FIS045
Introdução à Computação em Física
Introdução às redes neurais

Prof. Gustavo Guerrero (sala 4120)

Universidade Federal de Minas Gerais

2017

Redes neurais e aprendizado profundo

(*neural networs and deep learning*)

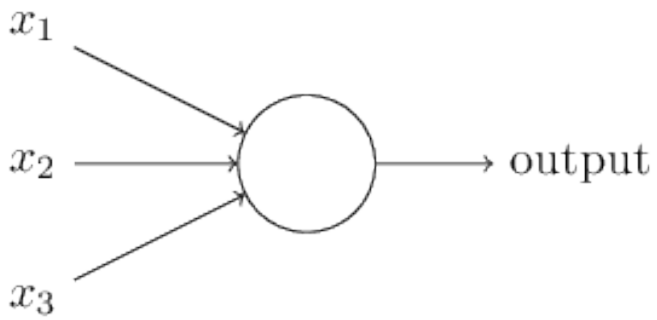
- **Redes neurais**: paradigma computacional inspirado na biologia que permite ao computador aprender a partir das observações. Nos modelos trabalhados até agora, passamos ao computador um conjunto específico de instruções para resolver um problema. No caso das redes neurais, o computador aprende a partir de "observações", encontrando por sim só a solução de um problema.
- **Aprendizado profundo**: técnicas para fazer o computador aprender nos modelos de redes neurais.

Provem as melhores soluções para reconhecimento de imagens e vozes, assim como no processamento de linguagens.
Apresentação baseada no livro *Neural networks and deep learning* by Michael Nielsen
`http://neuralnetworksanddeeplearning.com/about.html`
Veja também:
`http://www.wildml.com/2015/09/implementing-a-neural-network-from-scratch/`

Neuronios artificiais

Existem dois tipos de neuronios artificiais desenvolvidos no estudo das redes neurais:

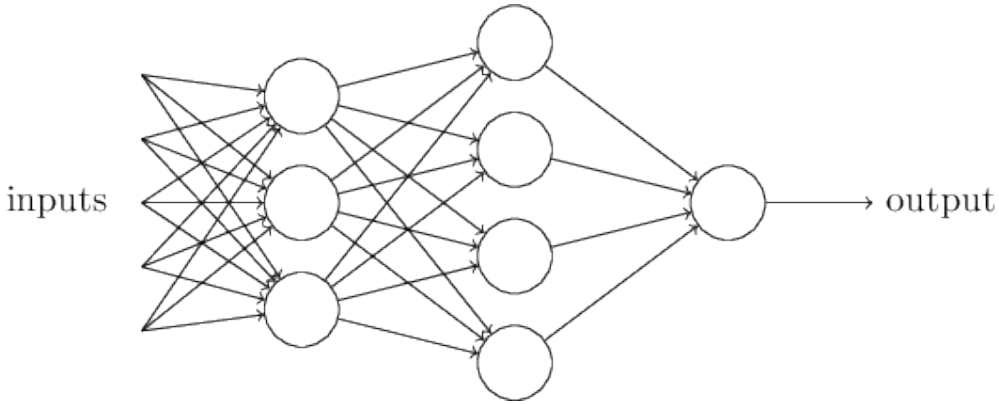
- **Perceptrons**: o modelo classico (Rosenblatt, 1960).



x_1, x_2, e x_3 são entradas que tem um pesso diferente w_1, w_2, e w_3. Os pessos são numeros reais que dão importancia aos parámetros de entrada. A saida pode ser:

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases} \quad (1)$$

uma rede de perceptrons pode ter uma configuração complexa:

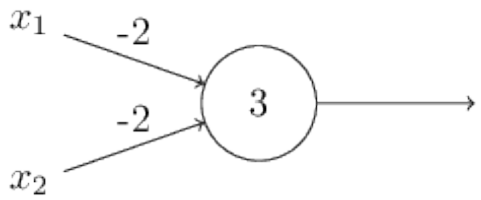


Os perceptrons da primeira camada tomam deicisões simples baseados nos parámetros de entrada. Cada perceptron tem uma saída unica. Os da segunda camada tomam decisões a partir dos resultados da primeira camada (decisões mais complexas e abstractas). Uma rede de multiples camadas pode tomar decisões altamente sofisticadas.
Podemos re-escrever a regra de funcionamento de um perceptron:

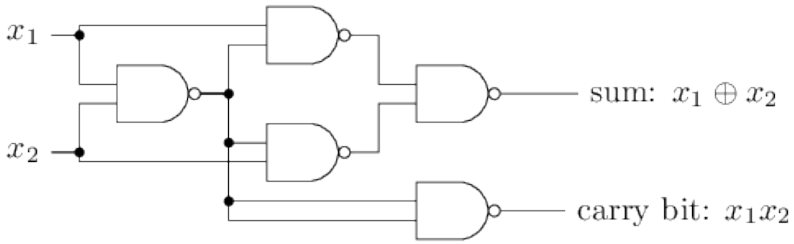
$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases} \tag{2}$$

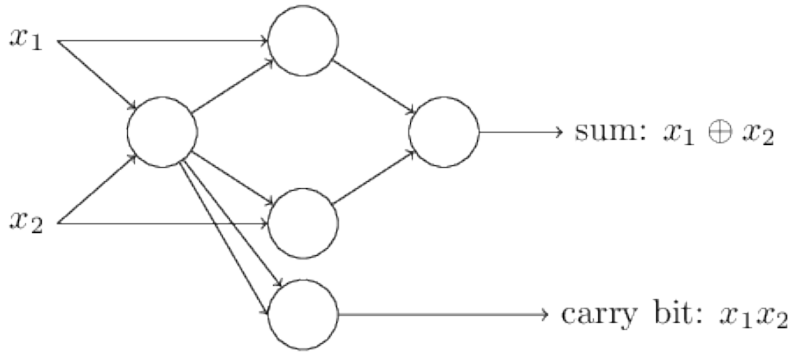
onde $b = -\text{thereshold}$, mede que tão fácil é ativar o perceptron. Note também que agora usamos um produto escalar, **x** e **w** são vetores.

Os perceptrons também podem ser utilizados como operações lógicas fundamentais: e.g., AND, OR ou NAND. Na seguinte figura cada uma das entradas tem pesso -2, e o bias é 3:



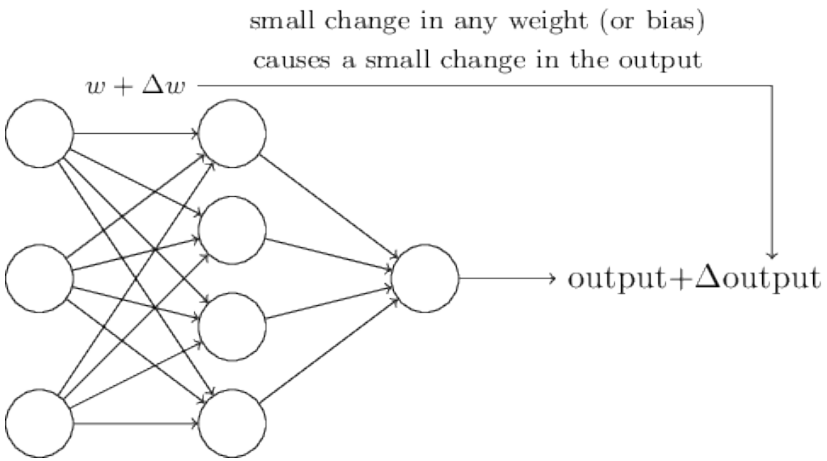
As entradas $0-0$, $0-1$ e $1-0$ geram uma saida positiva, mas a entrada $1-1$ gera uma saida negativa. Assim o perceptron é uma porta NAND.





Problema com o perceptron: pequenas mudanças nos pessos podem mudar a saida binaria. Para resolver:

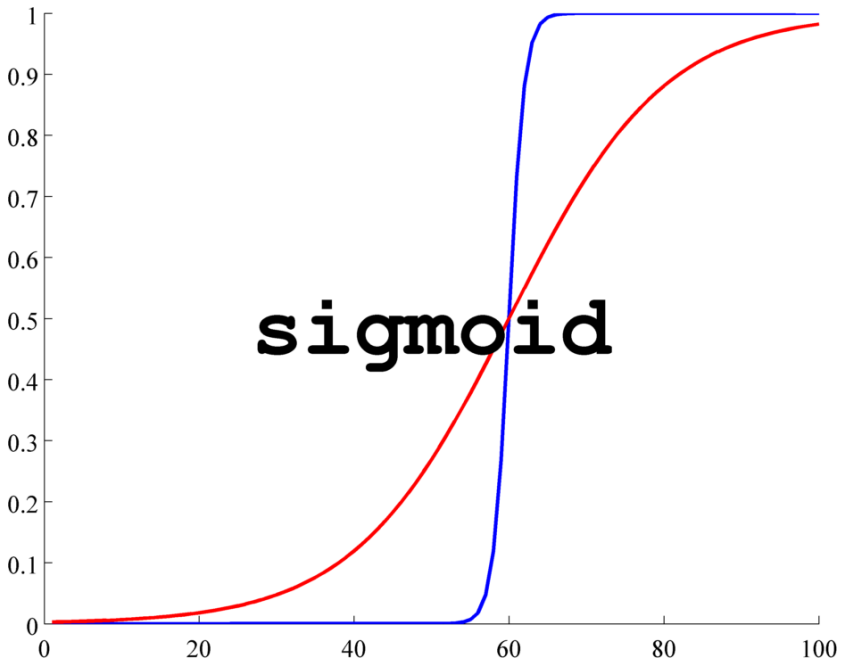
- **Neuronio sigmoide**: Similar ao perceptron, mas aceita pequenas mudanças nos pessos sem afetar consideravelmente a saida. Isto é feito usando a função de ativação sigmoide, σ



$$\sigma(w, x, b) = \frac{1}{1 + \exp(-\sum_j w_j x_j - b)}. \tag{3}$$

Outras funções podem ser usadas para ativar um neuronio: tanh, ReLus, ou no caso de saidas estatísticas, se usa a função softmax.

O sigmoide pode se comportar também como um perceptron

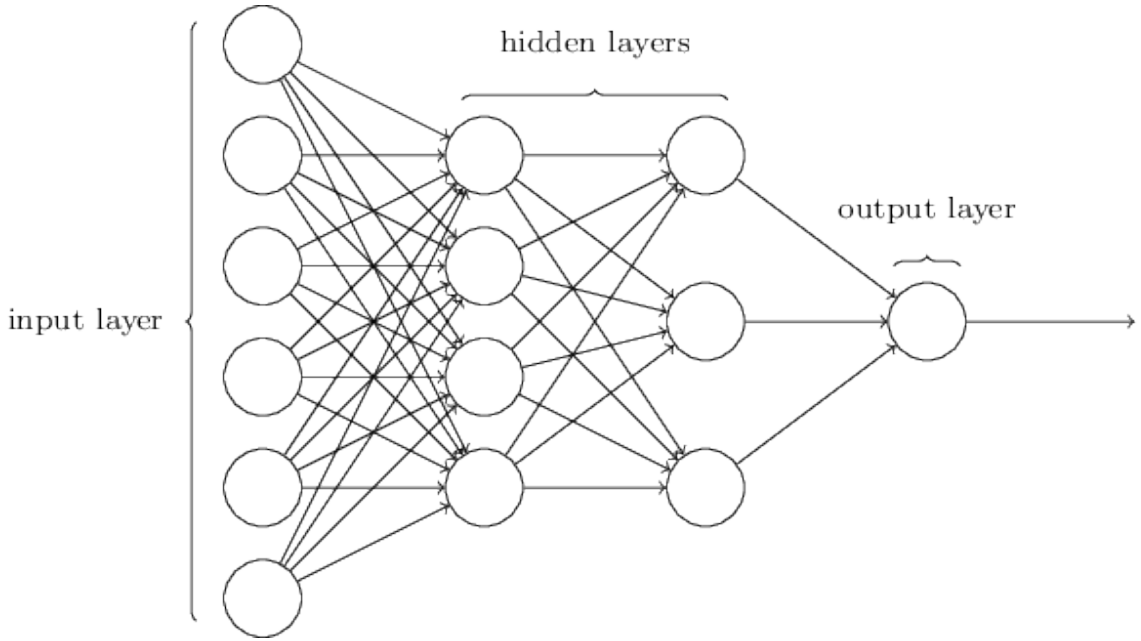


A variação na saida é linear em w e em b, pode ser calculada a partir de

$$\Delta \text{output} \approx \sum_j \frac{\partial \, \text{output}}{\partial w_j} \Delta w_j + \frac{\partial \, \text{output}}{\partial b} \Delta b, \tag{4}$$

Rede neural para classificar números escritos na mão

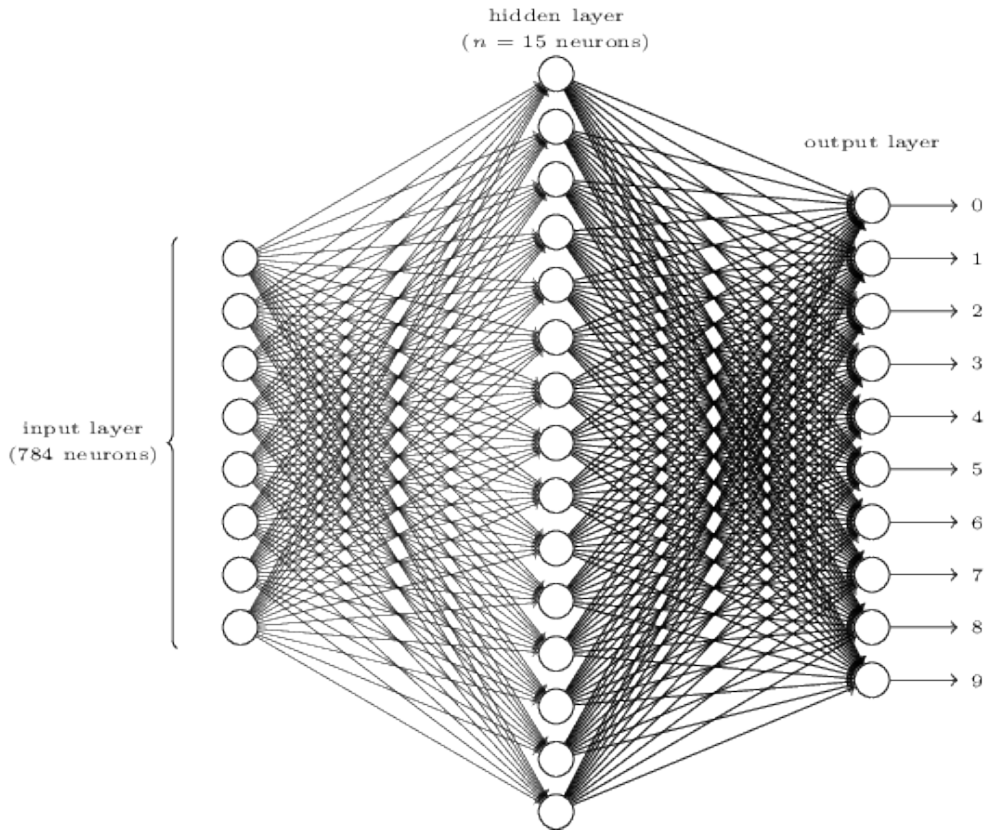
Arquitetura de uma rede neural



Rede feedforward : sem loops na rede, a informação vai passando sequêncialmente nas diferentes camadas.
Rede recorrente: existem loops entre as diferentes camadas. Isto é possível fazendo que cada neuronio fice ativado só por um tempo limitado.

Queremos reconhecer uma serie de números escritos a mão:

há dois problemas, (a) segmentar o número, e (b) reconhecer os digitos individualmente. Vamos nos focar no segundo problema usando uma rede neural de 3 camadas



Aprendizado com gradiente descendente

- Vamos a usar a base de dados MINST que contém milhares de números escritos a mão escanneados.
- x vai denotar cada entrada de treinamento, x é um vetor de dimensão $28 \times 28 = 784$
- a saida desejada é denotada com $y = y(x)$, que é um vetor de dimensão 10. E.g., se a saida é um 6, então $y(x) = (0, 0, 0, 0, 0, 0, 1, 0, 0, 0)^T$
- Queremos um algoritmo que encontre os pessos e os *biasses* tal que a saida se aproxima a $y(x)$ para todas as entradas de treinamento x. Para quantificar nosso objetivo usamos a função custo:

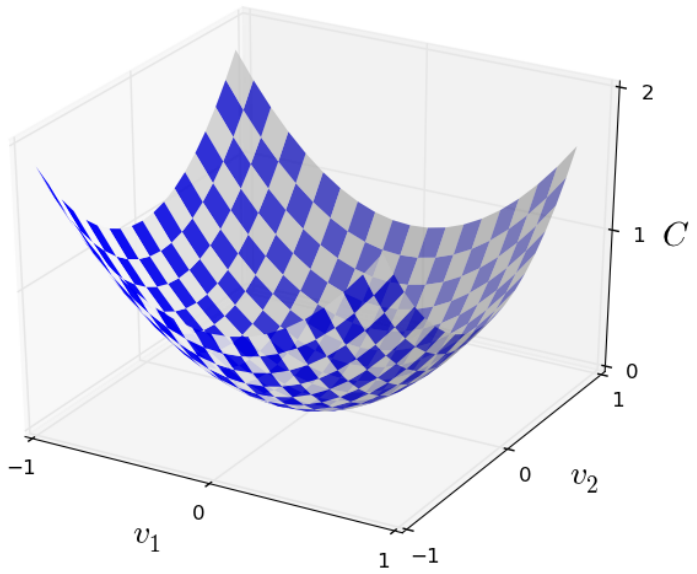
$$C(w, b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2. \tag{5}$$

onde a é o vetor de saidas. (Há outras possibilidades para a função custo.)

- o objetivo de nosso algorítmo é minimizar $C(w, b)$. Fazemos isso usando um método conhecido como **gradiente descendente**.

Gradiente descendente

- Queremos minimizar uma função $C(v)$, multivariável, com valor real.
- Imaginemos que C tem só duas varíaveis, v_1 e v_2



- queremos encontrar o ponto onde C tem um mínimo global.

Fazemos isso com um esquema numérico baseado nas seguintes ideias matemáticas

- se temos uma variação Δv_1 na direção v_1 e uma variação Δv_2 na direção v_2, a variação em C é

$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2. \tag{6}$$

- encontraremos uma forma de fazer ΔC negativa.
- sejam os vetores $\Delta v \equiv (\Delta v_1, \Delta v_2)^T$ e $\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}\right)^T$ assim

$$\Delta C \approx \nabla C \cdot \Delta v. \tag{7}$$

- Que tal escolher $\Delta v = -\eta \nabla C$,
- dessa forma $\Delta C \approx -\eta \nabla C \cdot \nabla C = -\eta \|\nabla C\|^2$, onde η é chamado *taxa de aprendizado*.
- essa escolha garante $\Delta C \leq 0$, i.e., C sempre decresce.
- o esquema é fácilmente generalizável a m variáveis.

A ideia é então usar o gradiente descendente para que a rede neural aprenda, i.e., encontrar os pessos, w_k e os biasses, b_l, que minimizam a função custo.

- Nosso espaço tem agora w_k e b_l, logo o vetor gradiente ∇C tem componentes, $\partial C/\partial w_k$ e $\partial C/\partial b_l$.
- Nossa regra para atualizar os valores de w_k e b_l fica:

$$w_k \rightarrow w_k' = w_k - \eta \frac{\partial C}{\partial w_k} \tag{8}$$

$$b_l \rightarrow b_l' = b_l - \eta \frac{\partial C}{\partial b_l}. \tag{9}$$

- aplicando essa regra repetidas vezes encontramos o mínimo de C.
- **principal inconveniente**: exigente em termos computacionais.
- gradiente descrescente estocastico

Gradiente descrescente estocastico

- escolhe randomicamente um número n das amostras de treinamento, as quales chamaremos $X_1, X_2, \ldots, X_n$
- esperamos que

$$\frac{\sum_{j=1}^{m} \nabla C_{X_j}}{m} \approx \frac{\sum_x \nabla C_x}{n} = \nabla C, \tag{10}$$

- assim, o treinamento será feito sobre uma amostragem menor

$$w_k \rightarrow w_k' = w_k - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial w_k} \tag{11}$$

$$b_l \rightarrow b_l' = b_l - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial b_l}, \tag{12}$$

Como implementar o GDE? Algoritmo de Backpropagation

O algoritmo foi introduzido por Rumelhart, Hinton & Williams (1986)
`https://www.nature.com/articles/323533a0.pdf`

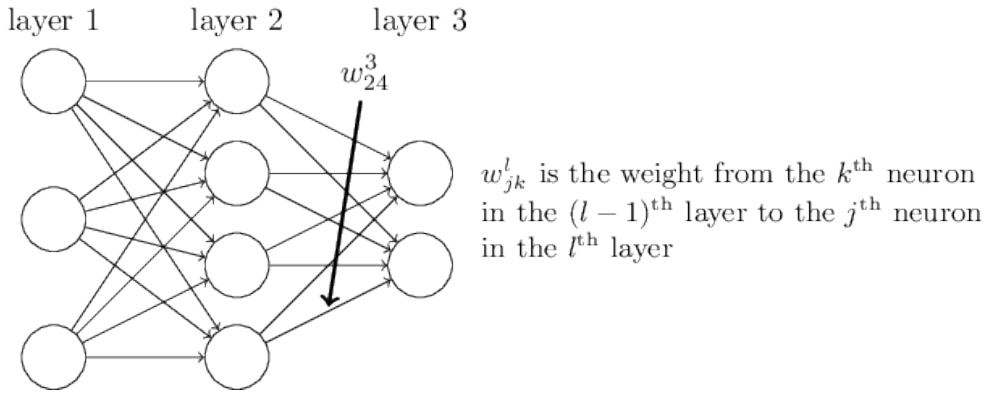
Learning representations by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton† & Ronald J. Williams*

* Institute for Cognitive Science, C-015, University of California, San Diego, La Jolla, California 92093, USA
† Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure[1].

- **Notação**: Vamos usar w^l_{jk} para denotar o pesso da conexão entre o k-essimo neuronio na camada $(l-1)$ com o j-essimo neuronio na camada l.



- b^l_j e a^l_j são o biass e a ativação do j-essimo neuronio na camada l.
- Com essa notação a ativação do j-essimo neuronio na camada l pode ser calculado como

$$a^l_j = \sigma\left(\sum_k w^l_{jk} a^{l-1}_k + b^l_j\right), \qquad \text{ou} \qquad a^l = \sigma(w^l a^{l-1} + b^l)$$

onde a soma é sobre todos os k neuronios na camada $l-1$.

- Chamamos $z^l \equiv w^l a^{l-1} + b^l$ o pesso de entrada do neuronio na camada l. Assim, $a^l = \sigma(z^l)$.
- Vamos assumir também que a função custo total, C pode ser escrita como a média sob funções custo individuais, C_x.
- Assumimos também que essa função pode ser escrita como função das saidas das redes neurais. Para nosso caso

$$C = \frac{1}{2}\|y - a^L\|^2 = \frac{1}{2}\sum_j (y_j - a_j^L)^2, \tag{13}$$

Essa suposição é verdadeira pois depende de a^L. A função também depende de y_j, porém y é uma variável independente (uma observação).

Quatro equações fundamentais do algoritmo de Backpropagation (demonstração no livro online)

- **Erro na camada de saida**:

$$\begin{aligned} \delta_j^L &= \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L), \quad \text{ou} \\ \delta^L &= \nabla_a C \odot \sigma'(z^L). \end{aligned}$$

- **Erro δ^l como função da camada seguinte**:

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

- **Taxa de variação do custo com respeito a qualquer biass**

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

- **Taxa de variação do custo com respeito ao pesso**

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

O algoritmo Backpropagation

O algorítmo para cada um dos treinamentos é:

1. **Entrada** x: defina a ativação neural, a^1 da camada de entrada
2. **FeedForward**: Para cada $l = 2, 3, ..., L$, calcule $z^l = w^l a^{l-1} + b^l$, e $a^l = \sigma(z^l)$
3. **Erro na saida**: Calcule o vetor $\delta^L = \nabla_a C \odot \sigma'(z^L)$
4. **Propague o erro para atras**: Para cada $l = L-1, L-2, \ldots, 2$, calcule $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$
5. **Saida**: O gradiente da função custo pode ser avaliado usando: $\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$, junto com, $\frac{\partial C}{\partial b^l_j} = \delta^l_j$.

O algoritmo Backpropagation

O algorítmo para para um gradiente decrescente estocástico:

1. **Entrada**: defina os exemplos de treinamento
2. Para cada um dos treinamentos, x: defina a ativação neural $a^{x,1}$ e siga os seguintes passos
   - **FeedForward**: Para cada $l = 2, 3, ..., L$, calcule $z^{x,l} = w^l a^{x,l-1} + b^l$, e $a^{x,l} = \sigma(z^{x,l})$
   - **Erro na saida**, $\delta^{x,L}$: Calcule o vetor $\delta^{x,L} = \nabla_a C_x \odot \sigma'(z^{x,L})$
   - **Propague o erro para atras**: Para cada $l = L-1, L-2, \ldots, 2$, calcule $\delta^{x,l} = ((w^{l+1})^T \delta^{x,l+1}) \odot \sigma'(z^{x,l})$
3. **Gradiente descendente**: Para cada $l = L, L-1, \ldots, 2$, atualize os pessos e os biasses segundo:

$$w^l \rightarrow w^l - \frac{\eta}{m} \sum_x \delta^{x,l} (a^{x,l-1})^T$$

$$b^l \rightarrow b^l - \frac{\eta}{m} \sum_x \delta^{x,l}$$

Duas perguntas fundamentais:

1. Dado um conjunto de entradas x, é possível construir uma rede neural que tenha como saida **qualquer** função f?

   R: Sim (ver capitulo 4 do livro online): universalidade das redes neurais.

2. Porque os aprendizados "deep"e "cheap"funcionam tão bem para resolver problemas físicos?

Why does deep and cheap learning work so well?

Henry W. Lin, Max Tegmark, and David Rolnick
Dept. of Physics, Harvard University, Cambridge, MA 02138
Dept. of Physics, Massachusetts Institute of Technology, Cambridge, MA 02139 and
Dept. of Mathematics, Massachusetts Institute of Technology, Cambridge, MA 02139

(Dated: March 11 2017)

Leia também:

The Extraordinary Link Between Deep Neural Networks and the Nature of the Universe:
`https://www.technologyreview.com/s/602344/`

- "... the universe is governed by a tiny subset of all possible functions. In other words, when the laws of physics are written down mathematically, they can all be described by functions that have a remarkable set of simple properties".
- "For reasons that are still not fully understood, our universe can be accurately described by polynomial Hamiltonians of low order."
- There is another property of the universe that neural networks exploit. This is the hierarchy of its structure. "Elementary particles form atoms which in turn form molecules, cells, organisms, planets, solar systems, galaxies, etc.". The layers in these networks can approximate each step in the causal sequence.
- "Artificial neural networks are famously based on biological ones. So not only do Lin and Tegmark's ideas explain why deep learning machines work so well, they also explain why human brains can make sense of the universe. Evolution has somehow settled on a brain structure that is ideally suited to teasing apart the complexity of the universe."