Machine Learning Aplicada à Agropecuária de Precisão
Dia 03 - Redes Neurais Artificiais

Giancarlo D. Salton, PhD

Applied Intelligence Research Centre & ADAPT Centre
School of Computing

Chapecó, 10 de maio de 2018

Redes Neurais Artificiais

Arquitetura das Redes Neurais

Neurônios Artificiais

Gradiente Descendente

Backpropagation

Redes Neurais Artificiais

- Redes neurais artificiais são um método de *machine learning* baseado em erro
- Uma das propriedades mais importantes das redes neurais é o fato delas serem "aproximadores universais"
  - em outras palavras, dado um número "suficiente" de neurônios, **elas conseguem aproximar/aprender qualquer função até uma certa margem de erro**

- O *setup* das redes neurais é o mesmo de todos os métodos baseados em erro
  - no entanto, a nomenclatura de alguns termos é diferente

- Dado uma *dataset* $\mathcal{D}$ com N *datapoints*: $(\mathbf{x}_n, y_n) \in (\mathcal{X}, \mathcal{Y})$
  - onde $\mathbf{x}_n$ são os *inputs* e y_n é o alvo relativos a um *datapoint* n

- algoritmo itera sobre o *dataset* e "aprende" uma função parametrizada $f : \mathcal{X} \to \mathcal{Y}$
  - esta função descreve a relação entre as *features* e o alvo
  - parâmetros também são chamados de "pesos" e controlam a saída retornada pela função
  - modelo é formado pela função e os parâmetros

- Para aprender f, o algoritmo usa função de perda/custo $\mathcal{C}(y_n,\ f(\mathbf{x}_n))$
  - *cost function*
  - Aprendizado do modelo também é chamado de *otimização*

- Embora sejam um dos mais avançados métodos baseados em erro, as redes neurais não se dão muito bem com os tipos de *features* que usamos em outros métodos

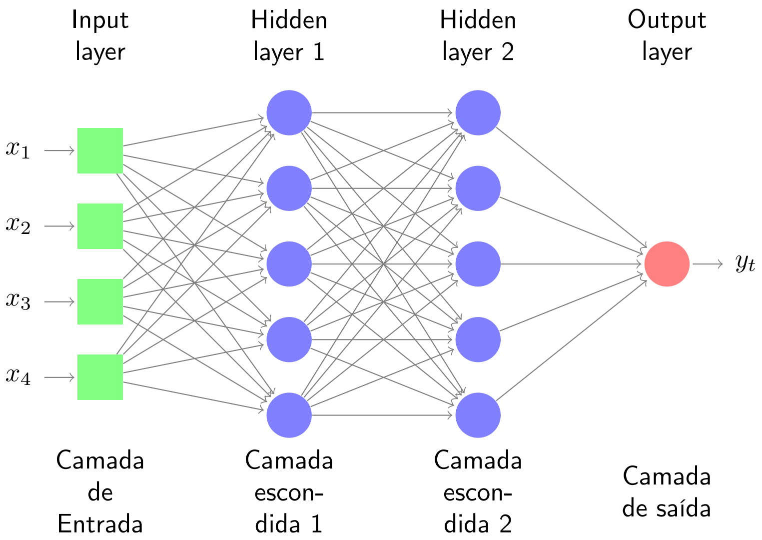
- Redes neurais "preferem" dados brutos como imagens, áudio, texto, ...

- De maneira geral, cada camada da rede neural aprende a extrair *features* dos dados brutos ou da saída da camada imediatamente anterior
  - Por isso dizemos que uma rede neural aprende a *hierarquia das features* que compõem os *datapoints*
  - Cada camada aprende um *nível* desta hierarquia
  - Quanto mais camadas, mais bem definidos os níveis dessa hierarquia
  - Daí vem o nome *Deep Learning* (ou *Aprendizado Profundo*)

Arquitetura das Redes Neurais

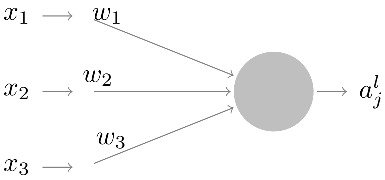
- Redes neurais são organizadas em camadas, também chamadas de *layers*
  - layers processam *features* sequencialmente, *i.e.*, um após o outro

- Cada camada é composta de unidades chamadas de "neurônios" que produzem um único número como saída
  - Portanto, a saída de uma camada da rede neural é um vetor (lista) numérico de N dimensões (uma para cada neurônio)


Input layer
Hidden layer 1
Hidden layer 2
Output layer
x_1
x_2
x_3
x_4
y_t
Camada de Entrada
Camada escon- dida 1
Camada escon- dida 2
Camada de saída

Neurônios Artificiais

- As unidades que compõem as camadas de uma rede neural são chamados neurônios
- Cada uma dessas unidades é uma função não-linear
    - A única exceção a esta regra são as unidades da camada de entrada que aplicam a função identidade $f : \mathcal{X} \to \mathcal{X}$
- Neurônios processam em paralelo $\rightarrow$ *features* processadas por um determinado neurônio de uma determinada camada também são processadas por todos os demais neurônios daquela camada ao mesmo tempo
    - A saída produzida pelos neurônios de uma camada será a entrada para os neurônios da camada subsequente



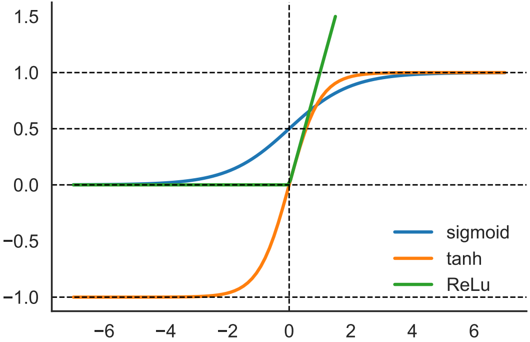
$$z = w_1 x_1 + w_2 x2 + w_3 x_3 + b = \sum_j w_j x_j + b_j = \mathbf{W}\mathbf{x} + \mathbf{b}$$

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

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

$$a = ReLu(z) = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases}$$

- a_j^l também é chamado de ativação do neurônio j na camada l


1.5
1.0
0.5
0.0
−0.5
−1.0
−6
−4
−2
0
2
4
6
sigmoid
tanh
ReLu

- Um dos motivos para usarmos este tipo de função como saída de cada neurônio é que se fizermos uma pequena alteração nos pesos, isso causará apenas uma pequena alteração na saída produzida

- O outro motivo se deve ao fato de que podemos calcular derivadas destas funções com respeito aos pesos dos neurônios e assim adaptá-los durante a fase de otimização

Gradiente Descendente

- O algoritmo de otimização padrão para redes neurais faz uso do método de Gradiente Descendente (ou *Gradient Descent*)
- Este é um método iterativo usado na área de otimização numérica para se encontrar o *mínimo* de uma função
- Fundamenta-se em cálculo diferencial

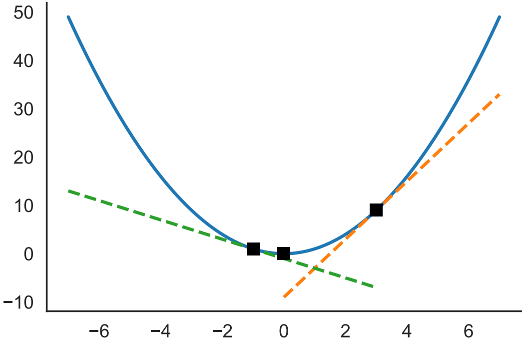
- Supondo que tenhamos a função de custo $\mathcal{C}$ e as variáveis v_1 e v_2 de uma função qualquer para otimizar
- Podemos agrupar as nossas variáveis em um vetor $\mathbf{v}$
- Definimos o vetor de gradientes como sendo

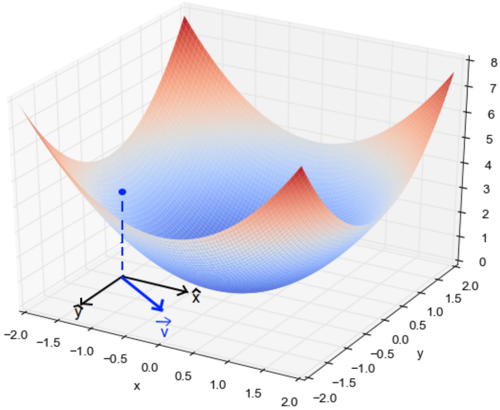
$$\nabla\mathcal{C} = \left(\frac{\partial\mathcal{C}}{\partial v_1}, \frac{\partial\mathcal{C}}{\partial v_2}\right)^T$$

- Desta forma, podemos escrever a regra de atualização em notação de matrizes/vetores

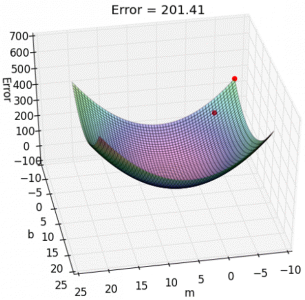
$$\hat{\mathbf{v}} = \mathbf{v} - \alpha \odot \nabla\mathcal{C}$$

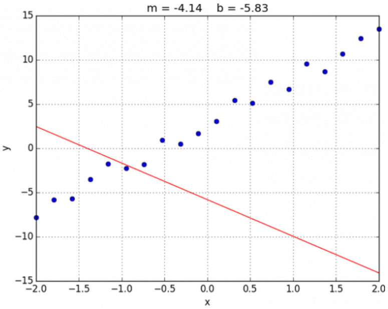
  - onde $\hat{\mathbf{v}}$ conterá os novos valores para v_1 e v_2
  - α é a taxa de aprendizado que controla o quanto do vetor de gradientes é usado para alterar cada variável
  - O símbolo $\odot$ indica multiplicação *component-wise*


50
40
30
20
10
0
−10
−6
−4
−2
0
2
4
6

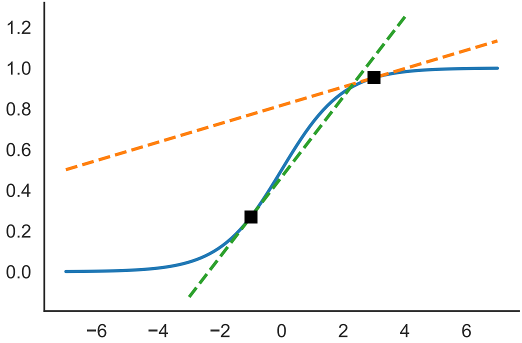


Fonte: `https://eli.thegreenplace.net/2016/understanding-gradient-descent`





Fonte: `https://towardsdatascience.com/what-i-have-understood-about-machine-learning-so-far-836d814dbe84`


1.2
1.0
0.8
0.6
0.4
0.2
0.0
−6
−4
−2
0
2
4
6

- Definições:
  - *Batch*: um grupo de N *datapoints* usado em uma iteração de Gradiente Descendente
  - Época (*epoch*): um laço de *batches* sobre o *dataset* inteiro
    - Exemplo: um *dataset* com 100 *datapoints* e $batch = 10$ *datapoints* precisará de 10 iterações (*batches*) para completar 1 época

Backpropagation

- A função de custo precisa obedecer a 2 regras:
  1. precisa ser escrita como uma média das funções de custo para cada *datapoint* no *dataset de treino*

$$\mathbf{C} = \frac{1}{n} \sum_{x} \mathcal{C}_x$$

  2. precisa ser uma função aplicável sobre a ativação dos neurônios da camada de saída

$$\mathbf{C} = C(a^L)$$

onde L indica a camada de saída da rede neural

- *Quadratic cost*

$$\mathbf{C} \equiv \frac{1}{n}\sum_{x} \|y - a^L(x)\|^2$$

- *Cross-entropy*

$$\mathbf{C} \equiv -\frac{1}{n}\sum_{x}[y \ln a + (1-y)\ln(1-a)]$$

- Equação para o erro na camada de saída (para ativação σ)

$$\delta_j^L = \frac{\partial}{\partial a_j^L}\sigma'(z_j^L) \tag{BP1}$$

- em forma de notação de matrizes:

$$\delta^L = \nabla_a \mathcal{C} \odot \sigma'(\mathbf{z}^L)$$

- Equação para o erro em uma camada (δ^l) em termos do erro na camada seguinte (δ^{l+1})

$$\delta^l = ((\mathbf{W}^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{BP2}$$

- Equação para a alteração no erro com respeito a qualquer bias (b) na rede neural

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \qquad \text{(BP3)}$$

- Em notação de matrizes

$$\frac{\partial C}{\partial b} = \delta$$

- Equação para a alteração no erro com respeito a qualquer peso (w) na rede

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \tag{BP4}$$

- Uma forma mais fácil de lembrar:

$$\frac{\partial C}{\partial w} = a_{\text{in}} \delta_{\text{out}}$$

- Resumo das equações:

$$\delta^L = \nabla_a \mathcal{C} \odot \sigma'(\mathbf{z}^L) \tag{BP1}$$

$$\delta^l = ((\mathbf{W}^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{BP2}$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{BP3}$$

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \tag{BP4}$$

1. Entrada: calcular a ativação para a camada de entrada

$$a^1 = \mathbf{x}$$

2. Feedforward: Para cada $l = 2, 3, \ldots, L$

$$z^l = w^l a^{l-1} + b^l \qquad \text{e} \qquad a^l = \sigma(z^l)$$

3. Calcular erro na saída:

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

4. Backpropagate: para cada $l = L-1, L-2, \ldots, 2$

$$\delta^l = ((\mathbf{W}^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \qquad \text{e} \qquad \frac{\partial C}{\partial b^l_j} = \delta^l_j$$

- Embora tenha sido usada a função de ativação σ nos exemplos, o processo para outras funções de ativação é o mesmo

- Basta substituir a derivada σ' pela derivada correspondente à função de ativação em uso

$$\sigma(z) = \frac{1}{1+e^{-z}} \qquad \sigma'(z) = \sigma(z) * (1 - \sigma(z))$$

$$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \qquad tanh'(z) = 1 - tanh(z)^2$$

$$ReLu(z) = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases} \qquad ReLu'(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$$

Laboratório

Implementando Backpropagation