

data

Redes Neurais

Gustavo Sutter e João G. M. Araújo •

12/05/2019

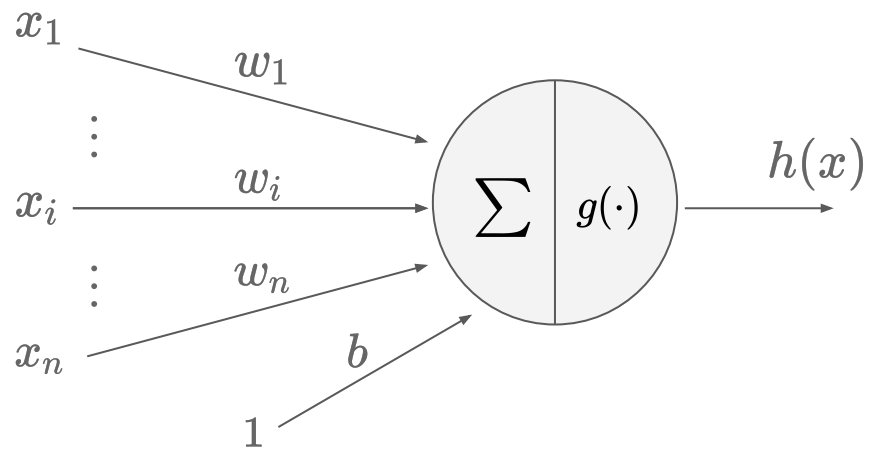
@suttergustav

Redes neurais

- Algoritmo muito poderoso utilizado para diferentes tarefas de **classificação** e **regressão**
- Possui diferentes variações que acrescentam ainda mais poder em suas representações
- Sua unidade mais básica é um neurônio



Neurônio



Neurônio

- Pré ativação do neurônio

$$a(\mathbf{x}) = b + \sum_i x_i w_i$$

- Ativação (saída) do neurônio

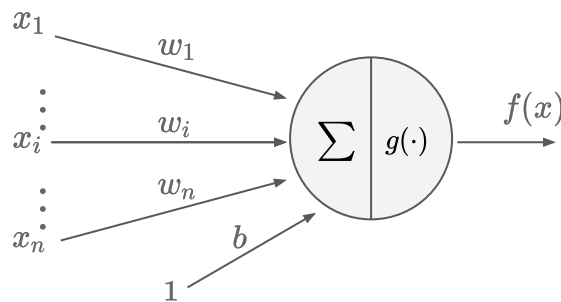
$$f(\mathbf{x}) = g(a(\mathbf{x}))$$

onde,

\mathbf{w} é o vetor de pesos

$g(\cdot)$ é a função de ativação

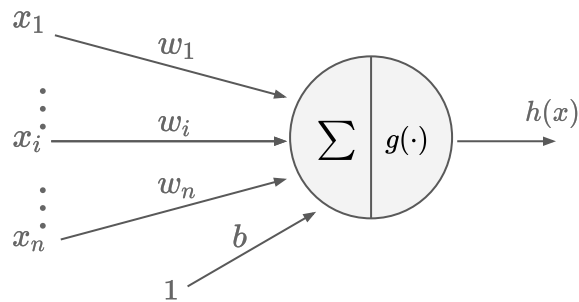
b é o termo de bias



Neurônio

- Ou simplesmente:

$$f(\mathbf{x}) = g(\mathbf{w}^\top \mathbf{x} + b)$$



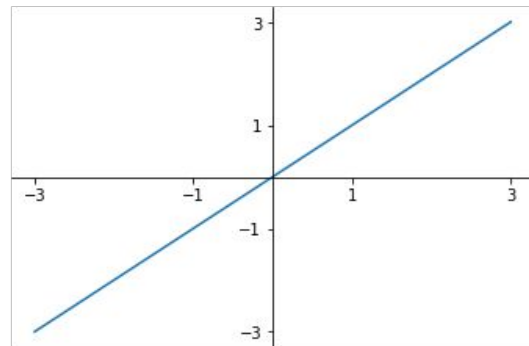
Note as semelhanças com o a regressão linear e logística



Funções de ativação

- Ativação linear
 - Não restringe saída a nenhum intervalo
 - Não é muito interessante
 - Não permite aproximação universal

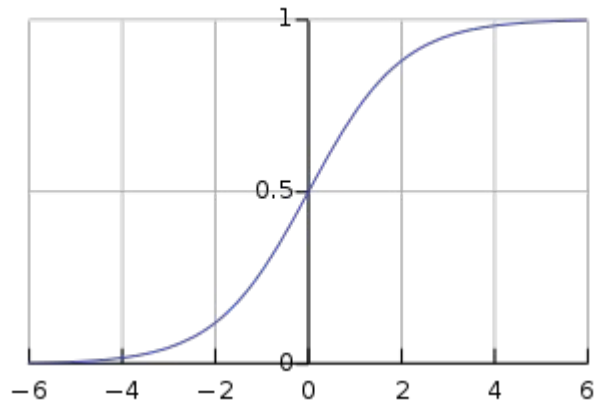
$$g(x) = x$$



Funções de ativação

- Sigmóide
 - Saída pertence ao intervalo $[0, 1]$
 - Estritamente crescente
 - Ativação clássica

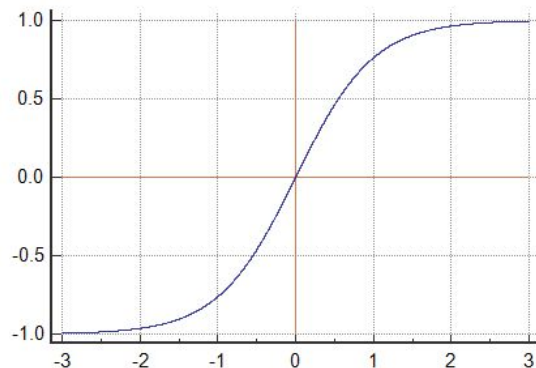
$$g(x) = \text{sigm}(x) = \frac{1}{1+e^{-x}}$$



Funções de ativação

- Tangente hiperbólica (tanh)
 - Saída pertence ao intervalo $[-1, 1]$
 - Estritamente crescente
 - Centrada no zero
 - Também clássica, porém com melhor aprendizado que a sigmóide

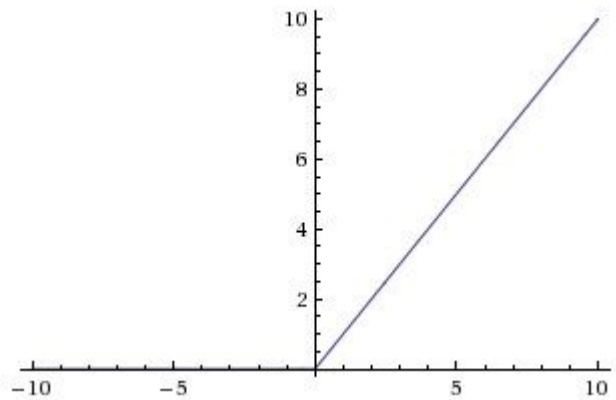
$$g(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



Funções de ativação

- Rectified linear unit (ReLU)
 - Saída pertence ao intervalo $[0, +\infty)$
 - Só é limitada inferiormente
 - Escolha popular em deep learning, facilita o treino em relação às anteriores
 - Superfícies de separação menos suaves

$$g(x) = \text{ReLU}(x) = \max(0, x)$$



Capacidade de um neurônio

- Podemos usar um neurônio para realizar classificação binária. Para isso utilizamos a função sigmóide para ativação e consideramos o seguinte

$$f(\mathbf{x}) = P(y = 1|\mathbf{x})$$



Capacidade de um neurônio

- Podemos usar um neurônio para realizar classificação binária. Para isso utilizamos a função sigmóide para ativação e consideramos o seguinte

$$f(\mathbf{x}) = P(y = 1|\mathbf{x})$$

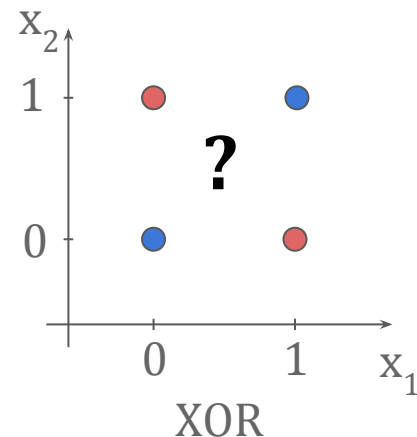
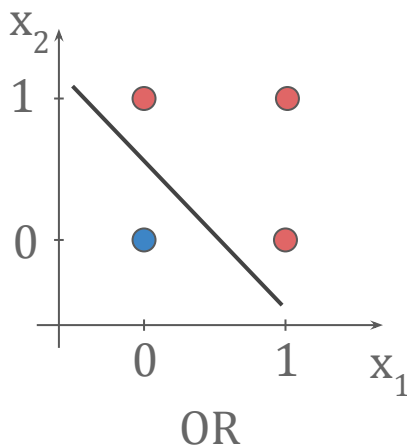
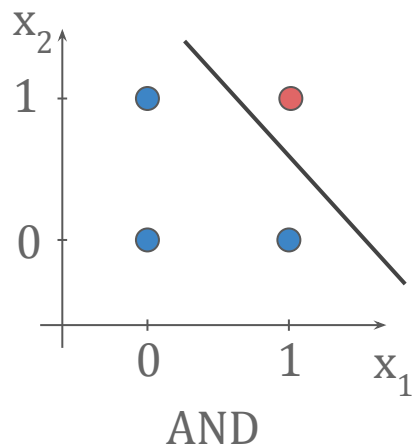


Esse é o classificador de regressão logística, que já vimos antes



Capacidade de um neurônio

- Como vimos quando estudamos a regressão logística isso produz um **classificador linear**



Multilayer neural network

- Chamamos as camadas no interior da rede de camadas escondidas ou, em inglês, *hidden layers*.
- Cada layer aprende uma representação de mais alto nível dos dados (pixels -> retas -> curvas -> objetos)
- Uma rede **pode ter várias hidden layers**, veremos agora a formulação para uma rede com apenas uma camada escondida (mas o processo é completamente análogo para mais de uma)



Multilayer neural network (one hidden layer)

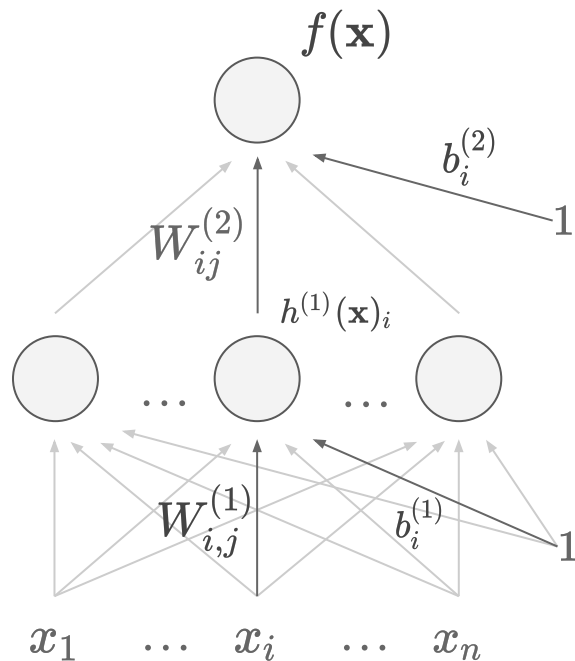
- Primeira camada

$$h(\mathbf{x}) = g(W^{(1)} \mathbf{x} + b^{(1)})$$

- Saída da rede

$$f(\mathbf{x}) = o(W^{(2)} h(\mathbf{x}) + b^{(2)})$$

output activation



Multilayer neural network - camada de saída

- Para classificação binária podemos usar um neurônio na camada de saída fazendo uso da ativação sigmóide
- Para classificação multi-classe utiliza-se algo semelhante a regressão multinomial, com um neurônio de saída para cada classe e softmax como ativação, tal que:

$$f(\mathbf{x})_c = P(y = c|\mathbf{x})$$



Softmax activation

- Para a última camada ter esse significado utilizamos a função de ativação **softmax**

$$\textit{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

- Isso garante que as saídas pertencem ao intervalo $[0, 1]$ e que a soma de todas as saídas é igual a 1.

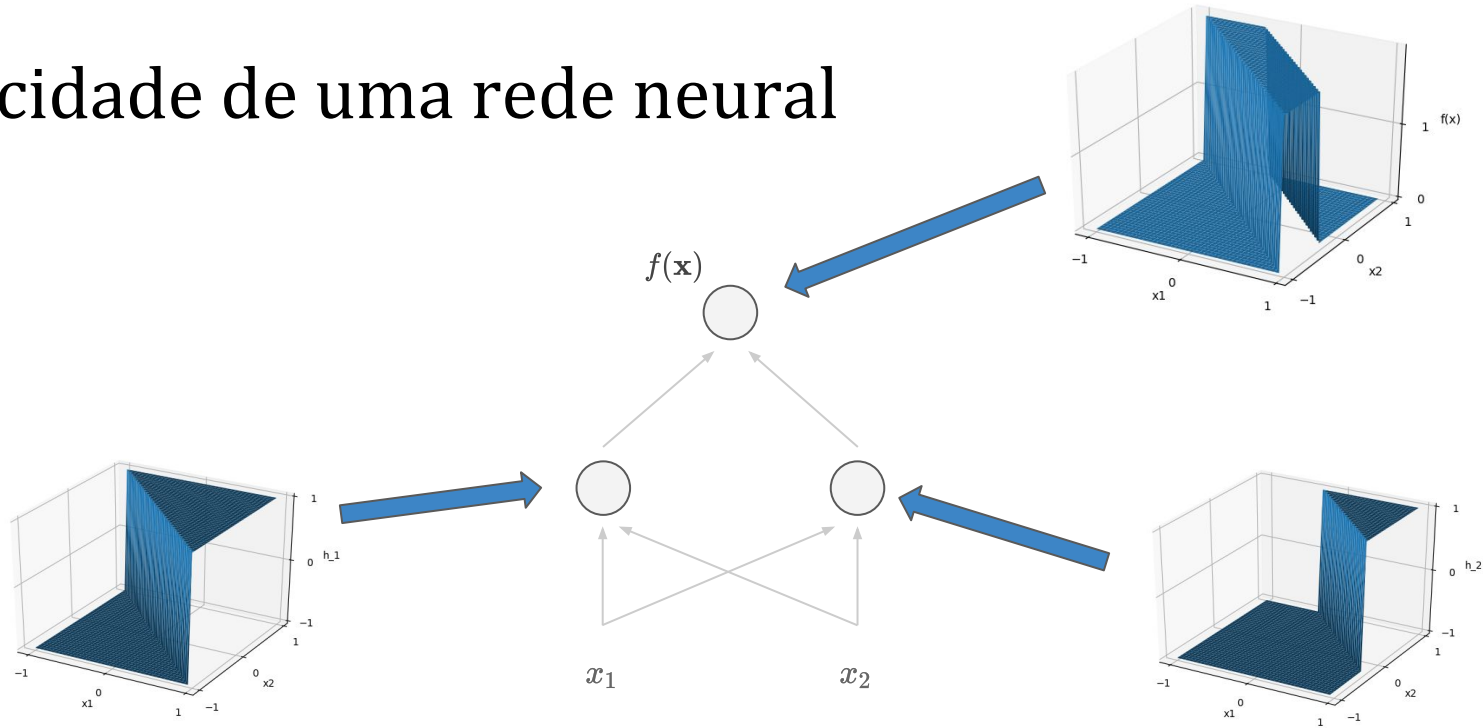


Capacidade de uma rede neural

- Teorema da aproximação universal (Hornik, 1991)
 - “Uma rede neural com uma camada escondida e função de ativação não polinomial, que faz uso de uma ativação linear em sua saída pode aproximar qualquer função contínua arbitrariamente bem em um conjunto compacto, se dado o número suficiente de neurônios escondidos”
- Resultados se aplicam para sigmóide, tanh e outras funções de ativação
- Mas ele só garante a existência de pesos que permitam essa aproximação, não que existe uma algoritmo capaz de encontrar esses valores



Capacidade de uma rede neural




Demonstração gráfica de como é possível criar funções com a forma desejada utilizando combinação dos neurônios da camada interna



Mas como encontramos os parâmetros da rede?

Treinando a rede neural

- Para treinar nossa rede iremos utilizar o algoritmo de gradient descent, assim como vimos antes.
- Nosso objetivo é minimizar a seguinte função:

$$\frac{1}{N} \sum_i l(f(\mathbf{x}^{(i)}), y^{(i)})$$


função de custo



Loss function

- Iremos usar uma loss function chamada **cross entropy loss**:

$$l(f(\mathbf{x}), y) = - \sum_c 1_{(y=c)} \log f(\mathbf{x})_c = -\log f(\mathbf{x})_y$$



Gradient Descent

- Inicializar valores de cada um dos parâmetros (muitos métodos de inicialização, importante para redes profundas)
- Por N iterações: (cada iteração é chamada de epoch)
 - Para cada exemplo do conjunto de treino:
 - Calcular gradiente de cada um dos parâmetros
 - Atualizar os parâmetros usando os gradientes calculados



Gradient Descent

- Inicializar valores de cada um dos parâmetros
- Por N iterações: (N é chamado de epoch)
 - Para cada exemplo do conjunto de treino:
 - **Calcular gradiente** de cada um dos parâmetros
 - Atualizar os parâmetros usando os gradientes calculados

Vamos ver como calculamos o gradiente dos parâmetros!
Algoritmo de backpropagation

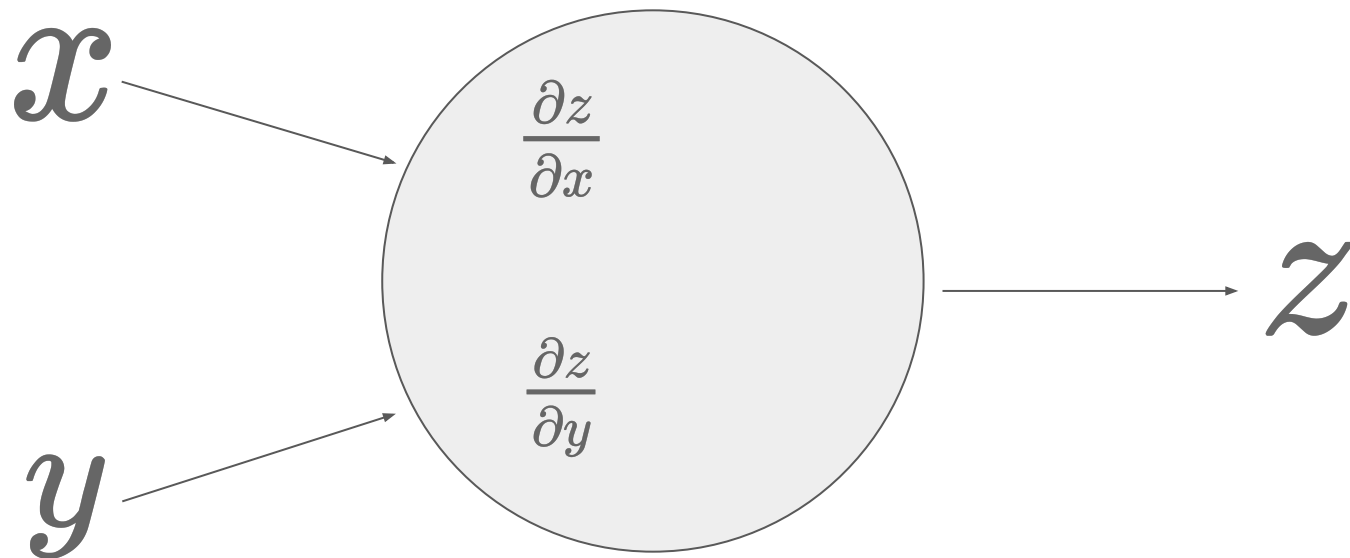


Grafo computacional

- Um grafo computacional é uma maneira mais computacional (quem diria, não?) de se pensar no processo de backpropagation
- Ele é composto por nós que possuem entrada e saída
- E a derivada da operação realizada pelo nó em relação às suas entradas é conhecida



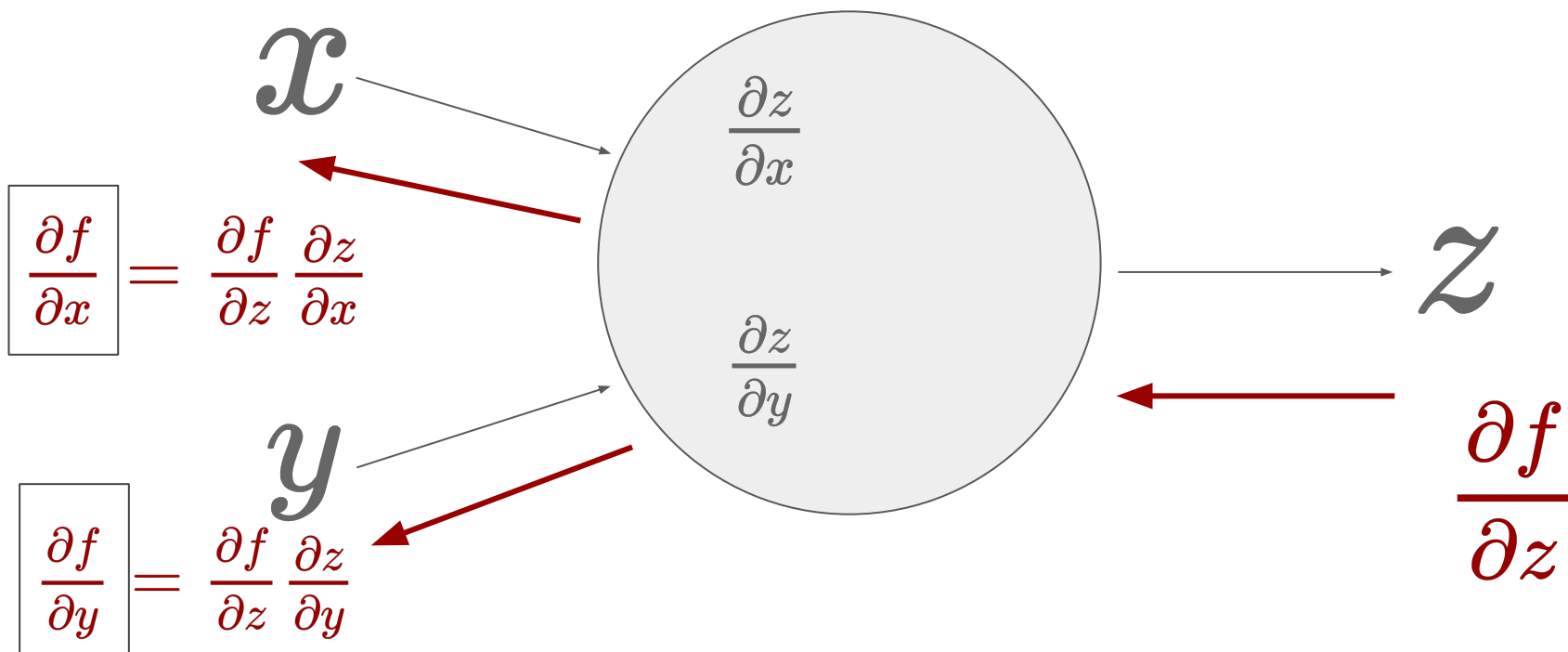
Grafo computacional: nó computacional



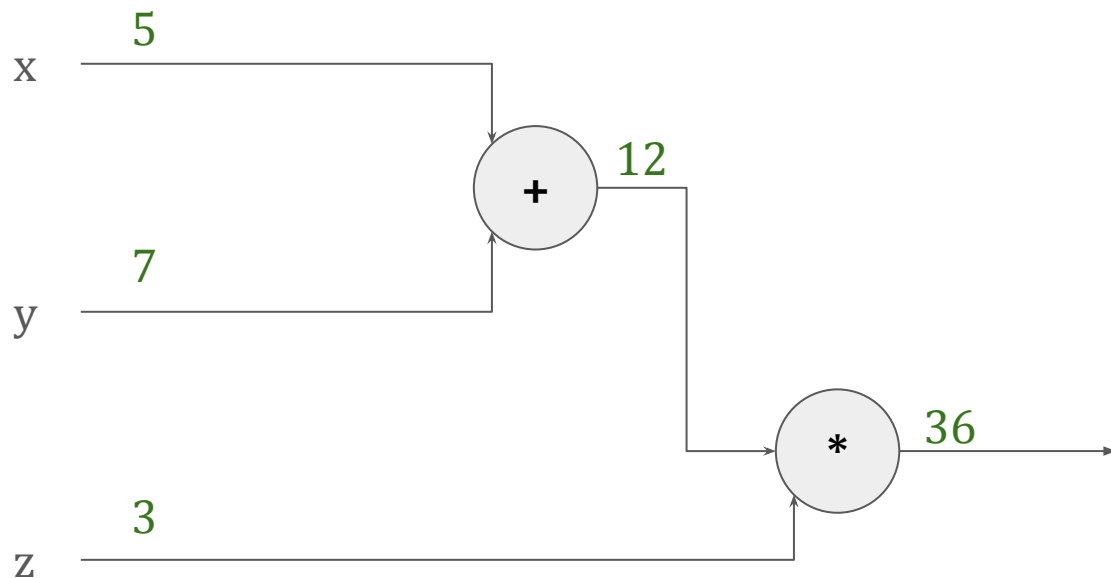
(Isso não é um neurônio, é um nó do grafo computacional!!!)



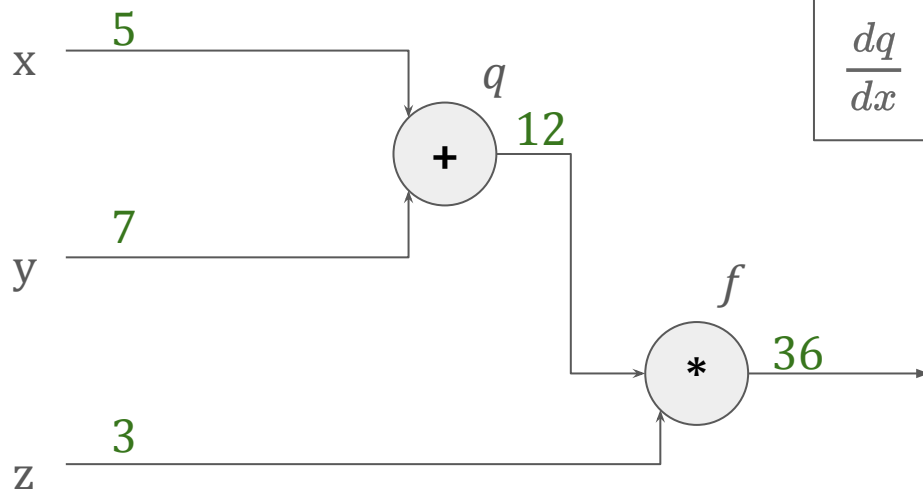
Grafo computacional: nó computacional



Grafo computacional



Grafo computacional

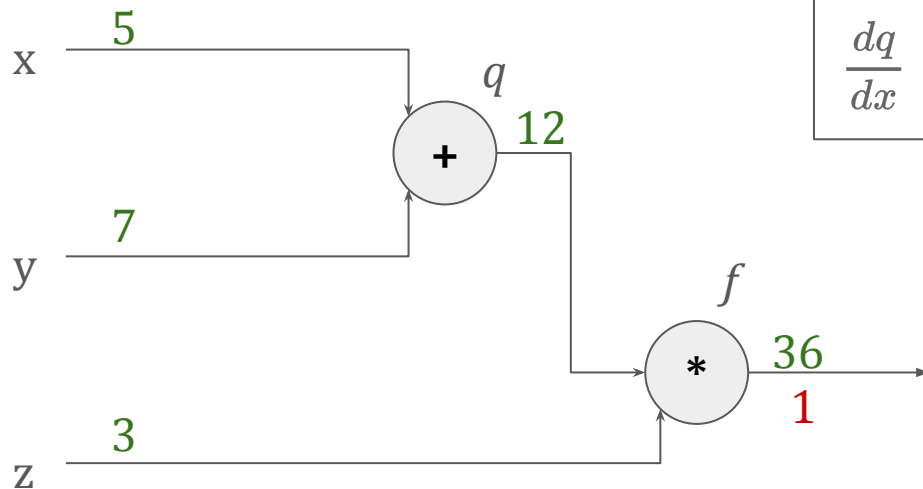


Sabemos

$$\frac{dq}{dx} = 1 \quad \frac{dq}{dy} = 1 \quad \frac{df}{dq} = z \quad \frac{df}{dz} = q$$



Grafo computacional: backpropagation



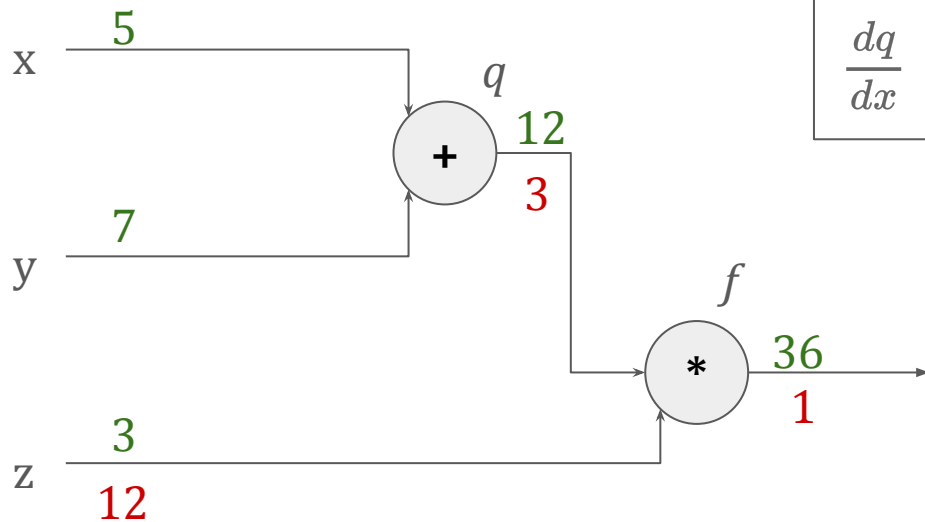
Sabemos

$$\frac{dq}{dx} = 1 \quad \frac{dq}{dy} = 1 \quad \frac{df}{dq} = z \quad \frac{df}{dz} = q$$

$$\frac{\partial f}{\partial f} = 1$$



Grafo computacional: backpropagation



Sabemos

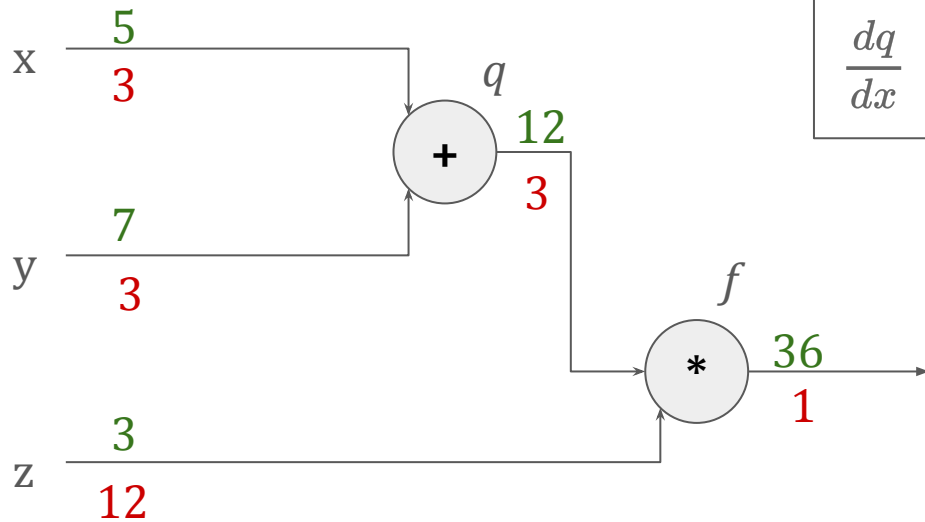
$$\frac{dq}{dx} = 1 \quad \frac{dq}{dy} = 1 \quad \frac{df}{dq} = z \quad \frac{df}{dz} = q$$

$$\frac{\partial f}{\partial q} = \frac{\partial f}{\partial q} \frac{\partial f}{\partial f} = z = 3$$

$$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial z} \frac{\partial f}{\partial f} = q = 12$$



Grafo computacional: backpropagation



Sabemos

$$\frac{dq}{dx} = 1 \quad \frac{dq}{dy} = 1 \quad \frac{df}{dq} = z \quad \frac{df}{dz} = q$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x} = z = 3$$

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y} = z = 3$$



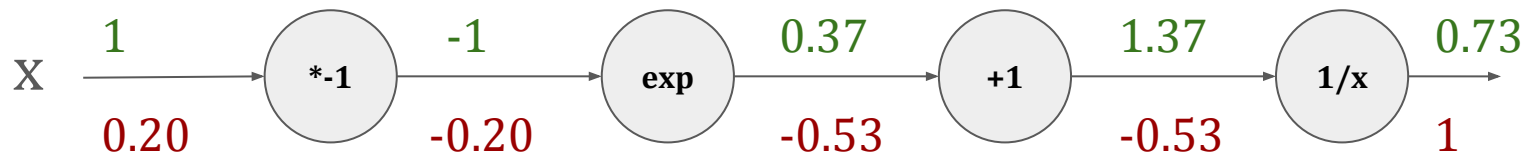
Grafo computacional

- Podemos ver alguns padrões para os nós de um grafo computacional
 - Soma: Distribui o gradiente que recebeu igualmente para suas entradas
 - Multiplicação: Multiplica o gradiente pela outra entrada
- Caso o nó tenha mais de uma saída temos que somar o gradiente de ambas



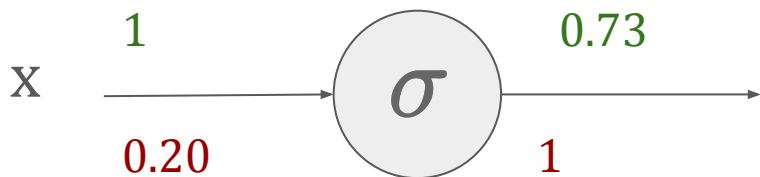
Grafo computacional

- O nível de abstração de um nó no grafo computacional é totalmente arbitrário, basta saber a derivada parcial da saída em relação a entrada



Grafo computacional

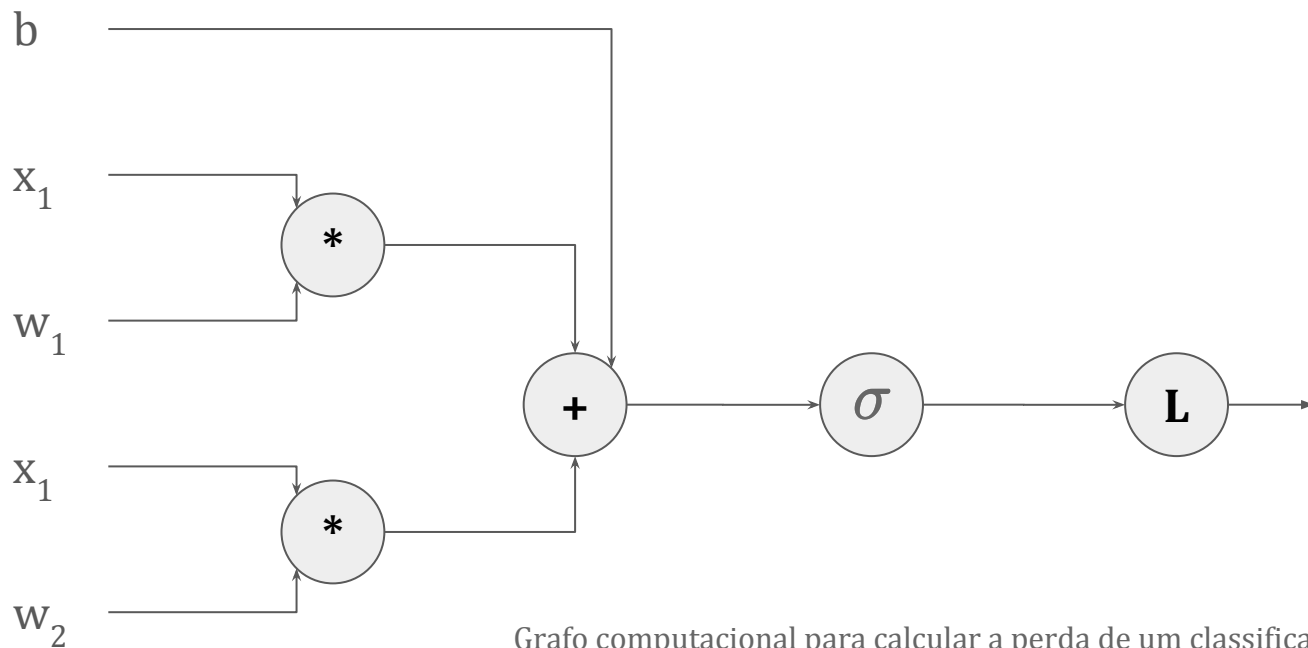
- O nível de abstração de um nó no grafo computacional é totalmente arbitrário, basta saber a derivada parcial da saída em relação a entrada



$$\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$$



Grafo computacional



Grafo computacional para calcular a perda de um classificador que consiste em um neurônio

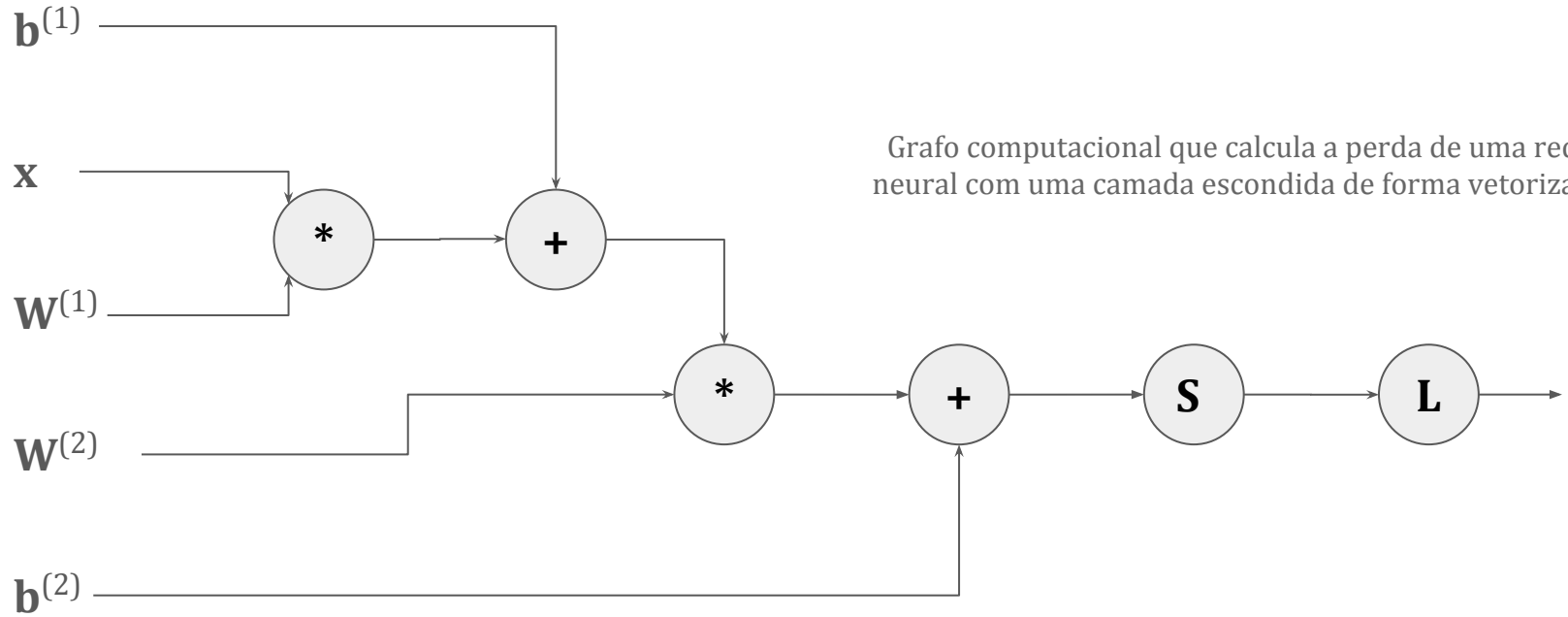


Grafo computacional

- Na prática construímos grafos computacionais que vetorizem as operações
 - Vetorizar é utilizar operações em todo o vetor ou matriz de uma só vez, ganhando em performance
- Conceito é o mesmo, mas o que flui no grafo não são escalares, sim vetores e matrizes ([material extra sobre derivadas com vetores e matrizes](#))



Grafo computacional



Gradient Descent

- Pronto! Agora sabemos como calcular o gradiente para cada um dos nossos parâmetros
- Com isso podemos realizar a atualização dos valores dos parâmetros a cada passo do algoritmo



Gradient Descent

- Inicializar valores de cada um dos parâmetros
- Por N iterações: (cada iteração é uma época)
 - Para cada exemplo do conjunto de treino:
 - Fazer *forward pass* do exemplo (computar a saída da rede dado o exemplo)
 - Calcular a *loss*
 - Atualizar a *loss total*
 - Utilizando o *backpropagation* calcular o gradiente da *loss* em relação aos parâmetros
 - Atualizar o valor de cada parâmetro seguindo o *learning rate*



Stochastic Gradient Descent

- Cada passo do Gradient Descent clássico demora a ser computado (precisamos calcular a loss e a para cada exemplo do conjunto de teste)
- Assim a rede computa vários outputs efetivamente aleatórios antes do primeiro passo de backpropagation
- (Mini-batch) Stochastic Gradient Descent separa o conjunto de treinamento em grupos (batches), e para cada grupo calcula a loss e faz backpropagation



Stochastic Gradient Descent

- Inicializar valores de cada um dos parâmetros
- Por N iterações: (cada iteração é uma época)
 - **Para cada batch do conjunto de treino:**
 - Para cada exemplo do **batch**:
 - Fazer *forward pass* do exemplo
 - Calcular a *loss*
 - Utilizando o *backpropagation* calcular o gradiente da *loss* em relação aos parâmetros
 - Atualizar o valor de cada parâmetro seguindo o *learning rate*



Mais redes neurais

- Chamamos esse tipo de rede que vimos de *feedforward neural network* ou *multilayer perceptron (MLP)*
- Rede neural convolucional (ConvNets)
 - Faz uso de operações de convolução
 - Utilizada para imagens
 - Também é feedforward
- Rede recorrente
 - Utilizada para modelar sequências
 - Utilizada em textos, por exemplo.



Na prática

- Sci-kit learn possui implementação do MLP
- Importante normalizar os dados
- Aplicar one-hot encoding em features categóricas
- Nunca use um learning rate fixo, uma boa ideia inicial é fazê-lo decair com o número da epoch



Na prática

- Pytorch e TensorFlow, possuem suporte para GPU, Automatic Differentiation (AD) e muitas camadas mais complexas
- Keras é um framework que encapsula o TensorFlow facilitando o uso
- Para usar outras funções de ativação é melhor usar um biblioteca com AD
- Para problemas com muitos dados é bom usar uma biblioteca que suporta aceleradores como GPU (pode ficar de 10 a 30 vezes mais rápido)



Na prática

- Redes neurais brilham em problemas com dados não estruturados, em geral são a melhor solução para esse tipo de problema
- Para dados tabelados também conseguem bons resultados, mas para redes profundas vão requerer bastante cuidado na escolha de hiperparâmetros
- Melhores para problemas com muitos dados



Uma aula é muito pouco

- Redes neurais e Deep Learning dão facilmente múltiplos cursos
- Venha para nossa frente de Deep Learning!
- Recomendo para quem se interessou pelo assunto procurar mais a respeito.

Seguem algumas recomendações:

- Mais sobre o gradient descent (momentum, ...)
- Outros otimizadores (RMSProp, Adam, ...)
- Inicialização de parâmetros
- Regularização dos parâmetros (L1, L2, Dropout)
- Otimização de hiperparâmetros



Referências

[\[1\]](#) - Neural networks class - Université de Sherbrooke by Hugo Larochelle

[\[2\]](#) - Stanford CS231n - Convolutional Neural Networks for Visual Recognition

[\[3\]](#) - Columbia W4995 - Applied Machine Learning

