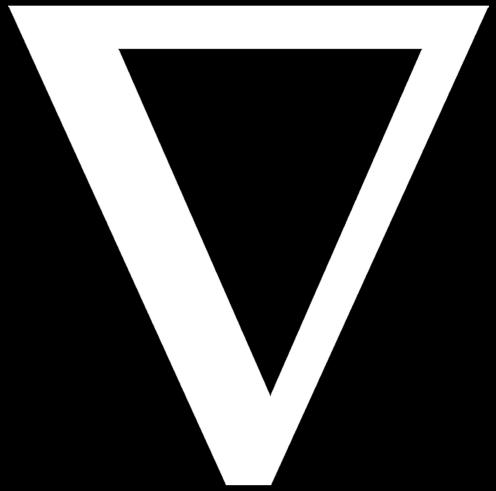


Introdução às Redes Neurais e Deep Learning

João G.M. Araújo • 09/06/2021
@_joaogui1

O que são?

Visão Geral

- Composta de blocos modulares
  - Parametrizados
  - Em geral diferenciáveis
  - Podem ser conectados de várias maneiras
  - G(x; W, b) = Wx + b
- Funções de ativação após os blocos
  - Não lineares
  - Evitam colapso

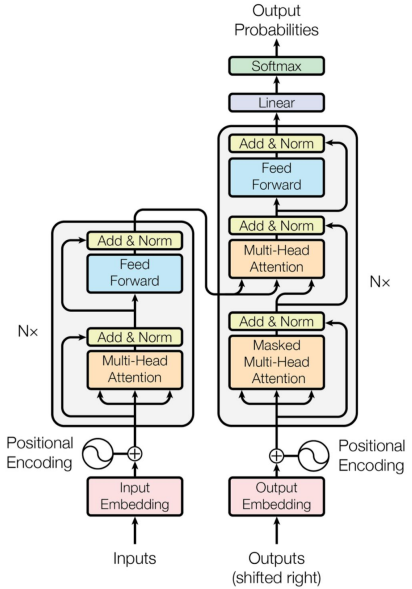
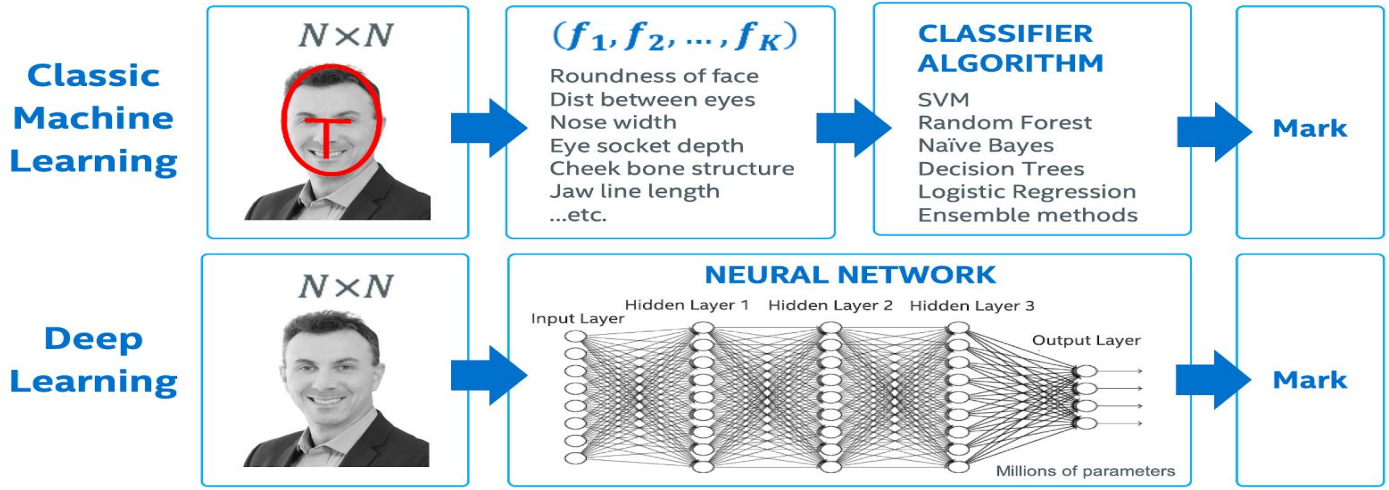


Figure 1: The Transformer - model architecture.

Deep Learning vs ML “clássica”

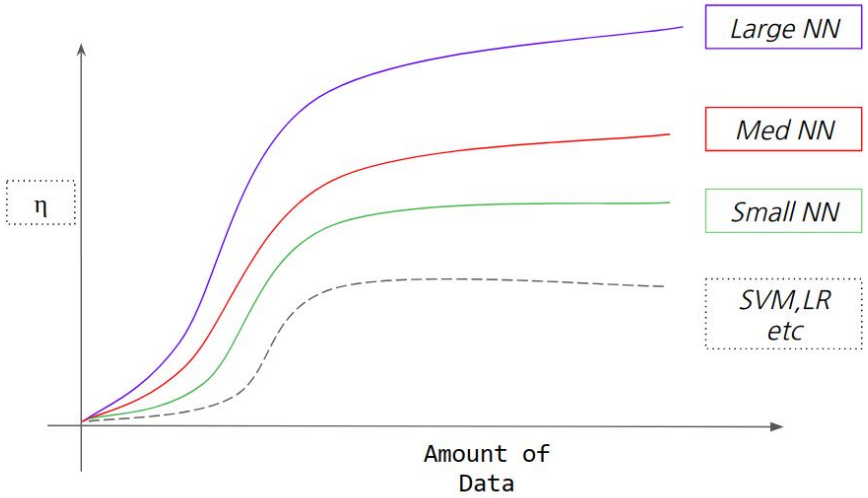
- O workflow padrão de ML é
  - Gerar features boas
  - Usar extratores de features interessantes
    - Os kernels do Hiro para imagem
    - TF-IDF e afins para texto
  - Treinar um classificador em cima dessas features extraídas
- O workflow padrão de DL é
  - Encontrar uma boa representação inicial para os dados
  - Treinar uma rede neural em cima dessa representação inicial

Deep Learning vs ML “clássica”

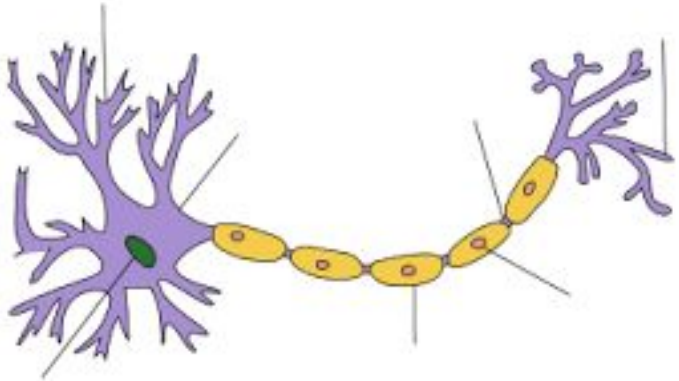


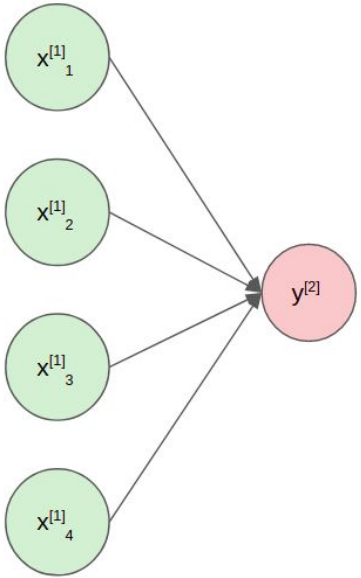
Fonte: Intel (https://www.intel.com.br/content/www/br/pt/artificial-intelligence/posts/difference-between-ai-machine-learning-deep-learning.html)

Deep Learning vs ML “clássica”



Intuição Biológica

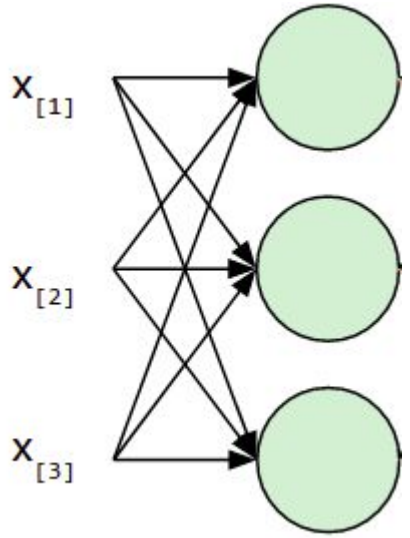




Componentes Principais

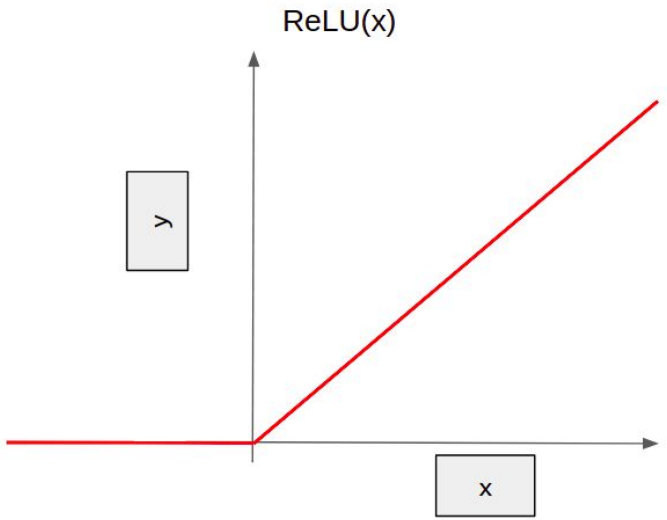
Camada Densa/Linear

- Camada mais simples
- Combinamos vários neurônios
    - Em vez de uma multiplicação de vetores temos uma multiplicação por matriz
- Bem otimizável por GPUs
- Muito custosa
- “Sem” viés indutivo



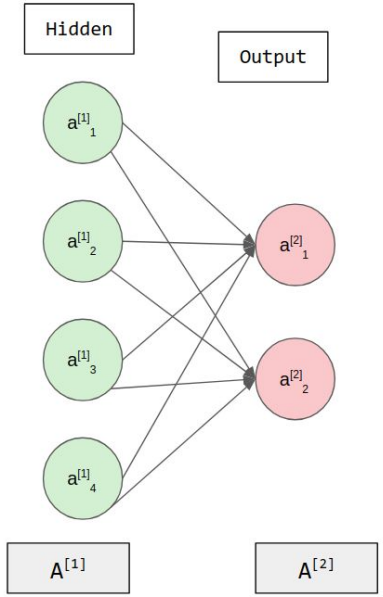
ReLU

- Função de ativação mais usada
- max(0, x)
- Inspiração biológica
  - Inativa/excitada
- Na verdade vem de um soma de sigmóides
- Não ótima, mas rápida



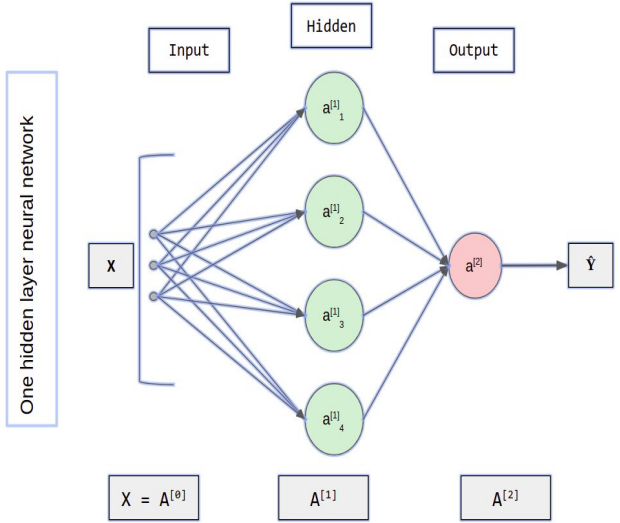
Rede Neural de Uma Camada

- Basicamente uma regressão
- Em certo sentido já generaliza vários modelos
    - Dependendo da função de ativação e perda



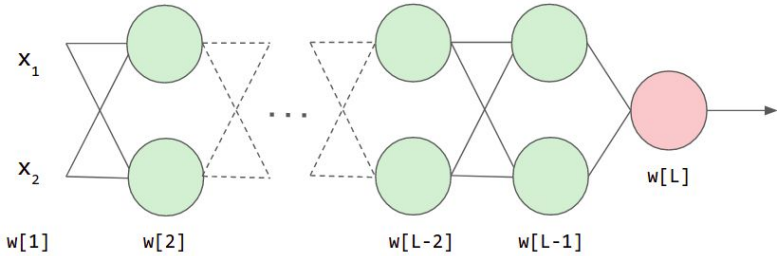
Rede Neural de Duas Camadas

- Função de ativação entre as camadas
- Aprendizado de representações
- Aproximação Universal
    - Aprox != Representar



Redes Profundas

- Aprendizado de representações
- Aproximador universal
- Menos parâmetros

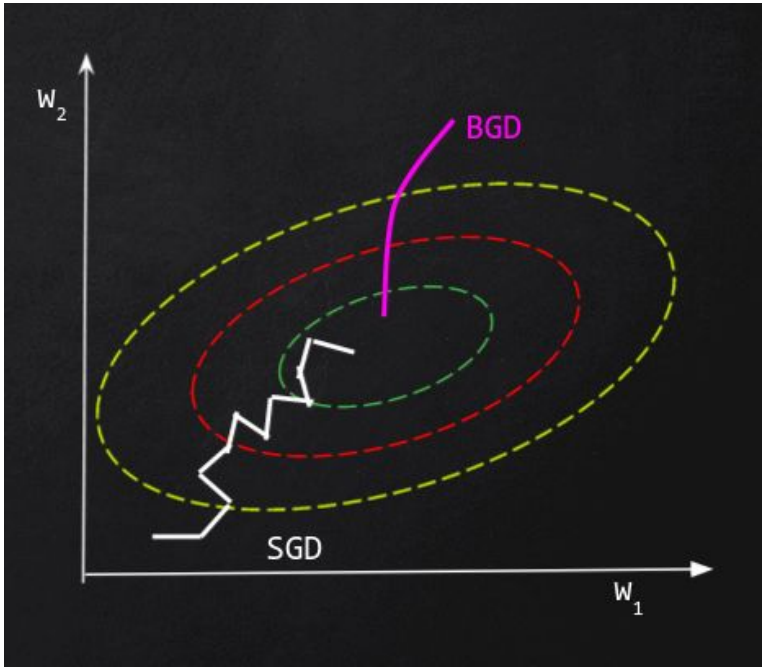


dúvidas?

Treinamento

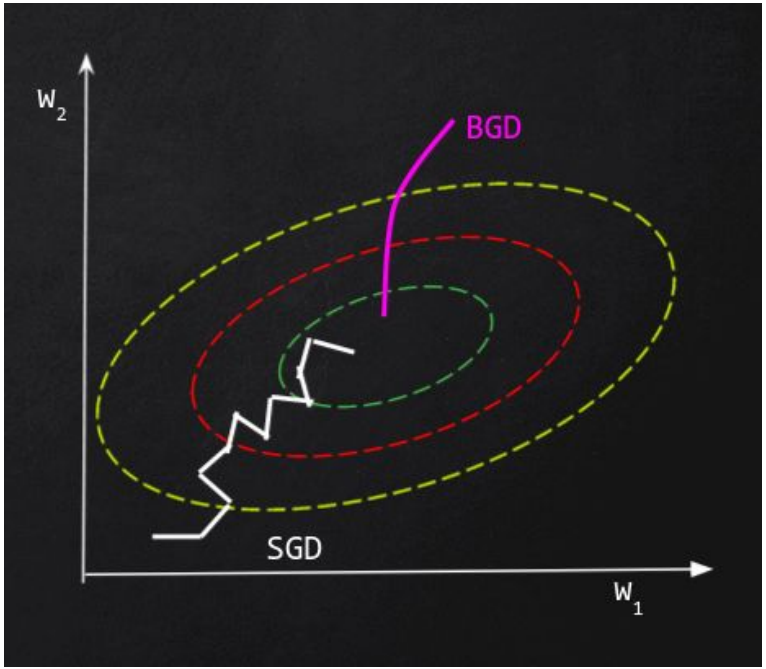
Visão geral

- Semelhante às regressões
  - Definimos uma função de perda
  - Otimizamos ela iterativamente
- Usamos derivados do SGD
  - Momentum
  - Adam
- Backpropagation
  - Computa o gradiente



Relembrando o SGD

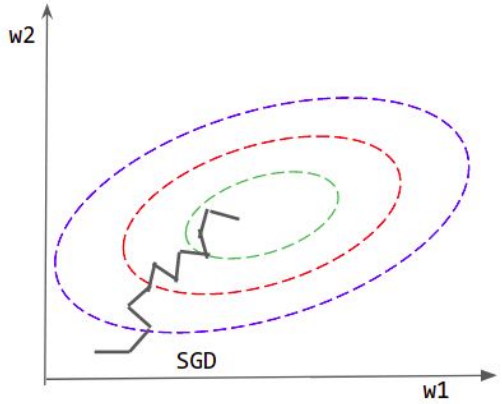
- Pegamos um batch/subconjunto de dados
- Geramos a previsão
- Calculamos o gradiente
- Update
  - $W_{t+1} = W_t - \alpha \nabla_w l(W_t, x_t)$
- Repetimos até convergir

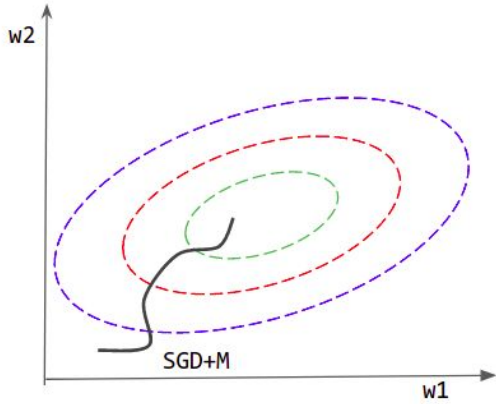


Momentum

- SGD é muito instável
  - Reduz os “zig zags”
  - Permite o uso de um lr maior
- Clássico

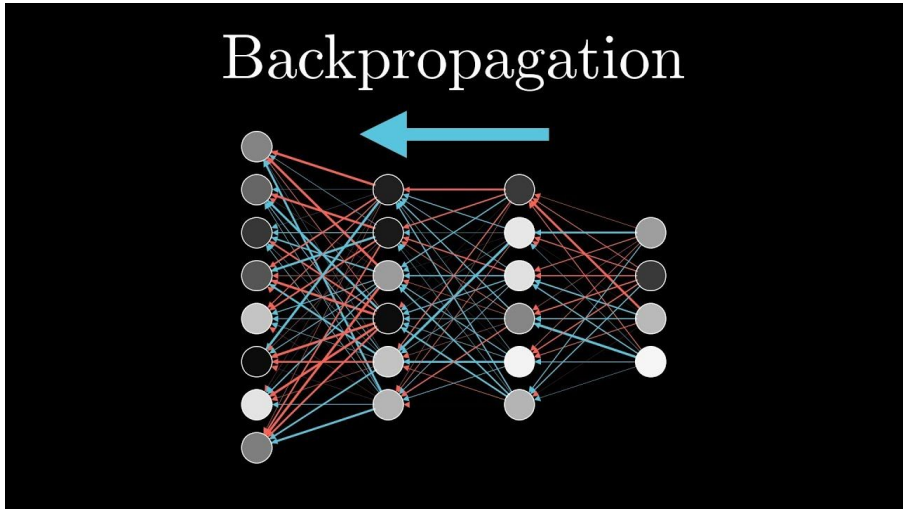
$$p_{k+1} = \beta_k p_k + \nabla f_i(w_k)$$
$$w_{k+1} = w_k - \gamma_k p_{k+1}$$
$$w_{k+1} = w_k - \gamma_k \nabla f_i(w_k) + \beta_k (w_k - w_{k-1})$$





Backpropagation

- Calcular o gradiente da perda em função dos parâmetros da rede
- $f(x, w) = f_n(f_{n-1}(...(f_1(x, w_1)..., w_{n-1})w_n)$
  - A rede neural é uma composição de funções
  - Regra da cadeia!



dúvidas?

JAX!!!onze!

Numpy + Aceleradores + Autodiff

- JAX implementa uma API extremamente semelhante a numpy
- Usa transformações de funções
  - jit acelera as funções com CPU/GPU/TPU
  - grad computa o gradiente da função

Rede Neural em JAX

Link pro colab: https://bit.ly/3weNxN2

dúvidas?

TensorFlow

FLAX

PyTorch

Axon

flux

Referências

- Minha playlist de Deep Learning