

Aprendizado de máquina

(1)

- Aprendizado supervisionado: "ensinar" o computador a dar respostas a dados de entrada desconhecidos a partir de dados cujas respostas são conhecidas.
(dados de treinamento)
- Onde entram os algoritmos de otimização?
↳ no ajuste dos parâmetros do sistema que dá as respostas! (é o treinamento)

Exemplo motivacional:

(2)

Suponha que queiramos treinar o computador para decidir se um dado texto discute sobre esportes.

- São dados n textos $(x_1, y_1), \dots, (x_n, y_n)$ para o treinamento, em que sabemos se são sobre esporte ou não:

- * $x_i \in \mathbb{R}^m$ vetor de características do texto (palavras, por exemplo)

- * $y_i \in \{-1, 1\}$; $1 = \text{esporte}$, $-1 = \text{não}$.

Devemos definir uma função de predição (3)
 h , cuja qualidade é medida contando o
número de respostas erradas ($h(x_i) \neq y_i$).

Neste sentido, queremos minimizar o
risco empírico de uma má classificação

$$R_n(h) = \frac{1}{n} \sum_{i=1}^n \mathbb{1}[h(x_i) \neq y_i],$$

onde

$$\mathbb{1}(P) = \begin{cases} 1 & \text{se } P \text{ é verdadeiro} \\ 0 & \text{caso contrário.} \end{cases}$$

Problemas:

14

- 1) R_n nem ao menos é contínua...
- 2) mesmo que $R_n(h) = 0$ (h nunca erra nos dados de treinamento), pode ser que h seja ruim para prever a resposta em dados desconhecidos (overfitting)

Ex.: $\tilde{h}(x) = \begin{cases} y_i & \text{se } x = x_i \text{ p/ algum } i \\ \pm 1 & \text{caso contrário (aleatório).} \end{cases}$

Veja que $R_n(\tilde{h}) = 0$, mas \tilde{h} pode ser (5)
qualquer coisa sobre dados desconhecidos...

Como resolver?

- Escolher h dentro de uma classe de funções (por exemplo, funções afins)
- Uma escolha adequada é feita por testes numéricos para o problema considerado.

- de certa forma, h deve capturar as \mathcal{L} características fundamentais do problema. Assim, teremos respostas corretas a dados desconhecidos mesmo que o treinamento seja feito com número limitado de dados conhecidos (evidentemente os dados de treinamento devem ser representativos)
- é desejável h diferenciável e fácil de computar (geralmente o treinamento é feito com muitos (x_i, y_i) ($n \gg 1$)).

Os conjuntos de dados coletados por experimentação (resposta manual, humana) são divididos em: (7)

- Dados de treinamento: usados para treinar o computador ($\min R_n(h)$).
- Dados de validação: usados para escolha da melhor função de predição h (já otimizada)
- Dados de teste: usados para medir a eficiência da h escolhida.

Escolha comum:

18

$$h(x) = h(x; w, b) = w^t x + b,$$

w : pesos;
 b : vies

> devem ser otimizados para
buscar $h(x_i; w^*, b^*) = y_i$.

Funciona bem em vários problemas.

$$R_n(w, b) = \frac{1}{n} \sum_{i=1}^n \text{ sinal}(h(x_i; w, b), y_i)$$

$\text{ sinal}(h(x_i; w, b), y_i) \in \{0, 1\}$ e é descontinua.

↳ intratável para $n \gg 1$ por métodos de PNL.

Em vez de utilizar "sinal", utilizamos 19
funções contínuas aproximadoras l (funções
de perda), escolhidas por experimentação numérica.

Exemplos

$$1) \quad l(h, y) = \log(1 + e^{-hy}) \quad (\text{para } h \in]-1, 1[)$$

Veja que $\min l(h, y)$ é atingido quando
 $-hy = 1$, isto é, h e y têm mesmo sinal.

$$2) \quad l(h, y) = (h - y)^2.$$

Função a ser minimizada:

$$R_m(\bullet) = \frac{1}{m} \sum_{i=1}^m \ell(h(x_i; \bullet), y_i).$$

h como sucessivas aplicações — Redes neurais

Dado um vetor entrada $x \in \mathbb{R}^d$, escrevemos

$$h(x) = f_L(f_{L-1}(\dots f_1(x) \dots))$$

(aplicação sucessiva de f_1, f_2, \dots, f_L).

• Trabalhar com h deste modo é efetivo em aplicações.

• Para avaliar $h(x)$, avaliamos em sequência

* $f_1 = f_1(x)$

* $f_2 = f_2(f_1)$

* $f_3 = f_3(f_2)$

⋮

* $h(x) = f_L(f_{L-1})$

"feed forward"

$(f_1 \rightarrow f_L)$

• Para avaliar $\nabla h(x)$, usamos a regra da cadeia. Para simplificar, vamos supor que $f_i: \mathbb{R} \rightarrow \mathbb{R}, \forall i$. Assim,

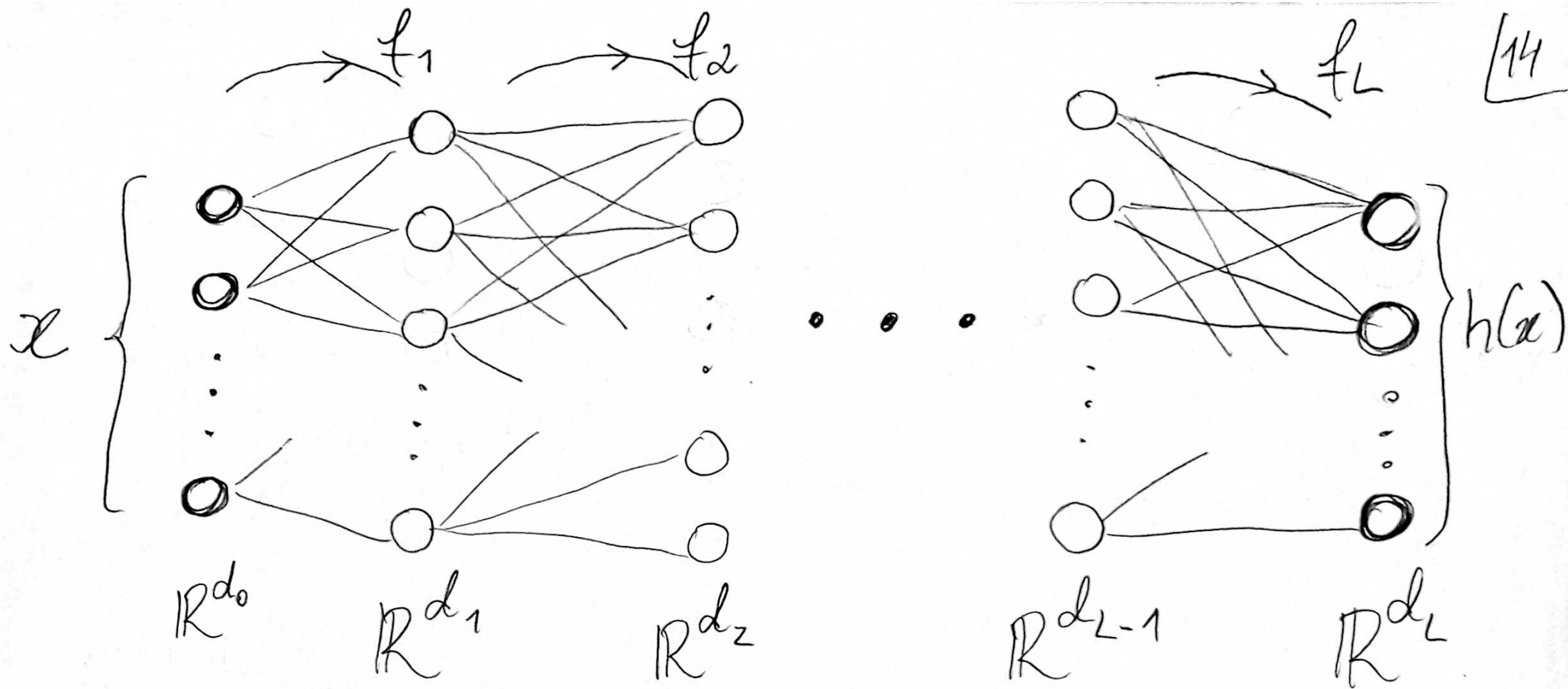
$$h'(x) = f_L'(f_{L-1}) \cdot f_{L-1}'(f_{L-2}) \cdots f_1'(x).$$

Então, calculados f_1, \dots, f_{L-1} , fazemos:

$$\left. \begin{array}{l} * f_L'(f_{L-1}) \\ * f_{L-1}'(f_{L-2}) \\ \vdots \\ * f_1'(x) \end{array} \right\} \text{ "backpropagation" } (f_L \rightarrow f_1).$$

É claro que, em geral, $f_i: \mathbb{R}^{d_{i-1}} \rightarrow \mathbb{R}^{d_i}$, (13)
e temos que usar a regra da cadeia para
várias variáveis...

A concatenação das f_i 's pode ser vista
por meio de um grafo. Por sua inspiração
em modelos para neurônios, tal grafo é
chamado rede neural. Os nós junto com
"seu mecanismo de controle do fluxo" é
chamado neurônio.



camadas (layers)

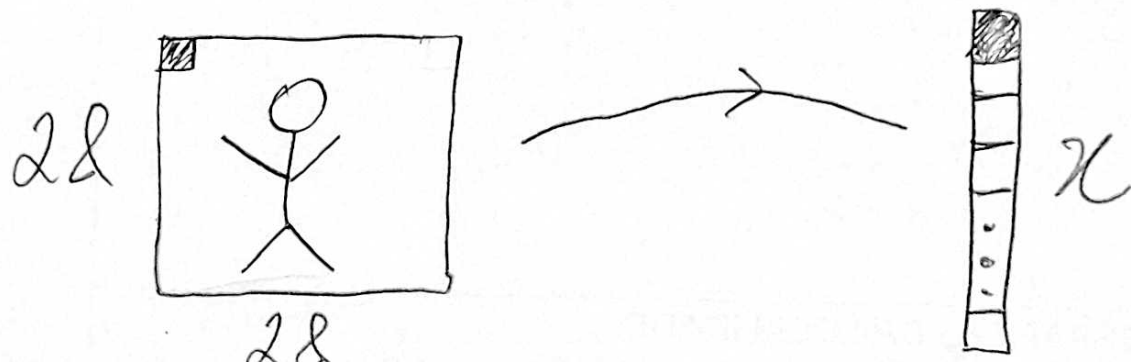
feed forward

back propagation

- Camada mais à esquerda: camada e entrada 15

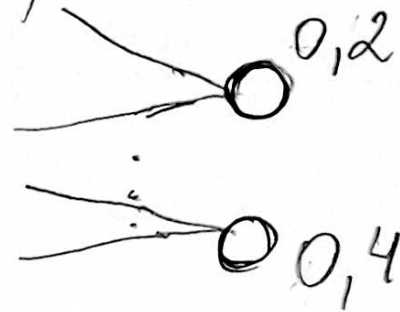
↳ é onde o dado x é codificado.

Ex.: $x \in [0, 256]^{784}$ representando a escala de cinza de uma imagem em tons de cinza de tamanho 28×28 .



- Camada mais à direita: camada de saída (16)
↳ é o valor de $h(x)$.

Ex. 1) resposta -1 ou 1.



$$\text{resposta} = \begin{cases} 1 & \text{se } h_1(x) \geq h_2(x) \\ -1 & \text{e.c.} \end{cases}$$

Ex. 2) x é a imagem de um dígito entre 0 e 9 escrito à mão.

$h(x) \in [0, 1]^{10}$ é um vetor de 10 coordenadas. A resposta à pergunta "qual o dígito da imagem x ?" é dada por

$$\arg \max_{0 \leq i \leq 9} h_{i+1}(x).$$

É comum que cada f_j seja a composta (18)
de uma aplicação afim com uma função
não linear a (função de ativação):

$$f_j: \mathbb{R}^{d_{j-1}} \rightarrow \mathbb{R}^{d_j}, \quad j = 1, \dots, L$$

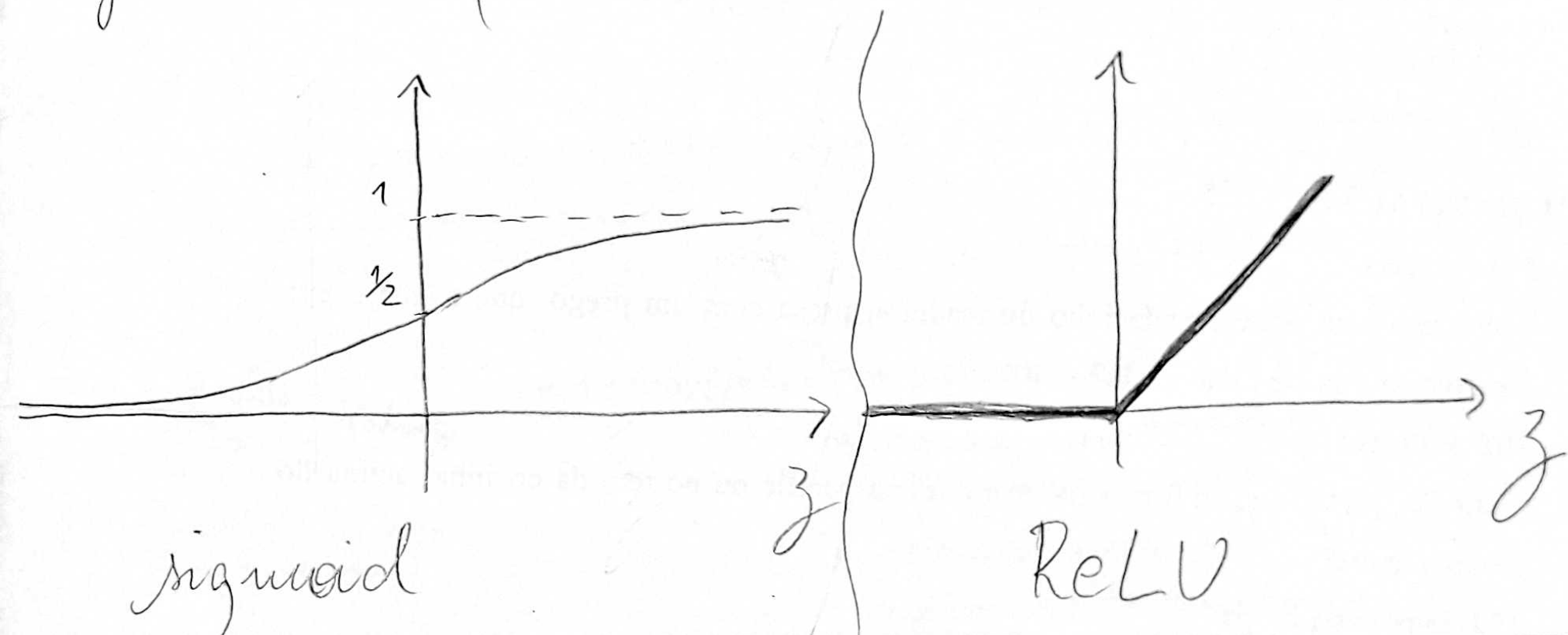
$$x_i^{(j)} = a(w_j x_i^{(j-1)} + b_j) \in \mathbb{R}^{d_j}, \quad \forall i$$

Algumas escolhas:

- $a(z) = \frac{1}{1 + e^{-z}}$ (sigmoid)

- $a(z) = \max\{0, z\}$ (rectified linear unit - ReLU). 19

As funções de ativação tem o intuito de ajustar o fluxo na rede neural.



Objetivo neste caso é minimizar

(20)

$$R_n(w, b) = \frac{1}{n} \sum_{i=1}^n \ell(h(x_i; w, b); y_i).$$

Essa função é não linear e não convexa. Felizmente isso não é grande problema nas aplicações \rightarrow métodos tipo gradiente dão boas soluções (isto é, bons (w^*, b^*) , que são a rede neural treinada).

O processo de otimização de (w, b) é o treinamento da rede neural. (21)

Obs: 1) a soma em $R_n(w, b)$ envolve muitos termos ($n \gg 1$) pois geralmente há muitos dados no treinamento. Portanto calcular ∇R_n é caro. (iterações do método do gradiente)

↳ solução: trocar ∇R_n pela soma de alguns poucos gradientes da soma a cada iteração do método \rightarrow gradiente incremental.

2) Como obter os parâmetros (w^*, b^*) da (22) rede neural (pesos e vieses de cada neurônio), podemos entrar com um \tilde{x} desconhecido e tomar a resposta na camada de saída. (rede treinada)

3) Podemos ver os dados de treinamento como amostras de uma variável aleatória. Nesse sentido, R_n é uma aproximação da função objetivo verdadeira: o risco esperado, ou

a esperança de má classificação: [23]

$$R(h) = \mathbb{P}[h(x) \neq y] = \mathbb{E}[\mathbb{1}[h(x) \neq y]].$$

Aqui, $\mathbb{P}[h(x) \neq y]$ é a probabilidade de h errar a resposta para x . Note que x é qualquer, não mais somente um dado da amostra de treinamento.

Não conhecemos \mathbb{P} , apenas uma aproximação pelos dados de treinamento. Porém,

queremos dizer algo em relação à (24)
 $R(h)$ (convergência em probabilidade do
método de gradiente incremental).

O método "resultante" da ideia de gradiente
incremental com a probabilidade é o
método do gradiente estocástico. Ele (e
variantes) são os mais utilizados no treina-
mento de redes (lembre-se que gradientes são
calculados por backpropagation).