MAE 5905: Introdução à Ciência de Dados

Pedro A. Morettin

Instituto de Matemática e Estatística
Universidade de São Paulo
pam@ime.usp.br
http://www.ime.usp.br/∼ pam

Aula 12

25 de abril de 2024

Sumário

Preliminares

- Uma rede neural (RN) é um conjunto de algoritmos construídos para identificar relações entre as variáveis de um conjunto de dados por intermédio de um processo que tenta imitar a maneira com que neurônios interagem no cérebro.
- Cada neurônio numa rede neural é uma função à qual dados são alimentados e transformados numa resposta como em modelos de regressão. Essa resposta é repassada para um ou mais neurônios da rede que ao final do processo produz um resultado ou recomendação.
- Quando falamos em arquitetura de uma RN, estamos nos referindo à maneira como os neurônios estão organizados em camadas. Há três classes diferentes de arquiteturas :

  1. RN com uma única camada. Nessa classe, há uma camada de entrada e uma de saída. Essa RN é do tipo feedforward.Essa rede também é chamada perceptron.
  2. RN multicamadas, também do tipo *feedforward*. São redes com uma ou mais camadas escondidas. Cada neurônio de uma camada tem como entradas os neurônios da camada precedente.
  3. RN recorrentes. Nesse caso, pelo menos um neurônio conecta-se com um neurônio da camada precedente, criando um ciclo (feedback).

História das RN

- As contribuições pioneiras para a área de Redes Neurais (também denominadas redes neuronais, termo derivado de neurônio) foram as de McCulloch e Pitts (1943), que introduziram a unidade lógica com limiar (*thresholded logic unit*) e de Hebb (1949), que postulou a primeira regra para aprendizado organizado.
- Rosenblatt (1957) introduziu o **perceptron** e Widrow e Hoff (1960) o **adaline** (*adaptive linear neuron*).
- Depois, seguiu-se o que foi chamado de primeiro inverno neural. Minsky e Papert (1969) escreveram um livro, chamado *Perceptron*, em que apresentam o problema XOR (*exclusive OR*). O problema consistia em usar redes neurais (o perceptron) para prever saídas da lógica XOR, dadas duas entrada binárias, como na Tabela 1. Uma função XOR deve retornar um valor verdadeiro (1, ponto verde na figura) se as duas entradas não são iguais e um valor falso (0, ponto vermelho na Figura 1) se elas são iguais.

História das RN

Tabela 1: Problema XOR

Entrada 1	Entrada 2	Saída
0	0	0
0	1	1
1	1	0
1	0	1

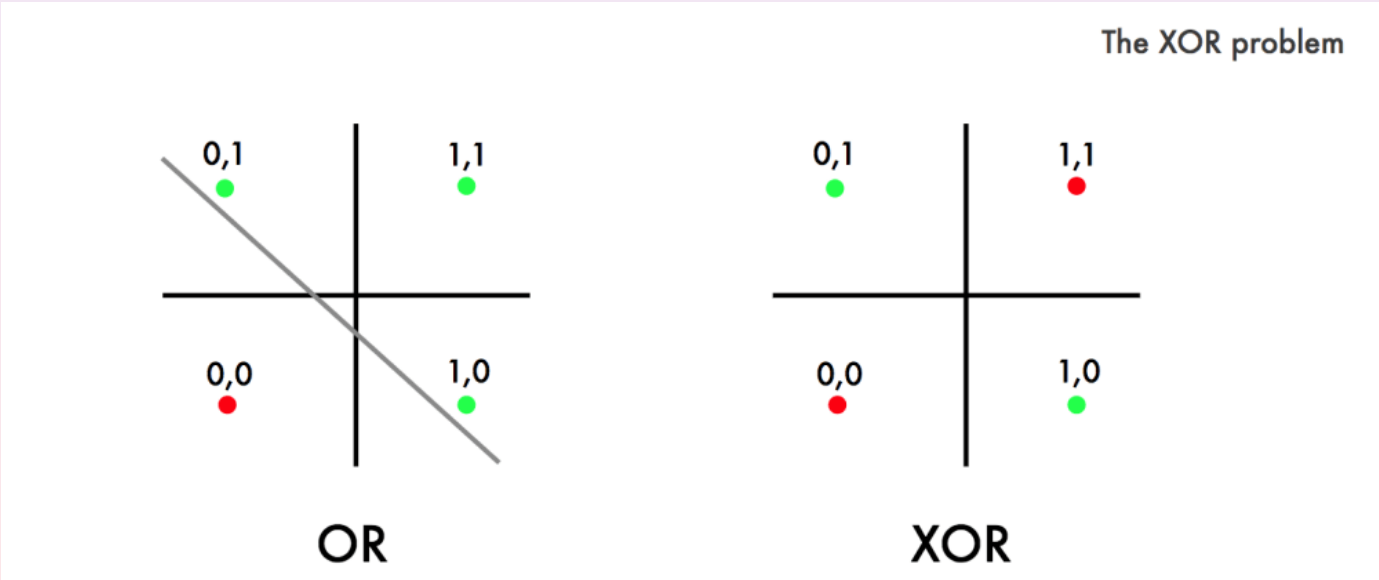


Figura 1: Os problemas lógicos (a) OR e (b) XOR.

História das RN

- Esse é um problema de classificação usando o perceptron, mas este supõe que as duas classes sejam linearmente separáveis, o que elas não são, no caso XOR, conforme a Figura 1. O problema OR (inclusive) é separável por uma reta.
- Este problema seria solucionado com a introdução, nas décadas de 1980 e 1990, das redes com várias camadas ocultas, o algoritmo de retroalimentação (*backpropagation*) e as redes neurais convolucionais. Veja LeCun (2015) para detalhes.
- Um segundo inverno neural ocorre na década de 1990, no qual a contribuição mais significativa, fora do contexto de redes neurais, foi a de Cortes e Vapnik (1995), que introduziram as máquinas de suporte vetorial (*support vector machines*).

História das RN

- A partir de 2006 foram desenvolvidas as redes neurais profundas (*deep neural networks*, DNN), com menção especial às redes neurais recorrentes (RNN) e as redes neurais convolucionais (CNN). É a chamada **era das GPU** (Graphic Processing Units).
- Em especial, destacamos uma competição promovida pela ImageNet, que mantém uma base de dados com milhões de imagens para fins de treinamento e classificação, usando técnicas de ciências de dados. Krizhevsky et al. (2012) venceram essa competição em 2012, usando uma CNN com sete camadas ocultas, denominada **Alex net** (primeiro nome do primeiro autor) e que levou seis dias para ser treinada. A proporção de erros dessa rede foi de $15,3\%$. Em 2015, o vencedor usou uma CNN com 150 camadas ocultas e a proporção de erros foi de $3,5\%$.
- A seguir faremos uma revisão dessas redes neurais.

Perceptron

- Rosenblatt (1958) introduziu o algoritmo *perceptron* como o primeiro modelo de aprendizado supervisionado. A ideia do algoritmo é atribuir pesos w_i aos dados de entrada $\mathbf{x}$, iterativamente, até que o processo tenha uma precisão pré-especificada para a tomada de decisão. No caso de classificação binária, o objetivo é classificar elementos do conjunto de dados com valor das variáveis preditoras $\mathbf{x}$ (dados de entrada) em uma de duas classes, rotuladas aqui por $+1$ e -1.
- O algoritmo *perceptron* (programado para um computador IBM 704) foi implementado em uma máquina chamada Mark I, planejada para reconhecimento de imagens. O modelo subjacente consiste de uma combinação linear das entradas, $\mathbf{x}$, com a incorporação de um viés externo, cujo resultado é comparado com um limiar, definido por meio de uma **função de ativação**. As funções de ativação usualmente empregadas são as funções **degrau** ou **sigmoide**. Outras funções de ativação serão vistas a seguir.
- Se $\mathbf{x} = (1, x_1, x_2, ..., x_p)^\top$ contém as entradas, $\mathbf{w} = (-b, w_1, w_2, ..., w_p)^\top$ são os pesos, a saída é dada por

$$v = \sum_{i=0}^{p} w_i x_i = \mathbf{w}^\top \mathbf{x}.$$

Perceptron

- Definamos uma função de ativação $f(v)$ tal que se $f(v) \geq b$, a amostra é classificada na classe $+1$ e se $f(v) < b$, é classificada na classe -1. Ou seja,

$$f(v) = \begin{cases} 1, & \text{se } v \geq b \\ -1, & \text{se } v < b. \end{cases}$$

- O parâmetro b é chamado de viés. A ideia de Rosenblatt era obter um algoritmo (regra de aprendizado), segundo a qual os pesos são atualizados a fim de se obter uma fronteira de decisão linear, que permite disciminar entre as duas classes linearmente separáveis. Veja a Figura 2.

Perceptron

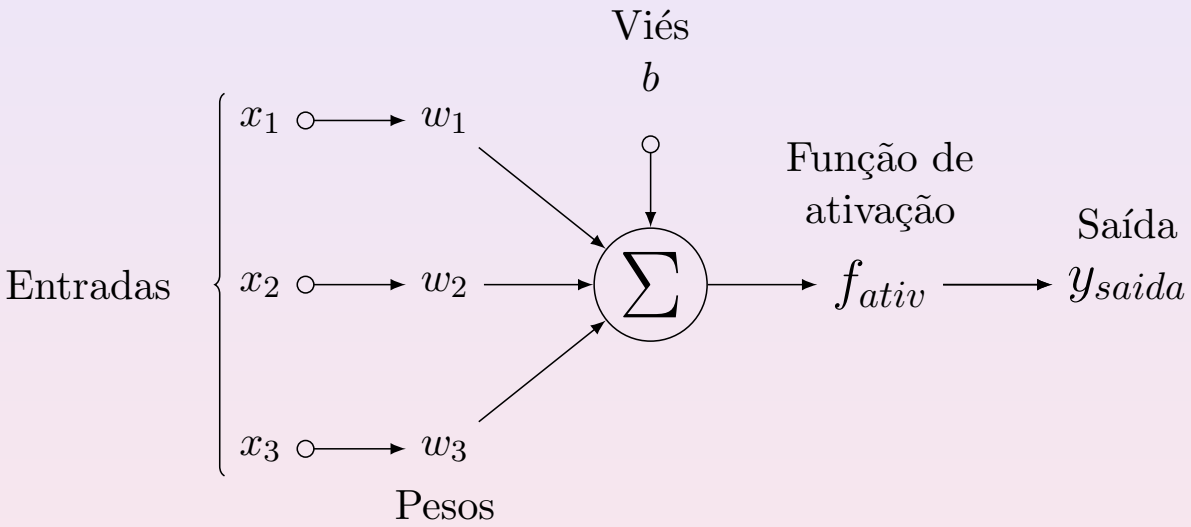


Figura 2: Diagrama de um perceptron.

Algoritmo do perceptron

[1] Inicialize todos os pesos como sendo zero ou valores aleatórios pequenos;

[2] Para cada amostra de treinamento $\mathbf{x}^{(i)}$:

(a) calcule os valores da saída;
(b) atualize os pesos.

A atualização dos pesos é feita segundo a regra de aprendizagem

$$\Delta w_j = \eta(\text{alvo}^{(i)} - \text{saída}^{(i)})x_j^{(i)},$$

onde η é a taxa de aprendizagem (um valor entre 0 e 1), alvo é o rótulo verdadeiro da classe e saída é o rótulo da classe prevista.Todos os pesos são atualizados simultaneamente. Por exemplo, no caso de duas variáveis, x_1 e x_2, teremos que atualizar w_0, w_1 e w_2.

Algoritmo do perceptron

- É fácil ver que nos dois casos para os quais o perceptron prevê o rótulo da classe verdadeira, $\Delta w_j = 0$, para todo j. No caso de previsão errônea, $\Delta w_j = 2\eta x_j^{(i)}$ ou $\Delta w_j = -2\eta x_j^{(i)}$.
- Uma observação importante é que a convergência do perceptron somente é garantida se as duas classes são linearmente separáveis. Se não forem, podemos fixar um número máximo de passadas sobre os dados de treinamento (épocas) ou um limiar para o número máximo de classificações erradas tolerável.

Perceptron – Exemplo 1

Exemplo 1. Consideremos um exemplo simulado. Simulamos 50 valores de X_1 e X_2, geradas segundo uma normal padrão e Y será igual a $+1$ se $X_2 > 0,5X_1 + 0,2$ e igual a -1, caso contrário. Veja a Figura 3.

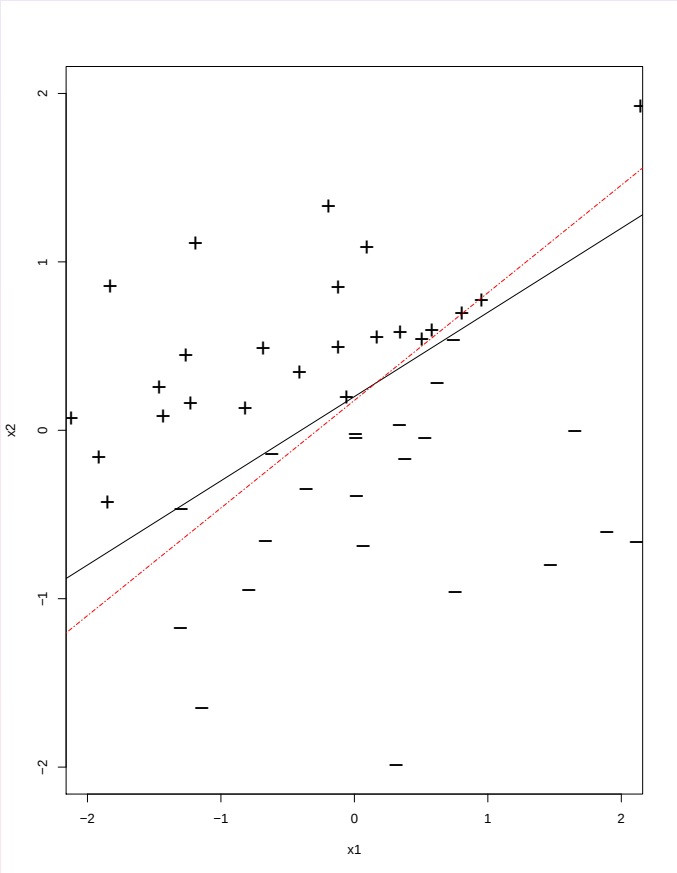


Figura 3. Exemplo 1: reta simulada (preto); perceptron (vermelho).

Perceptron – Exemplo 1

Usando o *script* para o perceptron (veja a página do livro), obtemos as estimativas dos coefcientes

```
$w
         x1          x2
-0.5384671  0.8426465

$b
[1] -0.1495257
```

e, portanto a reta $x_2 = 0,17745 + 0,639x_1$, que está apresentada também na Figura 2. Ambas as retas estão próximas.

Perceptron – Exemplo 2

Exemplo 2. Vamos considerar o conjunto de dados Iris e duas variáveis,

x_1: comprimento de sépala
x_2: comprimento de pétala
e duas espécies, Iris versicolor (+1) e Iris setosa (-1). Veja a Figura 4.

Usando, novamente, o *script* para o perceptron, obtemos

```
$w
      sepal     petal
1 0.1419446 0.9898746

$b
[1] -2.855304
```

de modo que obtemos a reta $x_2 = 2,8844 - 0,14335x_1$, representada na Figura 4.

Perceptron – Exemplo 2

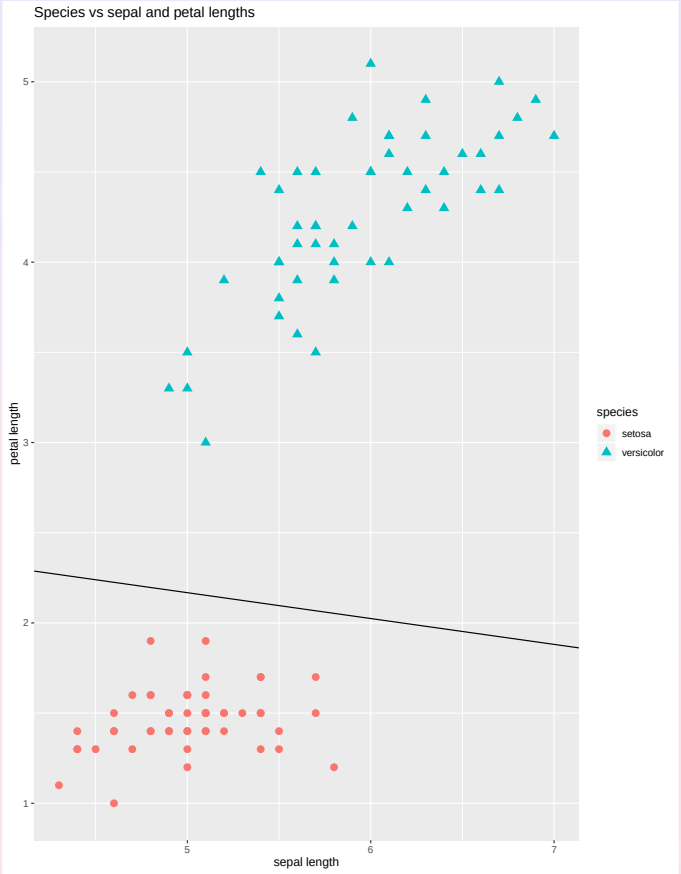


Figura 4. Iris data com duas variáveis e reta perceptron.

RN com uma camada

Atualmente, a RN mais simples consiste de entradas, de uma camada intermediária escondida e das saídas. Na Figura 5, se $K = 1$ temos o caso de regressão e, no caso de classificação, teremos K classes, sendo que $Z_i = -1$ ou $Z_i = +1$.

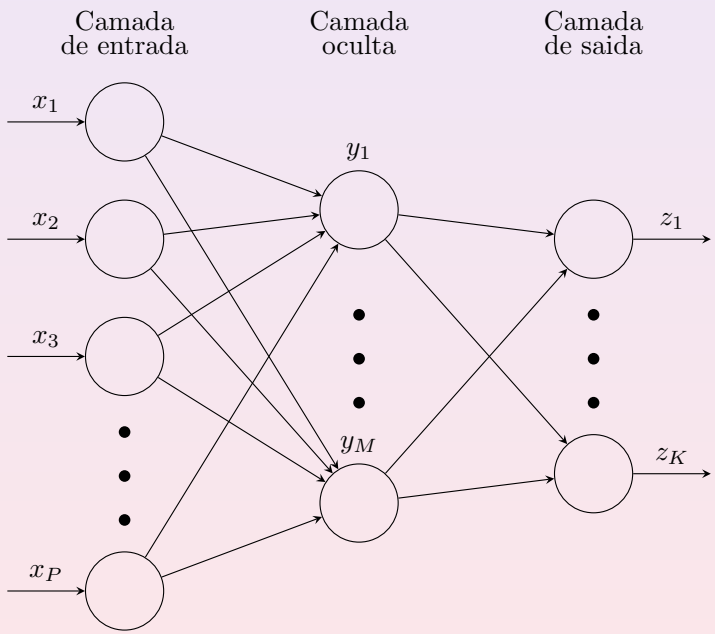


Figura 5. Rede neural com uma camada escondida.

RN com uma camada

- Consideremos os seguintes vetores de variáveis, $\mathbf{X} = (X_1, \ldots, X_p)^\top$, $\mathbf{Y} = (Y_1, \ldots, Y_M)^\top$ e $\mathbf{Z} = (Z_1, \ldots, Z_K)^\top$ e sejam, os vetores de pesos $\boldsymbol{\alpha}_j$, $j = 1, \ldots, M$, $\boldsymbol{\beta}_k$, $k = 1, \ldots, K$, de ordens $p \times 1$ e $M \times 1$, respectivamente.
- A RN da Figura 5 pode ser representada pelas equações:

$$Y_j = h(\alpha_{0j} + \boldsymbol{\alpha}_j^\top \mathbf{X}),\ j = 1, \ldots, M, \quad (1)$$

$$Z_k = g(\beta_{0k} + \boldsymbol{\beta}_k^\top \mathbf{Y}),\ k = 1, \ldots, K. \quad (2)$$

- As funções h e g são chamadas funções de ativação e geralmente são usadas as seguintes funções:

RN com uma camada

a) logística (ou sigmoide),

$$f(x) = \frac{1}{1 + e^{-x}},$$

b) tangente hiperbólica,

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}},$$

c) ReLU (rectified linear unit),

$$f(x) = \max(0, x).$$

d) Uma modificação da ReLU é *leaky ReLU*, dada por

$$f(x) = \begin{cases} x, & \text{se } x > 0, \\ 0,01x, & \text{se } x < 0. \end{cases}$$

Funções de ativação

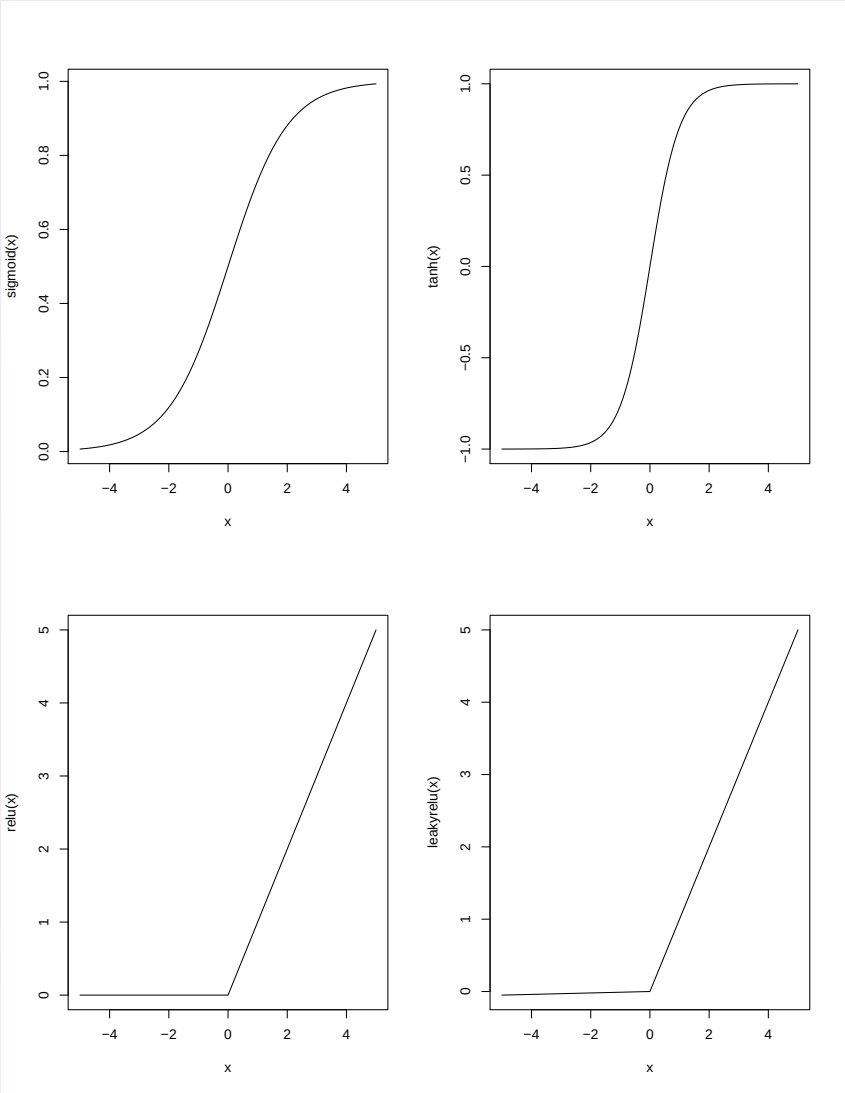


Figura 6. Algumas funções de ativação.

RN com uma camada

- A função (d) é bastante utilizada, pois é fácil de otimizar, o gradiente pode ser facilmente calculado e converge mais rápido do que sigmoides. Todavia, ela não é derivável na origem, e no algoritmo backpropagation, por exemplo, necessitamos de derivabilidade.
- Pode-se usar a mesma função de ativação em (1) e (2). Também é comum que haja uma saída única, Z, na Figura 5.
- Os pesos α_{0j} e β_{0k} têm o mesmo papel de b no perceptron e representam vieses. Os Y_j constituem a camada escondida e não são observáveis.
- No lugar de (1)–(2), podemos considerar a saída da RN expressa na forma

$$f(\mathbf{x}, \mathbf{w}) = \varphi\left(\sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x})\right), \tag{3}$$

onde $\varphi(\cdot)$ é a identidade, no caso de regressão e uma função não linear, no caso de classificação; w_j são pesos a serem determinados.

RN com uma camada

Com essa formulação, os seguintes passos são usualmente utilizados na análise de redes neurais (RN) (Bishop, 2006):

(i) Considere as ativações

$$a_j = \sum_{i=0}^{p} w_{ji}^{(1)} x_i, \quad j = 1, \ldots, M, \tag{4}$$

onde incluimos os vieses $w_{j0}^{(1)}$ nos pesos $\mathbf{w}_j^{(1)} = (w_{j0}, w_{j1}, \ldots, w_{jp})^\top$, $j = 1, \ldots, M$, fazendo $x_0 = 1$. O índice (1) indica a primeira camada da RN.

(ii) Cada ativação a_j é transformada usando uma função de ativação $h(\cdot)$, resultando

$$y_j = h(a_j). \tag{5}$$

Aqui, podemos usar uma das funções acima. Dizemos que os y_j são as unidades escondidas.

RN com uma camada

(iii) Considere as ativações de saída

$$a_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j, \quad k = 1, 2, \ldots, K, \tag{6}$$

onde, novamente incluimos os vieses $w_{k0}^{(2)}$ no vetor $\mathbf{W}$.

(iv) Finalmente, essas ativações são transformadas por meio de uma nova função de ativação, resultando nas saídas z_k da RN, sendo que no caso de regressão, $z_k = a_k$ e no caso de classificação,

$$Z_k = \sigma(a_k), \tag{7}$$

sendo que $\sigma(a)$ em geral é a logística dada em (a) acima.

RN com uma camada

- Combinando, obtemos

$$f_k(\mathbf{x}, \mathbf{W}) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{p} w_{ji}^{(1)} x_i\right)\right). \quad (8)$$

- O procedimento para obter (8) é chamado forward propagation da informação.
- Podemos generalizar considerando camadas adicionais.
- A nomenclatura de tal rede difere segundo autores e pacotes computacionais. Pode ser chamada de RN com 3 camadas ou uma RN com uma camada escondida. Bishop (2006) sugere chamá–la de uma RN com duas camadas, referindo-se aos pesos $w_{ji}^{(1)}$ e $w_{kj}^{(2)}$. Ainda, o pacote neuralnet estipula o número de neurônios, M, na camada escondida.

RN com uma camada–Exemplo 3

- **Exemplo 3.** Vamos considerar os dados do Exemplo 1 e vamos criar uma rede neuronal com uma camada escondida com 3 neurônios e função de ativação tangente hiperbólica. Para tanto, usamos o pacote neuralnet do R.
- `nn=neuralnet(y~x1+x2,data=df, hidden=3, act.fct="tanh", +linear.output=FALSE)# hidden=3: single layer with 3 neurons plot(nn)`
- Obtemos a Figura 7. Nesse exemplo, $p = 2$, $M = 3$ e $K = 1$. As equações (3)–(4) ficam:

$$\begin{aligned} Y_1 &= -4,179 - 2,761X_1 - 19,010X_2 \\ Y_2 &= 2,650 - 2,109X_1 - 0,978X_2 \\ Y_3 &= 0,582 + 1,472X_1 - 1,232X_2. \end{aligned}$$

- Também, teremos três vetores $\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2$ e $\boldsymbol{\alpha}_3$, de ordens 2×1, $\mathbf{X} = (X_1, X_2)$, $\mathbf{Y} = (Y_1, Y_2, Y_3)$ e $Z = g_1(W) = W = \beta_{01} + \boldsymbol{\beta}_1^\top \mathbf{Y}$, com $\boldsymbol{\beta}_1 = (\beta_1, \beta_2, \beta_3)$, $\beta_{01} = -0,540$. Ou seja,

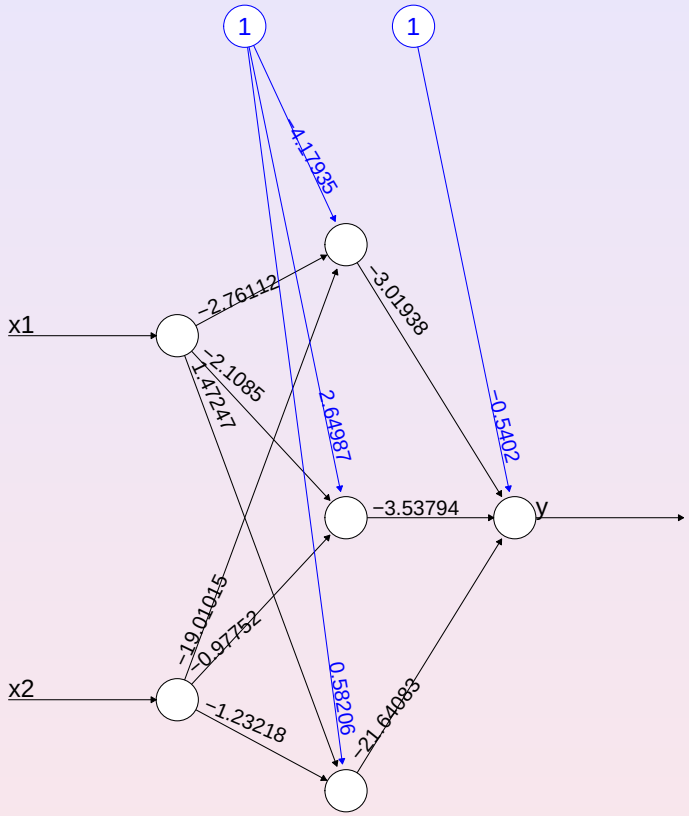
$$Z = -0,540 - 3,019Y_1 - 3,538Y_2 - 21,641Y_3.$$

RN com uma camada–Exemplo 3

Os pesos finais são dados por:

```
result.matrix
                                  [,1]
error                      0.003675542
reached.threshold          0.007275895
steps                    217.000000000
Intercept.to.1layhid1     -4.179347633
x1.to.1layhid1            -2.761123538
x2.to.1layhid1           -19.010153927
Intercept.to.1layhid2      2.649870079
x1.to.1layhid2            -2.108496830
x2.to.1layhid2            -0.977518529
Intercept.to.1layhid3      0.582063062
x1.to.1layhid3             1.472465125
x2.to.1layhid3            -1.232181252
Intercept.to.y            -0.540201763
1layhid1.to.y             -3.019378642
1layhid2.to.y             -3.537936157
1layhid3.to.y            -21.640828433
```

RN com uma camada–Exemplo 3



Error: 0.003676 Steps: 217

Figura 7. Rede neural com uma camada escondida para o Exemplo 3.

RN com uma camada–Exemplo 3

Vamos ver como fazer previsões para um conjunto de teste. Suponha que tenhamos 5 observações de **x**, a saber

```
   x1          x2
0.6180602   0.14261227
0.5053168   1.48586481
0.3615404  -0.04880704
-0.1707795 -1.09666571
-0.6284641 -0.54107065
```

que chamaremos de *test*. Usamos os comandos

```
test=data.frame(x1,x2)

Predict=compute(nn,test)

Predict\$net.result
```

```
          [,1]
[1,] -1.0000000
[2,]  1.0000000
[3,]  1.0000000
[4,] -0.9993473
[5,] -0.9997057
```

RN com uma camada–Exemplo 3

```
Convert probabilities into binary classes setting threshold level =0.5

prob<-Predict\$net.result
pred<-ifelse(prob>0.5,1,0)
pred

     [,1]
[1,]    0
[2,]    1
[3,]    1
[4,]    0
[5,]    0
```

onde agora 0 corresponde a -1 e 1 a $+1$. Ou seja, classificamos a segunda e terceira observações testes na classe $+1$ e a primeira, quarta e quinta na classe -1.

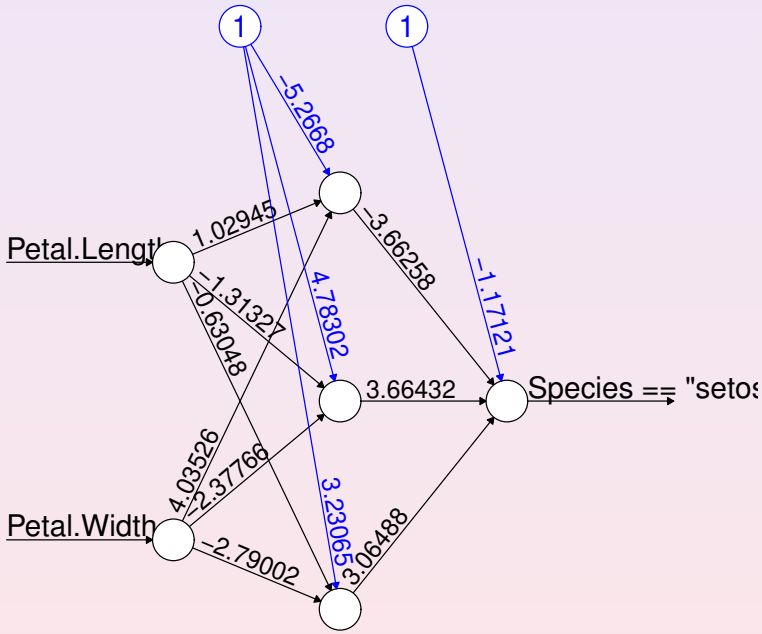
RN com uma camada–Exemplo 4

- **Exemplo 14.4.** Vamos considerar novamente o conjunto Iris, mas somente a espécie Setosa e dois preditores, Petal.Length e Petal.Width. Para uma rede neuronal com três neurônios na camada escondida, obtemos a Figura 8.
- Os pesos obtidos estão indicados abaixo, e vemos que foram necessárias 41 iterações, o limiar foi $0,00994$ e o erro de classificação $0,0125$.

RN com uma camada–Exemplo 4

error	0.012533834
reached.threshold	0.009935566
steps	41.000000000
Intercept.to.1layhid1	-5.266797930
Petal.Length.to.1layhid1	1.029448892
Petal.Width.to.1layhid1	4.035261475
Intercept.to.1layhid2	4.783018629
Petal.Length.to.1layhid2	-1.313269314
Petal.Width.to.1layhid2	-2.377663815
Intercept.to.1layhid3	3.230647942
Petal.Length.to.1layhid3	-0.630478727
Petal.Width.to.1layhid3	-2.790017234
Intercept.to.Species == "setosa"	-1.171205492
1layhid1.to.Species == "setosa"	-3.662576010
1layhid2.to.Species == "setosa"	3.664317725
1layhid3.to.Species == "setosa"	3.064883525

RN com uma camada–Exemplo 4



Error: 0.012534 Steps: 41

Figura 8. Rede neural com uma camada escondida para o Exemplo 4.

Software

- API (Application Programming Interface): software intermediary that allows two applications to talk to each other, eg each time you use an app, like Facebook, send an instant message or check the weather on your phone.
- Tensorflow: is a (R and Python) friendly open source library for numerical computation that makes ML faster and easier. Eases the process of acquiring data, training models, serving predictions and refining future results. Can train and run deep NN for handwritten digit classification, image recognition, recurrent NN, natural language processing etc.
- Keras: is a high-level, DL API developed by Google for implementing NNs. It provides a (R and Python) frontend with a high level of abstraction while having the option of multiple back-end for computation purposes. The frameworks supported by Keras are Tensorflow, Theano, MXNet, CNTK (Microsoft Cognitive Toolkit).
- TORCH: Torch is an open-source machine learning library, a scientific computing framework, and a scripting language based on Lua.

Os algoritmos de ML mais importantes (até 2018)

Classificação e Regressão

- **Boosting**
- **Random forest**
- **Support vector machine**
- **LSTM neural network**
- **Convolutional neural network**
- **K-nearest neighbors**

O algoritmo backpropagation

- O ajuste de modelos de RN é feito minimizando uma função perda, usualmente a soma dos quadrados dos resíduos, no caso de regressão, onde a minimização é feita sobre os pesos. No caso de classificação, usamos a entropia. Nos dois casos é usado um algoritmo chamado de backpropagation (BP).
- O algoritmo BP calcula o gradiente da função perda com respeito aos pesos da rede, atualizando para trás uma camada de cada vez, a fim de minimizar a perda. Comumente usa-se gradient descent ou variações, como stochastic gradient descent.
- É necessário escolher valores iniciais e regularização (usando uma função penalizadora), porque o algoritmo de otimização é não convexo e instável.

O algoritmo backpropagation

De modo geral, o problema de otimização da RN pode ser posto na forma:

$$\widehat{\mathbf{w}} = \arg\min_{\mathbf{w}} \widetilde{Q}_n(\mathbf{w}) = \arg\min_{\mathbf{w}} \left[\lambda_1 Q_n(\mathbf{w}) + \lambda_2 Q^*(\mathbf{w})\right], \tag{9}$$

com $\lambda_1, \lambda_2 > 0$, sendo

$$Q_n(\mathbf{w}) = \sum_{i=1}^{n} [y_i - f(\mathbf{x}_i, \mathbf{w})]^2, \tag{10}$$

e $Q^*(\mathbf{w})$ um termo de regularização, que pode ser escolhido entre aqueles estudados no Capítulo 6 (lasso, ridge ou elastic net).

O algoritmo backpropagation

- Podemos pensar uma rede neural como em (3), ou seja, uma função não linear paramétrica (determinística) de uma entrada $\mathbf{x}$, tendo $\mathbf{z}$ como saída.
- Suponha que tenhamos os vetores de treinamento $\mathbf{x}_i$, e os vetores alvos (saídas) $\mathbf{z}_i$, $i = 1, \ldots, n$ e considere a soma dos quadrados dos erros (10), ligeiramante modificada

$$Q_n(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{n} \|z_i - f(\mathbf{x}_i, \mathbf{w})\|^2, \tag{11}$$

que queremos minimizar.

O algoritmo backpropagation

[1] Vamos tratar primeiramente o problema de regressão e considerar a rede agora com um erro aleatório ε_i acrescentado antes da saída, de modo que agora temos um modelo probabilístico. Por simplicidade, consideremos a saída $\mathbf{z} = (z_1, \ldots, z_n)^\top$ e os erros com distribuição normal, com média zero e variância σ^2, de modo que

$$\mathbf{z} \sim N_n(f(\mathbf{x}_i, \mathbf{w}), \sigma^2 \mathbf{I}).$$

No caso em questão, a função de ativação de saída é a identidade. Chamando $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$, a verossimilhança pode ser escrita como

$$L(\mathbf{z}|\mathbf{X}, \mathbf{w}, \sigma^2) = \prod_{i=1}^{n} p(z_i|\mathbf{x}_i, \mathbf{w}, \sigma^2).$$

Maximizar a log-verossimilhança é equivalente a minimizar (10), de modo que obtemos $\hat{\mathbf{w}}_{MV}$. Encontrado $\hat{\mathbf{w}}_{MV}$, podemos obter o estimador de MV de σ^2. Como temos uma função não linear, $Q_n(\mathbf{w})$ é não convexa, portanto na prática podemos obter máximos não locais da verossimilhança.

O algoritmo backpropagation

(2) No caso de classificação binária, para a qual, por exemplo, $z = +1$ indica a classe C_1 e $z = 0$ indica a classe C_2, considere a RN com saída única z com função de ativação logística,

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

de modo que $0 \leq f(\mathbf{x}, \mathbf{w}) \leq 1$. Podemos interpretar $f(\mathbf{x}, \mathbf{w}) = P(C_1|\mathbf{x})$ e $P(C_2|\mathbf{x}) = 1 - f(\mathbf{x}, \mathbf{w})$. Segue que a distribuição de z, dado $\mathbf{x}$, é dada por

$$f(z|\mathbf{x}, \mathbf{w}) = f(\mathbf{x}, \mathbf{w})^z [1 - f(\mathbf{x}, \mathbf{w})]^{1-z}. \tag{12}$$

O algoritmo backpropagation

- Considerando as observações de treinamento i.i.d., a função erro será dada pela entropia cruzada

$$Q_n(\mathbf{w}) = -\sum_{i=1}^{n}[z_i \log z_i - (1 - z_i)) \log(1 - z_i)], \tag{13}$$

onde $z_i = f(\mathbf{x}_i, \mathbf{w})$.

- Temos que otimizar os pesos $\mathbf{w}$, ou seja, encontrar o valor que minimiza $Q_n(\mathbf{w})$. Usualmente, temos que obter o gradiente de Q_n, que vamos indicar por ∇Q_n, que aponta para a maior taxa de crescimento de Q_n. Supondo-se que Q_n seja uma função contínua suave de $\mathbf{w}$, o valor mínimo ocorre no ponto onde o gradiente anula-se.

O algoritmo backpropagation

- Procedimentos numéricos são usados e há uma literatura extensa sobre o assunto. As técnicas que usam o gradiente, começam fixando-se um valor inicial $\mathbf{w}^{(0)}$ e os pesos são iterados na forma

$$\mathbf{w}^{(r+1)} = \mathbf{w}^{(r)} - \lambda \nabla Q_n(\mathbf{w}^{(r)}),$$

  na qual λ é chamada a taxa de aprendizado.
- Esse método é chamado de gradiente descendente e usa todo o conjunto de treinamento. É um algoritmo não muito eficiente e, na prática, usa-se o algoritmo backpropagation (BP) para calcular o gradiente da SQE em uma RN.

O algoritmo backpropagation

- Aqui vamos nos basear em (1)–(2) e em Hastie et al. (2017). Podemos usar também (4)–(8), veja Bishop (2006). Vamos chamar de $\mathbf{w} = (\alpha_0, \ldots, \alpha_M, \beta_0, \ldots, \beta_K)^\top$ e considerar $Q_n(\mathbf{w})$ escrita de modo geral como

$$Q_n(\mathbf{w}) = \sum_{k=1}^{K} \sum_{i=1}^{n} (z_{ik} - f(\mathbf{x}_i, \mathbf{w}))^2, \tag{14}$$

e seja

$$y_{mi} = h(\boldsymbol{\alpha}_m^\top \mathbf{x}_i), \quad \mathbf{y}_i = (y_{1i}, \ldots, y_{Mi})^\top. \tag{15}$$

- Considerando os $\mathbf{x}_i$ i.i.d, como em (14), escrevemos $Q_n(\mathbf{w}) = \sum_{i=1}^{n} Q_i(\mathbf{w})$ e as derivadas

$$\frac{\partial Q_i}{\partial \beta_{km}} = -2(z_{ik} - f(\mathbf{x}_i, \mathbf{w}))g'(\boldsymbol{\beta}_k^\top \mathbf{y}_i) y_{mi}, \tag{16}$$

$$\frac{\partial Q_i}{\partial \alpha_{m\ell}} = -\sum_{k=1}^{K} 2(z_{ik} - f(\mathbf{x}_i, \mathbf{w}))g'(\boldsymbol{\beta}_k^\top \mathbf{y}_i)\beta_{km} h'(\boldsymbol{\alpha}_m^\top \mathbf{x}_i) x_{i\ell}. \tag{17}$$

O algoritmo backpropagation

- O gradiente na $(r+1)$-ésima iteração é dado por

$$\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \lambda_r \sum_{i=1}^{n} \frac{\partial Q_i}{\partial \beta_{km}^{(r)}}, \tag{18}$$

$$\alpha_{m\ell}^{(r+1)} = \alpha_{m\ell}^{(r)} - \lambda_r \sum_{i=1}^{n} \frac{\partial Q_i}{\partial \alpha_{m\ell}^{(r)}}, \tag{19}$$

sendo λ_r a taxa de aprendizagem.

- Escrevamos as derivadas (16) e (17) como

$$\frac{\partial Q_i}{\partial \beta_{km}} = \delta_{ki} y_{mi}, \tag{20}$$

$$\frac{\partial Q_i}{\partial \alpha_{m\ell}} = s_{mi} x_{i\ell}, \tag{21}$$

onde δ_{ki} e s_{mi} são os erros do modelo atual na saída e camadas escondidas, respectivamente.

O algoritmo backpropagation

- De suas definições, esses erros satisfazem

$$s_{mi} = h(\alpha_m^\top \mathbf{x}_i) \sum_{k=1}^{K} \beta_{km} \delta_{ki}, \tag{22}$$

conhecidas como equações de backpropagation, que podem ser usadas para implementar (18) e (19) em duas passadas:

(a) uma para a frente, onde os pesos atuais são fixos e os valores previstos de $f(\mathbf{x}, \mathbf{w})$ são calculados a partir de (22);

(b) uma para trás, os erros δ_{ki} são calculados e propagados para trás via (20) e (21)para obter os erros s_{mi}.

- Os dois conjuntos de erros são então usados para calcular os gradientes para atualizar (18) e (19) via (20) e (21). Uma época de treinamento refere-se a uma passada por todo o conjunto de treinamento.

Referências

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. New York: Springer.

Hastie, T., Tibshirani, R. and Friedman, J. (2017). *The Elements of Statistical Learning*, 2nd Edition, Springer.

Hebb, D. O. (1949). *The organization of behavior*. New York: Wiley.

McCulloch, W. S. and Pitts, W. A. (1943). Logical calculus of the ideas immanent in nervous activity. *Butt. Math. Biophysics*, S, 115–133.

Pereira, B.B., Rao, C.R. and Oliveira, F. B. (2020). *Statistical Learning Using Neural Networks: A Guide for Statisticians and Data Scientists with Python*. Chapman and Hall.

Referências

Rosenblatt, F. (1958). The perceptron: A theory of statistical separability in cognitive systems. Buffalo: Cornell Aeronautical Laboratory, Inc. Report No. VG-1196-G-1.

Rumelhart, D. E., Hinton, G. E. and Williams, R. J. (1986a). Learning representations by back-propagating errors. *Nature*, **323**, 533–536.

Rumelhart, D. E., Hinton, G. E. and Williams, R. J. (1986b). Learning internal representations by by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Vol. 1: Foundations (eds Rumelhart, D. E. and McClelland, J. L.) 318–362 (MIT, Cambridge, 1986).