

XI Encontro de Física e Astronomia da UFSC

Introdução às redes neurais com Python e Keras

Prof. Dr. Robson da Silva Oliboni



Introdução às redes neurais com Python e Keras

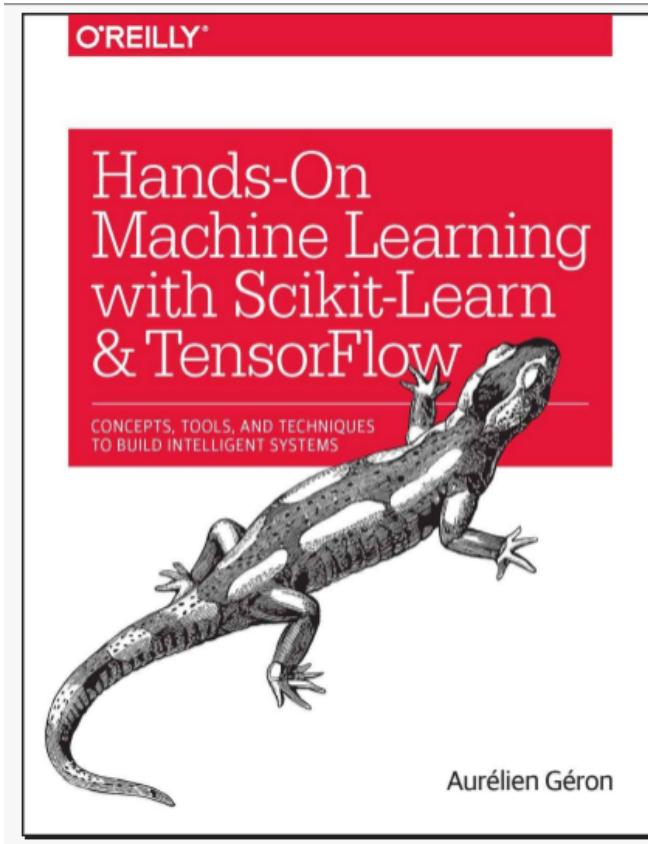
Sumário

1. Conceitos iniciais
2. O neurônio de McCulloch-Pitts
3. O neurônio de Rosenblatt
4. Redes neurais (artificiais)
5. O algoritmo de retropropagação
6. Redes neurais com o Keras

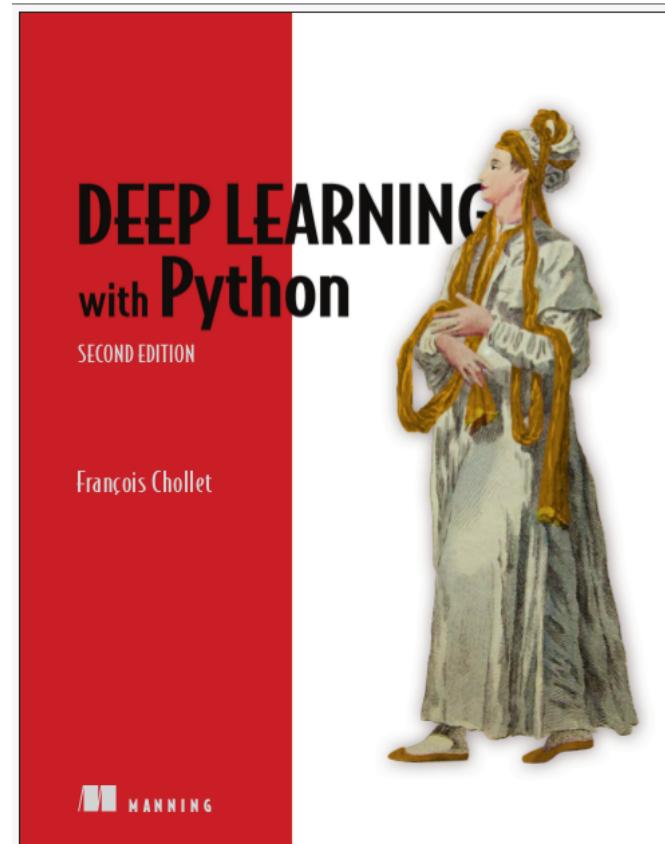
Introdução às redes neurais com Python e Keras

Referências

[1]



[2]



O que é aprendizado de máquina (*machine learning*)?



Arthur Samuel (1959): Campo de estudo que fornece ao computador a capacidade de *aprender* sem ser explicitamente programado.

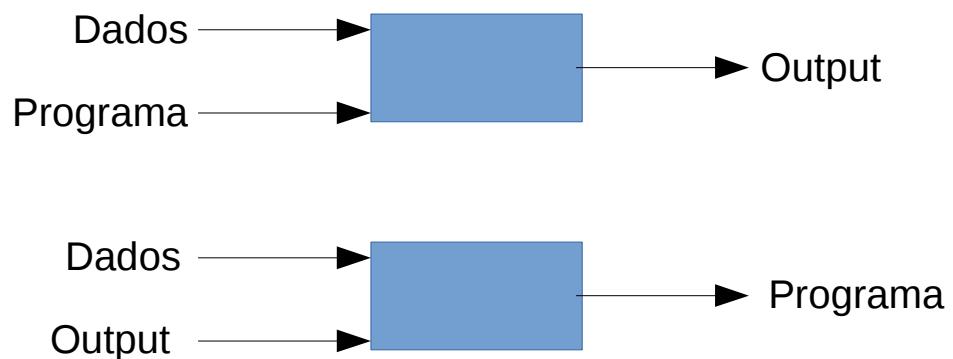


Tom Mitchell (1998): Um programa de computador é dito *aprender* da experiência E em relação a certa tarefa T e certa medida de performance P , se sua performance em T , como medida por P , melhora com a experiência E .

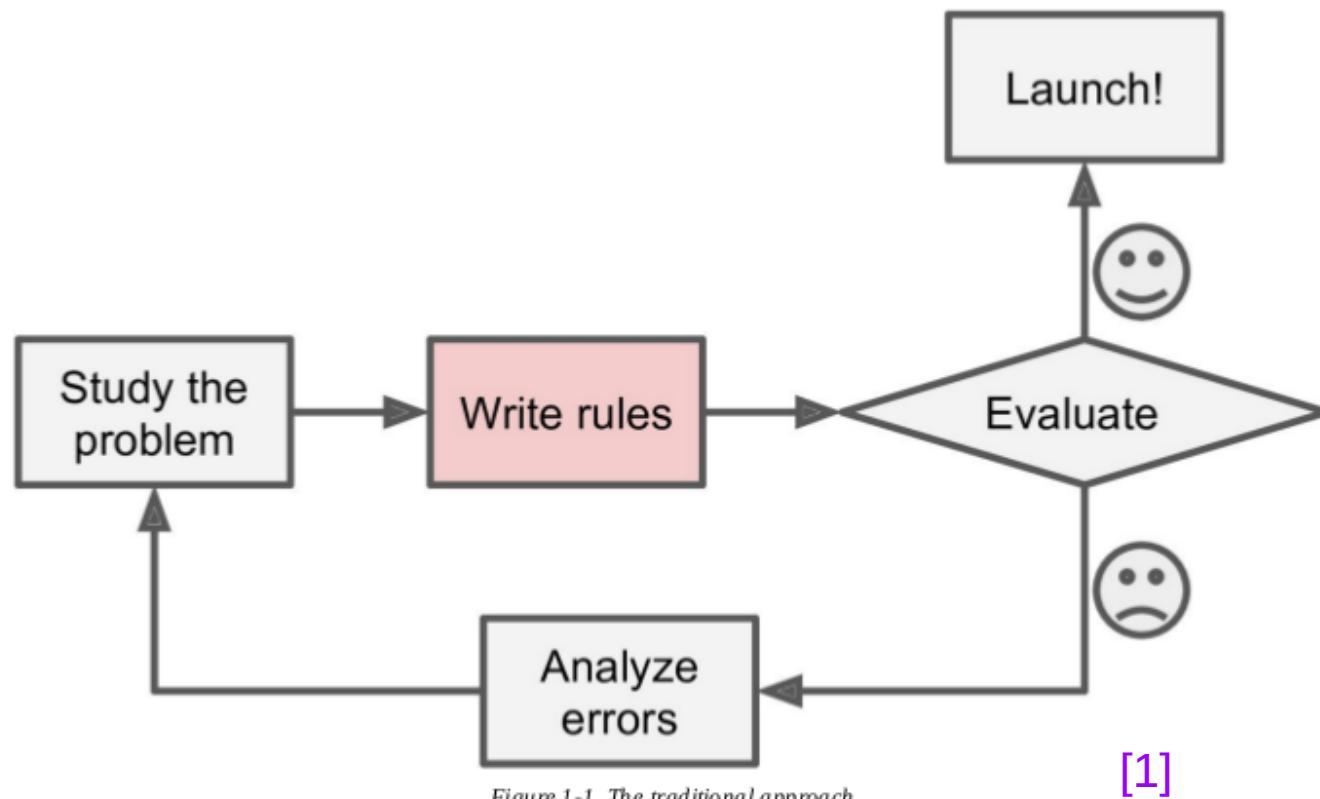
Premissa básica do aprendizado: “*Utilizar um conjunto de observações para descobrir um processo subjacente*”.

A essência do aprendizado de máquinas:

- Um padrão existe;
- Não podemos resumí-lo matematicamente;
- Temos dados para trabalhar.

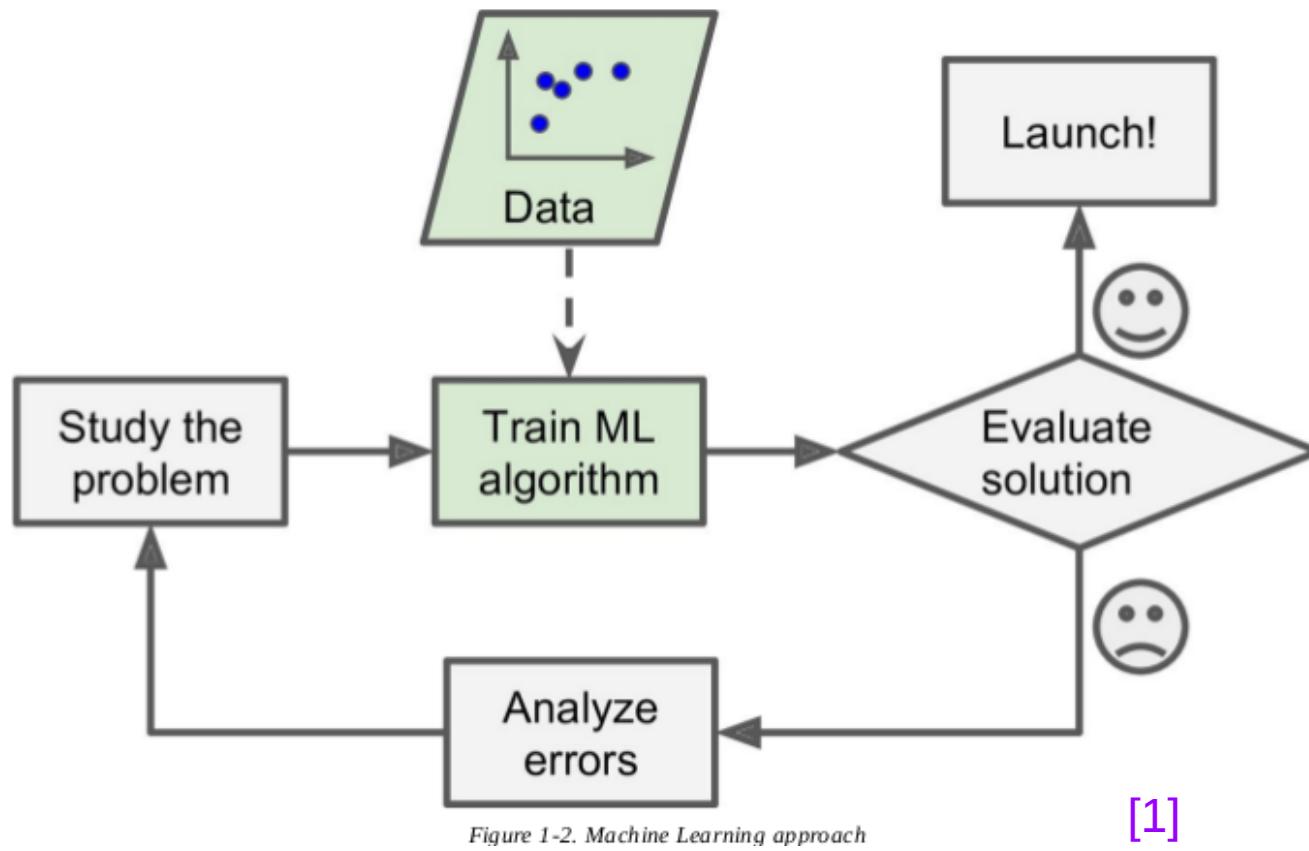


Exemplos de aplicação: **Detector de spam**



A abordagem tradicional à solução de problemas

Exemplos de aplicação: **Detector de spam**



A abordagem de aprendizado de máquinas (*machine learning*, ML)

Exemplos de aplicação: **Detector de spam**

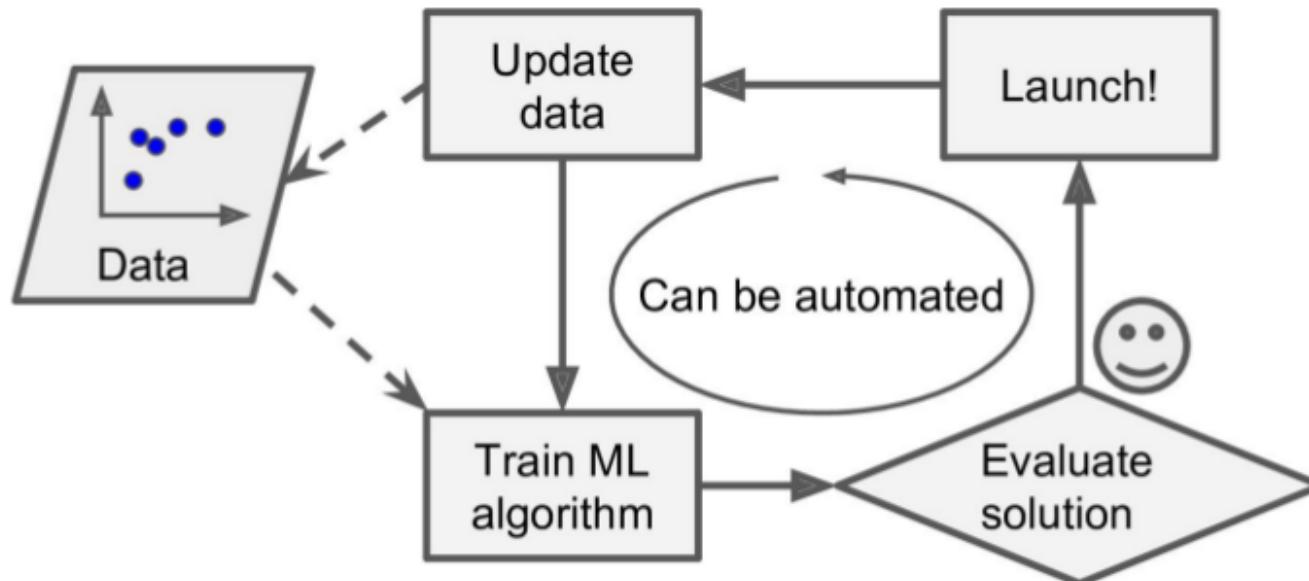


Figure 1-3. Automatically adapting to change

[1]

Um programa de ML é altamente adaptável a novos exemplos

Exemplos de aplicação

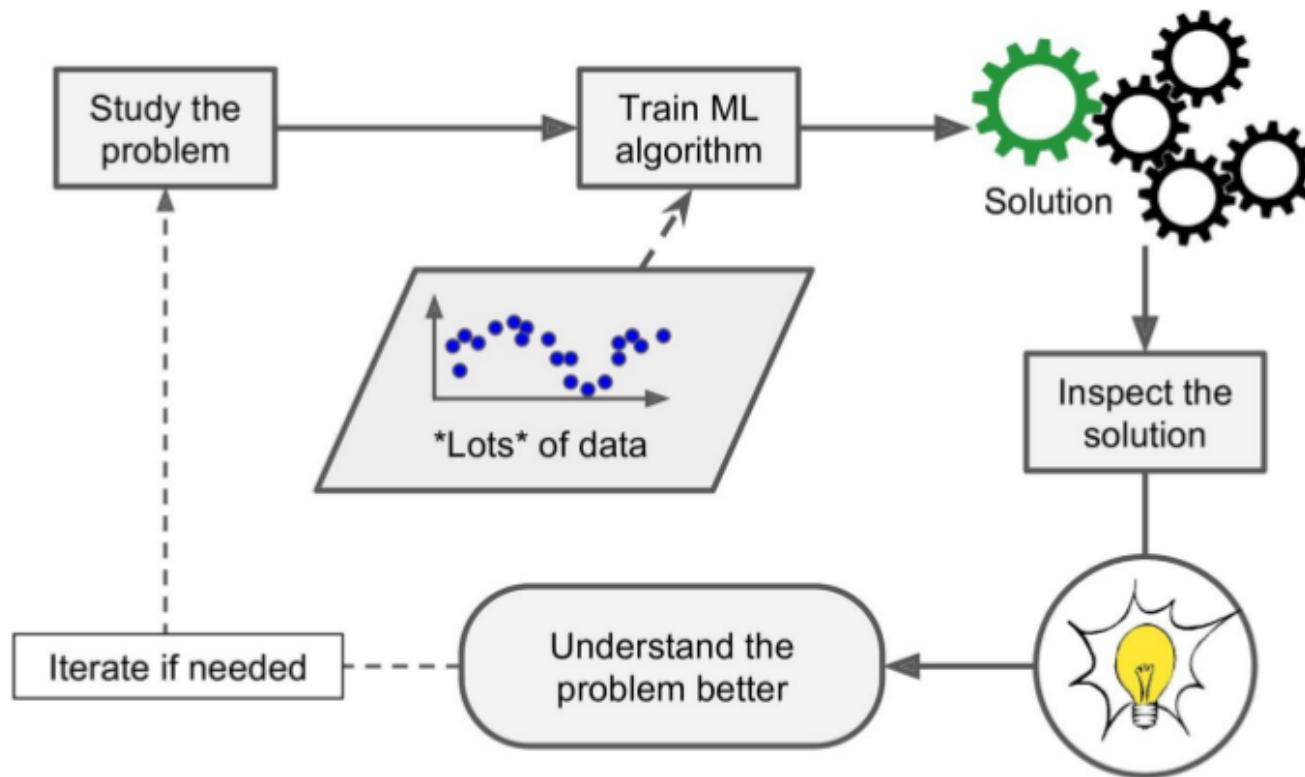


Figure 1-4. Machine Learning can help humans learn

[1]

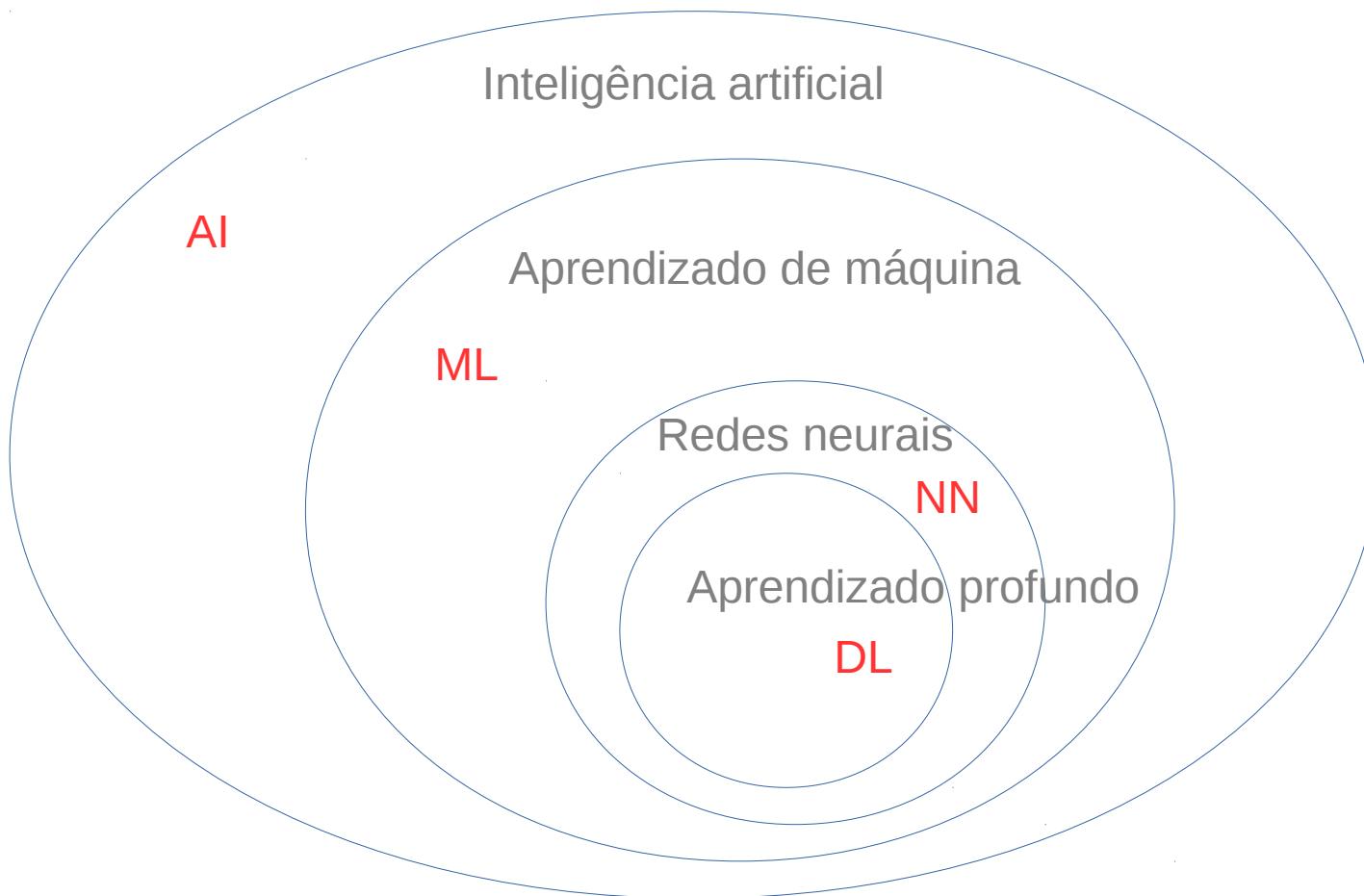
Mineração de dados: Descoberta de padrões subjacentes aos dados

Diferença entre AI, ML, NN, DL...

ML é uma subárea da IA que lida com aprendizado; habilidade que possibilita todas as outras.

IA = computador faz coisas que apenas humanos conseguem;

“É o planeta que queremos alcançar. ML é o foguete que nos levará lá, e big data é o combustível que nutre esse foguete.” – Pedro Domingos



IA: o esforço de automatizar tarefas intelectuais normalmente executadas por humanos.

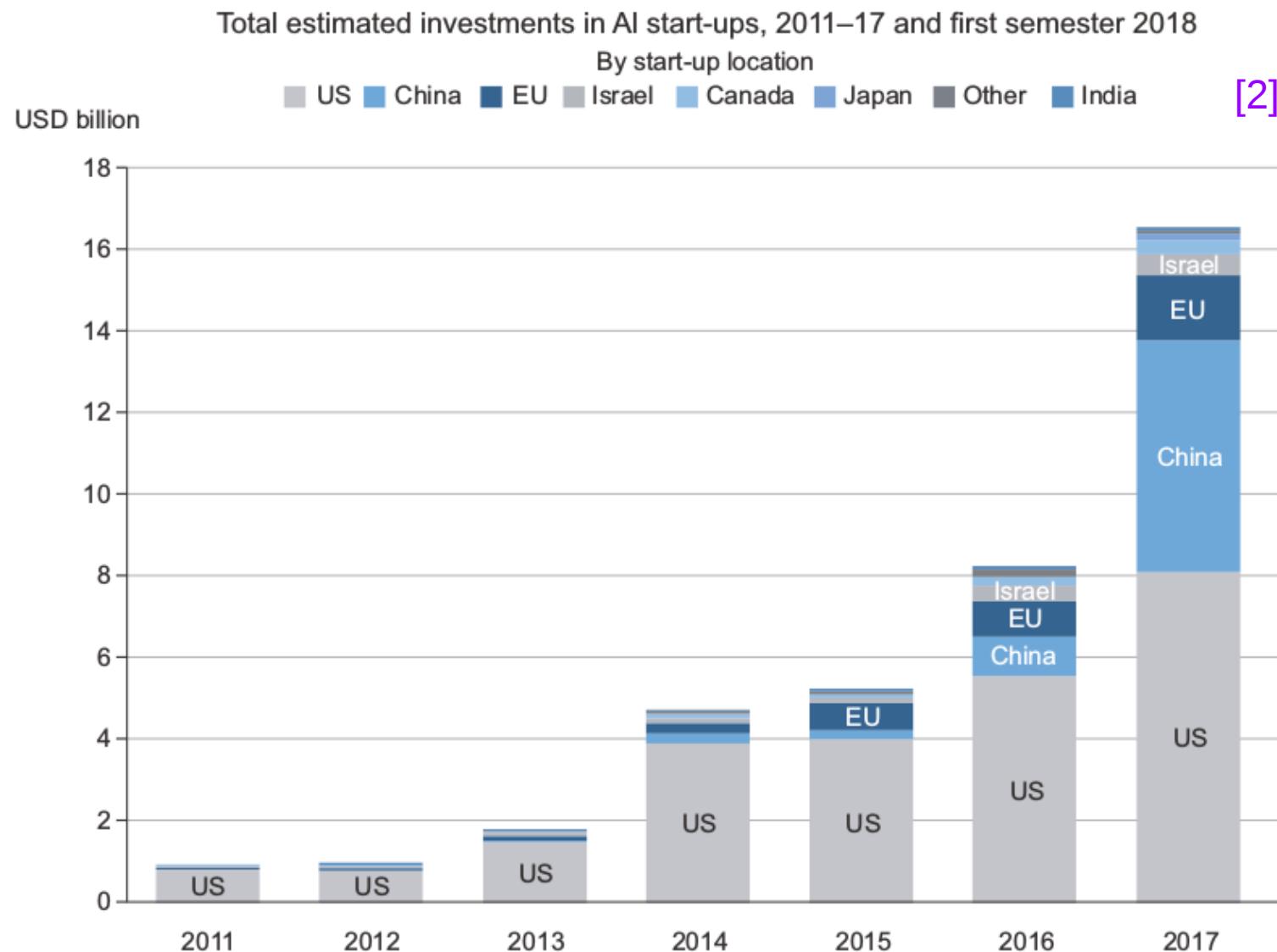
Recente desenvolvimento do *machine learning*

Década de 1940 até 1960 → Inverno da AI → Década 1980-90 → Outro inverno

Razões do ressurgimento:

1. Quantidade massiva de dados;
2. Aumento espantoso do processamento computacional desde os anos 90 (Lei de Moore+games/GPUs);
3. Melhores algoritmos de treinamento;
4. Algumas limitações teóricas se provaram inofensivas.

Recente desenvolvimento do *machine learning*



Amplo investimento + democratização das técnicas de ML.

Tipos de sistemas de aprendizado de máquina

Categorias simples:

- Serem ou não treinados com supervisão humana: ([Paradigmas de aprendizado](#))
Aprendizado supervisionado, não-supervisionado, semissupervisionado ou aprendizado por reforço;
- Se podem ou não aprender gradativamente em tempo real:
Aprendizado online ou aprendizado em batch;
- Se funcionam simplesmente comparando novos pontos de dados com pontos de dados conhecidos, ou se detectam padrões em dados de treinamento e criam um modelo preditivo, como cientistas:
Aprendizado baseado em instâncias ou aprendizado baseado em modelo.

Exemplos de aprendizado supervisionado: Classificação e regressão

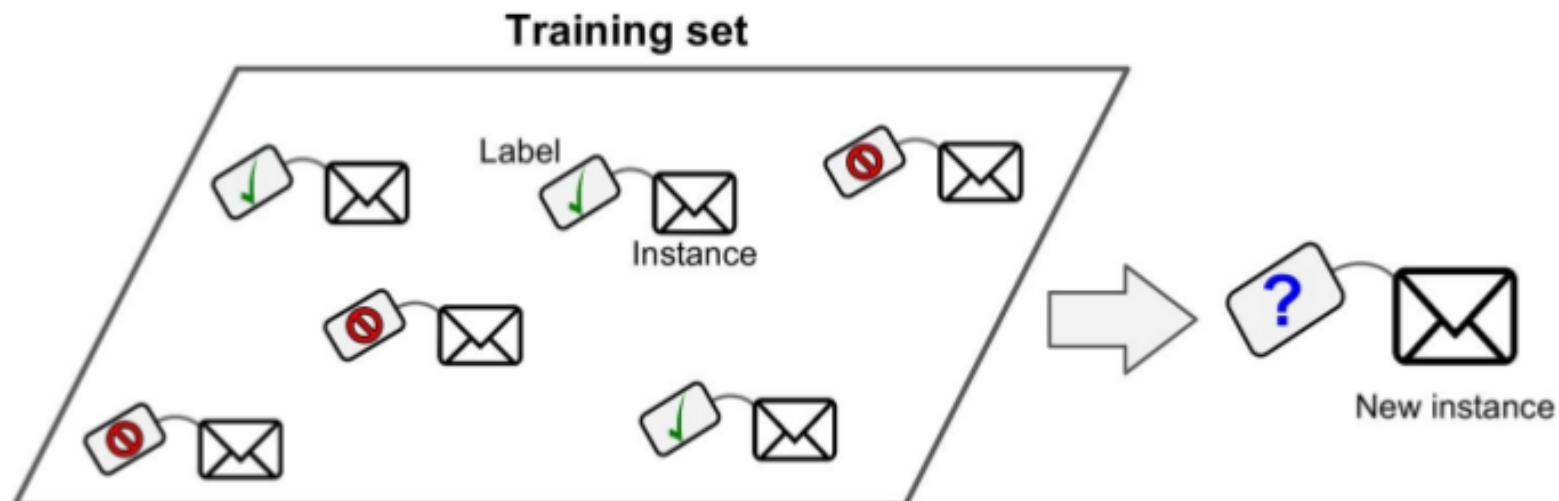


Figure 1-5. A labeled training set for supervised learning (e.g., spam classification)

[1]

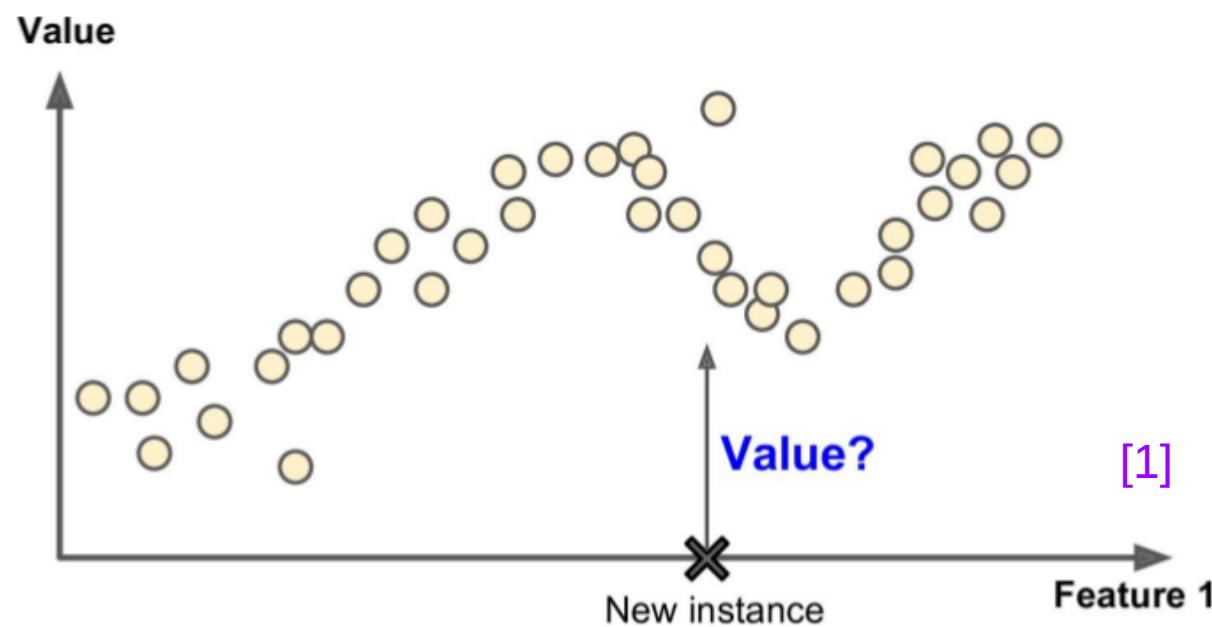
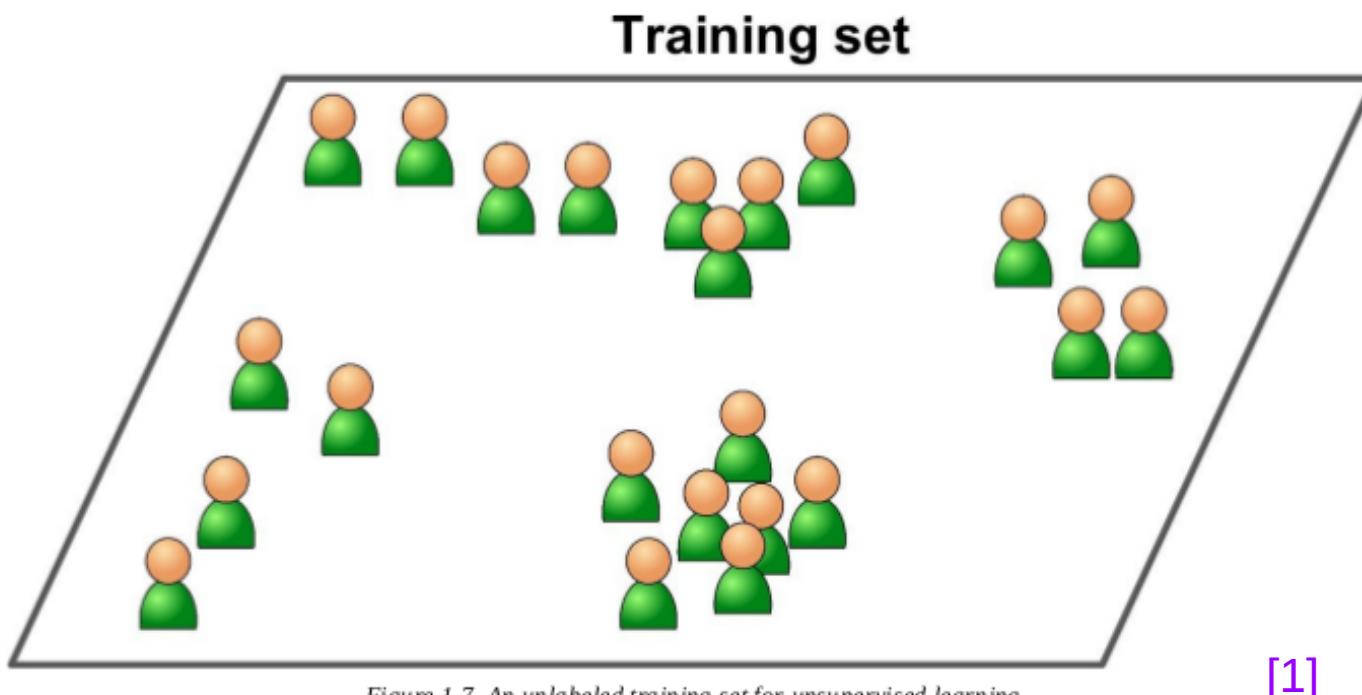


Figure 1-6. Regression

13

Aprendizado não-supervisionado



Um conjunto de dados não-supervisionado não contém rótulos (padrões).

Exemplos de aprendizado não-supervisionado: Clusterização e detecção de anomalias

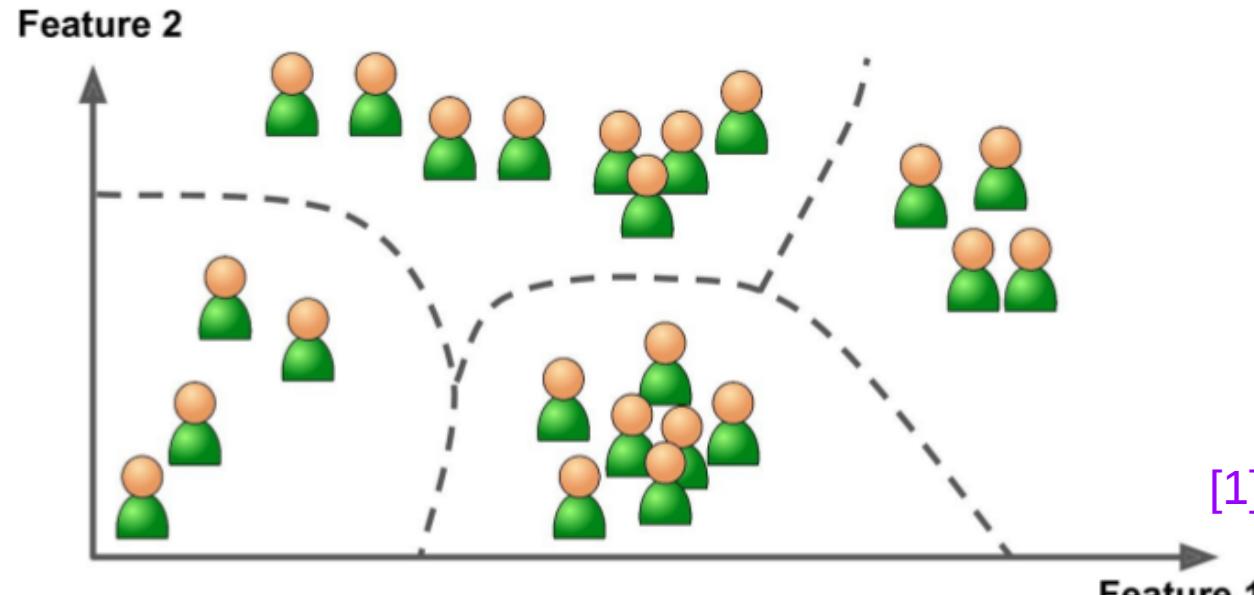


Figure 1-8. Clustering

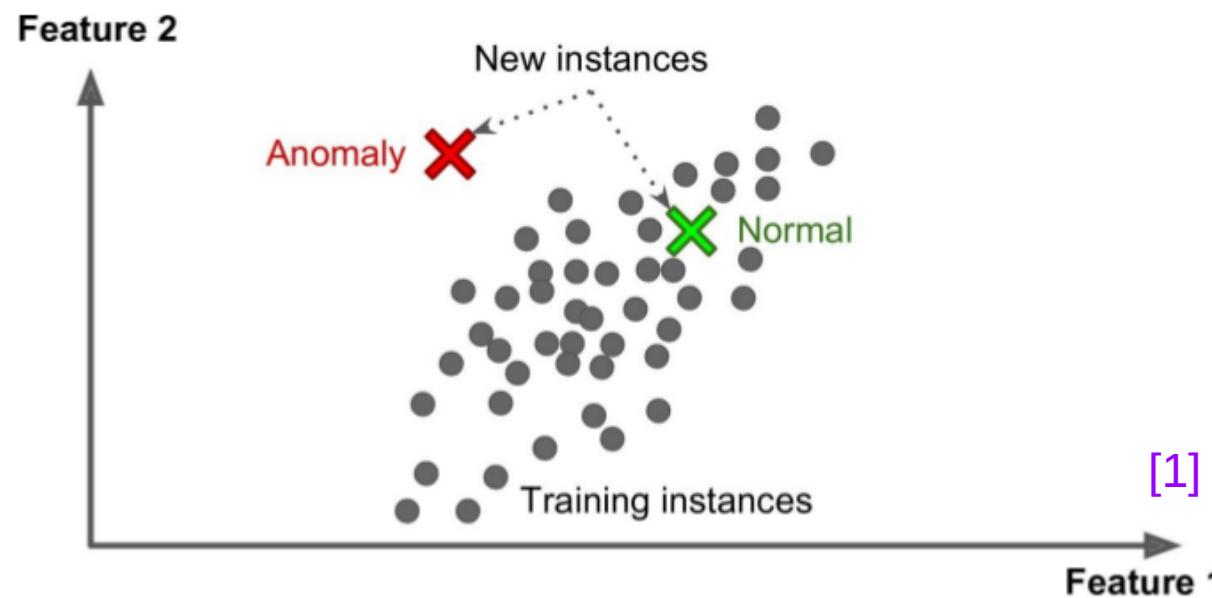


Figure 1-10. Anomaly detection

Aprendizado semissupervisionado

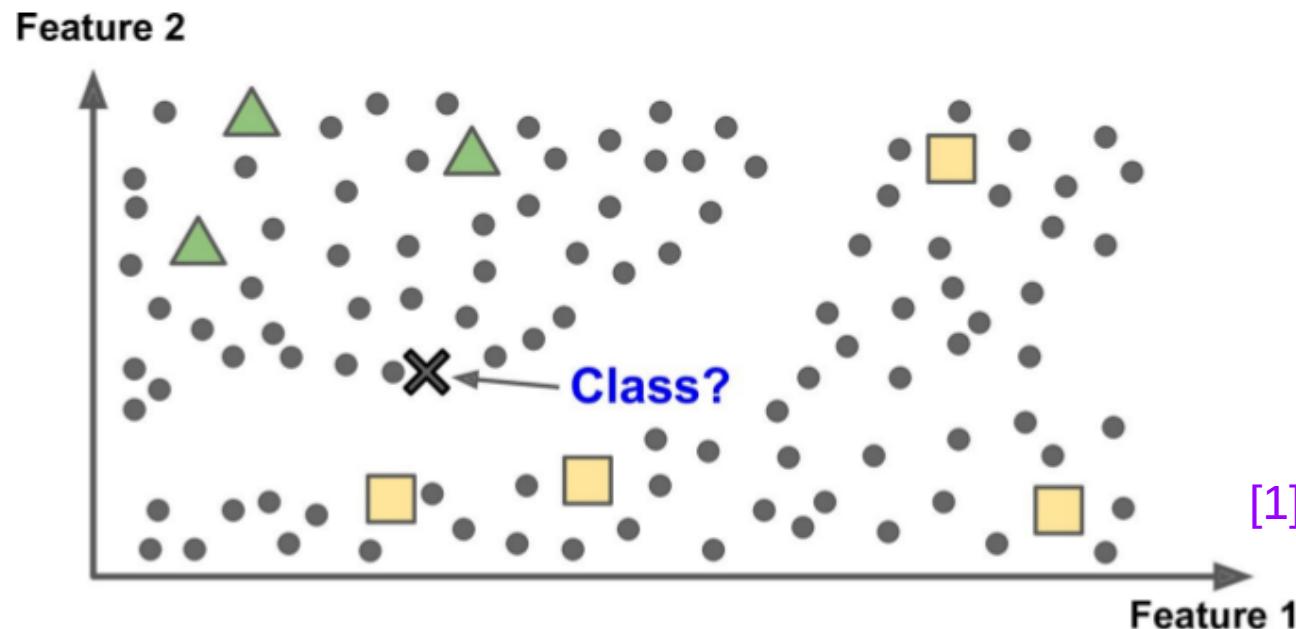


Figure 1-11. Semisupervised learning

Um conjunto de dados semissupervisionado contém rótulos para alguns exemplos, porém não todos.

Aprendizado por reforço (*Reinforcement learning*, RL)

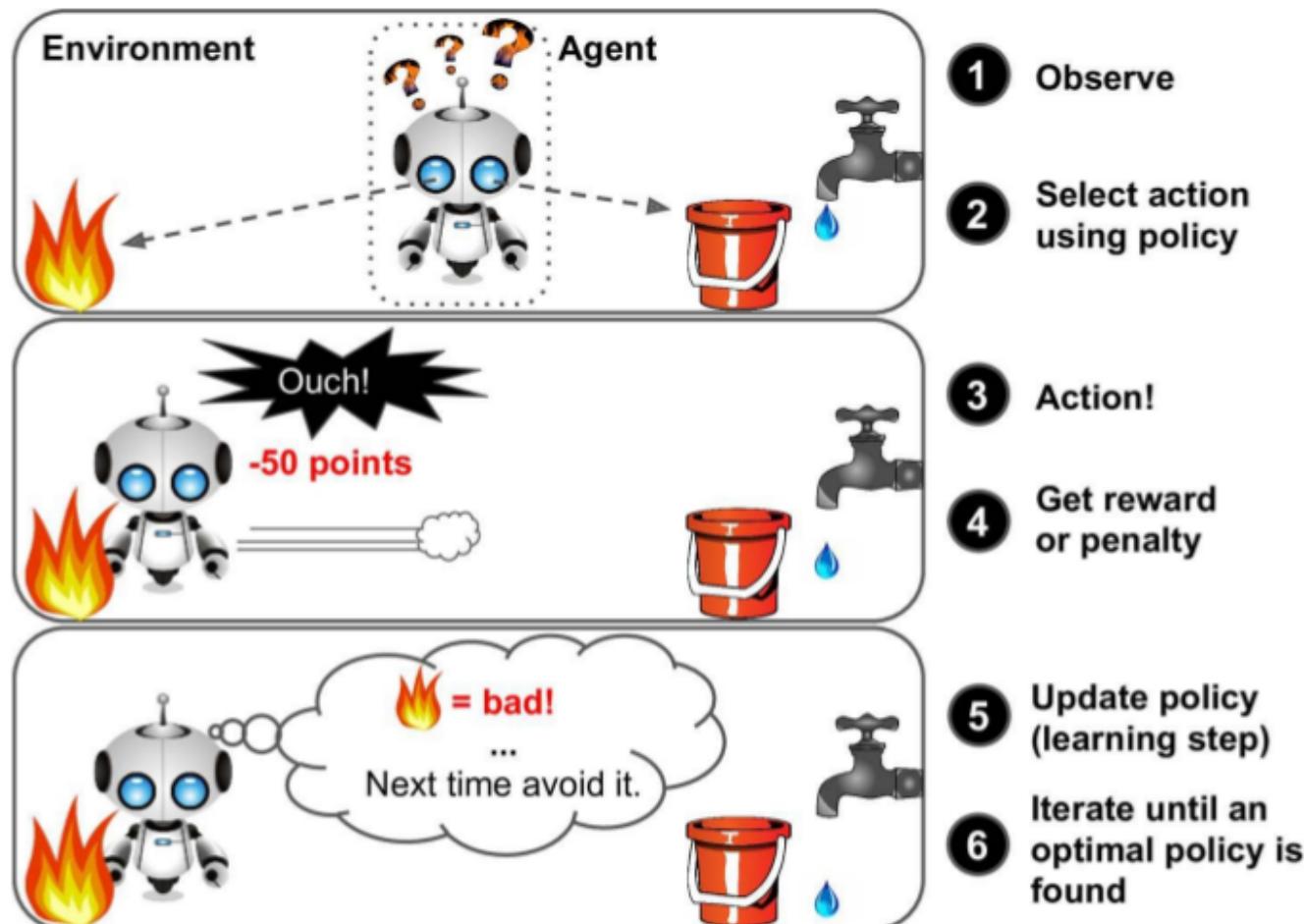


Figure 1-12. Reinforcement Learning

[1]

No aprendizado de reforço não há rótulos, porém há uma resposta, positiva ou negativa, para cada exemplo.

Aprendizado baseado em instâncias vs aprendizado baseado em modelos

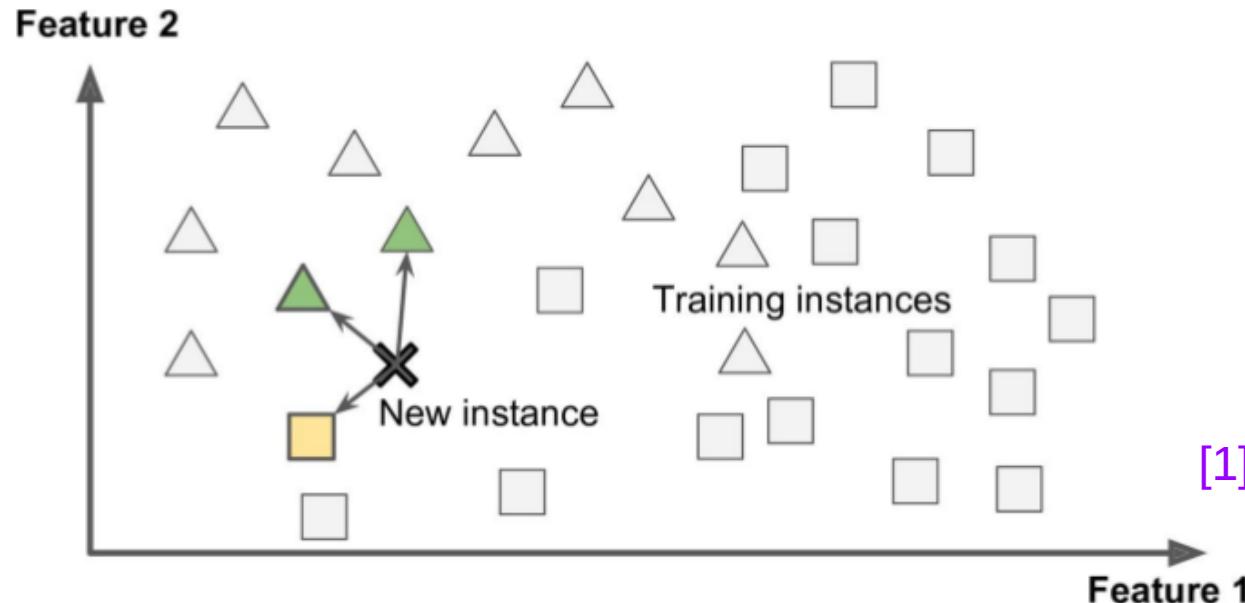


Figure 1-15. Instance-based learning

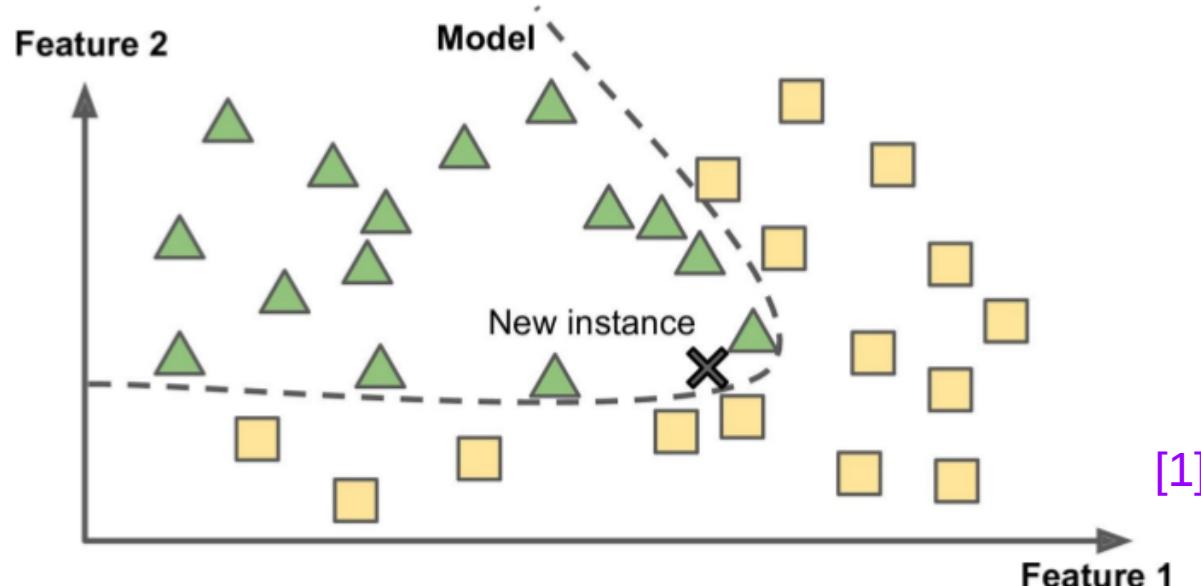


Figure 1-16. Model-based learning

Como funciona o aprendizado em ML

O problema central em ML é transformar os dados de maneira significativa; é aprender representações úteis dos dados de entrada.

Aprendizado, no contexto de ML, descreve um processo de busca automática por transformações de dados que produzem representações úteis dos inputs, guiado por um sinal de feedback.

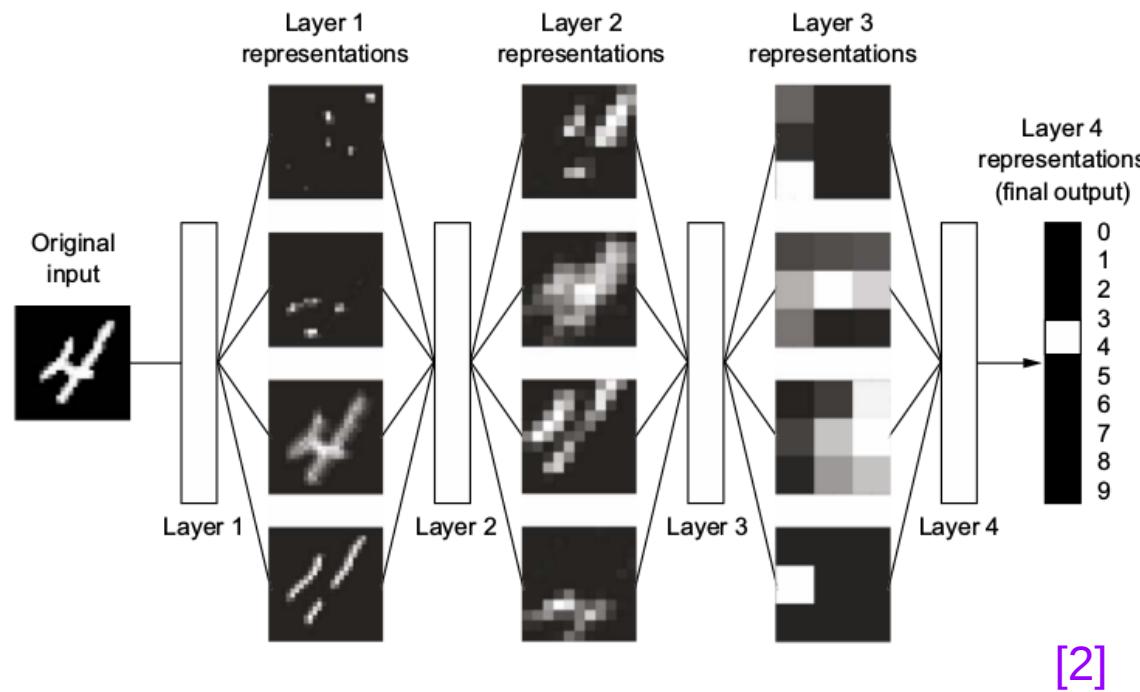
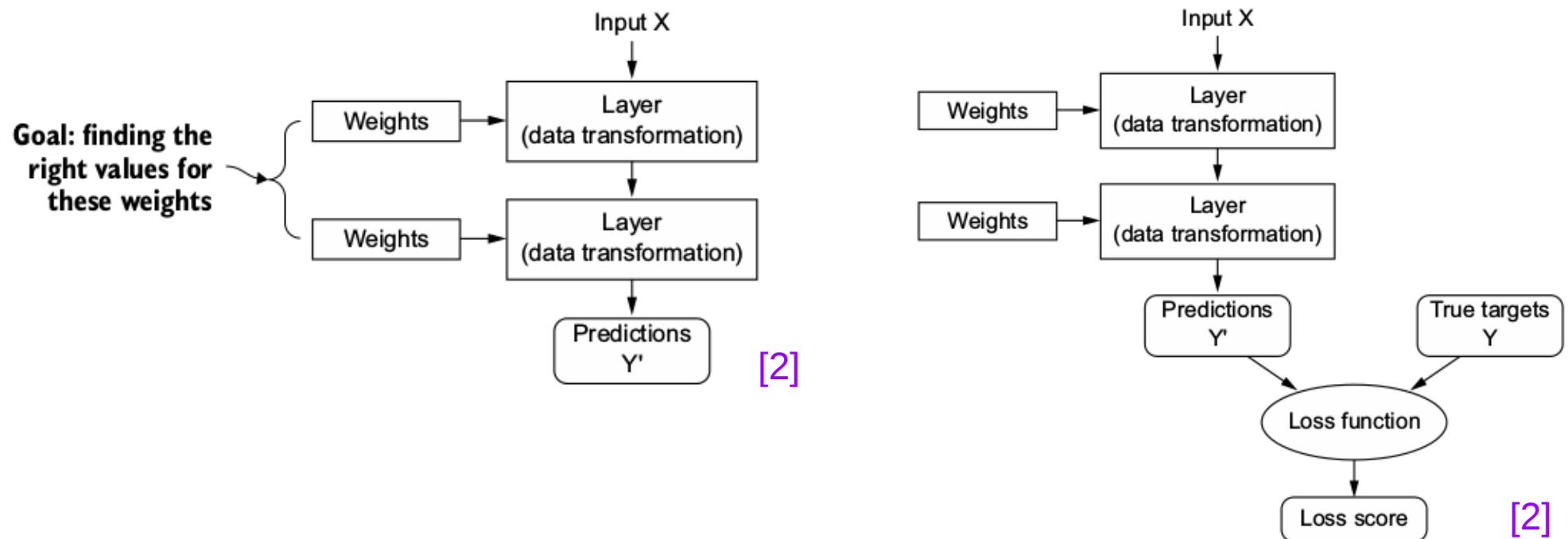


Figure 1.6 Data representations learned by a digit-classification model

Como funciona o aprendizado em ML/NN

Aprendizado de máquina consiste de um mapeamento de inputs a alvos, realizado ao observar vários exemplos de inputs e alvos. As redes realizam esse mapeamento via uma sequência de transformações simples de dados (camadas), com essas transformações aprendidas por exemplos.

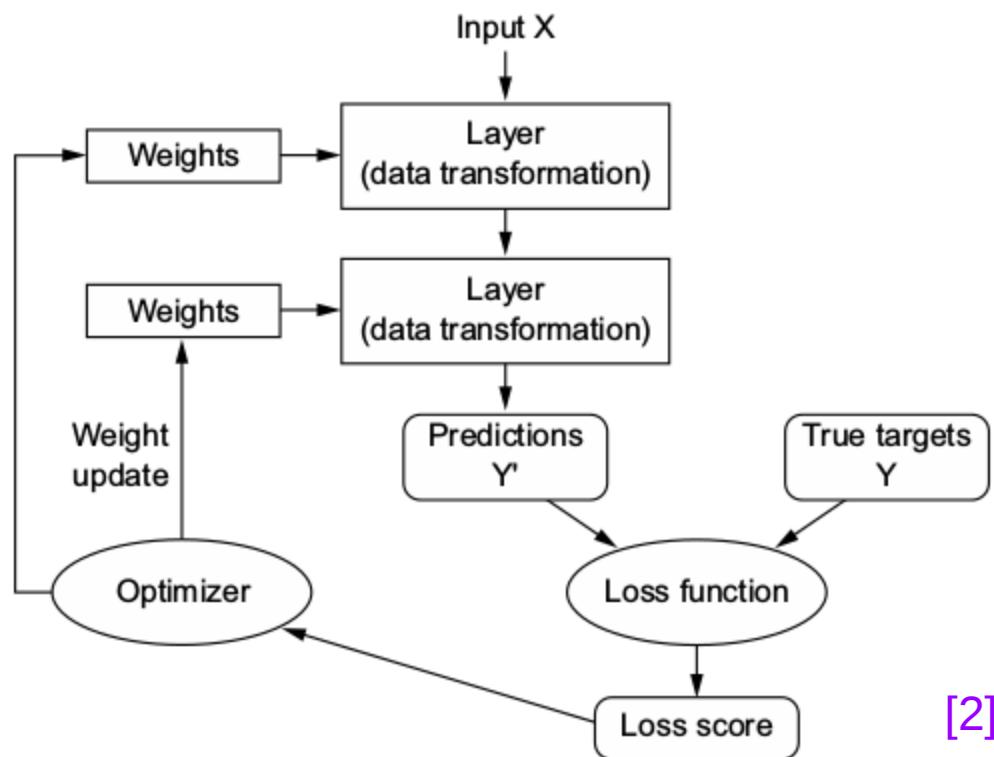
Neste contexto, *aprendizado* significa encontrar um conjunto de valores para os pesos (parâmetros) de todas as camadas da rede, tal que a rede irá corretamente mapear os exemplos de input aos seus respectivos alvos.



Para controlar o output de uma rede neural, precisamos quantificar quão distante este output está do valor esperado. Isto é controlado pelo função perda/custo/erro.

Como funciona o aprendizado em ML/NN

O truque fundamental nas NNs é utilizar o escore da função custo como um sinal de feedback para ajustar o valor dos pesos, na direção que irá diminuir a função custo para o exemplo atual. Esse ajuste é realizado pelo *otimizador*, implementado através do algoritmo de retropropagação (*backpropagation*).



O loop de aprendizado nas NNs

1. Selecione um batch de exemplos de treinamento, x , e alvos correspondentes, y_{true} ;
2. Rode o modelo em x para obter as previsões, y_{pred} (*forward pass*);
3. Calcule a perda do modelo no batch, uma medida do mismatch entre y_{pred} e y_{true} ;
4. Atualize os pesos do modelo de forma a reduzir suavemente a perda neste batch.

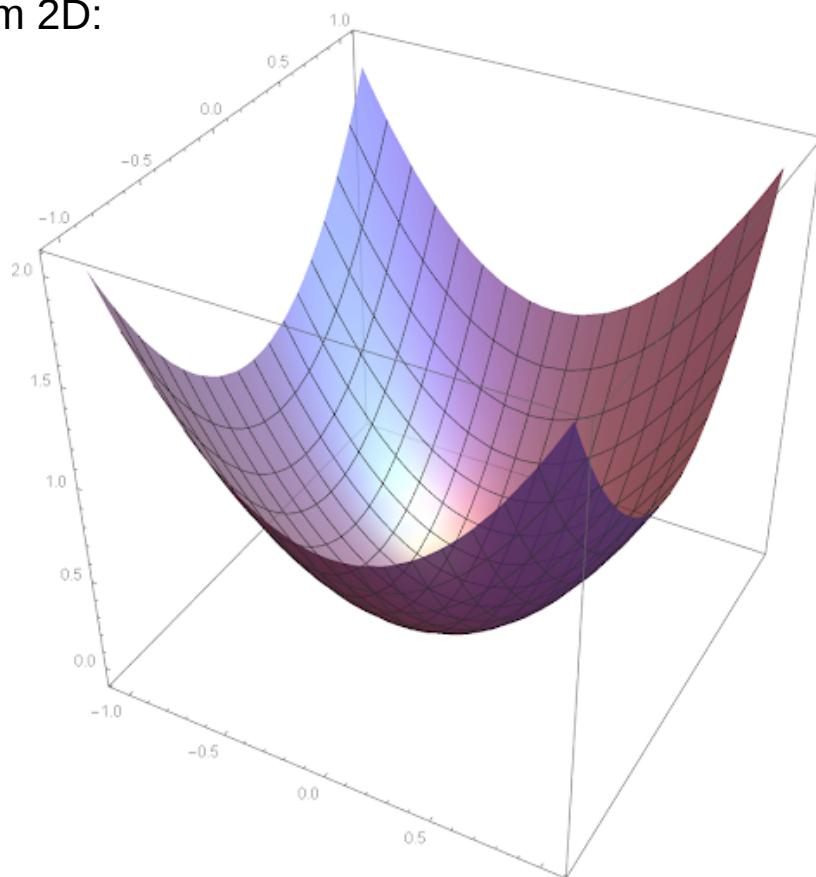
A etapa 4 é difícil: como atualizar os pesos de forma a sistematicamente reduzir a função custo?

Uso do *gradient descent* (GD, gradiente descendente), um método geral de otimização utilizado em NNs para minimizar a função custo, baseado no gradiente da função.

O método do gradiente descendente

O *gradiente* de uma função indica a direção e taxa de maior aumento da função em relação a suas variáveis.

Em 2D:

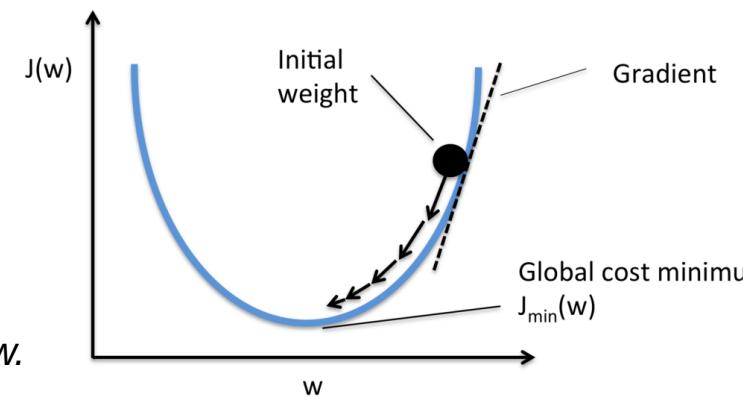
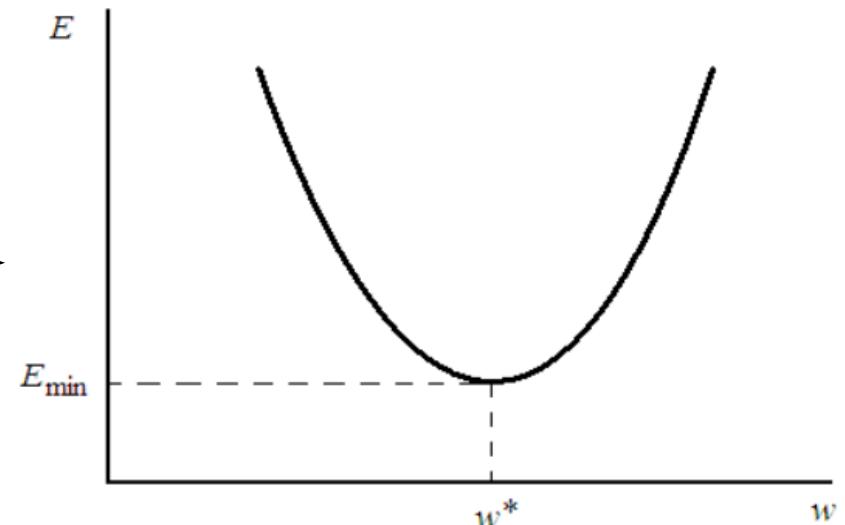


$$E = E(w_0, w_1) = E(\vec{w})$$

Equivalentemente, função custo: $J(\vec{w})$

Objetivo é encontrar o mínimo da função no “espaço de pesos” w .

Em 1D, para uma função erro quadrática:



O loop de aprendizado nas NNs

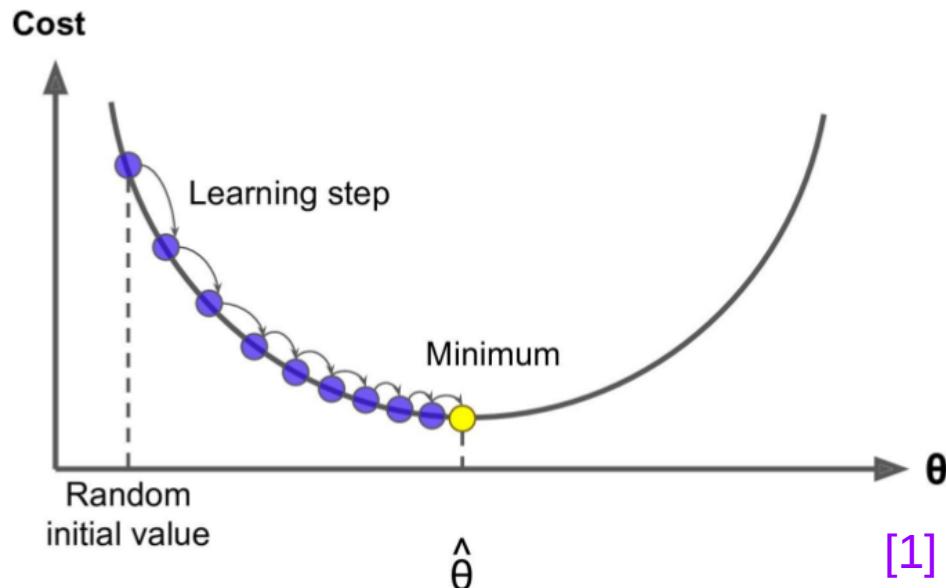
1. Selecione um batch de exemplos de treinamento, x , e alvos correspondentes, y_{true} ;
 2. Rode o modelo em x para obter as previsões, y_{pred} (*forward pass*);
 3. Calcule a perda do modelo no batch, uma medida do mismatch entre y_{pred} e y_{true} ;
 4. Calcule o gradiente da perda em relação aos parâmetros do modelo (*backward pass*);
 5. Atualize os parâmetros na direção oposta ao gradiente, reduzindo suavemente a perda neste batch:
- $$\Delta w = -\eta \nabla E$$

Este processo é chamado de *mini-batch stochastic gradient descent* (mini-batch SGD). Algumas variantes calculam o gradiente em cada iteração, ao invés de um batch: *true SGD*; no outro extremo, poderíamos rodar em todos os dados disponíveis: *batch gradient descent*.

Existe várias variantes do SGD, que levam em consideração atualizações anteriores dos pesos (*otimizadores* ou métodos de otimização): RMSprop, Adagrad, ...

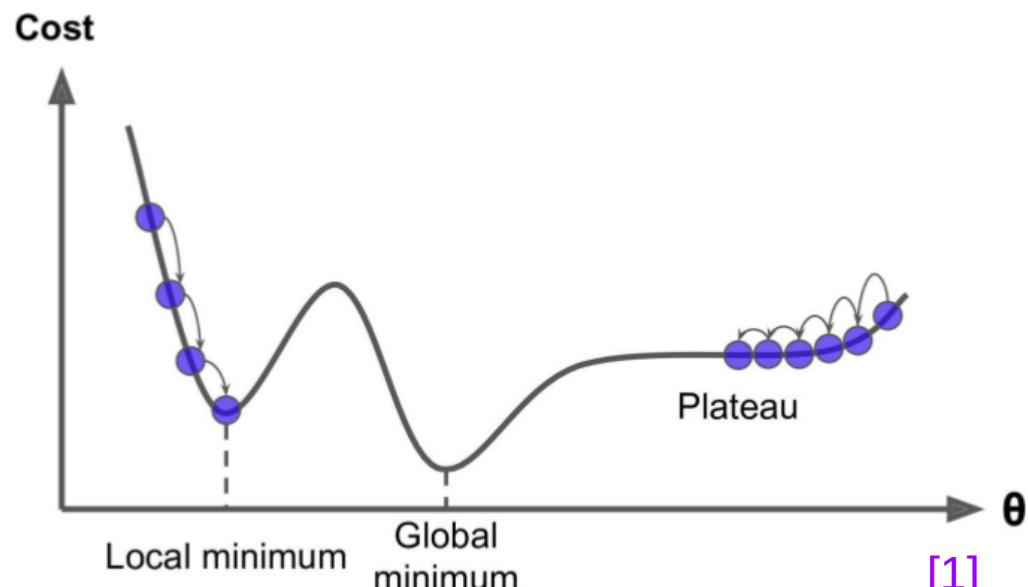
O método do gradiente descendente

Convergência do método depende da forma da função custo: $J(\theta)$
(ou, de maneira equivalente, da função erro $E(w)$)



[1]

Figure 4-3. Gradient Descent



[1]

Figure 4-6. Gradient Descent pitfalls

Em um modelo linear, o gradiente descendente garantirá a convergência ao mínimo (com uma razoável taxa de aprendizagem); em outros modelos, pode encontrar problemas.

Gradiente descendente

Gradiente descendente: $\Delta w = -\eta \nabla E$

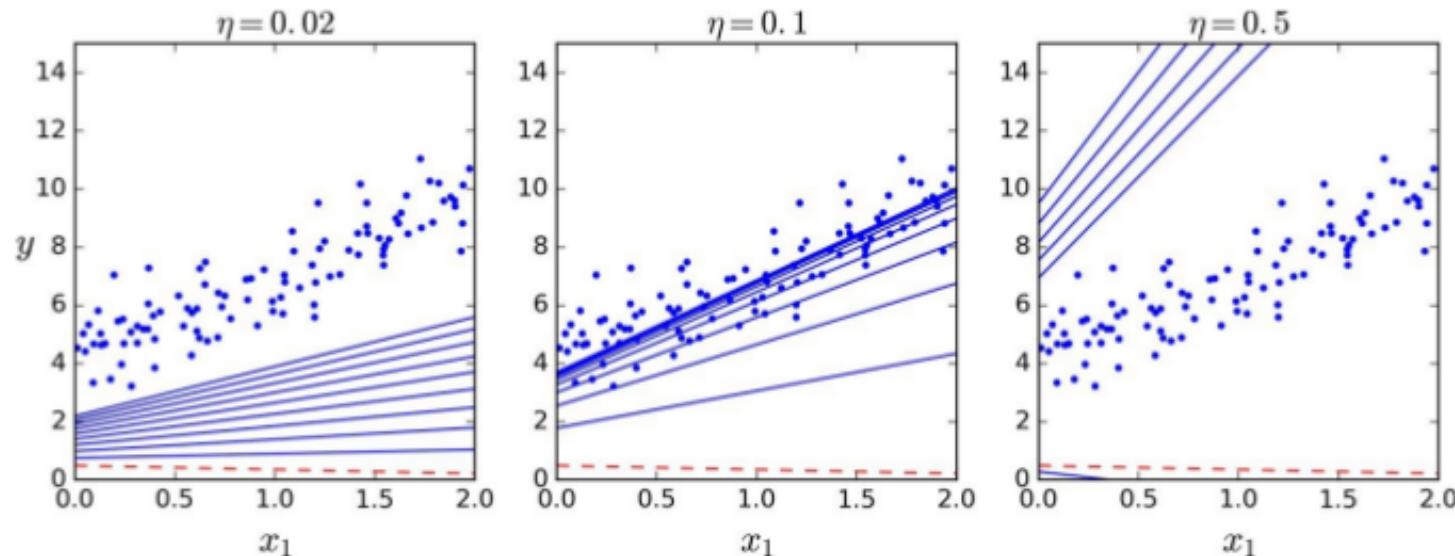


Figure 4-8. Gradient Descent with various learning rates

[1]

Convergência de um modelo linear de acordo com a taxa de aprendizagem.

Gradiente descendente

Convergência é mais eficiente se as características estiverem aproximadamente na mesma escala → *Escalonamento de características*.

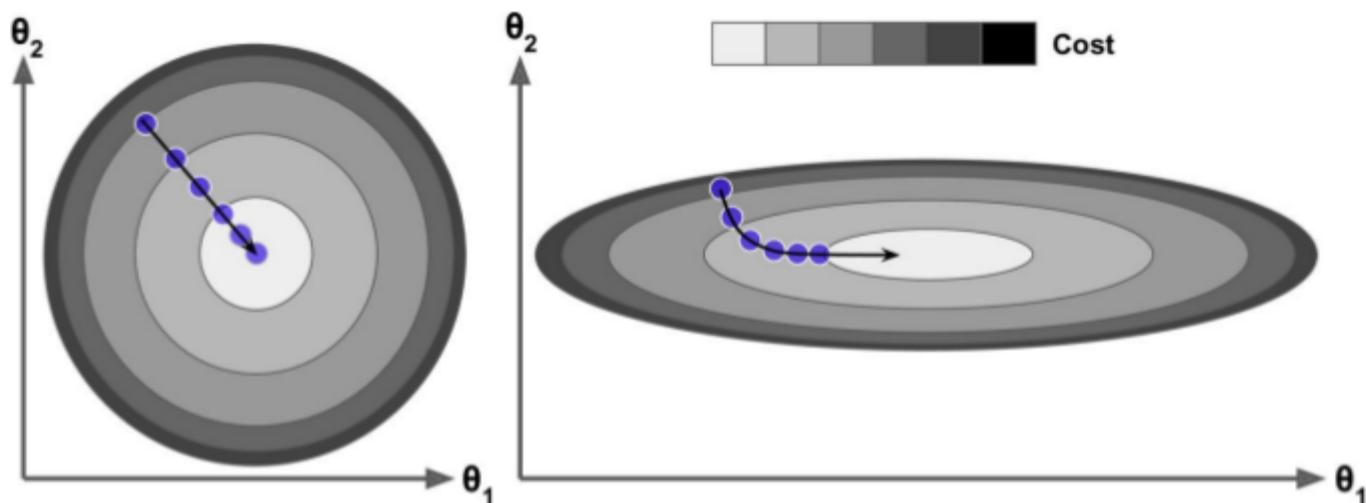


Figure 4-7. Gradient Descent with and without feature scaling

[1]

Pré-tratamento das características => *feature engineering*.

Gradiente descendente

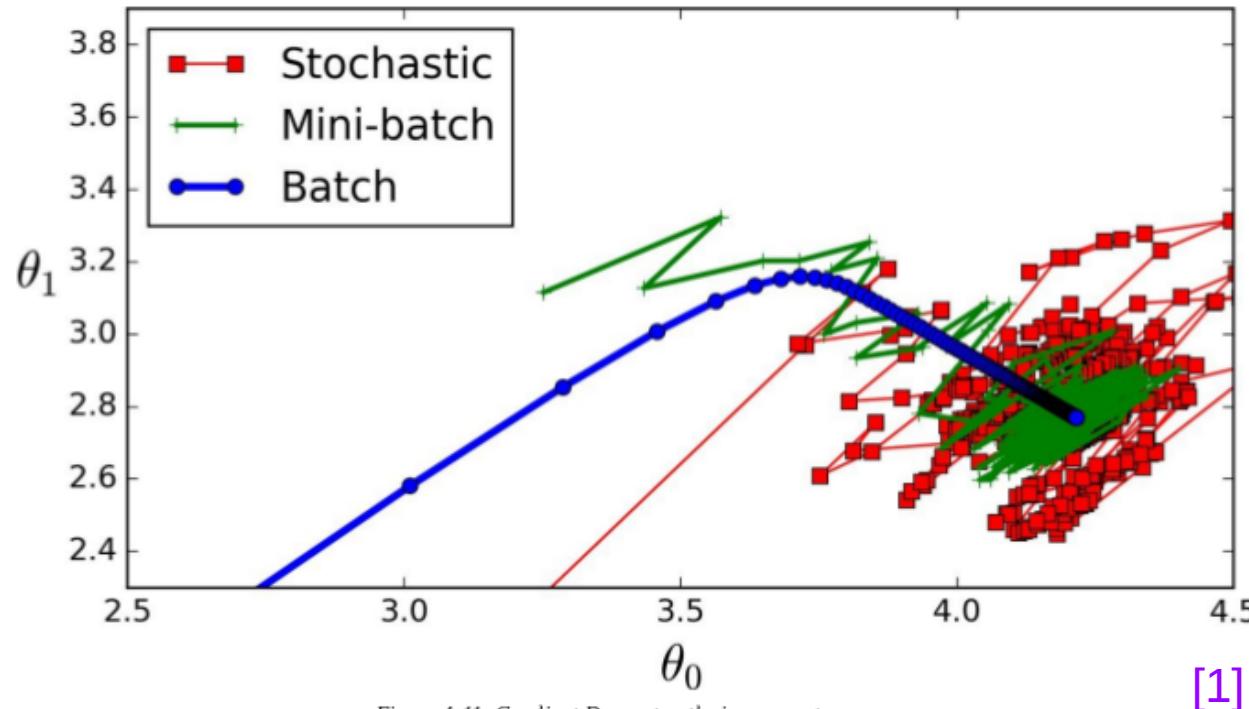


Figure 4-11. Gradient Descent paths in parameter space

Variação de convergência para o gradiente descendente estocástico, batch e mini-batch.

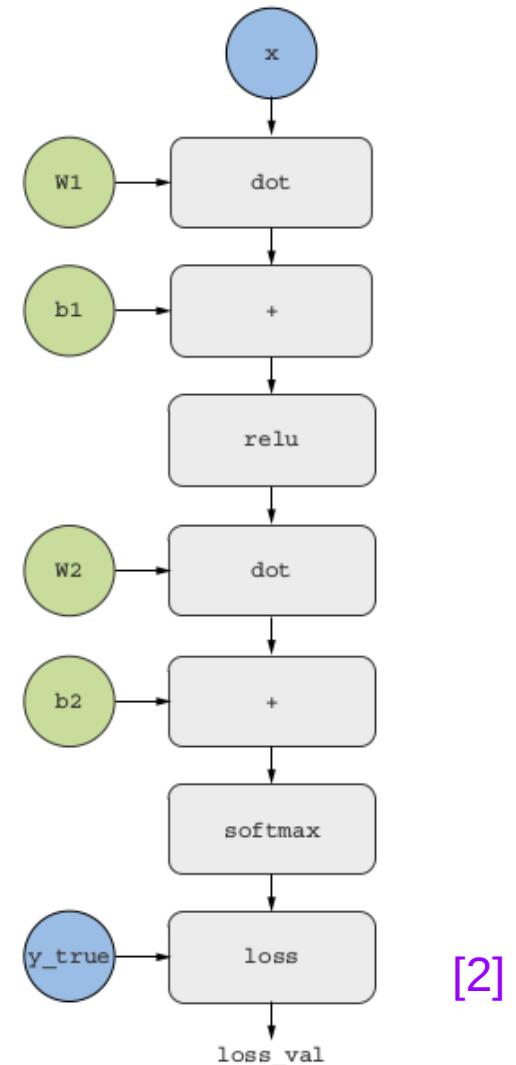
Taxa de convergência depende do tamanho do batch; ruído pode ajudar na convergência.

O algoritmo de *backpropagation*

Backpropagation é um método eficiente para calcular gradientes; é basicamente uma interpretação simples da regra da cadeia, que permite calcular todas as derivadas parciais linearmente em termos do tamanho do grafo.

Backprop = uso da regra da cadeia da última camada para a primeira

Diferenciação automática com grafos computacionais:
(automatic differentiation)



O algoritmo de *backpropagation*

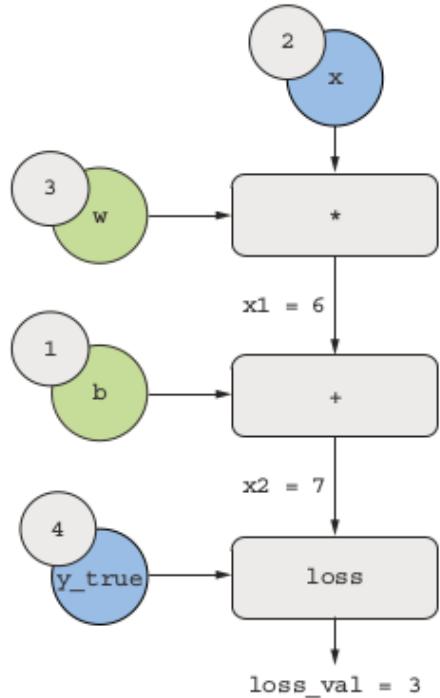


Figure 2.23 Running a forward pass

$$x1 = x \cdot w = 2 \cdot w$$

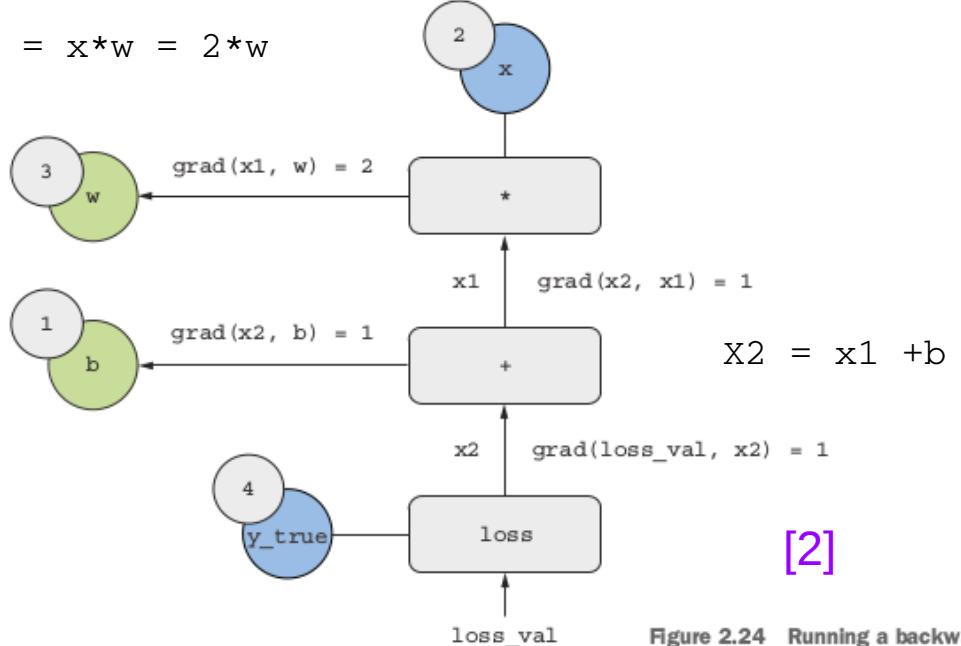


Figure 2.24 Running a backward pass

$$\text{loss_val} = \text{abs}(y_{\text{true}} - x2)$$

O algoritmo de *backpropagation*

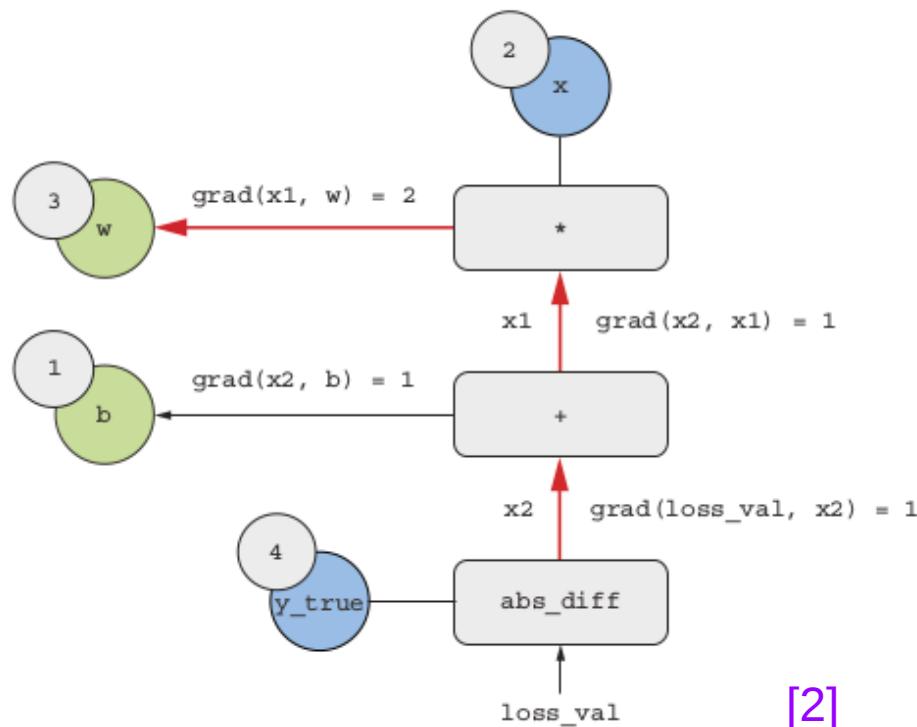


Figure 2.25 Path from $loss_val$ to w in the backward graph

$grad(loss_val, w) = grad(loss_val, x2) * grad(x2, x1) * grad(x1, w)$

$grad(loss_val, b) = grad(loss_val, x2) * grad(x2, b)$

O algoritmo de *backpropagation*

O GradientTape() no TensorFlow: É um scope do Python que irá “gravar” as operações de tensor que rodam em si, na forma de grafos computacionais (chamados de “tape”).

```
Instantiate a scalar Variable  
with an initial value of 0.  
  
import tensorflow as tf  
x = tf.Variable(0.)  
with tf.GradientTape() as tape:  
    y = 2 * x + 3  
grad_of_y_wrt_x = tape.gradient(y, x)  
  
Open a GradientTape scope.  
  
Inside the scope, apply  
some tensor operations  
to our variable.  
  
Use the tape to retrieve the  
gradient of the output y with  
respect to our variable x.
```



```
x = tf.Variable(tf.random.uniform((2, 2)))  
with tf.GradientTape() as tape:  
    y = 2 * x + 3  
grad_of_y_wrt_x = tape.gradient(y, x)
```


Instantiate a Variable with shape (2, 2) and an initial value of all zeros.

grad_of_y_wrt_x is a tensor of shape (2, 2) (like x) describing the curvature of $y = 2 * a + 3$ around $x = [[0, 0], [0, 0]]$.


```
W = tf.Variable(tf.random.uniform((2, 2)))  
b = tf.Variable(tf.zeros((2,)))  
x = tf.random.uniform((2, 2))  
with tf.GradientTape() as tape:  
    y = tf.matmul(x, W) + b  
grad_of_y_wrt_W_and_b = tape.gradient(y, [W, b])
```

matmul is how you say “dot product” in TensorFlow.

grad_of_y_wrt_W_and_b is a list of two tensors with the same shapes as W and b, respectively.

O que são o Keras e o TensorFlow?

TensorFlow [<https://tensorflow.org/>] é uma plataforma de aprendizado de máquina baseada em Python, livre e *open source*, desenvolvida principalmente pelo Google.

Objetos do TF: Tensores, variáveis, operações sobre tensores, *gradient tape*...

Keras [<https://keras.io/>] é uma API de aprendizado profundo para Python, construída sobre a TensorFlow, que fornece uma maneira conveniente de definir e treinar qualquer tipo de modelo de aprendizado profundo.

Classe central do Keras: Layer.

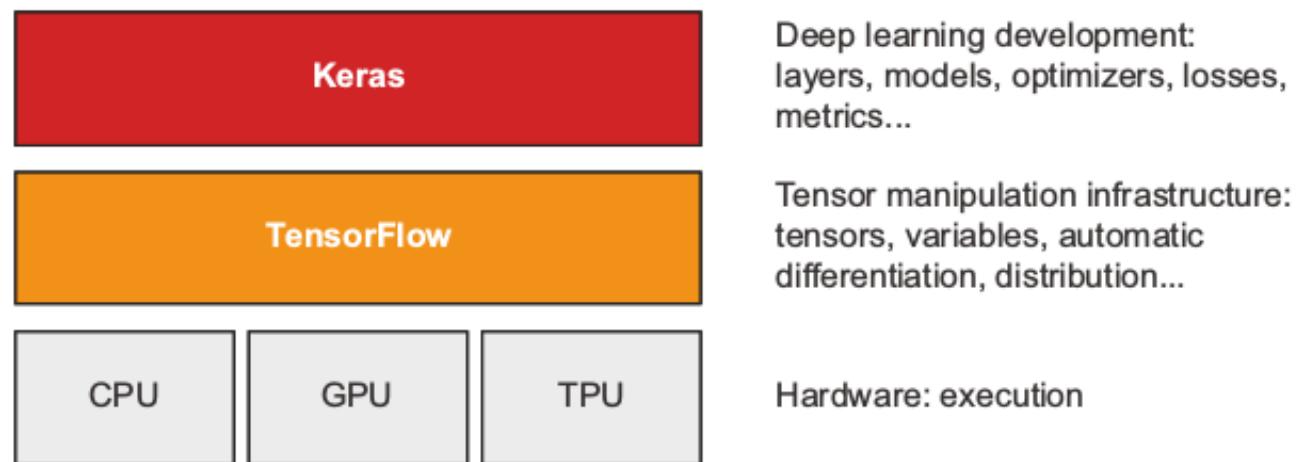


Figure 3.1 Keras and TensorFlow: TensorFlow is a low-level tensor computing platform, and Keras is a high-level deep learning API

Por que o Keras?

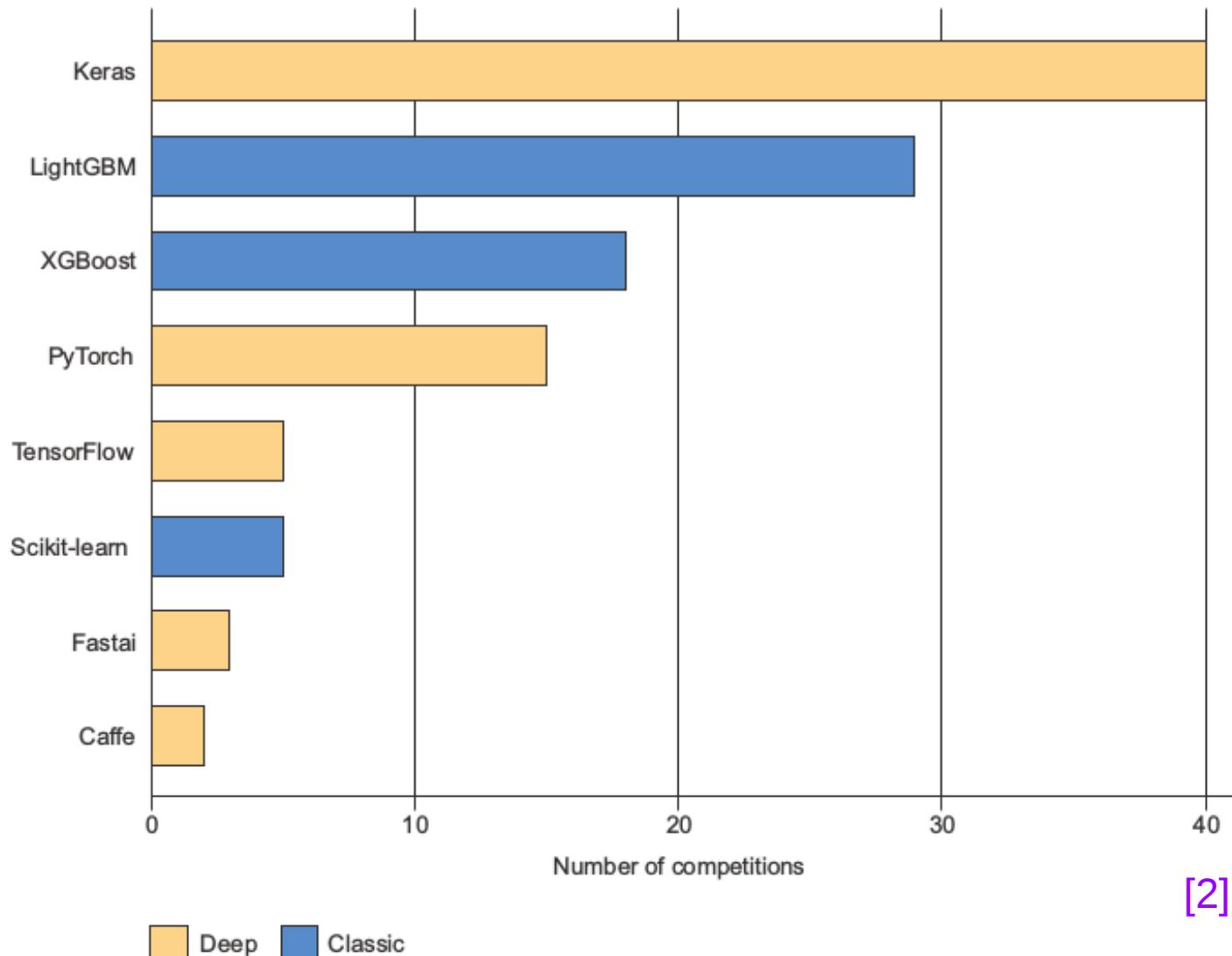


Figure 1.12 Machine learning tools used by top teams on Kaggle

Por que o Keras?

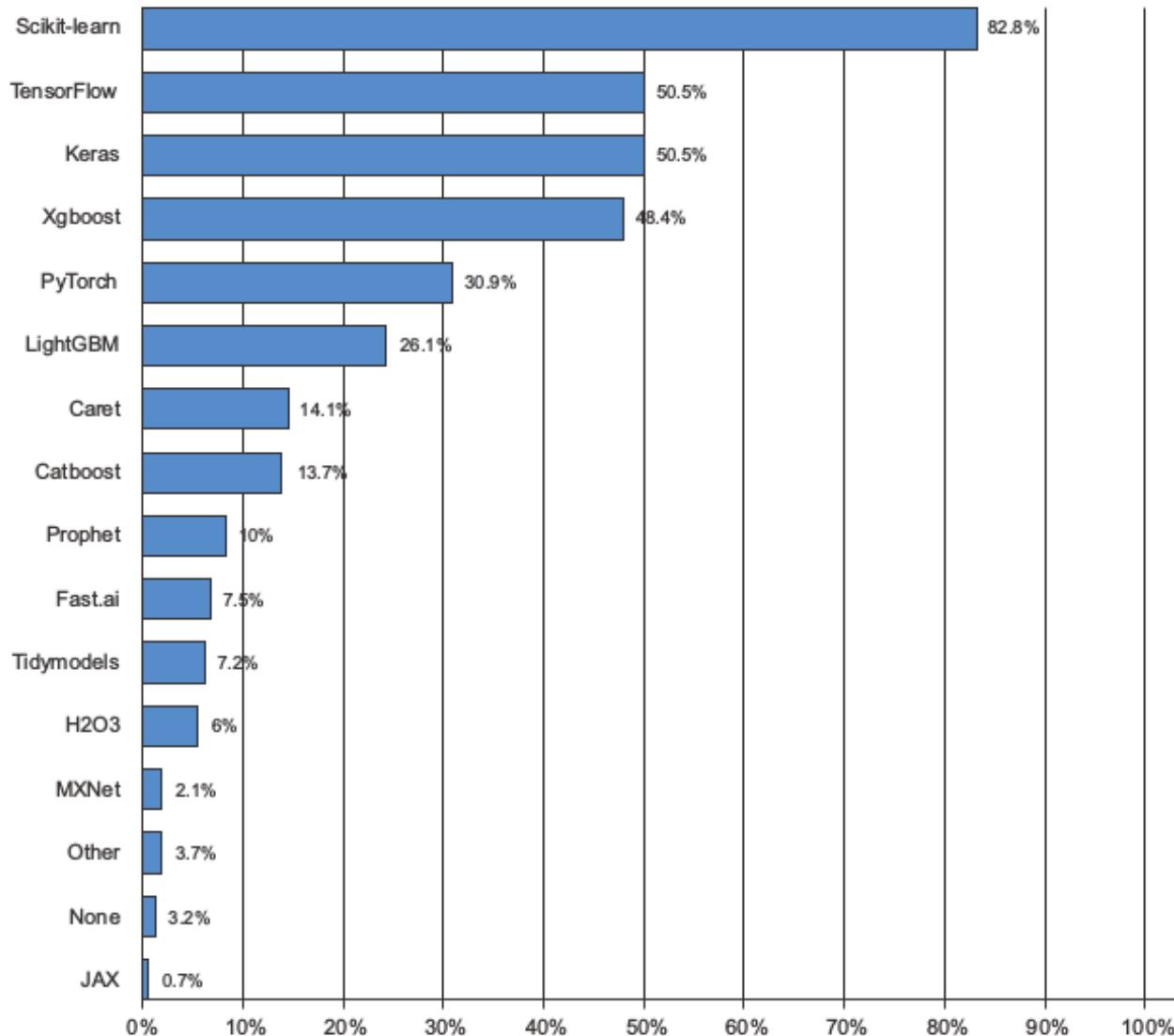


Figure 1.13 Tool usage across the machine learning and data science industry (Source: www.kaggle.com/kaggle-survey-2020)

Exemplos no Google colab

<https://colab.research.google.com>

Arquivos em:

<https://github.com/roboLiboni/2025-EFA>

Etapas básicas no uso do Keras

1. Carregar os dados:

```
from tensorflow.keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

Dados (sempre) separados em um conjunto de treinamento e outro de teste!

2. Definir a arquitetura da rede:

```
from tensorflow import keras
from tensorflow.keras import layers
model = keras.Sequential([
    layers.Dense(512, activation="relu"),
    layers.Dense(10, activation="softmax")
])
```

3. Compilar o modelo:

```
model.compile(optimizer="rmsprop",
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])
```

4. Preparar os dados:

```
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype("float32") / 255
test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype("float32") / 255
```

5. Ajustar (*fitar*) o modelo:

```
>>> model.fit(train_images, train_labels, epochs=5, batch_size=128)
Epoch 1/5
60000/60000 [=====] - 5s - loss: 0.2524 - acc: 0.9273
Epoch 2/5
51328/60000 [=====>.....] - ETA: 1s - loss: 0.1035 - acc: 0.9692
```

6. Utilizar o modelo para fazer previsões:

```
>>> test_digits = test_images[0:10]
>>> predictions = model.predict(test_digits)
>>> predictions[0]
array([1.0726176e-10, 1.6918376e-10, 6.1314843e-08, 8.4106023e-06,
       2.9967067e-11, 3.0331331e-09, 8.3651971e-14, 9.9999106e-01,
       2.6657624e-08, 3.8127661e-07], dtype=float32)
```

7. Avaliar o modelo em novos dados:

```
>>> test_loss, test_acc = model.evaluate(test_images, test_labels)
>>> print(f"test_acc: {test_acc}")
test_acc: 0.9785
```

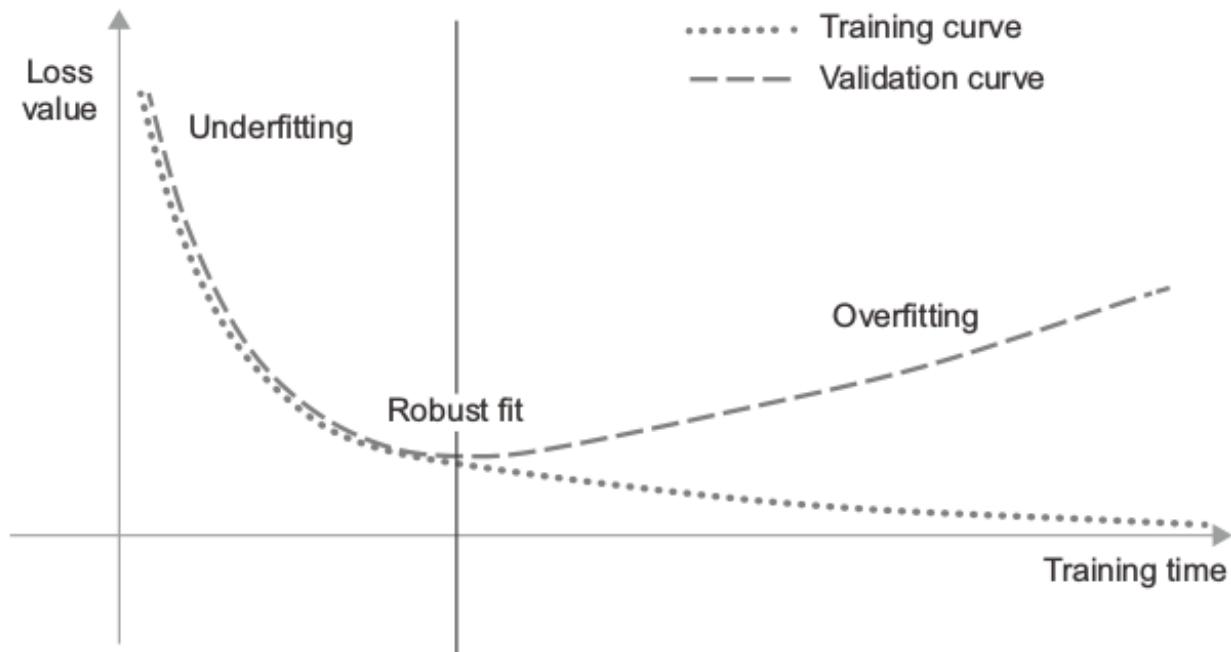
Generalização vs otimização

Geralmente o desempenho no conjunto de teste é menor que o obtido no conjunto de treinamento.

Obter um bom desempenho no conjunto de treinamento é relativamente simples; a rede estaria *memorizando*, e não *generalizando* o problema, que é o objetivo do aprendizado.

Normalmente separa-se uma parte do conjunto de treinamento como *conjunto de validação*, para avaliar o potencial de generalização do modelo.

Em todo modelo de ML, há um balanço entre otimização e generalização; entre *subajuste* (*underfitting*) e *sobreajuste* (*overfitting*).



[2]

Figure 5.1 Canonical overfitting behavior

Possíveis razões de sobreajuste nos dados de treinamento: Ruídos, incerteza, características raras...

Exemplos: Regressão e classificação

1. *Prática:* Estimar o valor de solubilidade de compostos;
2. *Prática:* Classificar compostos em ativos ou inativos;
3. *Exemplo:* Desenvolvimento de campo de força com redes neurais.

Exemplo: Regressão

Estimar o valor de solubilidade de compostos

J. Chem. Inf. Comput. Sci. **2004**, *44*, 1000–1005

ESOL: Estimating Aqueous Solubility Directly from Molecular Structure

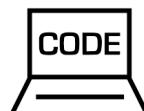
John S. Delaney*

Syngenta, Jealott's Hill International Research Centre, Bracknell, Berkshire, RG42 6EY, United Kingdom

Received October 29, 2003

This paper describes a simple method for estimating the aqueous solubility (ESOL – Estimated SOLubility) of a compound directly from its structure. The model was derived from a set of 2874 measured solubilities using linear regression against nine molecular properties. The most significant parameter was calculated $\log P_{\text{octanol}}$, followed by molecular weight, proportion of heavy atoms in aromatic systems, and number of rotatable bonds. The model performed consistently well across three validation sets, predicting solubilities within a factor of 5–8 of their measured values, and was competitive with the well-established “General Solubility Equation” for medicinal/agrochemical sized molecules.

<https://doi.org/10.1021/ci034243x>



regressao.ipynb

Exemplo: Regressão

Estimar o valor de solubilidade de compostos

Conjunto de validação: Verificar a capacidade de generalização do modelo (10-30% dos dados)

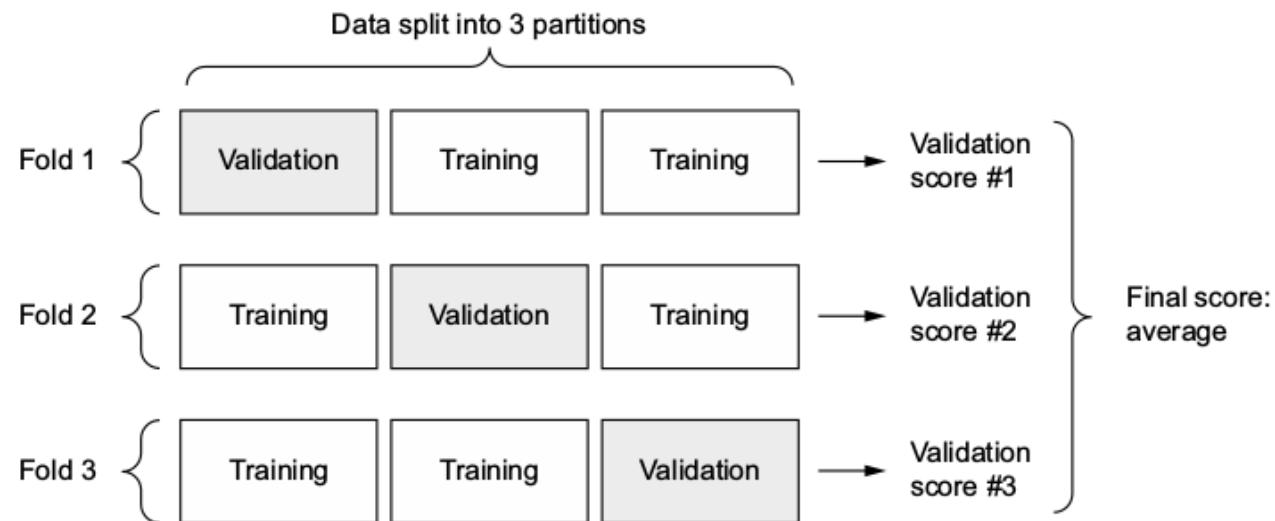


Figure 5.13 K-fold cross-validation with $K=3$

[2]



regressao.ipynb

Exemplo: Classificação

Qual composto é ativo?

RDKit é uma ferramenta *open source* para bioinformática. <https://www.rdkit.org/>
<https://www.rdkit.org/docs/GettingStartedInPython.html>

ChemBL é um base de dados de moléculas bioativas com propriedades farmacêuticas.
<https://www.ebi.ac.uk/chembl/>

Formato Simplified Molecular Input Line Entry System (SMILES).

<https://doi.org/10.1021/ci00057a005>

https://en.wikipedia.org/wiki/Simplified_Molecular_Input_Line_Entry_System



classificacao.ipynb

Exemplo

Desenvolvimento de campo de força com redes neurais

PRL 98, 146401 (2007)

PHYSICAL REVIEW LETTERS

week ending
6 APRIL 2007

Generalized Neural-Network Representation of High-Dimensional Potential-Energy Surfaces

Jörg Behler and Michele Parrinello

Department of Chemistry and Applied Biosciences, ETH Zurich, USI-Campus, Via Giuseppe Buffi 13, CH-6900 Lugano, Switzerland

(Received 27 September 2006; published 2 April 2007)

The accurate description of chemical processes often requires the use of computationally demanding methods like density-functional theory (DFT), making long simulations of large systems unfeasible. In this Letter we introduce a new kind of neural-network representation of DFT potential-energy surfaces, which provides the energy and forces as a function of all atomic positions in systems of arbitrary size and is several orders of magnitude faster than DFT. The high accuracy of the method is demonstrated for bulk silicon and compared with empirical potentials and DFT. The method is general and can be applied to all types of periodic and nonperiodic systems.

DOI: [10.1103/PhysRevLett.98.146401](https://doi.org/10.1103/PhysRevLett.98.146401)

PACS numbers: 71.15.Pd, 61.50.Ah, 82.20.Kh

<https://doi.org/10.1103/PhysRevLett.98.146401>

Exemplo

Desenvolvimento de campo de força com redes neurais

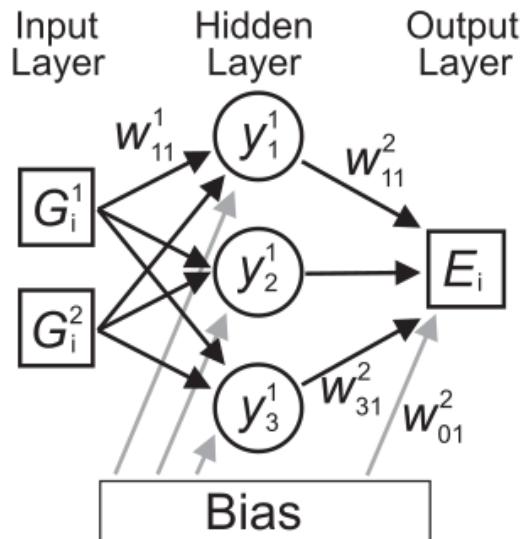


FIG. 1. Example of a standard neural network employed for fitting potential-energy surfaces [5,6]. The node in the output layer yields the energy E_i , which in this case depends on the values of the two input nodes, G_i^1 and G_i^2 . In between the input and the output layer there is a hidden layer with three nodes represented by the circles. The arrows correspond to the 13 weight parameters w_{ij}^k , which connect node j in layer k with node i in layer $k - 1$. The bias node is used to adapt the nonlinearity region of the activation functions. The functional form of this small network is given in Eq. (1).

Desenvolvimento de HDNNPs, *high-dimensional neural network potentials*

Desvantagens da abordagem da Fig. 1:
→ Ordem das coordenadas inseridas importa!
→ Tamanho fixo (# átomos);

Exemplo

Desenvolvimento de campo de força com redes neurais

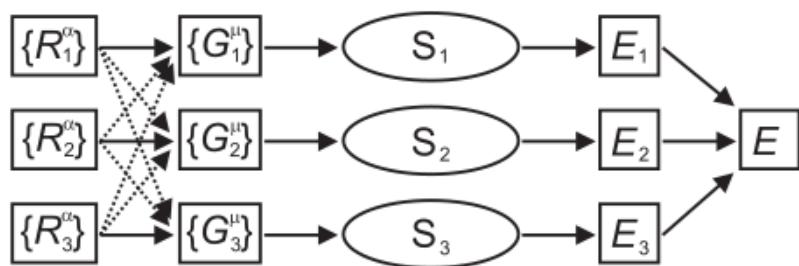


FIG. 2. Structure of the neural network as applied in this Letter to a system containing three atoms. The Cartesian coordinates of atom i are given by R_i^α . These are transformed to a set of μ symmetry function values G_i^μ describing the local geometric environment of atom i , which depends on the positions of all atoms in the system as indicated by the dotted arrows. The symmetry function values of atom i then enter the subnet S_i yielding the energy contribution E_i of atom i to the total energy of the system E . The structure of the subnets corresponds to the neural network shown in Fig. 1.

Uso de funções de simetria G_i
(atom-centered symmetry fcts, ACSFs)

G_1^i : Dois parâmetros η, R_s
 G_2^i : Três parâmetros λ, η, ξ

Si (64 átomos), com base em 9000 estruturas (experimentais+MD); RMSE \sim 5-6 meV, forças \sim 0.2 eV/Angs

2 camadas ocultas com 40 nós cada,
48 funções de simetria

$$G_i^1 = \sum_{j \neq i}^{\text{all}} e^{-\eta(R_{ij}-R_s)^2} f_c(R_{ij}). \quad G_i^2 = 2^{1-\zeta} \sum_{j,k \neq i}^{\text{all}} (1 + \lambda \cos \theta_{ijk})^\zeta \\ \times e^{-\eta(R_{ij}^2 + R_{ik}^2 + R_{jk}^2)} f_c(R_{ij}) f_c(R_{ik}) f_c(R_{jk}),$$

Exemplo

Desenvolvimento de campo de força com redes neurais

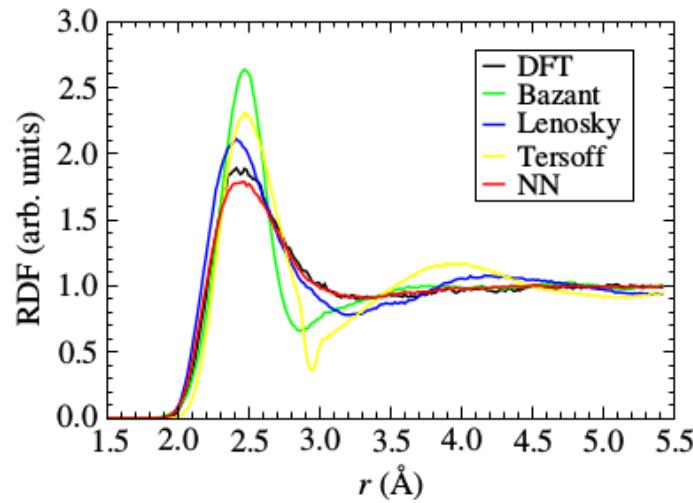


FIG. 3 (color online). Radial distribution function (RDF) of a silicon melt at 3000 K as obtained using a cubic 64 atom cell ($a = 20.526$ bohr). The curves shown were obtained from the Bazant [17,19], the Lenosky [15,19], the Tersoff [16,20], a neural network (NN) potential, and from density-functional theory (DFT) [18].

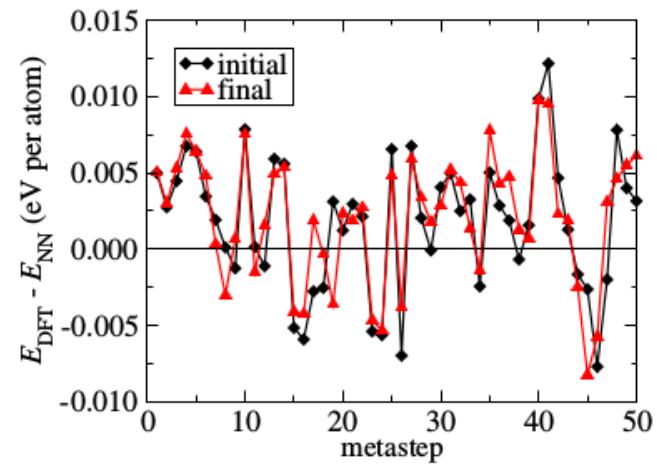


FIG. 4 (color online). Difference between the energies predicted by the neural network (NN) and recalculated energies obtained from density-functional theory (DFT) for the initial and final structures in each step of a metadynamics simulation [14] of bulk silicon. Each metastep involves a molecular dynamics simulation of 2 ps. The metadynamics simulation for the 64 atom cell starts from the β -tin structure with a pressure of 15 GPa at 300 K.

Library-Based LAMMPS Implementation of High-Dimensional Neural Network Potentials. <https://doi.org/10.1021/acs.jctc.8b00770>

Constructing high-dimensional neural network potentials: A tutorial review. <https://doi.org/10.1002/qua.24890>

Exemplo

Desenvolvimento de campo de força com redes neurais

nature communications



Article

<https://doi.org/10.1038/s41467-024-46772-0>

Dynamics of the charge transfer to solvent process in aqueous iodide

Received: 21 September 2023

Jinggang Lan ^{1,2,3} , Majed Chergui ^{4,5} & Alfredo Pasquarello ¹

Accepted: 5 March 2024

Published online: 21 March 2024

Check for updates

Charge-transfer-to-solvent states in aqueous halides are ideal systems for studying the electron-transfer dynamics to the solvent involving a complex interplay between electronic excitation and solvent polarization. Despite extensive experimental investigations, a full picture of the charge-transfer-to-solvent dynamics has remained elusive. Here, we visualise the intricate interplay between the dynamics of the electron and the solvent polarization occurring in this process. Through the combined use of ab initio molecular dynamics and machine learning methods, we investigate the structure, dynamics and free energy as the excited electron evolves through the charge-transfer-to-solvent process, which we characterize as a sequence of states denoted charge-transfer-to-solvent, contact-pair, solvent-separated, and hydrated electron states, depending on the distance between the iodine and the excited electron. Our assignment of the charge-transfer-to-solvent states is supported by the good agreement between calculated and measured vertical binding energies. Our results reveal the charge transfer process in terms of the underlying atomic processes and mechanisms.

<https://doi.org/10.1038/s41467-024-46772-0>

A seguir

- Predição de séries temporais e o uso de redes neurais recorrentes;
- Aprendizado profundo aplicado a texto: Modelos geratitivos.