# Tutorial de Deep Learning

**Luiz Gustavo Hafemann**
LIVIA
École de Technologie Supérieure – Montréal

# Organização do tutorial

**Conceitos básicos para treinamento de redes neurais convolucionais**

**Orientado para aplicar métodos na prática:**

- Apresentação: ~30 minutos
- Exercício prático: 3-4h

# Organização do tutorial

- **Dia 1:**
  - Motivação - aplicações
  - Introdução à aprendizagem de máquina
  - Computação simbólica com Theano
- **Dia 2**
  - Redes neurais convolucionais

  **Dia 3**
  - Transfer Learning

# Motivação

# Motivação

## Deep learning

- Modelos de aprendizagem de máquina (em geral redes neurais) com arquitetura profunda - múltiplas camadas

# Motivação

## Deep learning

- Modelos de aprendizagem de máquina (em geral redes neurais) com arquitetura profunda - múltiplas camadas
- Atualmente são o estado-da-arte em vários problemas, principalmente em visão computacional e processamento de linguagem natural:
  - Classificação de imagens, localização de objetos, entre outros

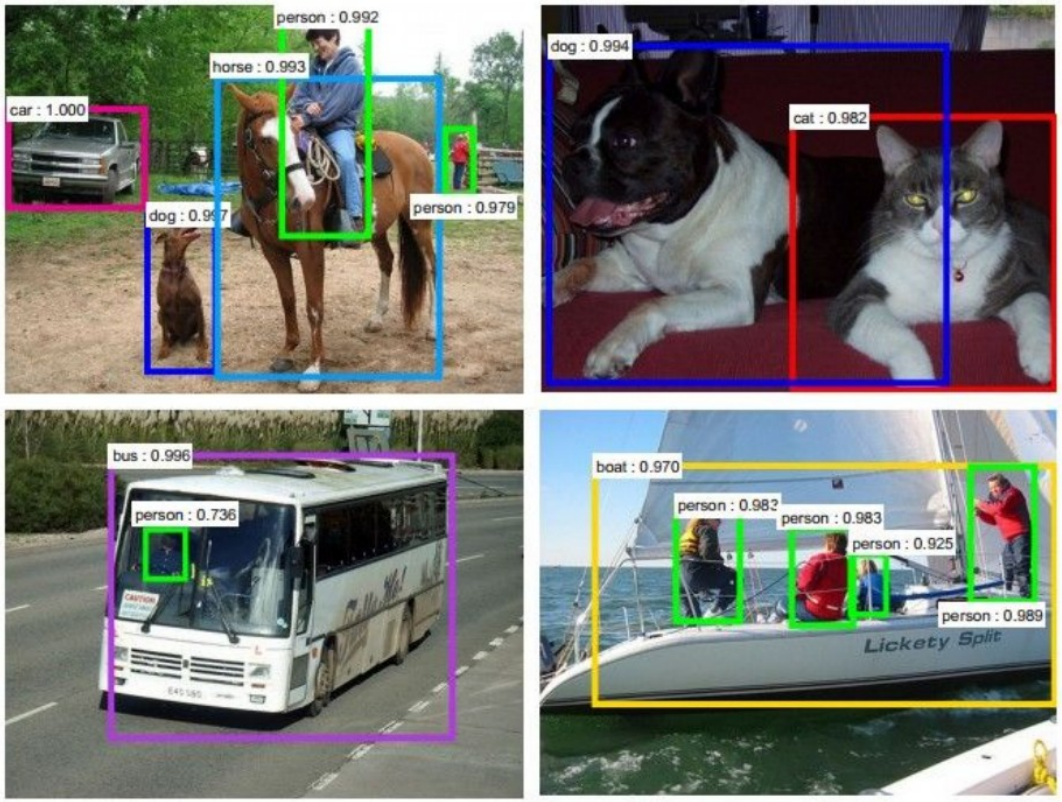
# Aplicações

## Classificação de imagens, busca de imagens semelhantes


(Krizhevsky 2012)

# Outras aplicações

Localização de objetos




Ren et al. 2015

Classificação de placas de trânsito

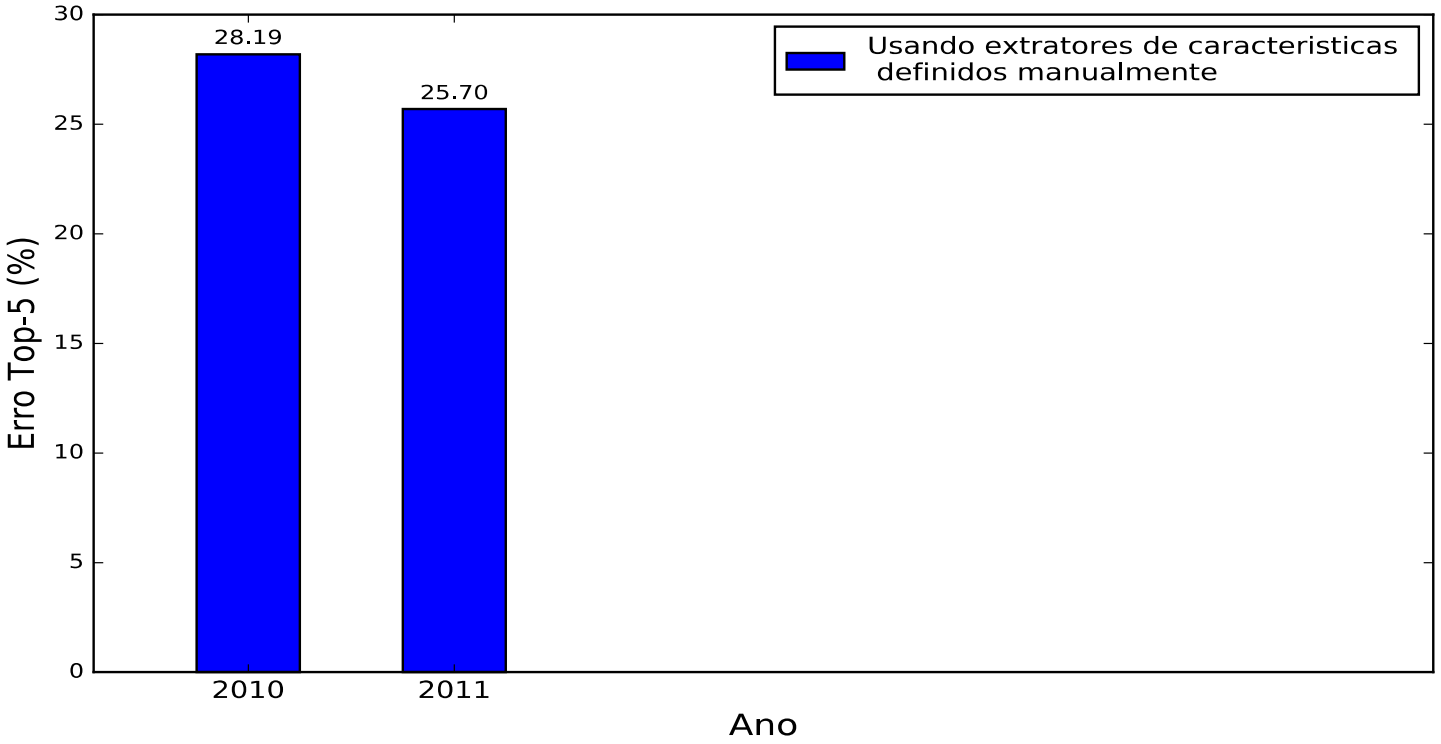
Ciresan et al. 2011

# Melhora de performance usando CNNs

## Base de dados Imagenet

Mais de 1 milhão de imagens

1000 classes

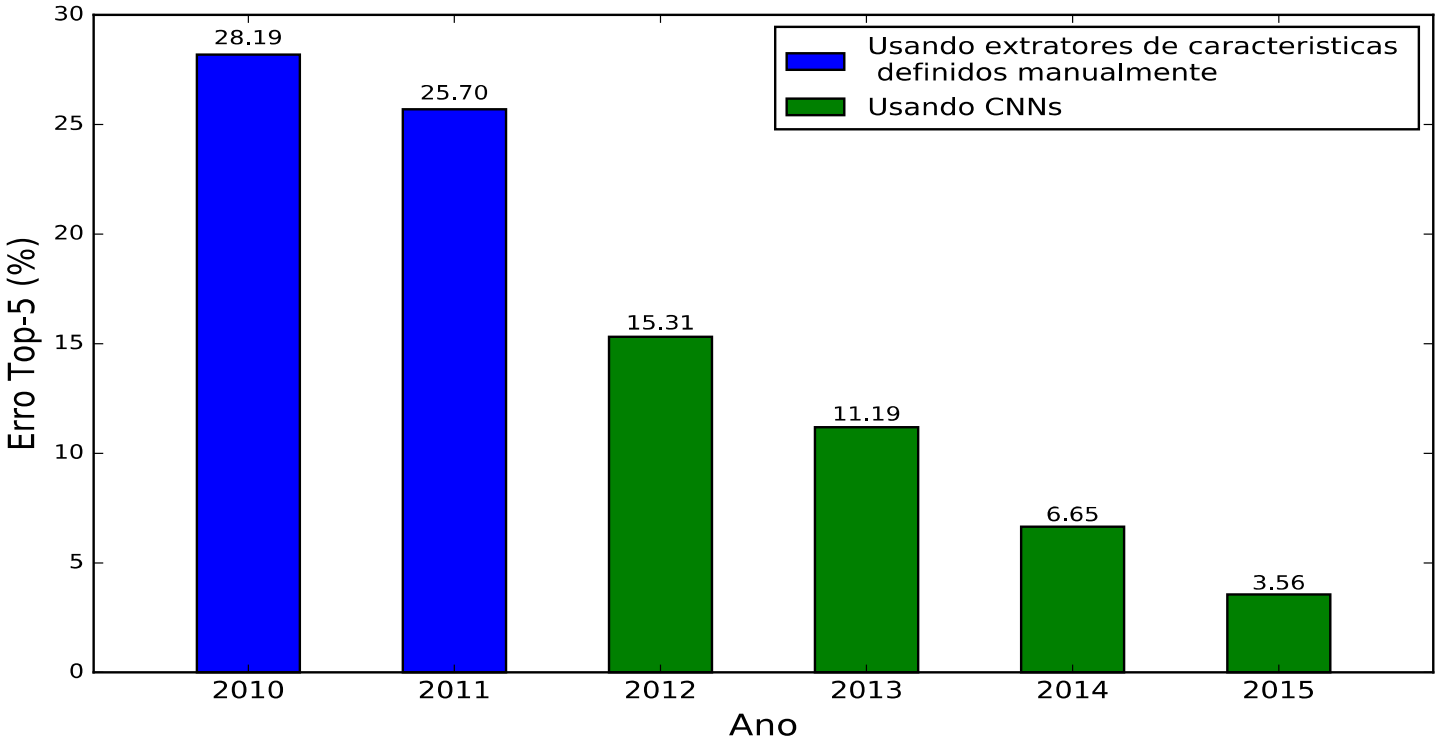


# Melhora de performance usando CNNs

## Base de dados Imagenet

Mais de 1 milhão de imagens

1000 classes



# Introdução à aprendizagem de máquina

# Introdução à aprendizagem de máquina

## Aprender funções à partir de dados

# Introdução à aprendizagem de máquina

**Aprender funções à partir de dados**

- **Aprendizagem supervisionada:**
  - Classificação: classificar SPAM, reconhecimento de objetos
    - Saída categórica: $y = f(\mathbf{x})$ $y \in \{y_1, ..., y_n\}$

# Introdução à aprendizagem de máquina

**Aprender funções à partir de dados**

- **Aprendizagem supervisionada:**
  - Classificação: classificar SPAM, reconhecimento de objetos
    - Saída categórica: $y = f(\mathbf{x}) \qquad y \in \{y_1, ..., y_n\}$
  - Regressão: Preço de imóveis, análise de series de dados
    - Saída contínua: $y = f(\mathbf{x}) \qquad y \in \mathbb{R}$

# Introdução à aprendizagem de máquina

**Aprender funções à partir de dados**

- **Aprendizagem supervisionada:**
  - Classificação: classificar SPAM, reconhecimento de objetos
    - Saída categórica: $y = f(\mathbf{x}) \qquad y \in \{y_1, ..., y_n\}$
  - Regressão: Preço de imóveis, análise de series de dados
    - Saída contínua: $y = f(\mathbf{x}) \qquad y \in \mathbb{R}$

- **Aprendizagem não-supervisionada**
  - Clustering (agrupamento), detecção de anomalias

# Aprendizagem supervisionada - classificação

# Aprendizagem supervisionada - classificação

## Formulação do problema

# Aprendizagem supervisionada - classificação

## Formulação do problema

**Dado um conjunto de exemplos $(\mathbf{x}^{(i)}, y^{(i)})$, onde**

# Aprendizagem supervisionada - classificação

## Formulação do problema

**Dado um conjunto de exemplos $(\mathbf{x}^{(i)}, y^{(i)})$, onde**

$\mathbf{x}^{(i)} = [x_1, x_2, ..x_n]$ **são medidas de entrada**

# Aprendizagem supervisionada - classificação

## Formulação do problema

**Dado um conjunto de exemplos $(\mathbf{x}^{(i)}, y^{(i)})$, onde**

$\mathbf{x}^{(i)} = [x_1, x_2, ..x_n]$ **são medidas de entrada**

$y^{(i)}$ **é a classe correta**

# Aprendizagem supervisionada - classificação

## Formulação do problema

**Dado um conjunto de exemplos $(\mathbf{x}^{(i)}, y^{(i)})$, onde**

$\mathbf{x}^{(i)} = [x_1, x_2, ..x_n]$ **são medidas de entrada**

$y^{(i)}$ **é a classe correta**

**O objetivo é aprender uma função para estimar y dado x:**

$$\hat{y} = f(\mathbf{x})$$

**Que generalize para novas entradas x**

# Aprendizagem supervisionada - classificação

**Exemplo:**

Problema de 2 classes: classificar um peixe entre truta e salmão:

$$y \in \{\text{truta}, \text{salmão}\}$$

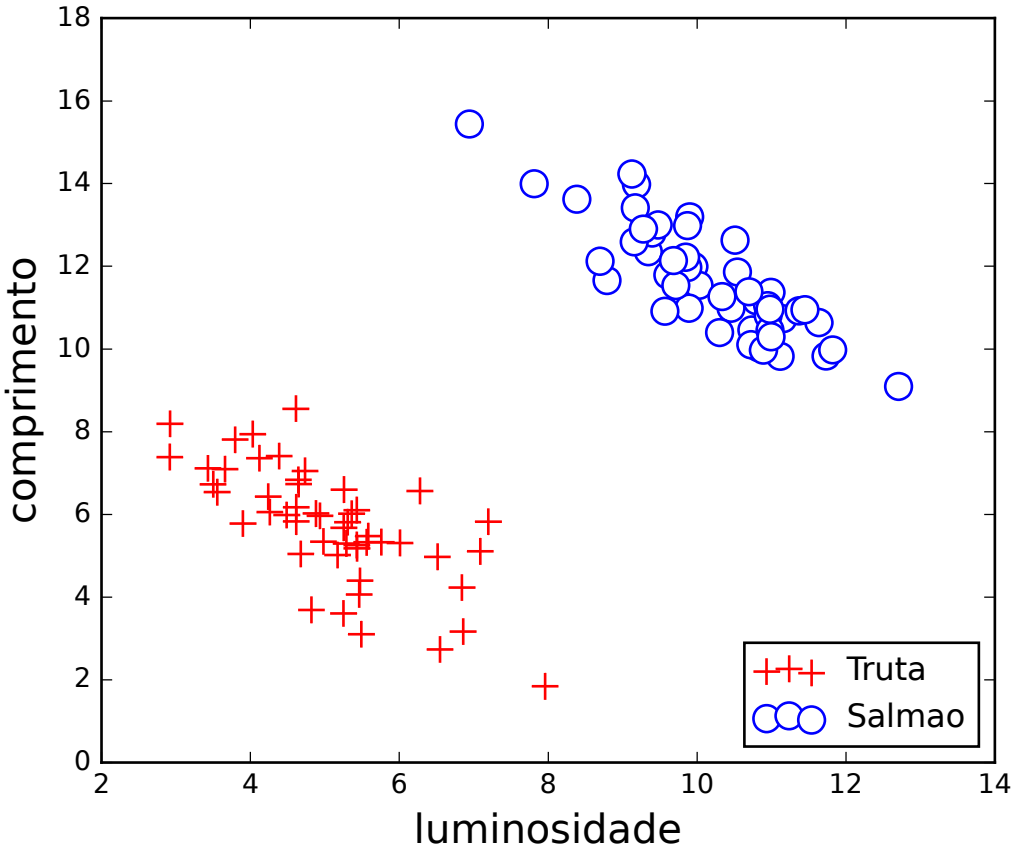
Duas medidas de entrada: comprimento e luminosidade:

$$\mathbf{x} = \{x_1, x_2\}$$

Objetivo: aprender um classificador que, dado medidas de um novo exemplo, diga à qual classe ele pertence.

# Aprendizagem supervisionada - classificação

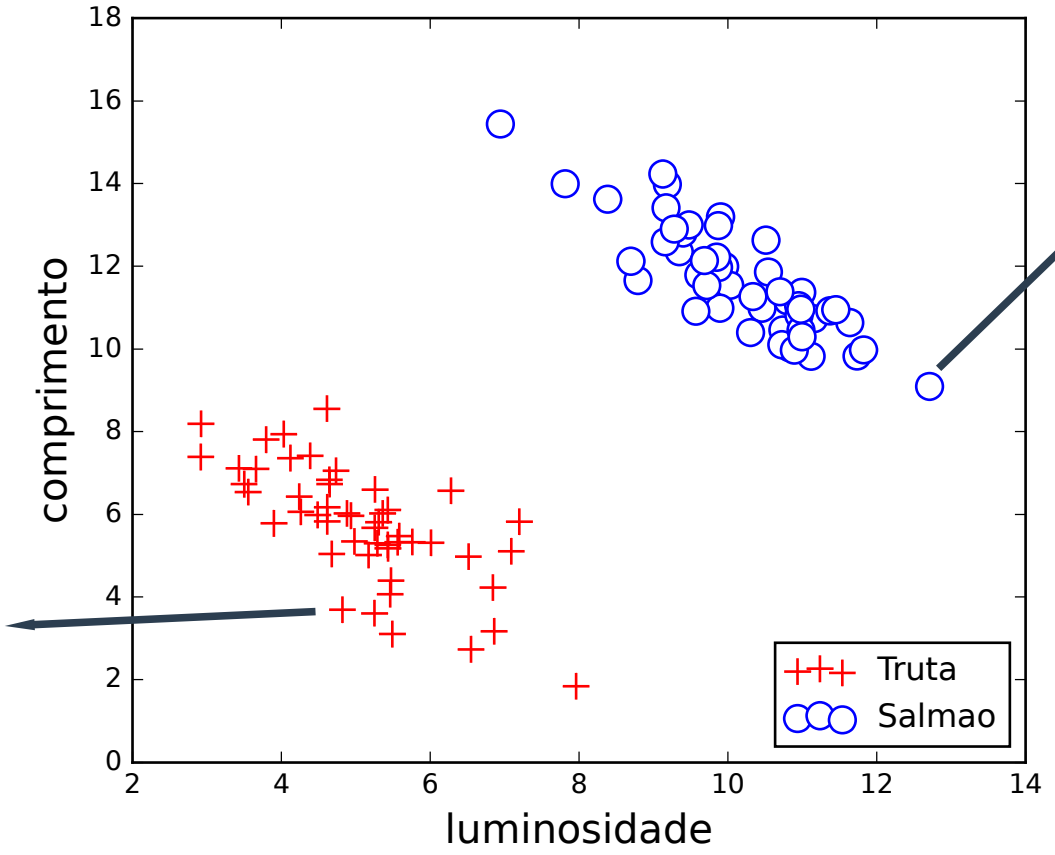
## 50 exemplos são coletados para cada classe:



# Aprendizagem supervisionada - classificação

**50 exemplos são coletados para cada classe:**

$\mathbf{x} = [4.3, 3.8]$
$y = 0$

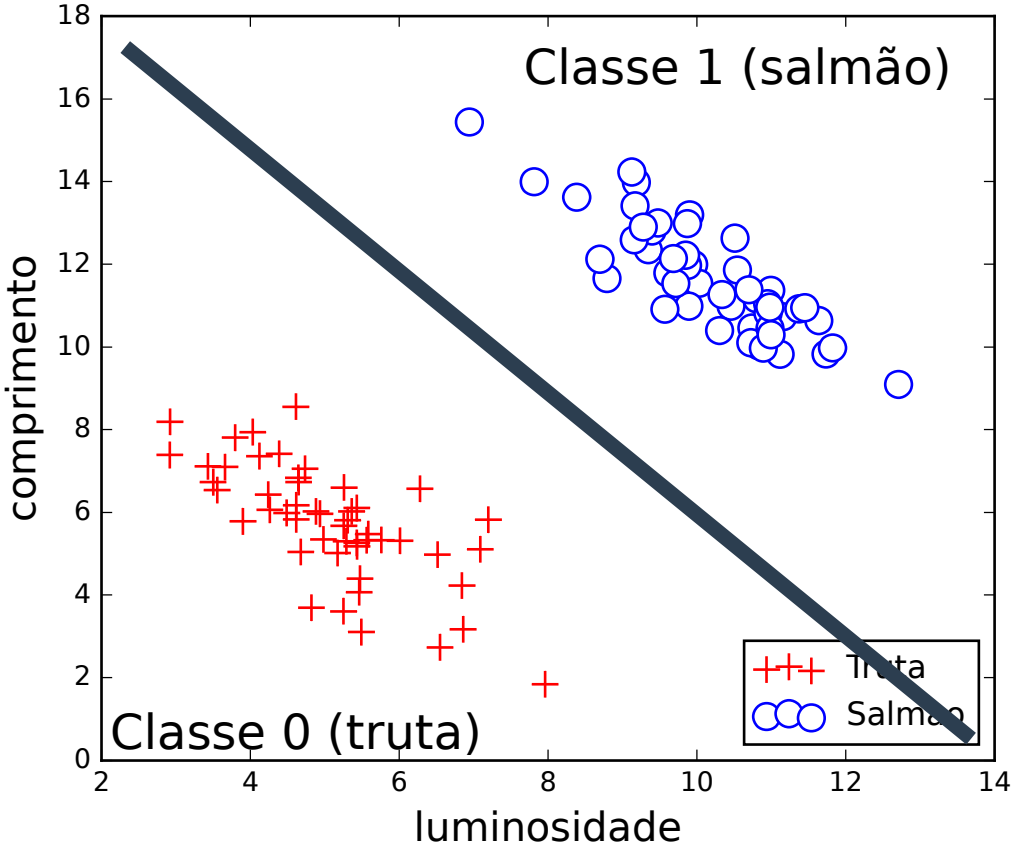



$\mathbf{x} = [12.5, 9]$
$y = 1$

# Aprendizagem supervisionada - classificação

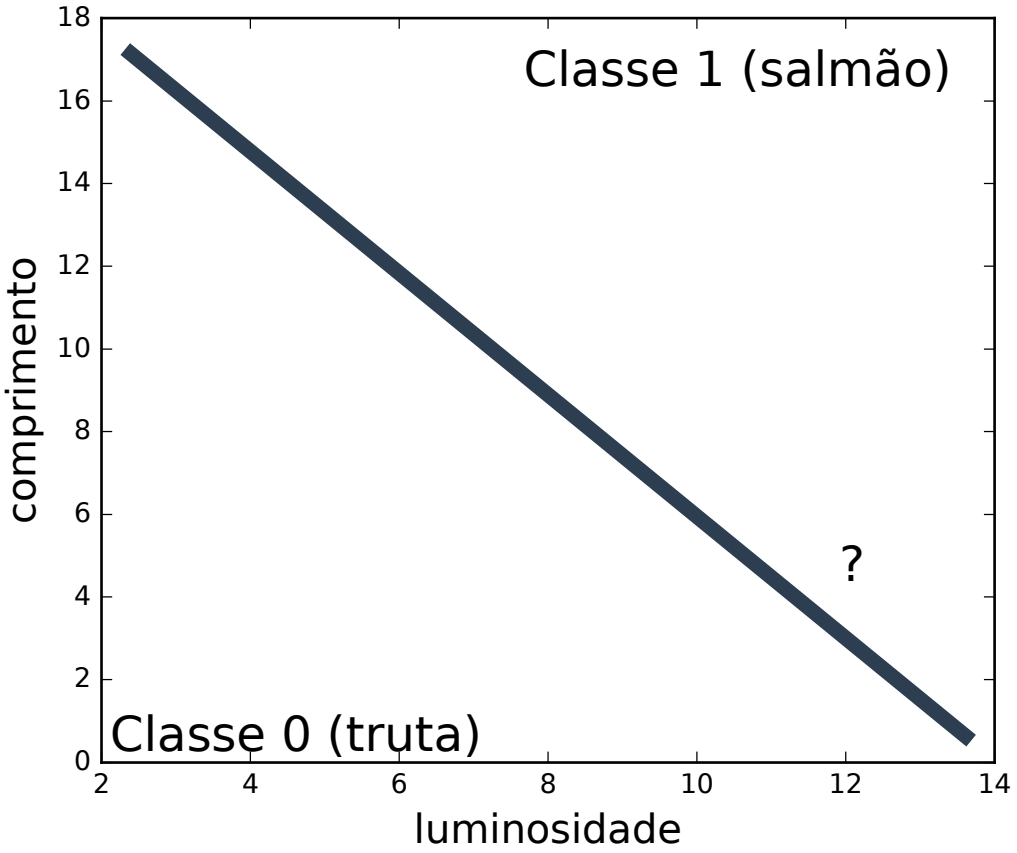
## Treinamento de um modelo

(nesse exemplo, um modelo paramétrico:)

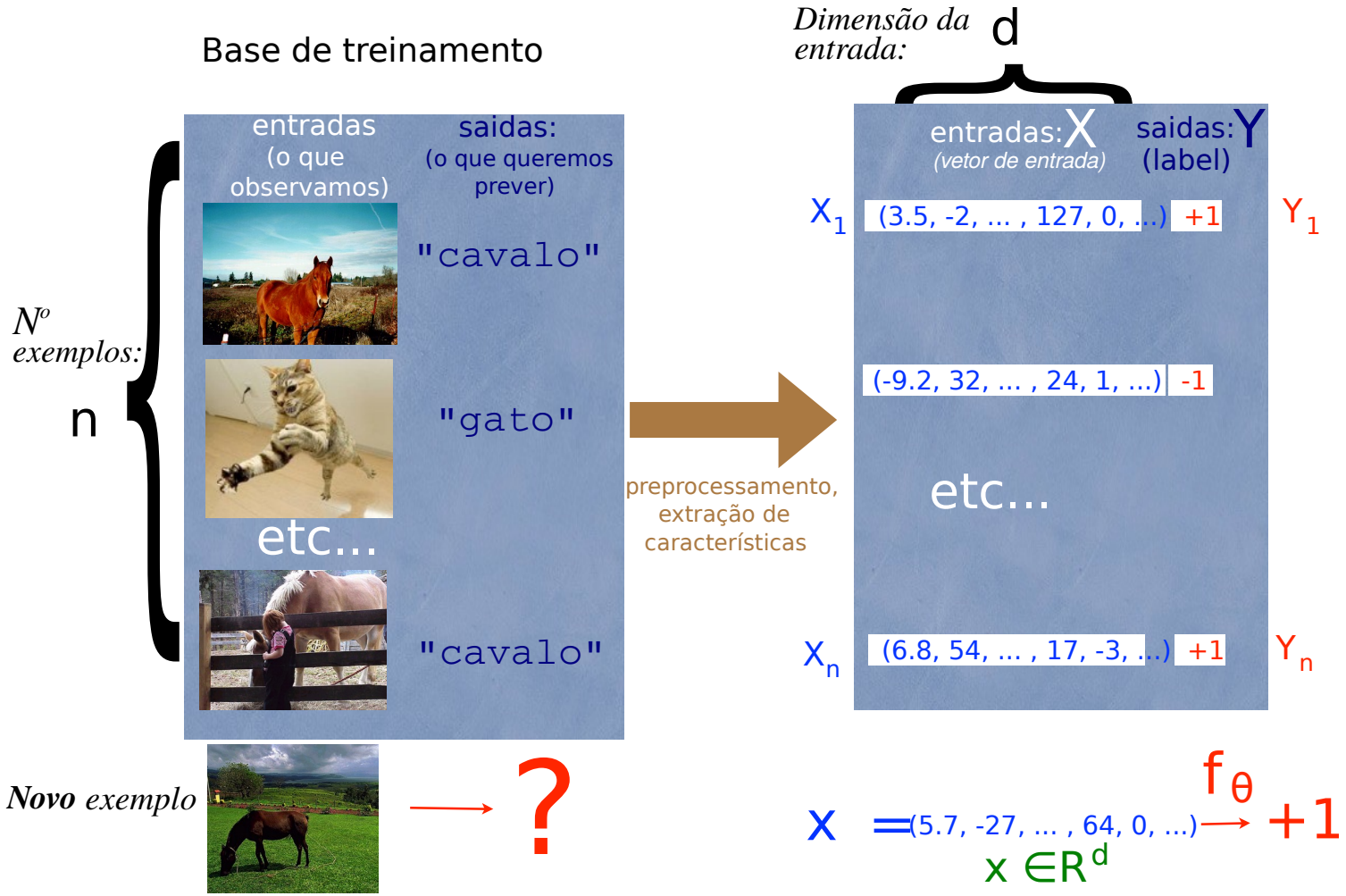


# Aprendizagem supervisionada - classificação

## Generalização para novos exemplos



# Aprendizagem supervisionada




Slide de Pascal Vincent

# 3 elementos básicos

# 3 elementos básicos

## Escolher a família de funcões

Geralmente paramétricas, (e.g. “Regressão logística”)

# 3 elementos básicos

**Escolher a família de funcões**

Geralmente paramétricas, (e.g. “Regressão logística”)

**Uma medida para avaliar f**

Função de custo → quanto menor, melhor o modelo

# 3 elementos básicos

**Escolher a família de funcões**

Geralmente paramétricas, (e.g. “Regressão logística”)

**Uma medida para avaliar f**

Função de custo → quanto menor, melhor o modelo

**Uma forma de buscar a melhor f**

Processo de otimização → como encontrar os parâmetros que minimizem a função de custo

# Regressão logística

# Regressão logística

**Família de funções:**

Combinação linear da entrada: $w_1 x_1 + w_2 x_2 ... w_m x_m$

Usa-se uma função não linear ao fim para obter resultados entre [0,1]

$$\hat{y} = P(y = 1|\mathbf{x}) = \sigma(\mathbf{w}^\intercal \mathbf{x})$$

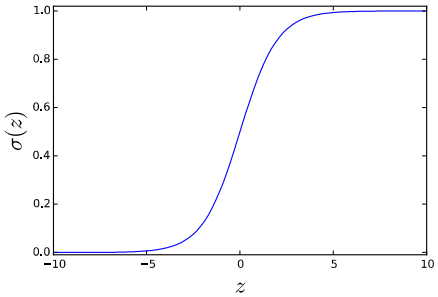
# Regressão logística

**Família de funções:**

Combinação linear da entrada: $w_1 x_1 + w_2 x_2 ... w_m x_m$

Usa-se uma função não linear ao fim para obter resultados entre [0,1]

$$\hat{y} = P(y = 1|\mathbf{x}) = \sigma(\mathbf{w}^\intercal \mathbf{x})$$

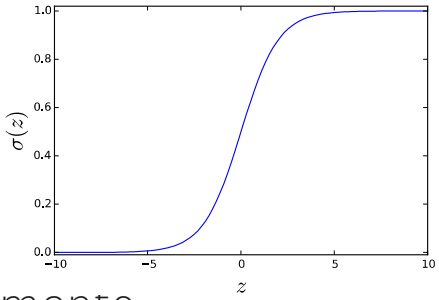


# Regressão logística

**Família de funções:**

Combinação linear da entrada: $w_1 x_1 + w_2 x_2 ... w_m x_m$

Usa-se uma função não linear ao fim para obter resultados entre [0,1]




$$\hat{y} = P(y = 1|\mathbf{x}) = \sigma(\mathbf{w}^\intercal \mathbf{x})$$

**Objetivo:**

Maximizar $P(y = y^{(i)}|\mathbf{x}^{(i)})$ para os exemplos na base de treinamento

Equivalente à minimizar:

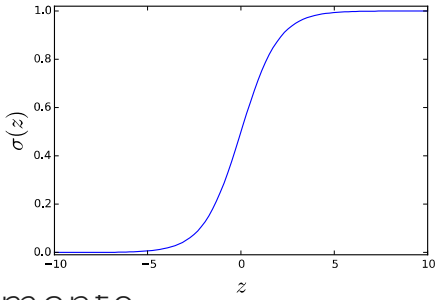
$$-\sum_i \log P(y = y^{(i)}|\mathbf{x}^{(i)})$$

# Regressão logística

**Família de funções:**

Combinação linear da entrada: $w_1 x_1 + w_2 x_2 ... w_m x_m$

Usa-se uma função não linear ao fim para obter resultados entre [0,1]




$$\hat{y} = P(y = 1|\mathbf{x}) = \sigma(\mathbf{w}^\intercal \mathbf{x})$$

**Objetivo:**

Maximizar $P(y = y^{(i)}|\mathbf{x}^{(i)})$ para os exemplos na base de treinamento

Equivalente à minimizar:

$$-\sum_i \log P(y = y^{(i)}|\mathbf{x}^{(i)})$$

**Optimização:**

Descida de gradiente: Começando com w aleatório, dando pequenos passos para diminuir a função de custo

# Regressão logística

# Regressão logística

**Função de custo, dado saída:** $\hat{y} = P(y = 1|\mathbf{x}) = \sigma(\mathbf{w}^\intercal \mathbf{x})$

$$L = -\frac{1}{N} \sum_{i=1}^{N} \log P(y = y^{(i)}|\mathbf{x}^{(i)})$$

$$L = -\frac{1}{N} \sum_{i=1}^{N} y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

# Regressão logística

**Função de custo, dado saída:** $\hat{y} = P(y = 1|\mathbf{x}) = \sigma(\mathbf{w}^\intercal \mathbf{x})$

$$L = -\frac{1}{N} \sum_{i=1}^{N} \log P(y = y^{(i)}|\mathbf{x}^{(i)})$$

$$L = -\frac{1}{N} \sum_{i=1}^{N} y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

**Otimização:**

Início: $\mathbf{w}^{(0)} = \text{random}$

Por T iterações:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \alpha \nabla_{\mathbf{w}} L$$

# Regressão logística

**Função de custo, dado saída:** $\hat{y} = P(y = 1|\mathbf{x}) = \sigma(\mathbf{w}^\intercal \mathbf{x})$

$$L = -\frac{1}{N} \sum_{i=1}^{N} \log P(y = y^{(i)}|\mathbf{x}^{(i)})$$

$$L = -\frac{1}{N} \sum_{i=1}^{N} y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

**Otimização:**

Início: $\mathbf{w}^{(0)} = \text{random}$

Por T iterações:

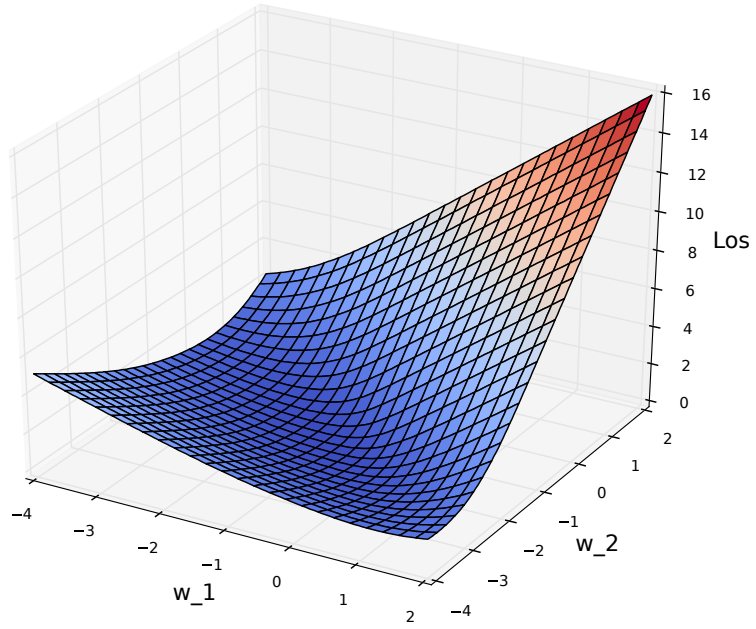
$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \alpha \nabla_{\mathbf{w}} L$$

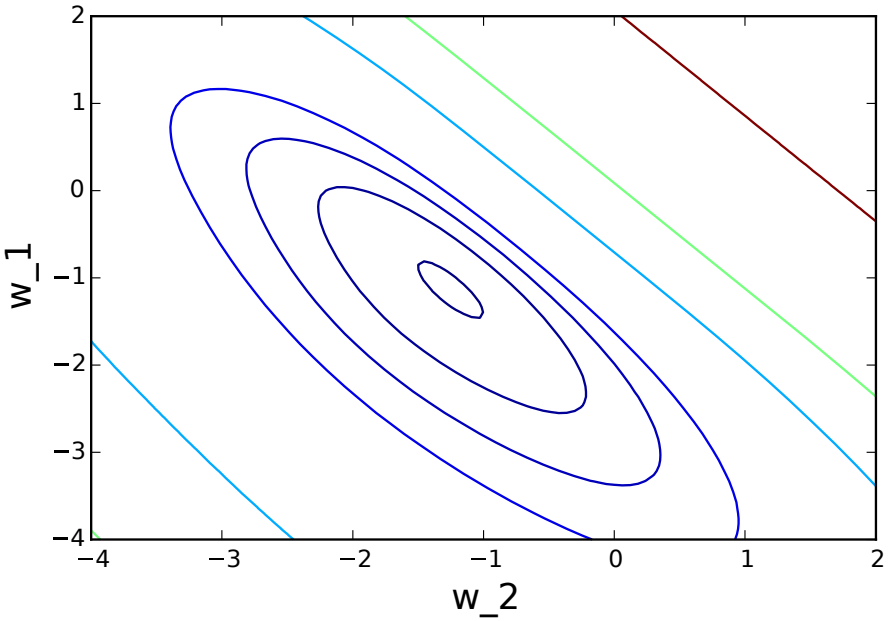
**Calculando $\nabla_{\mathbf{w}} L$:**

Usamos a regra de cadeia –ou software que a calcule automaticamente (e.g. Theano)

# Visualizando a função de custo

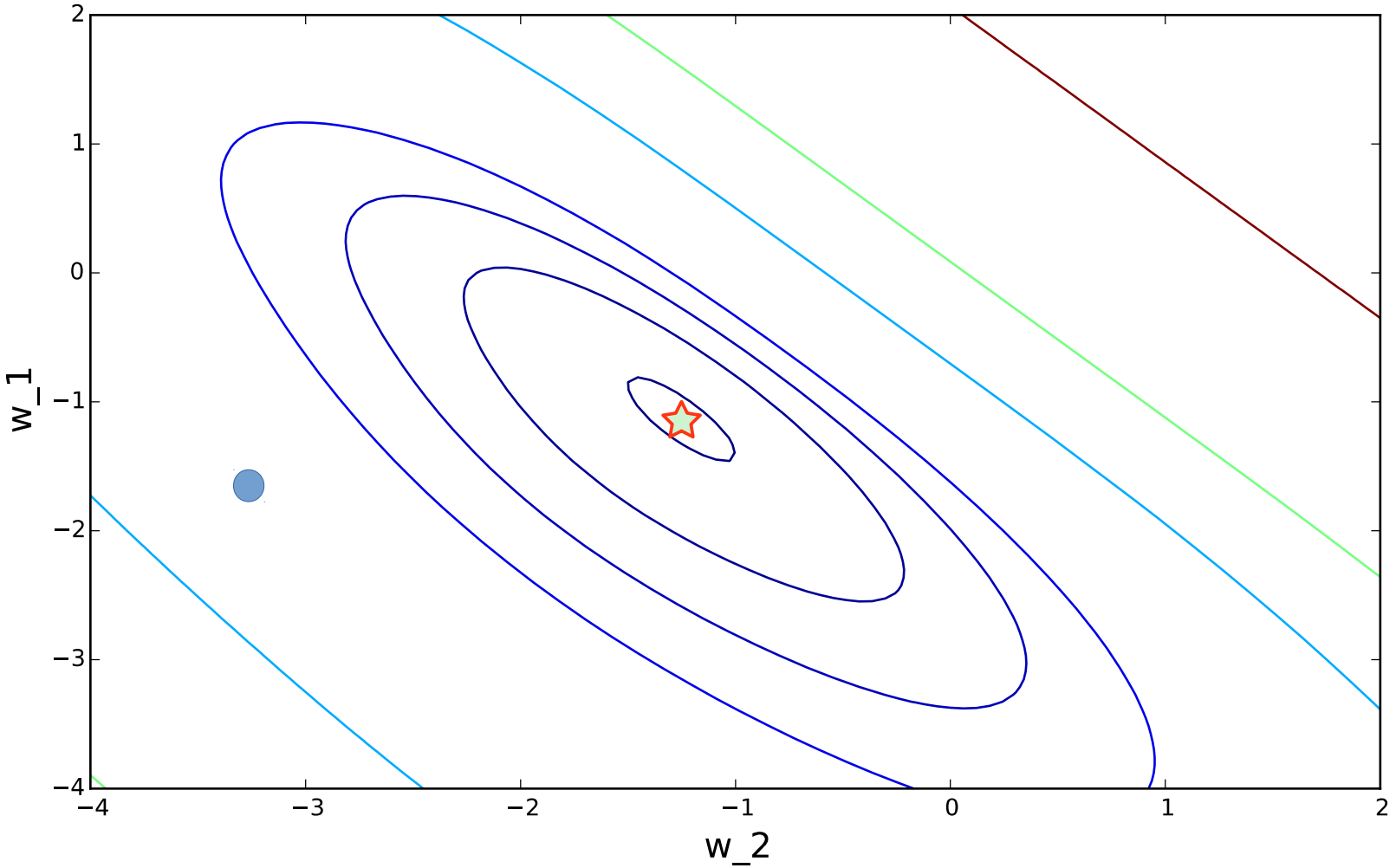
## Custo como função dos parametros w_1, w_2:



# Descida de gradiente

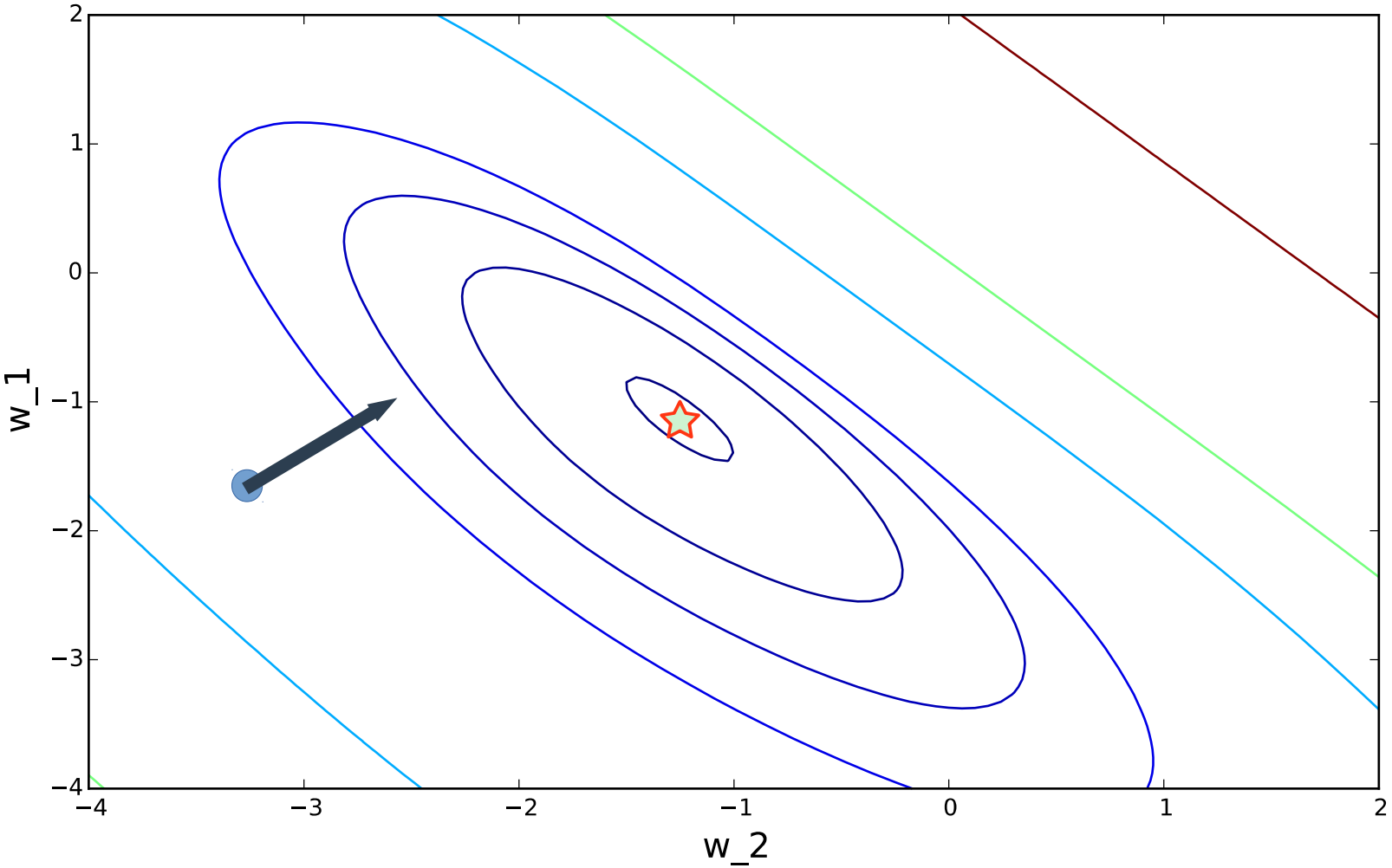
$\mathbf{w}^{(0)}$
(aleatório)



# Descida de gradiente

$\mathbf{w}^{(0)}$

Gradiente
$-\nabla_{\mathbf{w}} L$

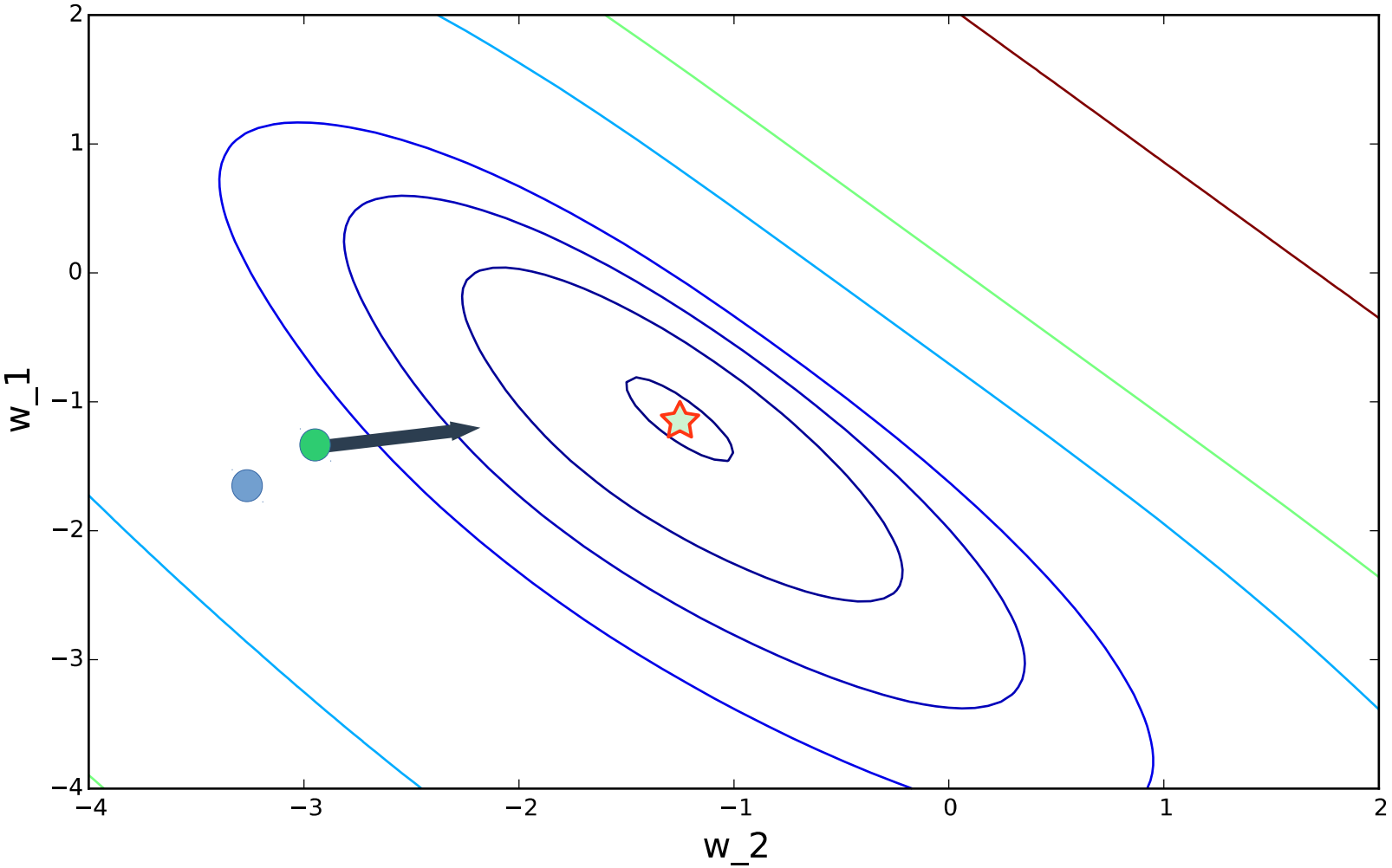


# Descida de gradiente

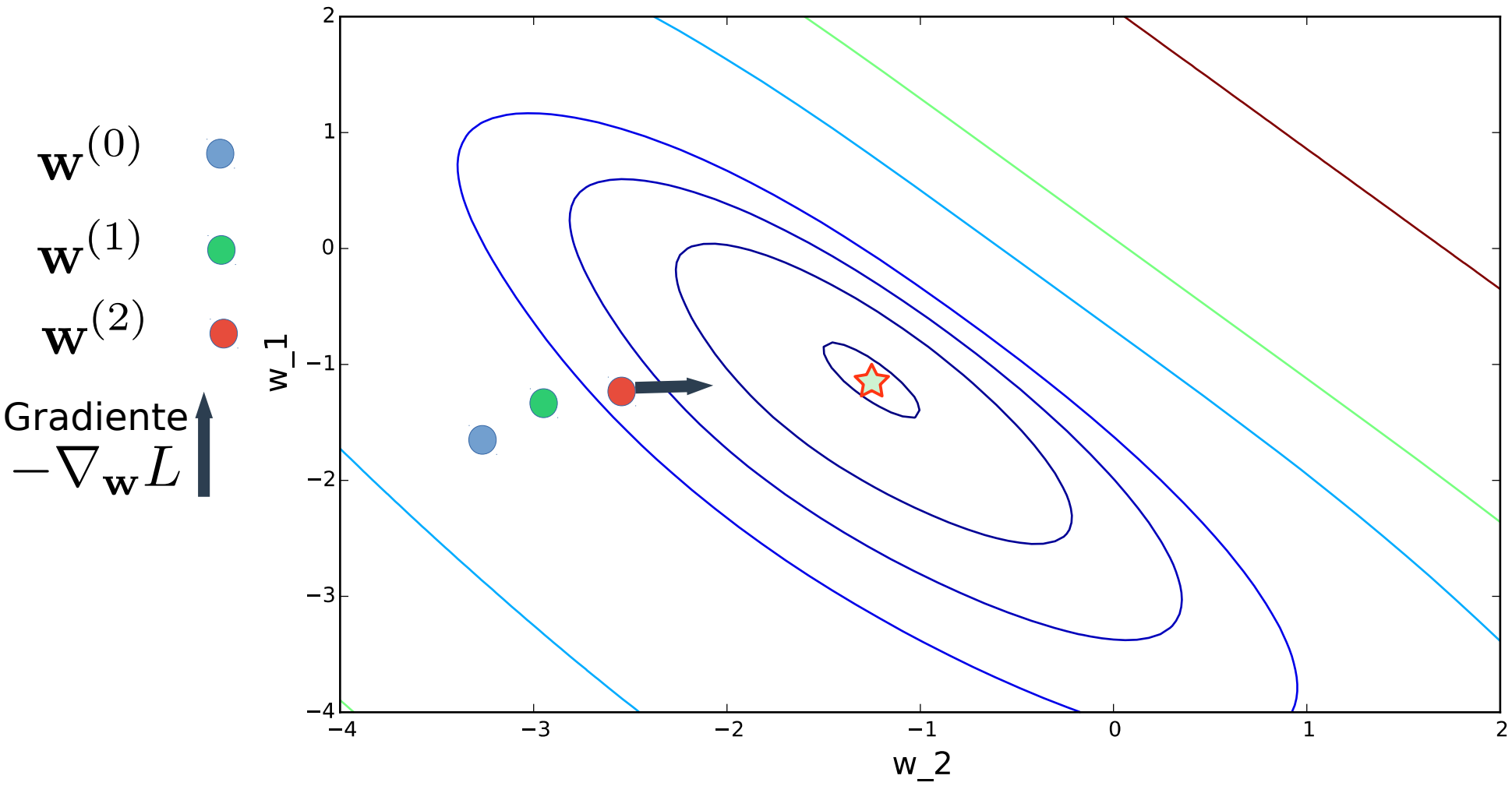
$\mathbf{w}^{(0)}$

$\mathbf{w}^{(1)}$

Gradiente
$-\nabla_{\mathbf{w}} L$



# Descida de gradiente

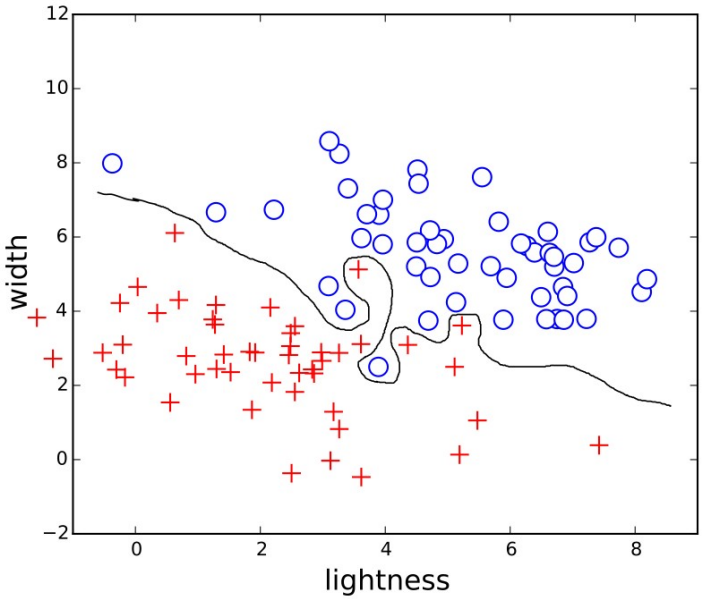


# Overfitting

**O importante é o <u>erro para novos exemplos</u> (generalização),**

Mas durante o treinamento, minimizamos o erro na base de treinamento

Se os modelos forem multo complexos, podem entrar em "overfitting"



# Overfitting

# Overfitting

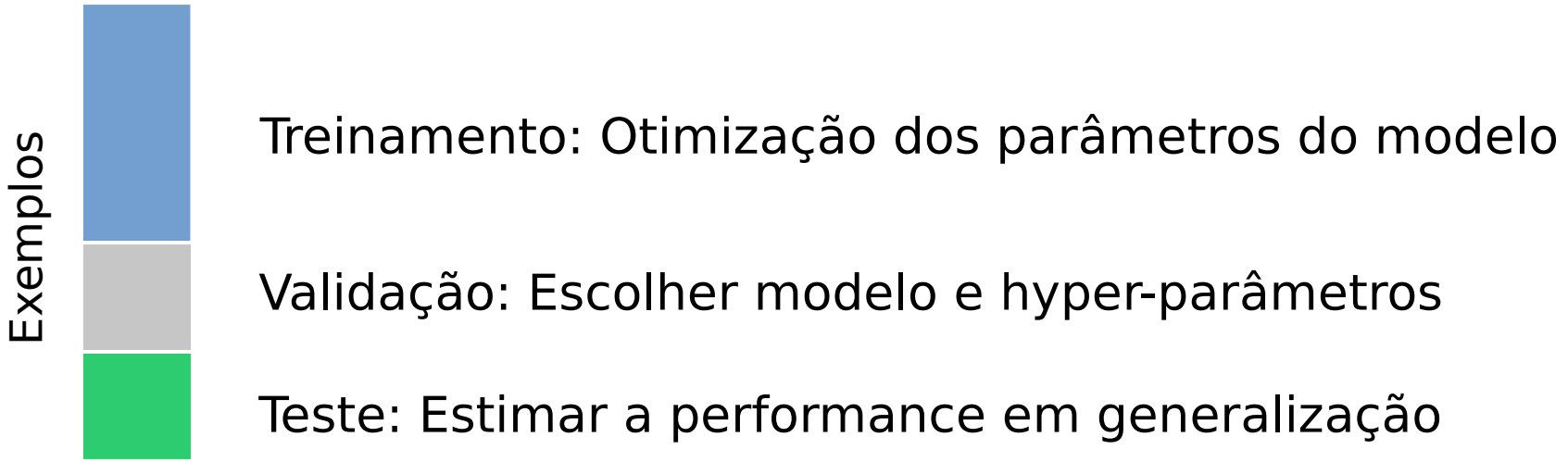
- **Estimar o erro em generalização:**
  - Manter uma base de teste separada (precisa conter exemplos diferentes, para evitar viés / bias)

# Overfitting

- **Estimar o erro em generalização:**
  - Manter uma base de teste separada (precisa conter exemplos diferentes, para evitar viés / bias)
  - Para escolher hyper-parâmetros, (e.g. tipo de modelo, caractéristicas a serem usadas), usar uma outra base de dados:

# Overfitting

- **Estimar o erro em generalização:**
  - Manter uma base de teste separada (precisa conter exemplos diferentes, para evitar viés / bias)
  - Para escolher hyper-parâmetros, (e.g. tipo de modelo, caractéristicas a serem usadas), usar uma outra base de dados:

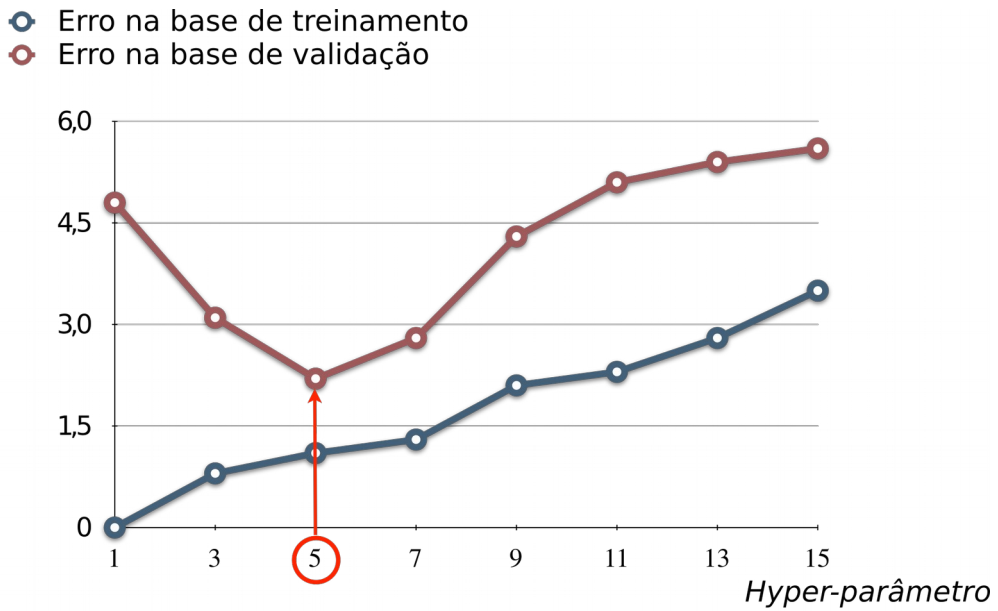


# Exemplo: seleção de modelos

**Treine diferentes modelos na base de treinamento**

**Avalie a performance em validação. Escolha o melhor modelo**

**Teste a performance do modelo na base de teste**



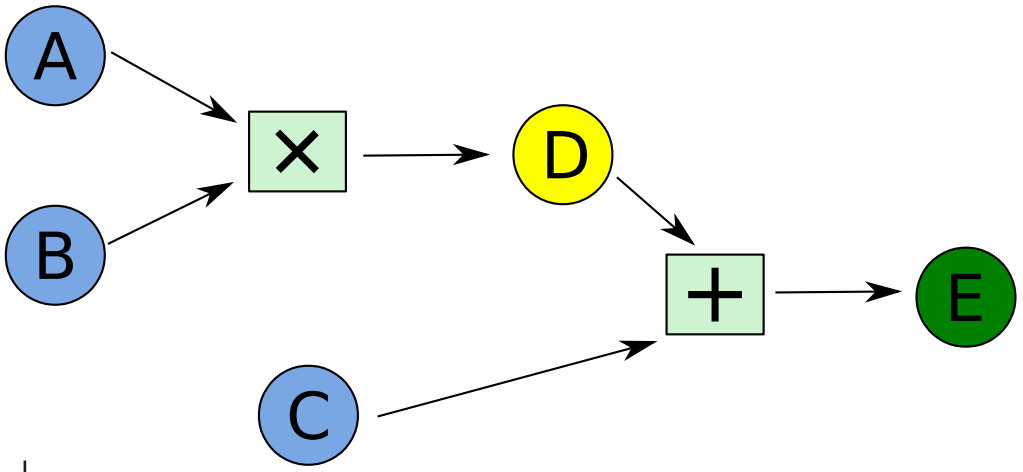

Melhor hyper-parâmetro segundo base de validação: **5**
(Segundo a base de treinamento: **1**)

# Introdução ao Theano

# Introdução ao Theano

## Computação simbólica

Expressões são definidas em grafos. Exemplo: e = ab + c

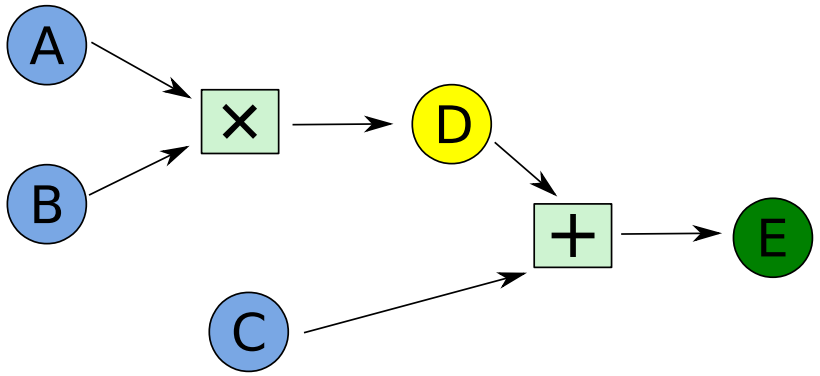



azul: entradas

amarelo: vértices intermediários

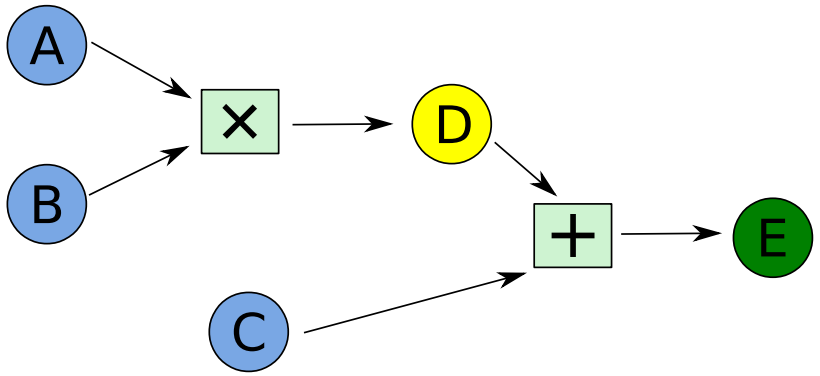
Verde: saídas

# Introdução ao Theano



# Introdução ao Theano

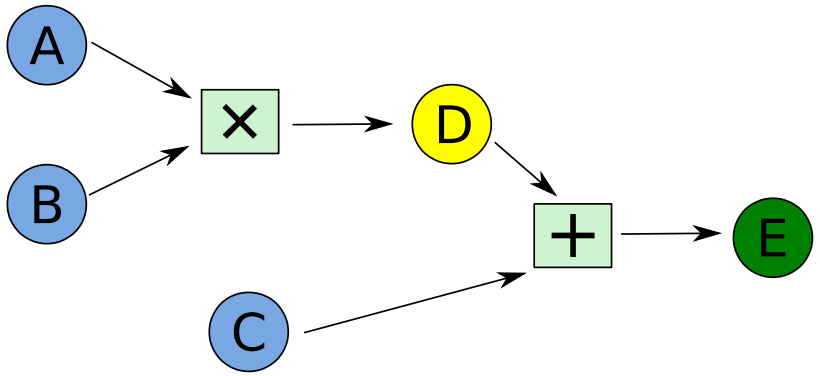
## Expressões precisam ser compiladas



# Introdução ao Theano

## Expressões precisam ser compiladas

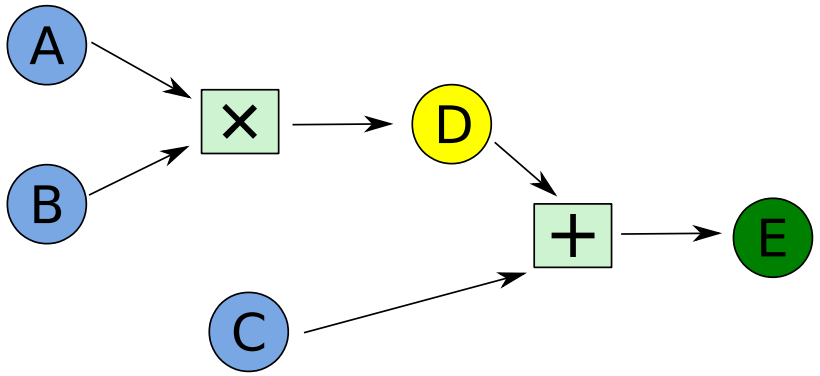
```
a = T.scalar()
```



# Introdução ao Theano

## Expressões precisam ser compiladas

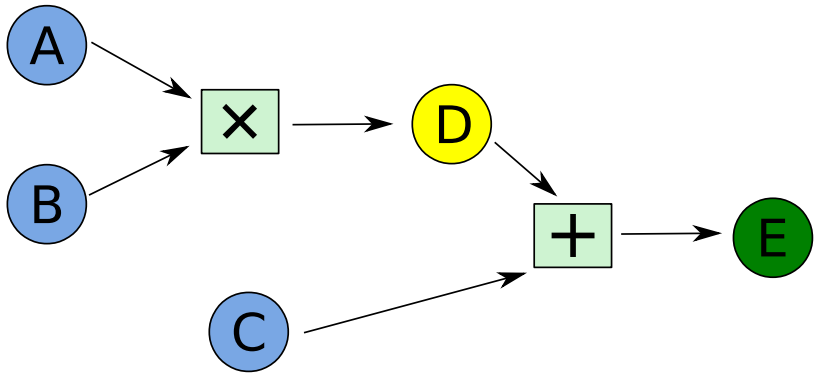
```
a = T.scalar()
b = T.scalar()
```



# Introdução ao Theano

## Expressões precisam ser compiladas

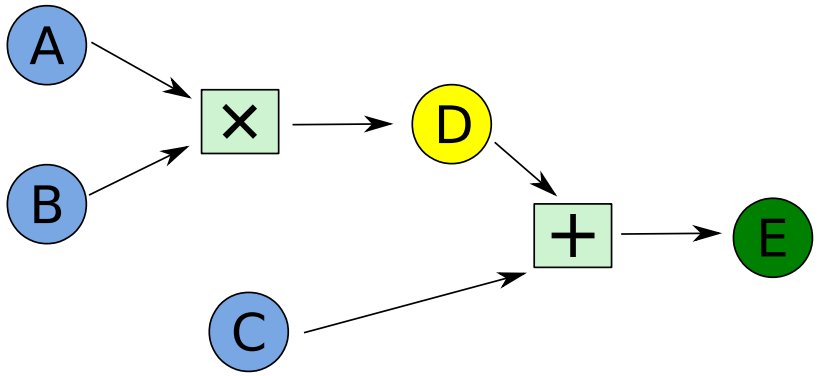
```
a = T.scalar()
b = T.scalar()
c = T.scalar()
```



# Introdução ao Theano

## Expressões precisam ser compiladas

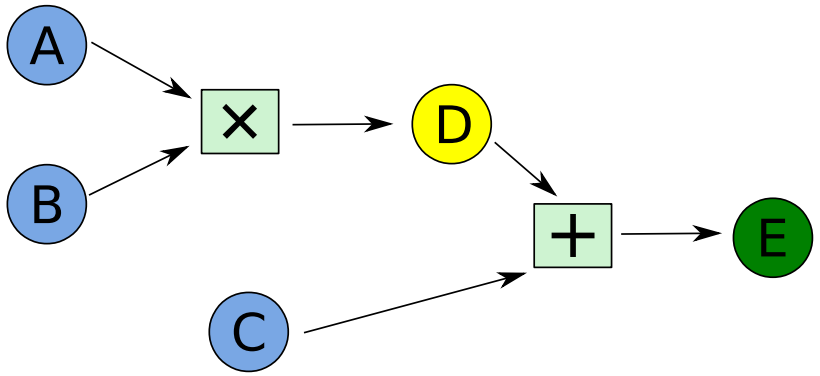
```
a = T.scalar()
b = T.scalar()
c = T.scalar()
e = a*b + c
```



# Introdução ao Theano

## Expressões precisam ser compiladas

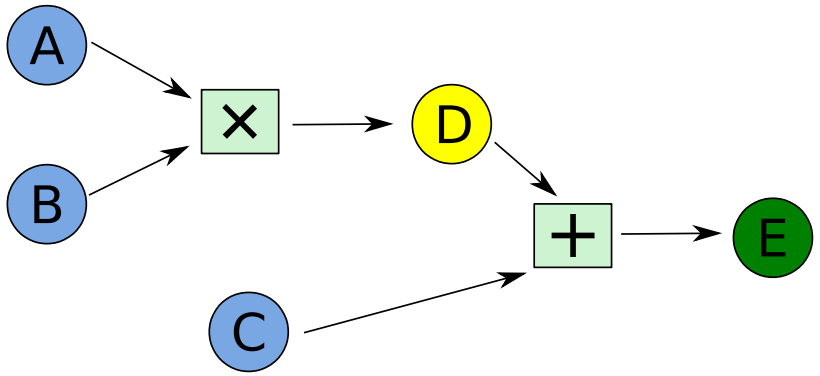
```
a = T.scalar()
b = T.scalar()
c = T.scalar()
e = a*b + c
f = theano.function([a,b,c],e)
```



# Introdução ao Theano

## Expressões precisam ser compiladas

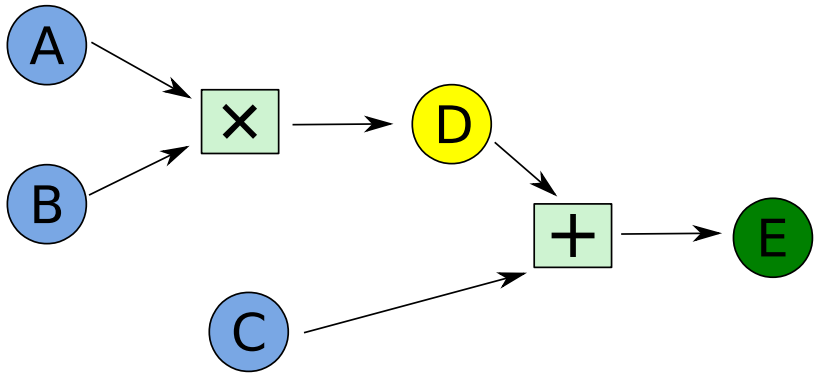
```
a = T.scalar()
b = T.scalar()
c = T.scalar()
e = a*b + c
f = theano.function([a,b,c],e)
f(2,4,10) #retorna 2*4+10 = 18
```



# Introdução ao Theano

## Expressões precisam ser compiladas

```
a = T.scalar()
b = T.scalar()
c = T.scalar()
e = a*b + c
f = theano.function([a,b,c],e)
f(2,4,10) #retorna 2*4+10 = 18
```




## Permite derivação automática:

```
de_da = T.grad(e, a)
g = theano.function([a,b,c], de_da)
g(2,4,10) # retorna 4
```

# Dois tipos de varíaveis

## Variável simbólica

- Não possui estado.
- É informada na chamada de uma função, ou computada à partir de outras variáveis

- **Variável compartilhada**
  - Possui estado

# Dois tipos de varíaveis

## Variável simbólica

- Não possui estado.
- É informada na chamada de uma função, ou computada à partir de outras variáveis

- **Variável compartilhada**
  - Possui estado

```
a = T.scalar()          # Variável simbólica
```

# Dois tipos de varíaveis

## Variável simbólica

- Não possui estado.
- É informada na chamada de uma função, ou computada à partir de outras variáveis

- **Variável compartilhada**
  - Possui estado

```
a = T.scalar()             # Variável simbólica
b = theano.shared(2)  # Variável compartilhada, com valor 2
```

# Dois tipos de varíaveis

## Variável simbólica

- Não possui estado.
- É informada na chamada de uma função, ou computada à partir de outras variáveis

- **Variável compartilhada**
  - Possui estado

```
a = T.scalar()             # Variável simbólica
b = theano.shared(2)  # Variável compartilhada, com valor 2
c = theano.shared(1)  # Variável compartilhada, com valor 1
```

# Dois tipos de varíaveis

## Variável simbólica

- Não possui estado.
- É informada na chamada de uma função, ou computada à partir de outras variáveis

- **Variável compartilhada**
  - Possui estado

```
a = T.scalar()              # Variável simbólica
b = theano.shared(2)  # Variável compartilhada, com valor 2
c = theano.shared(1)  # Variável compartilhada, com valor 1
e = a*b + c
```

# Dois tipos de varíaveis

## Variável simbólica

- Não possui estado.
- É informada na chamada de uma função, ou computada à partir de outras variáveis

- **Variável compartilhada**
  - Possui estado

```
a = T.scalar()              # Variável simbólica
b = theano.shared(2)  # Variável compartilhada, com valor 2
c = theano.shared(1)  # Variável compartilhada, com valor 1
e = a*b + c

f = theano.function([a],e)
```

# Dois tipos de varíaveis

## Variável simbólica

- Não possui estado.
- É informada na chamada de uma função, ou computada à partir de outras variáveis

- **Variável compartilhada**
  - Possui estado

```
a = T.scalar()              # Variável simbólica
b = theano.shared(2)  # Variável compartilhada, com valor 2
c = theano.shared(1)  # Variável compartilhada, com valor 1
e = a*b + c

f = theano.function([a],e)
f(3)  # na chamada, informamos o valor de a.
```

# Dois tipos de varíaveis

## Variável simbólica

- Não possui estado.
- É informada na chamada de uma função, ou computada à partir de outras variáveis

- **Variável compartilhada**
  - Possui estado

```
a = T.scalar()              # Variável simbólica
b = theano.shared(2)  # Variável compartilhada, com valor 2
c = theano.shared(1)  # Variável compartilhada, com valor 1
e = a*b + c

f = theano.function([a],e)
f(3)  # na chamada, informamos o valor de a.
#retorna 3*2+1 = 7
```

# Ipython Notebook

**DEMO**