a **relação da palavra central com as palavras do vocabulário**.
- Mas o que fazer para **ajustar** esse vetor de 300 dimensões?
  - Utilizaremos Gradiente Descendente Estocástico e calcularemos as derivadas.
  - A demonstração dos cálculos não faz parte do escopo da aula, mas pode ser vista [aqui](#).

- Na arquitetura ilustrada anteriormente, construímos um vocabulário com 10000 palavras e uma rede com 300 neurônios, ou seja, tanto a camada oculta quanto a camada de saída terão uma matriz com **3 milhões de pesos**.
- Executar o SGD nessa rede se torna uma tarefa muito complexa e lenta e, pensando nesse tipo de estrutura, é necessário uma grande quantidade de dados de treinamento para evitar **overfitting**.

- Assim, precisamos de um método que melhore a etapa de treinamento.
- Para endereçar esse problema, os autores utilizaram uma técnica chamada *Negative Sampling*.

- Treinar uma rede neural significa pegar um exemplo de treinamento e ajustar todos os pesos dos neurônios de modo que ele preveja que a amostra de treinamento seja mais precisa.
- Como mencionado, a rede neural do *Skip-gram* possui um número gigantesco de pesos, os quais serão atualizados por cada uma das milhares de amostras de treinamento

- *Negative Sampling* resolve esse problema modificando para cada amostra de treinamento uma **pequena porcentagem dos pesos ao invés deles todos**.
- Quando na etapa de treinamento eu tenho um par (contexto, central) a saída correta é um *one-hot vector*, ou seja, 1 para 'contexto' e 0 para as demais.
- Com o *negative sampling*, vamos aleatoriamente selecionar um pequeno número (5, digamos) de palavras 'negativas' para atualizar os pesos ('negativa' significa uma palavra que queremos que a rede produza um 0)



- Ainda continuaremos atualizando os pesos para a palavra 'positiva'.
- Assim, vamos ajustar os pesos da palavra positiva mais os pesos de 5 palavras negativas, o que corresponde a um total de 6 neurônios e 1800 pesos no total. Isso é apenas 0,06% dos 3M de pesos da camada de saída.

- Como selecionamos tais amostras?
  - Elas são selecionadas usando uma ‘distribuição unigrama’, em que palavras mais frequentes são mais prováveis de serem selecionadas como *negative sampling*.
- Supondo um vocabulário de tamanho  $n$ , a probabilidade de selecionar aleatoriamente uma palavra é dada por:

$$- p(w_i) = \frac{f(w_i)}{\sum_{j=0}^n f(w_j)}$$

- Os autores afirmam que testaram diversas variações desta equação e aquele que obteve o melhor resultado foi a que elevou a contagem de palavras à potência de  $3/4$ .

- $$p(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=0}^n f(w_j)^{3/4}}$$


- Por que?
  - is:  $0,9^{3/4} = 0,92$
  - Constitution:  $0,09^{3/4} = 0,16$
  - Bombastic:  $0,01^{3/4} = 0,032$

# Demo e Case End-to-End



# Obrigado!

profdheny.fernandes@fiap.com.br

 /dhenyfernandes

FIAP MBA<sup>+</sup>

Copyright © 2022 | Professor Dheny R. Fernandes

Todos os direitos reservados. Reprodução ou divulgação total ou parcial deste documento, é expressamente proibido sem consentimento formal, por escrito, do professor/autor.

FIAP