(1) Introdução ao Aprendizado Profundo

Redes Neurais e Arquiteturas Profundas

Moacir Antonelli Ponti
CeMEAI/ICMC, Universidade de São Paulo
MBA em Ciência de Dados

www.icmc.usp.br/~moacir — moacir@icmc.usp.br

São Carlos-SP/Brasil – 2020

Agenda

Uma tarefa de classificação

Mudando o pipeline do aprendizado

Machine vs Deep Learning

Rede neural: do raso ao profundo

- Perceptron e Multilayer Perceptron

Tarefa: aprender a distinguir dois tipos de imagens

- **deserto/desert**;
- **praia/beach**.

Aprendizado supervisionado: dadas imagens anotadas (rotuladas) gere um modelo que seja capaz de classificar imagens desconhecidas (não vistas) em uma dessas duas classes.

Tarefa: aprender a distinguir dois tipos de imagens

- **deserto/desert**;
- **praia/beach**.

Aprendizado supervisionado: dadas imagens anotadas (rotuladas) gere um modelo que seja capaz de classificar imagens desconhecidas (não vistas) em uma dessas duas classes.

Tarefa: aprender a distinguir dois tipos de imagens

- **deserto/desert**;
- **praia/beach**.

Aprendizado supervisionado: dadas imagens anotadas (rotuladas) gere um modelo que seja capaz de classificar imagens desconhecidas (não vistas) em uma dessas duas classes.

Uma tarefa de classificação

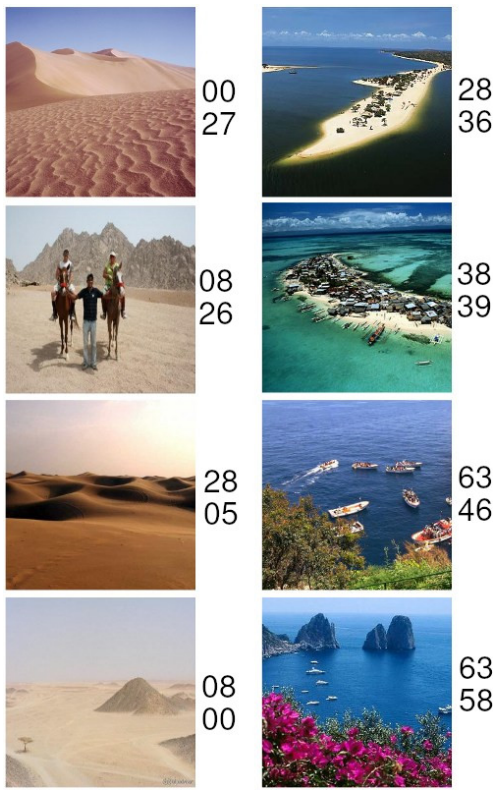
- **Atributos**: necessários para permitir medir a (dis)similaridade entre exemplos (imagens)

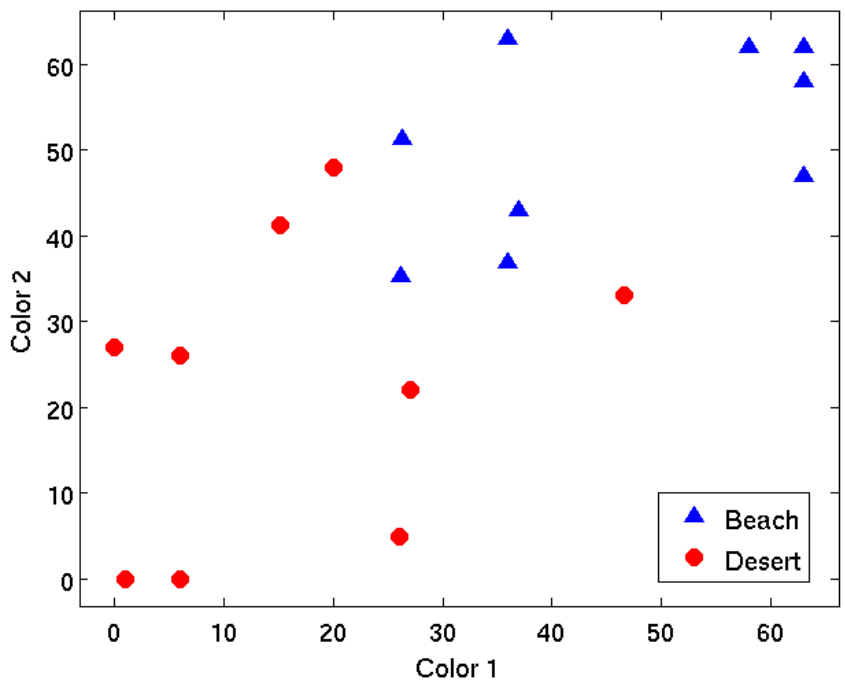
Uma tarefa de classificação

- **Atributos**: necessários para permitir medir a (dis)similaridade entre exemplos (imagens)
  - podemos usar os próprios pixels mas há desvantagens

Uma tarefa de classificação

- **Atributos**: necessários para permitir medir a (dis)similaridade entre exemplos (imagens)
  - podemos usar os próprios pixels mas há desvantagens
  - vamos codificar as cores em 64 valores e representar as imagens por dois valores relativos as duas cores mais frequentes





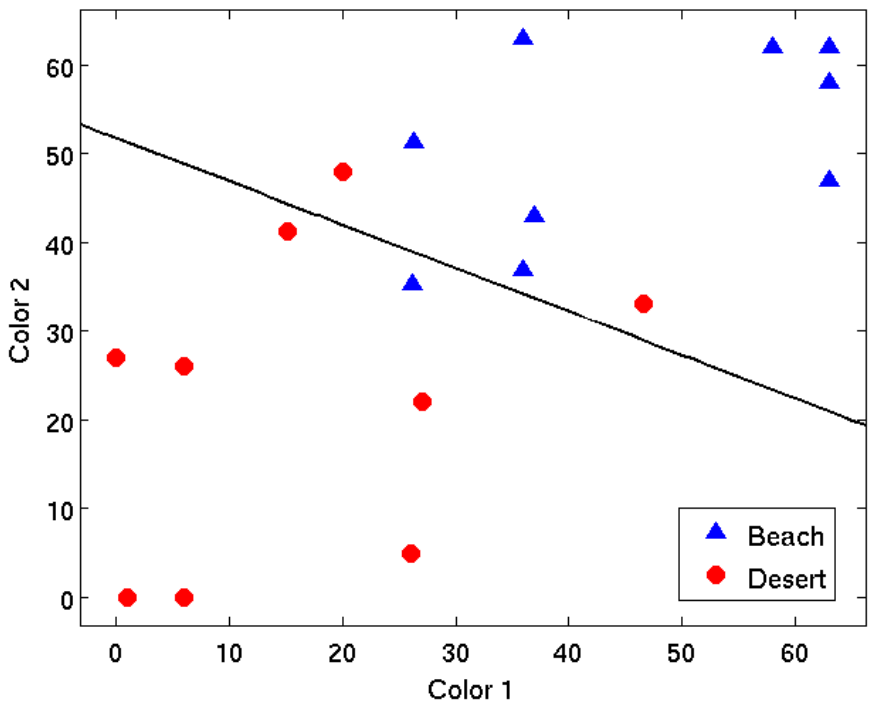
Uma tarefa de classificação

- **Classificador**: é o modelo (e não o algoritmo) criado a partir de um conjunto de dados anotado.
- ... deve permitir, com base na representação de uma nova imagem, classificá-la

,

Uma tarefa de classificação

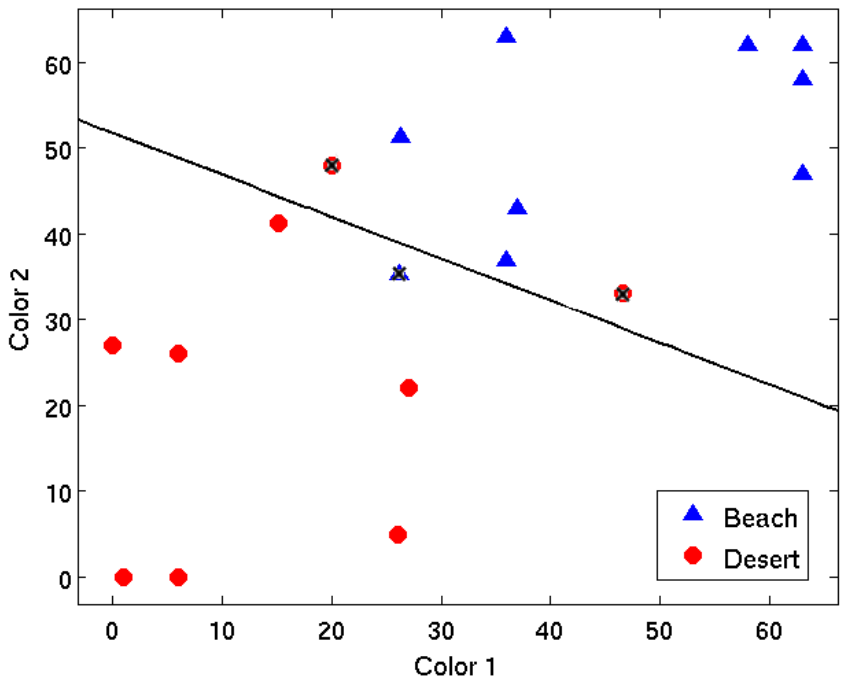
- **Classificador**: é o modelo (e não o algoritmo) criado a partir de um conjunto de dados anotado.
- ... deve permitir, com base na representação de uma nova imagem, classificá-la
  - usando a navalha de Occam: um classificador linear



,

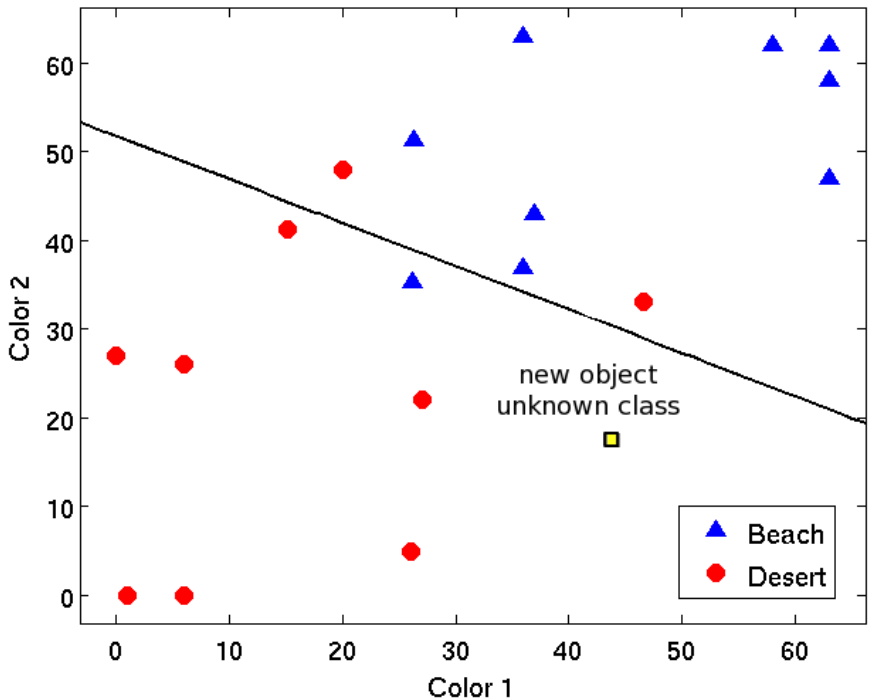
Uma tarefa de classificação

- Exemplos usados para obter o modelo/classificador: **conjunto de treinamento**.
- Dados de treinamento comumente não são completamente separáveis
- O erro do classificador nesses dados é o erro de treinamento.



Uma tarefa de classificação

- The model, or **classifier**, can then be used to predict/infer the class of **new data**.

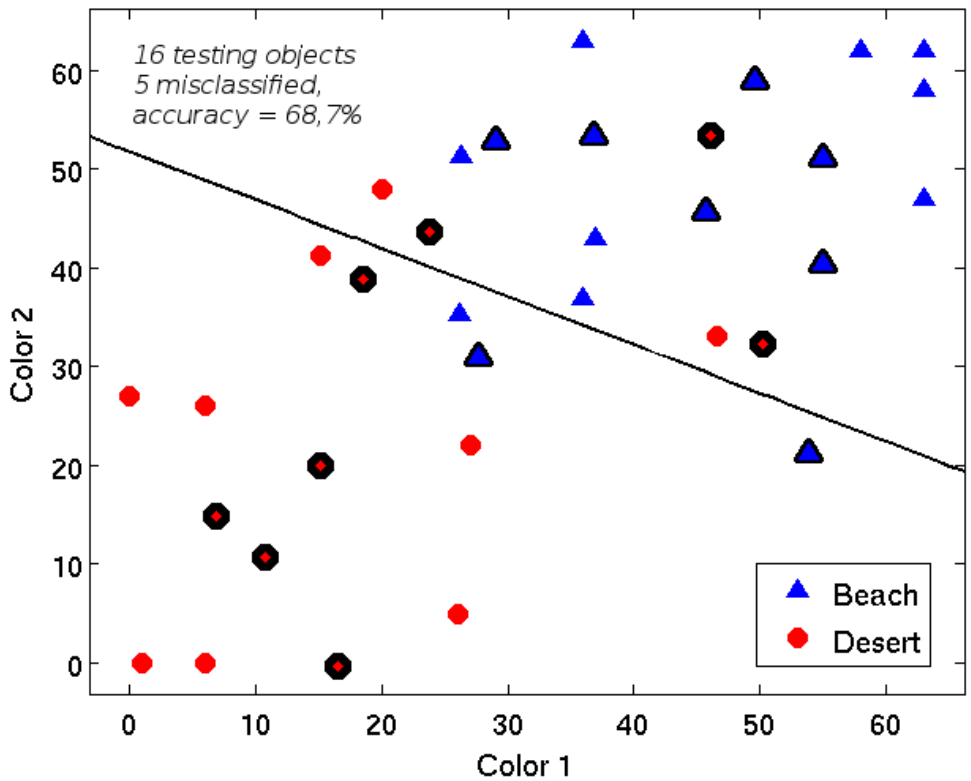


Uma tarefa de classificação

- Como saber quão bom é um modelo? Apenas testando em dados não vistos/desconhecidos pelo classificador

Uma tarefa de classificação

- Como saber quão bom é um modelo? Apenas testando em dados não vistos/desconhecidos pelo classificador
- ... **conjunto de teste**.



Agenda

Uma tarefa de classificação

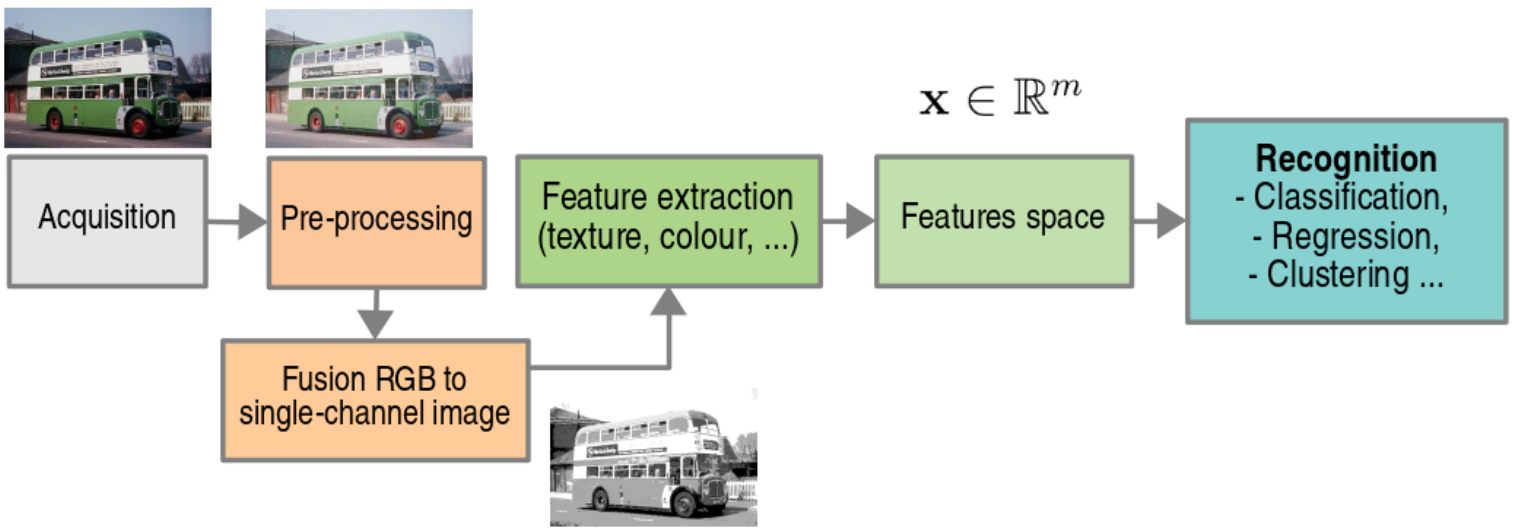
Mudando o pipeline do aprendizado

Machine vs Deep Learning

Rede neural: do raso ao profundo

Perceptron e Multilayer Perceptron

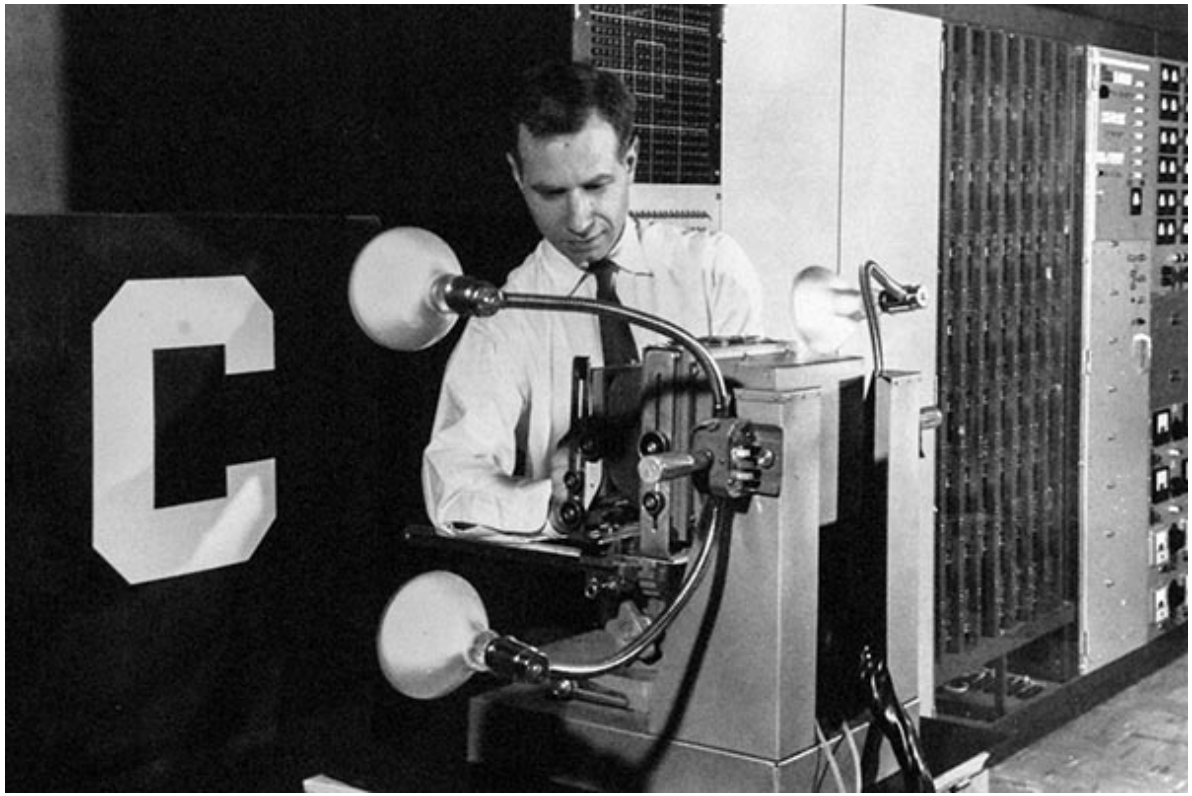
Pipeline de classificação de imagens ($\sim$ 1970 – 2010)



... mas e as redes neurais?

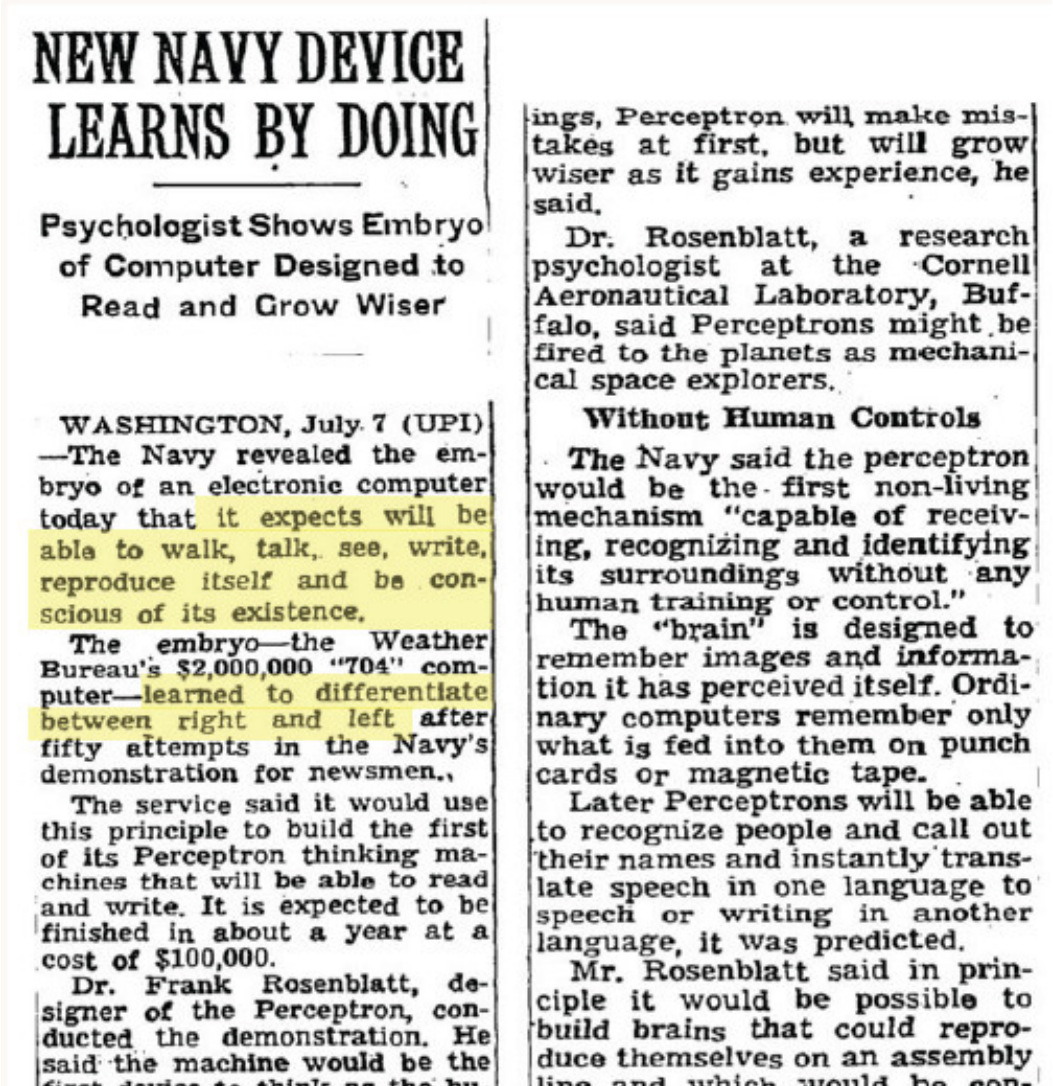
Primavera das redes neurais

- 1958: Frank Rosenblatt propõe o Perceptron como um **modelo conexionista** bioinspirado;



Primavera das redes neurais

- The New York Times, 1958 : "... o embrião de um computador ... que será capaz de andar, falar, ver, escrever, se reproduzir e ser consciente de sua existência."



NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

WASHINGTON, July 7 (UPI) —The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

The embryo—the Weather Bureau's $2,000,000 "704" computer—learned to differentiate between right and left after fifty attempts in the Navy's demonstration for newsmen.

The service said it would use this principle to build the first of its Perceptron thinking machines that will be able to read and write. It is expected to be finished in about a year at a cost of $100,000.

Dr. Frank Rosenblatt, designer of the Perceptron, conducted the demonstration. He said the machine would be the

ings, Perceptron will make mistakes at first, but will grow wiser as it gains experience, he said.

Dr. Rosenblatt, a research psychologist at the Cornell Aeronautical Laboratory, Buffalo, said Perceptrons might be fired to the planets as mechanical space explorers.

Without Human Controls

The Navy said the perceptron would be the first non-living mechanism "capable of receiving, recognizing and identifying its surroundings without any human training or control."

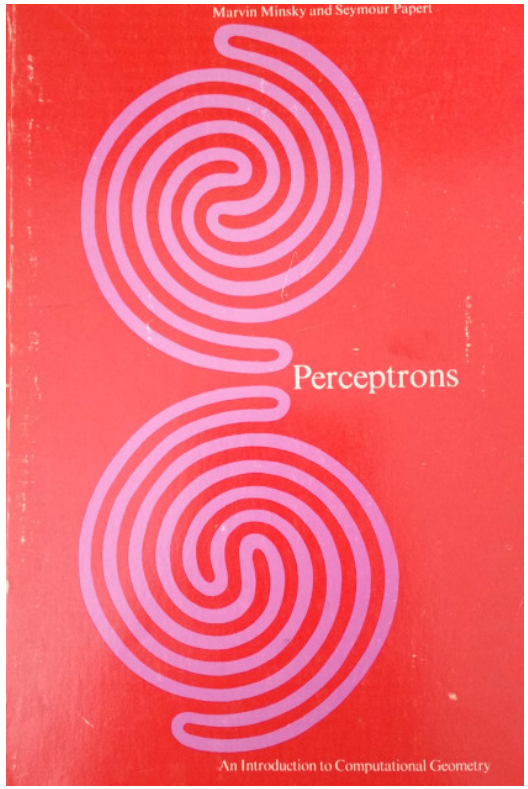
The "brain" is designed to remember images and information it has perceived itself. Ordinary computers remember only what is fed into them on punch cards or magnetic tape.

Later Perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech or writing in another language, it was predicted.

Mr. Rosenblatt said in principle it would be possible to build brains that could reproduce themselves on an assembly

Primavera das redes neurais

- Limitações em treinar o Perceptron original foram encontradas
  - Minsky e Papert (MIT). Perceptrons, 1969



Inverno

- de 1970 até 2010 grande parte da pesquisa em "Inteligência Artificial" se referia a criar sistemas especialistas
- havia maior atenção para métodos com embasamento teórico e garantias matemáticas (como SVM):

Inverno

- de 1970 até 2010 grande parte da pesquisa em "Inteligência Artificial" se referia a criar sistemas especialistas
- havia maior atenção para métodos com embasamento teórico e garantias matemáticas (como SVM):
  - "espaços de atributos/características"

Inverno

- de 1970 até 2010 grande parte da pesquisa em "Inteligência Artificial" se referia a criar sistemas especialistas
- havia maior atenção para métodos com embasamento teórico e garantias matemáticas (como SVM):
  - "espaços de atributos/características"
  - "generalização": confiar no treinamento de um modelo;

Inverno

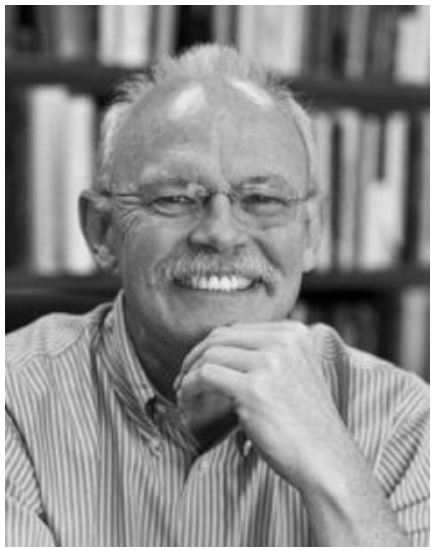
- de 1970 até 2010 grande parte da pesquisa em "Inteligência Artificial" se referia a criar sistemas especialistas
- havia maior atenção para métodos com embasamento teórico e garantias matemáticas (como SVM):
  - "espaços de atributos/características"
  - "generalização": confiar no treinamento de um modelo;
  - "viés" ou "complexidade": reduzir a complexidade do modelo aumenta as garantias de aprendizado;

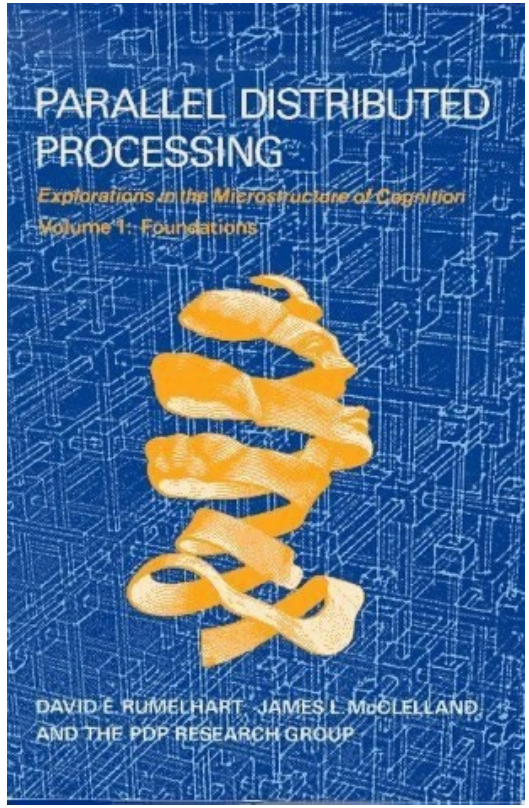
Inverno

- de 1970 até 2010 grande parte da pesquisa em "Inteligência Artificial" se referia a criar sistemas especialistas
- havia maior atenção para métodos com embasamento teórico e garantias matemáticas (como SVM):
  - "espaços de atributos/características"
  - "generalização": confiar no treinamento de um modelo;
  - "viés" ou "complexidade": reduzir a complexidade do modelo aumenta as garantias de aprendizado;
  - "tamanho da amostra" (Lei dos Grandes números): o estimador se aproxima do valor esperado conforme aumenta a quantidade de exemplos anotados.

Nova primavera

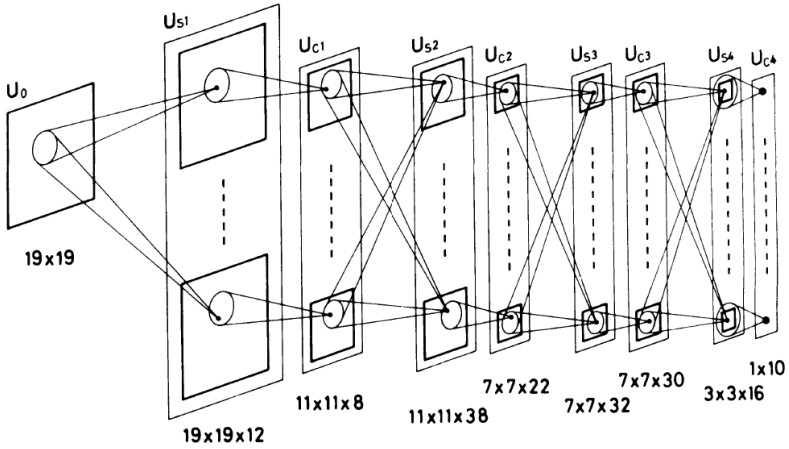
- Pesquisadores que trabalharam no inverno (1980's).
  - Rumelhart e McClelland (1986)



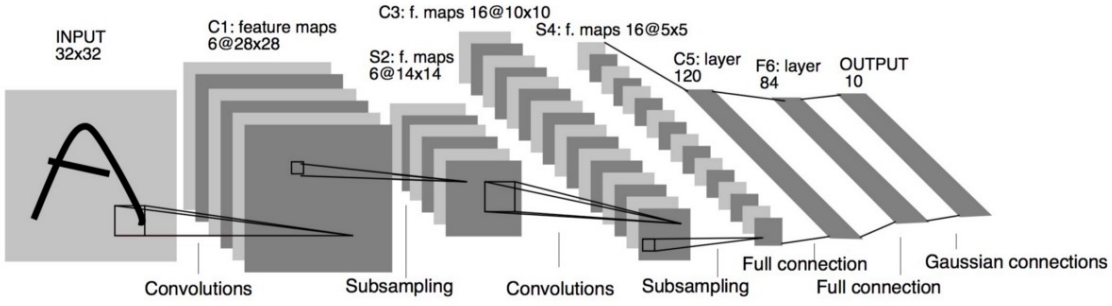


Trabalhos precursores do aprendizado profundo...

Fukushima's Neocognitron (1989)

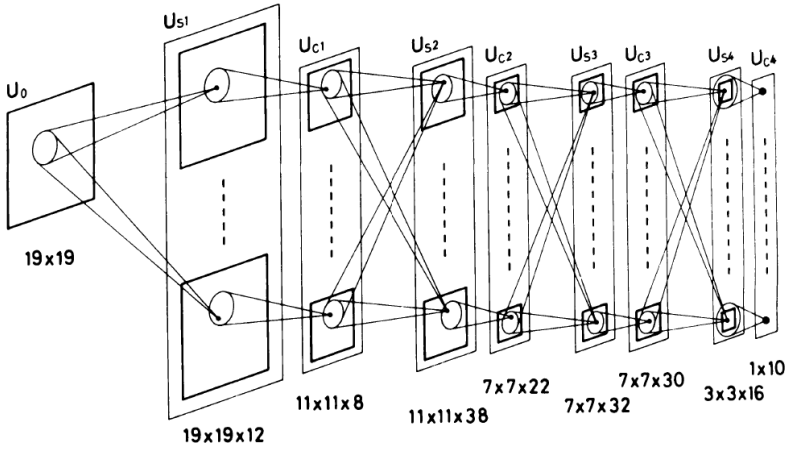


LeCun's LeNet (1998)

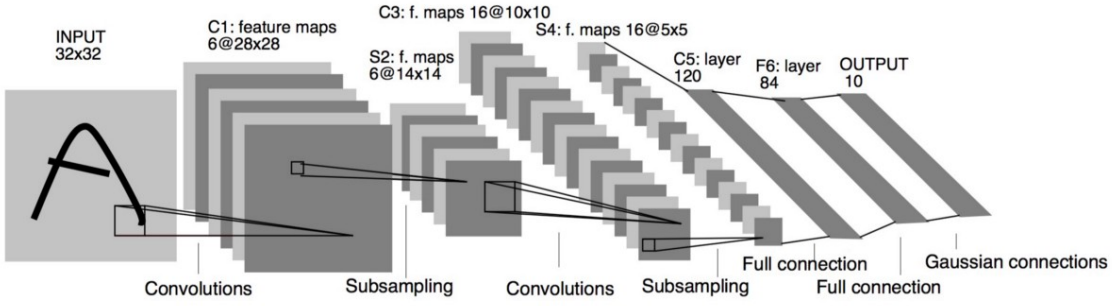


Trabalhos precursores do aprendizado profundo...

Fukushima's Neocognitron (1989)



LeCun's LeNet (1998)

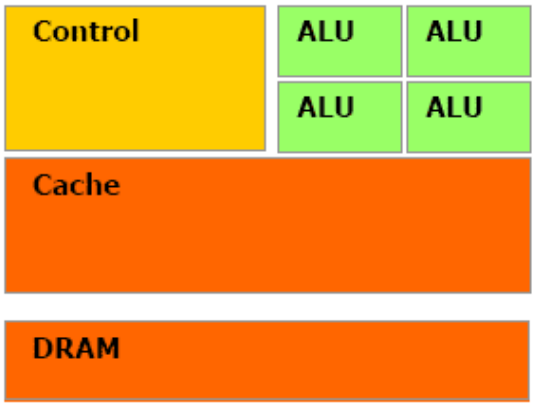


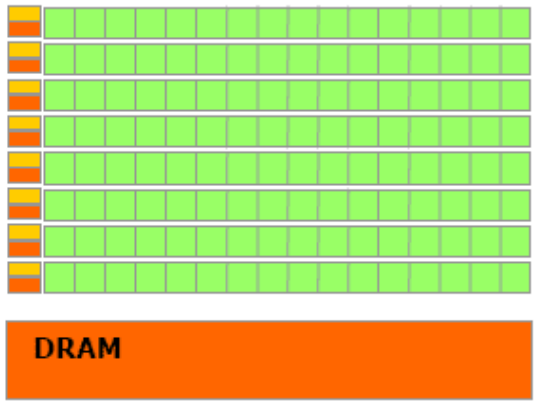
- ... e então uma nova primavera surgiu.

Razão 1: disponibilidade de dados anotados

ImageNet Challenge: $\sim$ 1.4 milhões de imagens, 1000 classes.

Razão 2: poder de processamento gráfico paralelo





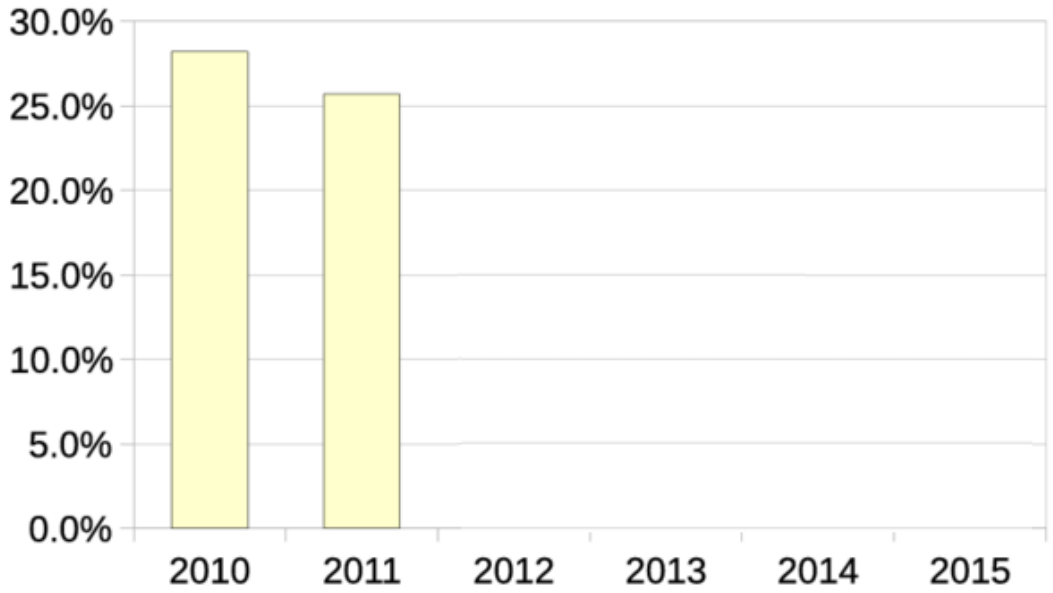
CPU

Baixa densidade de processamento
Controle complexo
Maior tamanho de cache
Baixa tolerância a latência

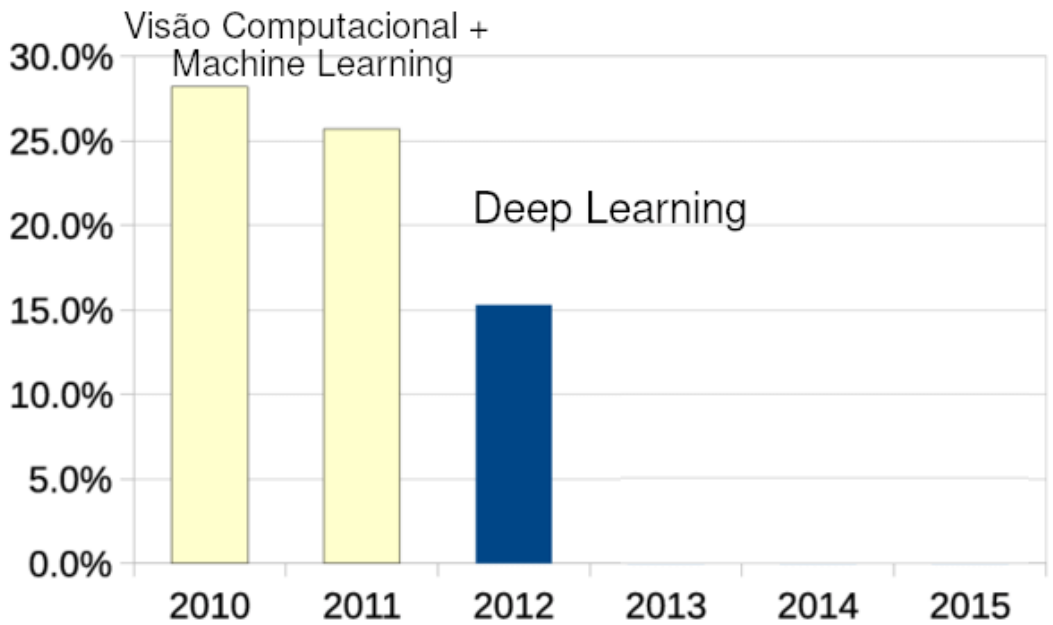
GPU

Alta densidade de processamento
Controle simples
Cache pequeno
Alta tolerância a latência

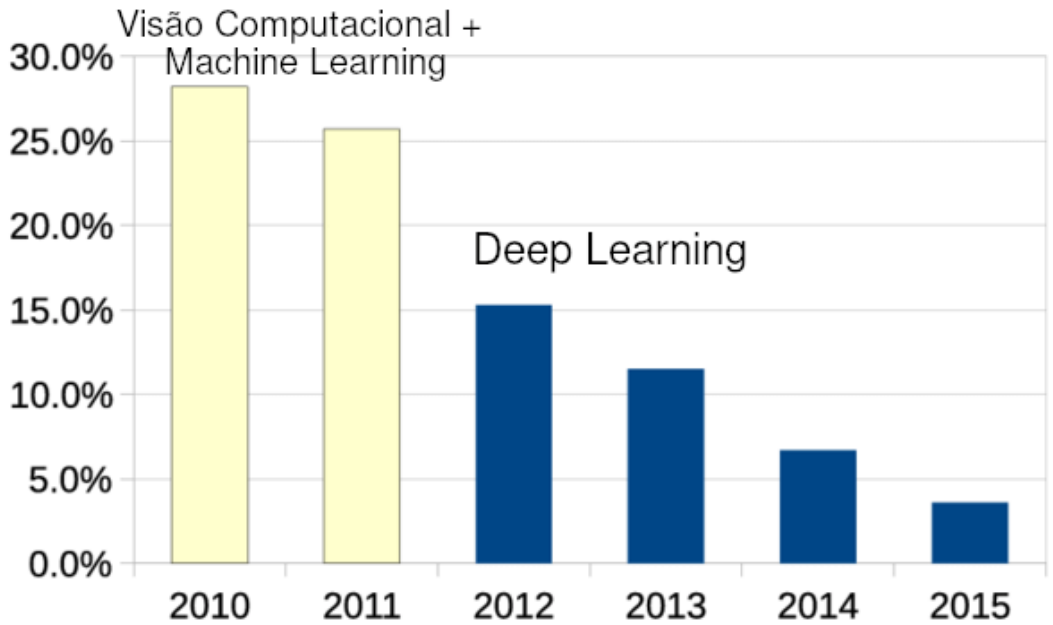
Resultados do desafio (erro de classificação top-5)



Resultados do desafio (erro de classificação top-5)

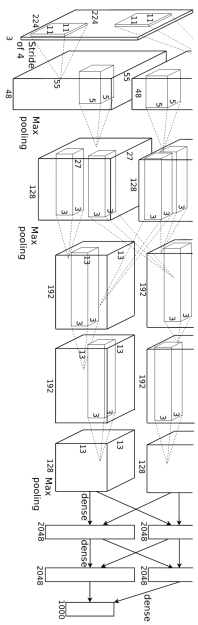


Resultados do desafio (erro de classificação top-5)



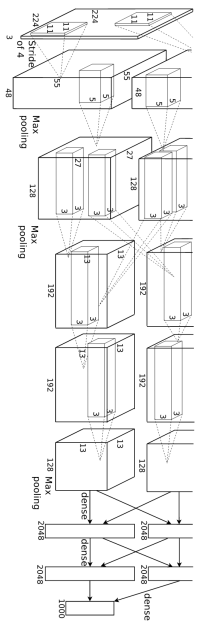
Redes neurais convolucionais profundas (e suas camadas)

AlexNet (9)

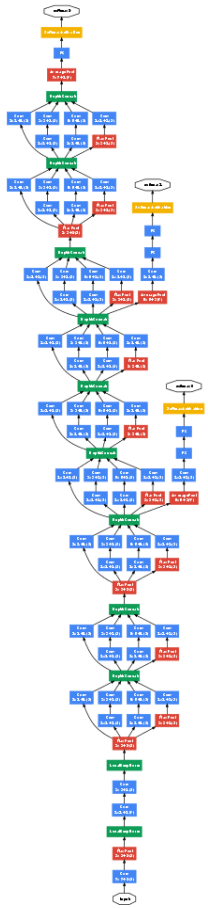


Redes neurais convolucionais profundas (e suas camadas)

AlexNet (9)

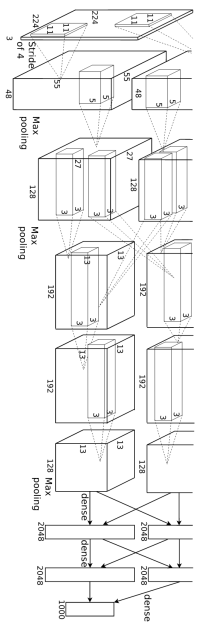


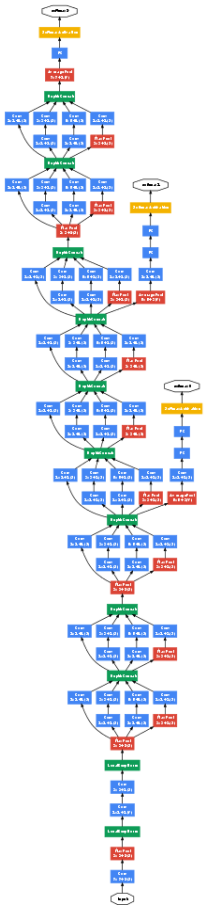
Inception (22)

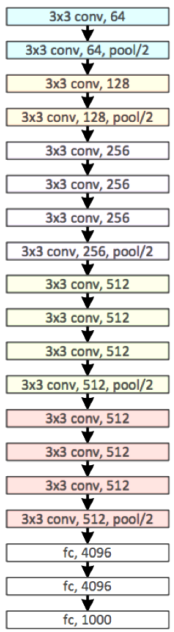


Redes neurais convolucionais profundas (e suas camadas)

AlexNet (9) Inception (22) VGGNet (16/19)

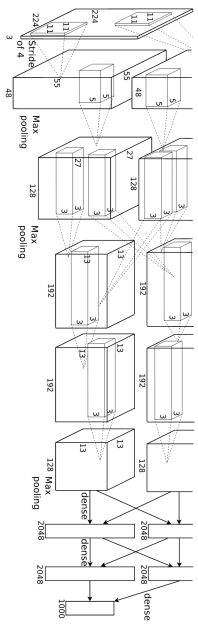


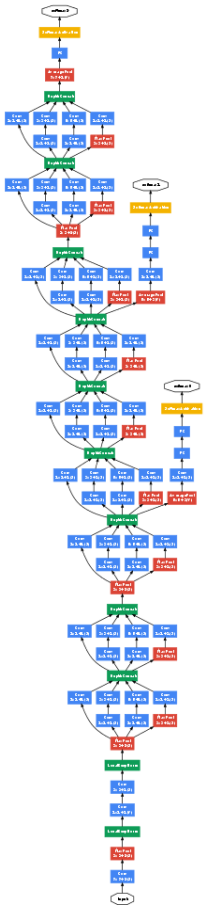


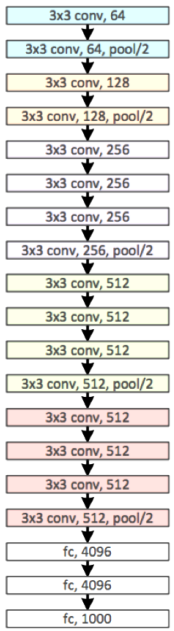


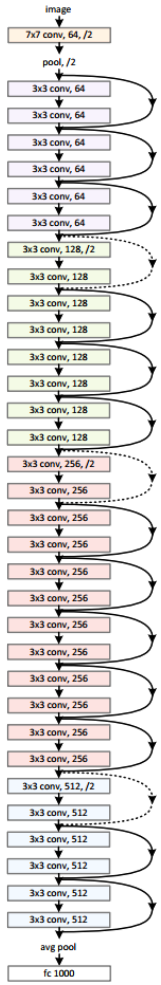
Redes neurais convolucionais profundas (e suas camadas)

AlexNet (9) Inception (22) VGGNet (16/19) ResNet (34-1000)





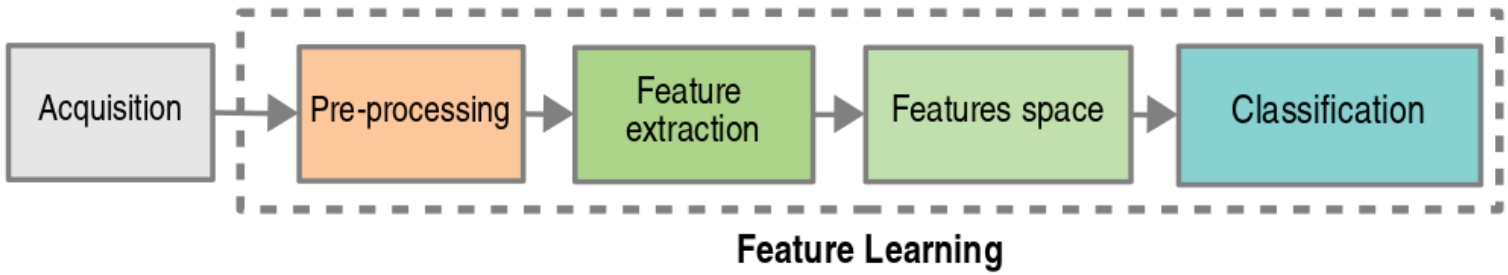




Novo pipeline de reconhecimento: aprendizado profundo



$\mathbf{x} \in \mathbb{R}^m$



Primavera das redes neurais

- Turing Award, 2018

Agenda

Uma tarefa de classificação

Mudando o pipeline do aprendizado

Machine vs Deep Learning

Rede neural: do raso ao profundo

Perceptron e Multilayer Perceptron

Machine learning: dois exemplos

Precisamos inferir uma função $f(\mathbf{x}) = \mathbf{y}$

— o significado de f, $\mathbf{x}$ e $\mathbf{y}$ dependem da tarefa

Machine learning: dois exemplos

Precisamos inferir uma função $f(\mathbf{x}) = \mathbf{y}$
— o significado de f, $\mathbf{x}$ e $\mathbf{y}$ dependem da tarefa

1 – classificação de imagens de paisagens

- Dados disponíveis: pares (imagens, rótulos) obtidas de desertos e praias,
- Entrada: pixels da imagem organizados na forma $\mathbf{x}$,
- Saída: rótulo y (e.g. praia) atribuído à imagem de entrada.

Machine learning: dois exemplos

2 – predição de fraude em transação de cartão de crédito

- Dados disponíveis: transações legítimas de um cliente,
- Entrada: dados incluindo: localização, moeda, valor, data e hora, na forma $\mathbf{x}$,
- Saída: probabilidade y de observar uma transação fraudulenta (anômala).

Machine Learning (ML) vs Deep Learning (DL)

Machine Learning

Uma área mais geral que inclui DL.

Algoritmos comumente aprendem uma função $f : X \to Y$, a partir de um espaço de funções admissíveis f e dados de treinamento

- métodos rasos ("shallow") comumente inferem uma única $f(.)$. e.g. uma função linear $f(x) = w \cdot x + b$,
  - aprendizado de máquina seria ajustar os valores para w e b
  - examplos: Perceptron, Support Vector Machines (SVM), Logistic Regression Classifier, Linear Discriminant Analysis (LDA).

Machine Learning (ML) vs Deep Learning (DL)

Deep Learning

Envolve aprender uma representações, aprendidas de forma hierárquica por funções compostas.
Por exemplo, dada uma entrada $\mathbf{x}_1$ produzir diversas representações intermediárias:

$$\mathbf{x}_2 = f_1(\mathbf{x}_1)$$
$$\mathbf{x}_3 = f_2(\mathbf{x}_2)$$
$$\mathbf{x}_4 = f_3(\mathbf{x}_3)$$
$$\dots$$

A saída é obtida pelo aninhamento de L funções:

$$f_L\left(\cdots f_3(f_2(f_1(\mathbf{x}_1, \Theta_1), \Theta_2), \Theta_3) \cdots, \Theta_L\right),$$

Θ_i são os parâmetros associados a cada função i.

Agenda

Uma tarefa de classificação

Mudando o pipeline do aprendizado

Machine vs Deep Learning

Rede neural: do raso ao profundo

- Perceptron e Multilayer Perceptron

Montando um classificador raso

Entrada  $\rightarrow \mathbf{x}$

Seja Θ uma matriz W de pesos e um vetor $\mathbf{b}$ de termos "bias"

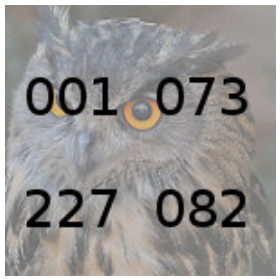
matriz de pesos | imagem | bias

$$f(\Theta, \mathbf{x}) = W \quad \mathbf{x} \quad + \quad \mathbf{b}$$

$$= \text{ scores para possíveis classes de } \mathbf{x}$$

Montando um classificador raso

- Entrada: imagem (com $N \times M \times 3$ pixels) vetorizada em $\mathbf{x}$
- Classes: gato, tartaruga, coruja
- Saída: scores para cada classe



$= \mathbf{x} = [1, 73, 227, 82]$

saída $f(\Theta, \mathbf{x}) = s \quad \rightarrow$ 3 números com os scores das classes

Montando um classificador raso

- Entrada: imagem (com $N \times M \times 3$ pixels) vetorizada em $\mathbf{x}$
- Classes: gato, tartaruga, coruja
- Saída: scores para cada classe

$= \mathbf{x} = [1, 73, 227, 82]$

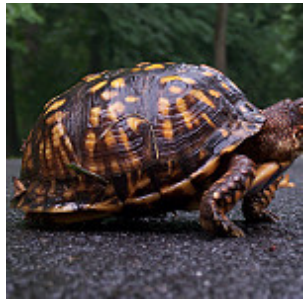
saída $f(\Theta, \mathbf{x}) = s \quad \rightarrow$ 3 números com os scores das classes

$$W\mathbf{x} + \mathbf{b}$$

$$\begin{bmatrix} 0.1 & -0.25 & 0.1 & 2.5 \\ 0 & 0.5 & 0.2 & -0.6 \\ 2 & 0.8 & 1.8 & -0.1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 73 \\ 227 \\ 82 \end{bmatrix} + \begin{bmatrix} -2.0 \\ 1.7 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -37.3 \\ 6.3 \\ 38.6 \end{bmatrix}$$

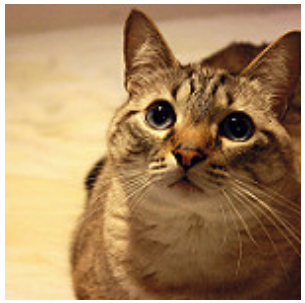
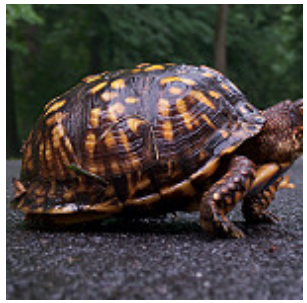
Montando um classificador raso: scores obtidos





gato	-37.3	**180.3**	18.6
coruja	6.3	90.3	**26.3**
tartaruga	**38.6**	17.6	21.8

Montando um classificador raso: scores obtidos

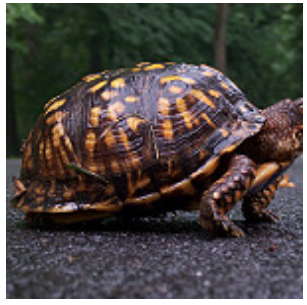




gato	-37.3	**180.3**	18.6
coruja	6.3	90.3	**26.3**
tartaruga	**38.6**	17.6	21.8

O que precisamos para melhorar esse modelo?

Montando um classificador raso: scores obtidos





gato	-37.3	**180.3**	18.6
coruja	6.3	90.3	**26.3**
tartaruga	**38.6**	17.6	21.8

O que precisamos para melhorar esse modelo?

- uma forma de quantificar o **custo** de escolher o modelo atual
- um **algoritmo de otimização** para ajustar Θ de forma a minimizar o custo.

Montando um classificador raso: otimização

- Queremos otimizar uma função para selecionar o **melhor classificador**
- Essa função é chamada de perda (**loss**) ou custo (**cost**),

Estatisticamente, minimizar a perda esperada (**expected loss**:)

$$E\left[\mathcal{L}\left(f(\Theta, X\right), Y] = \int_{\mathbb{R}^{dX} \times \mathbb{R}^{dY}} \mathcal{L}\left(f(\Theta, X), Y\right) dP(X, Y)$$

Montando um classificador raso: otimização

- Queremos otimizar uma função para selecionar o **melhor classificador**
- Essa função é chamada de perda (**loss**) ou custo (**cost**),

Estatisticamente, minimizar a perda esperada (**expected loss**:)

$$E\left[\mathcal{L}\left(f(\Theta, X), Y\right] = \int_{\mathbb{R}^{dX} \times \mathbb{R}^{dY}} \mathcal{L}\left(f(\Theta, X), Y\right) dP(X, Y)\right.$$

$P(x, y)$ (que modela a relação entrada e saída) é **desconhecida**, então:

- seja (x_i, y_i) um exemplo de treinamento, e $f : x \rightarrow y$ uma função de classificação atual, com parâmetros Θ.

Montando um classificador raso: otimização

- Queremos otimizar uma função para selecionar o **melhor classificador**
- Essa função é chamada de perda (**loss**) ou custo (**cost**),

Estatisticamente, minimizar a perda esperada (**expected loss**:)

$$E\left[\mathcal{L}\left(f(\Theta, X\right), Y\right] = \int_{\mathbb{R}^{dX} \times \mathbb{R}^{dY}} \mathcal{L}\left(f(\Theta, X), Y\right) dP(X, Y)$$

$P(x, y)$ (que modela a relação entrada e saída) é **desconhecida**, então:

- seja (x_i, y_i) um exemplo de treinamento, e $f : x \rightarrow y$ uma função de classificação atual, com parâmetros Θ.
- a perda empírica (**empirical loss**) pode ser computada:

$$\ell\left(f(\Theta, \mathbf{x}_i), y_i\right)$$

Montando um classificador raso: otimização

Perdas empíricas $\mathcal{L}(f(\Theta, X, Y))$ computadas em N exemplos

Mean squared error (valores contínuos) / perda quadrática

$$\ell(f(\Theta, X), Y) = \frac{1}{N} \sum_{i=1}^{N} (\ \overbrace{\hat{y}_i}^{\text{estimado}} - \overbrace{y_i}^{\text{real}}\ )^2$$

Montando um classificador raso: otimização

Perdas empíricas $\mathcal{L}(f(\Theta, X, Y))$ computadas em N exemplos

Mean squared error (valores contínuos) / perda quadrática

$$\ell(f(\Theta, X), Y) = \frac{1}{N} \sum_{i=1}^{N} (\ \overbrace{\hat{y}_i}^{\text{estimado}} - \overbrace{y_i}^{\text{real}}\ )^2$$

Cross entropy (bits ou vetores de probabilidade) / entropia cruzada

$$\ell(f(\Theta, X), Y) = \frac{1}{N} \sum_{i=1}^{N} y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)$$

Minimizando a perda

Comumente usa-se a inclinação da função de perda considerando os parâmetros do modelo
Para cada direção j do espaço de parâmetros Θ

Minimizando a perda

Comumente usa-se a inclinação da função de perda considerando os parâmetros do modelo
Para cada direção j do espaço de parâmetros Θ

$$\frac{d\ell\left(f\left(w_j, \mathbf{x}_i\right)\right)}{dw_j} = \lim_{\delta \to 0} \frac{f\left(w_j + \delta, \mathbf{x}_i\right) - f\left(w_j, \mathbf{x}_i\right)}{\delta}$$

Minimizando a perda

Comumente usa-se a inclinação da função de perda considerando os parâmetros do modelo
Para cada direção j do espaço de parâmetros Θ

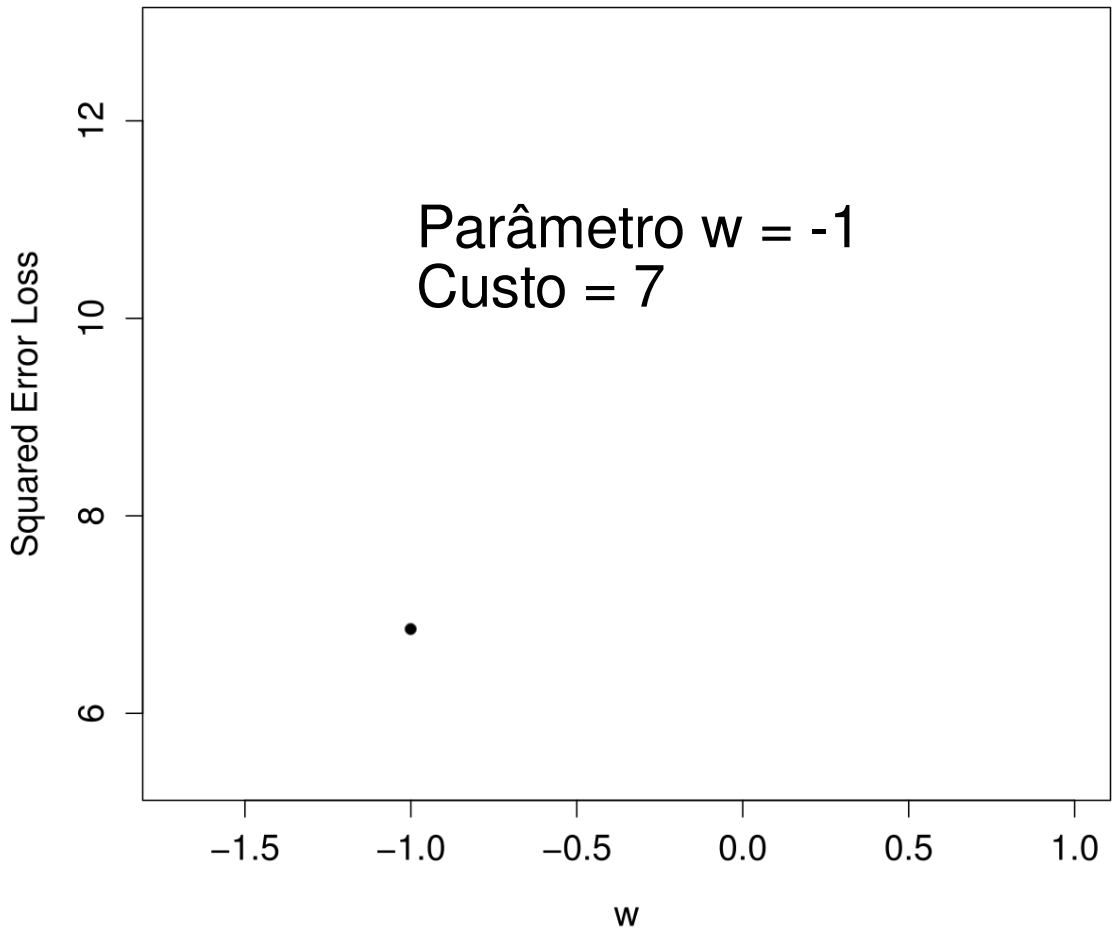
$$\frac{d\ell\left(f(w_j, \mathbf{x}_i)\right)}{dw_j} = \lim_{\delta \to 0} \frac{f(w_j + \delta, \mathbf{x}_i) - f(w_j, \mathbf{x}_i)}{\delta}$$

Temos muitas dimensões (parâmetros) e portanto um gradiente (vetor de derivadas).

Na prática computamos o gradiente numérico e buscamos pelo vale (mínimo) da função por meio da descida do gradiente (**Gradient descent**).

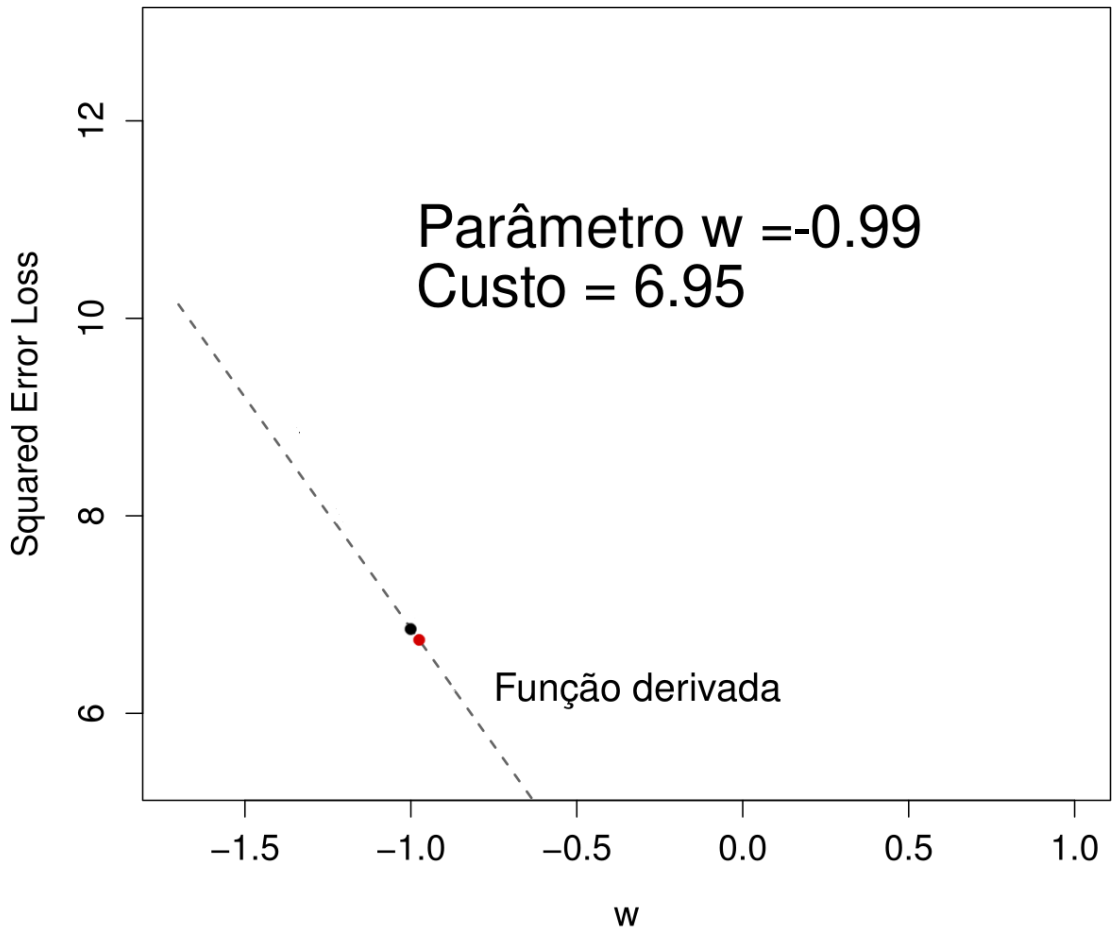
Intuição do método

Inicializamos o parâmetro com um valor arbitrário e computamos o custo



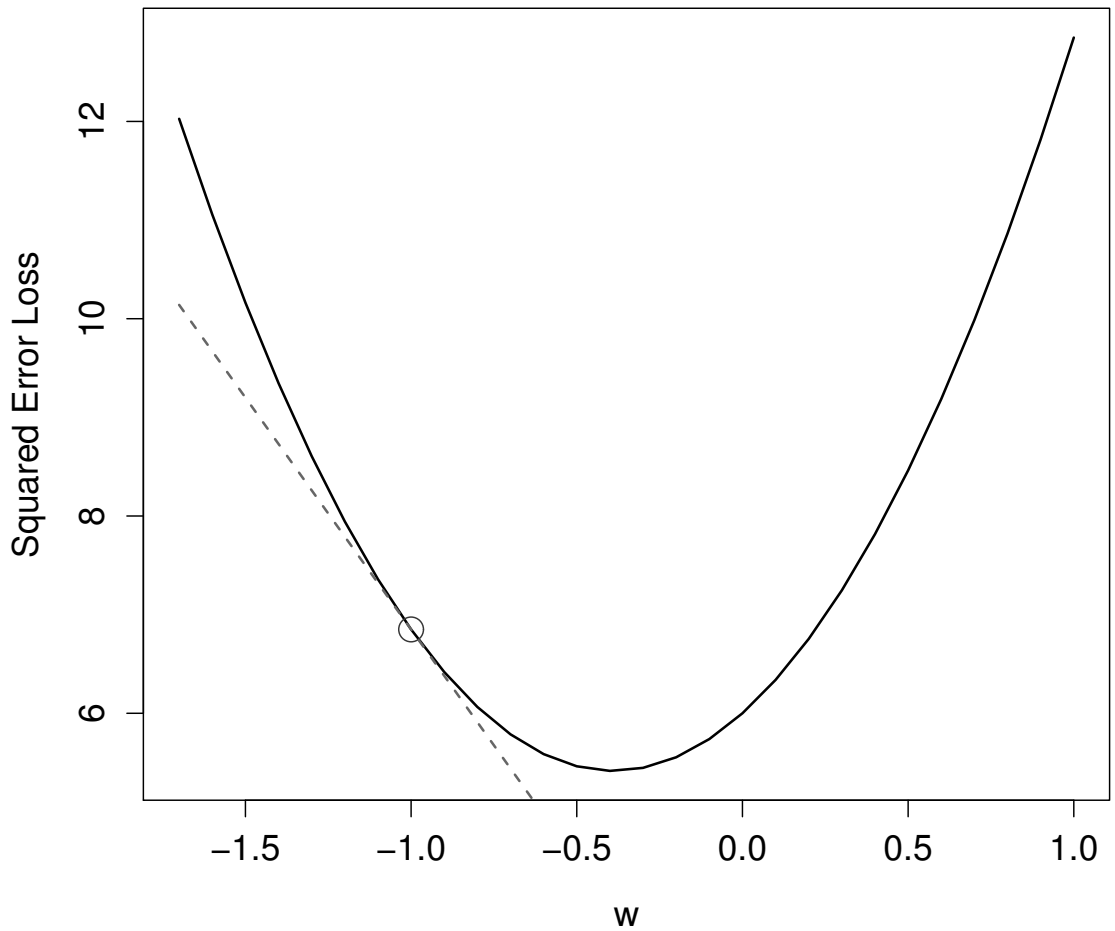
Intuição do método

A inclinação da derivada (diferença entre os custos) indica a direção que devemos seguir



Intuição do método

O que gostaríamos é que a função de custo fosse convexa, i.e. com mínimo global



Gradient descent

Dado um exemplo de treinamento ajustamos cada parâmetro do modelo

W

$$\begin{bmatrix} 0.1, \\ -0.25, \\ 0.1, \\ 2.5, \\ 0, \\ \ldots, \\ -0.1 \end{bmatrix}$$

$$\ell(f(W)) = 2.31298$$

$w_i + \delta$

$$\begin{bmatrix} 0.1 + \mathbf{0.001}, \\ -0.25, \\ 0.1, \\ 2.5, \\ 0, \\ \ldots, \\ -0.1 \end{bmatrix}$$

$$\ell(f(W')) = 2.31\mathbf{201}$$

dw_i

$$\begin{bmatrix} ?, \\ , \\ , \\ , \\ , \\ \ldots, \\ \end{bmatrix}$$

$$(f(w_i+\delta)-f(w_i))/\delta$$

Gradient descent

W

$$\begin{bmatrix} 0.1, \\ -0.25, \\ 0.1, \\ 2.5, \\ 0, \\ ..., \\ -0.1 \end{bmatrix}$$

$\ell(f(W)) = 2.31298$

$w_i + \delta$

$$\begin{bmatrix} 0.1 + \mathbf{0.001}, \\ -0.25, \\ 0.1, \\ 2.5, \\ 0, \\ ..., \\ -0.1 \end{bmatrix}$$

$\ell(f(W')) = 2.31\mathbf{201}$

dw_i

$$\begin{bmatrix} -0.97, \\ , \\ , \\ , \\ , \\ ..., \end{bmatrix}$$

$(f(w_i+\delta)-f(w_i))/\delta$

Gradient descent

W

$$\begin{bmatrix} 0.1, \\ -0.25, \\ 0.1, \\ 2.5, \\ 0, \\ ..., \\ -0.1 \end{bmatrix}$$

$\ell(f(W)) = 2.31298$

$w_i + \delta$

$$\begin{bmatrix} 0.1, \\ -0.25 + \mathbf{0.001}, \\ 0.1, \\ 2.5, \\ 0, \\ ..., \\ -0.1 \end{bmatrix}$$

$\ell(f(W')) = 2.31298$

dw_i

$$\begin{bmatrix} -0.97, \\ 0.0, \\ , \\ , \\ , \\ ..., \end{bmatrix}$$

$(f(w_i+\delta)-f(w_i))/\delta$

Gradient descent

W

$$\begin{bmatrix} 0.1, \\ -0.25, \\ 0.1, \\ 2.5, \\ 0, \\ ..., \\ -0.1 \end{bmatrix}$$

$\ell(f(W)) = 2.31298$

$w_i + \delta$

$$\begin{bmatrix} 0.1, \\ -0.25, \\ 0.1 + \mathbf{0.001}, \\ 2.5, \\ 0, \\ ..., \\ -0.1 \end{bmatrix}$$

$\ell(f(W1)) = 2.31\mathbf{459}$

dw_i

$$\begin{bmatrix} -0.97, \\ 0.0, \\ +1.61, \\ -, \\ -, \\ ..., \\ - \end{bmatrix}$$

$(f(w_i+\delta)-f(w_i))/\delta$

Gradient descent

W

$$\begin{bmatrix} 0.1, \\ -0.25, \\ 0.1, \\ 2.5, \\ 0, \\ ..., \\ -0.1 \end{bmatrix}$$

$\ell(f(W)) = 2.31298$

$w_i + \delta$

$$\begin{bmatrix} 0.1, \\ -0.25, \\ 0.1, \\ 2.5, \\ 0, \\ ..., \\ -0.1 \end{bmatrix}$$

$\ell(f(W')) =$ **2.08720**

dw_i

$$\begin{bmatrix} -0.93, \\ 0.0, \\ -1.61, \\ +0.02, \\ +0.5, \\ ..., \\ -3.7 \end{bmatrix}$$

$(f(w_i+\delta)-f(w_i))/\delta$

Stochastic Gradient Descent (SGD)

Mas é computacionamente caro computar o gradiente para N grande

SGD:
aproximar a perda empírica usando um lote aleatório **minibatch** de instâncias: algo entre 16 and 128.

- mais rápido
- mais grosseiro, necessitando mais iterações

Agenda

Uma tarefa de classificação

Mudando o pipeline do aprendizado

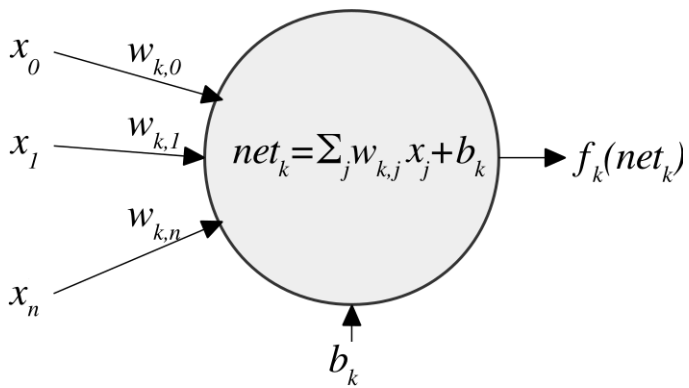
Machine vs Deep Learning

Rede neural: do raso ao profundo

- Perceptron e Multilayer Perceptron

Neurônio

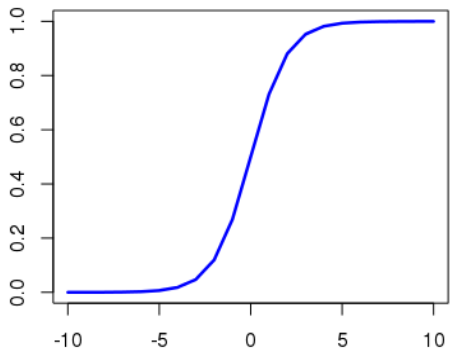
- entrada: valores organizados em um vetor
- saída: um único valor
  - cada valor de entrada é associado a um peso w (força da conexão)
  - o bias b funciona como intercepto
- aprender é ajustar w's e b's aos dados de treinamento
- há uma função aplicada nessa soma, a qual é chamada função de ativação



Algumas funções de ativação

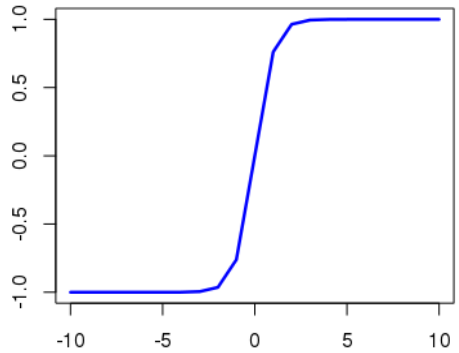
Sigmoid

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



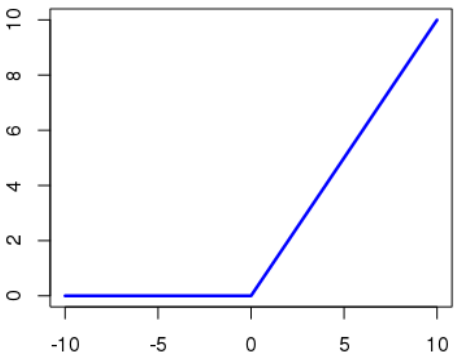
Hiperbolic Tangent

$$f(x) = \tanh(x)$$



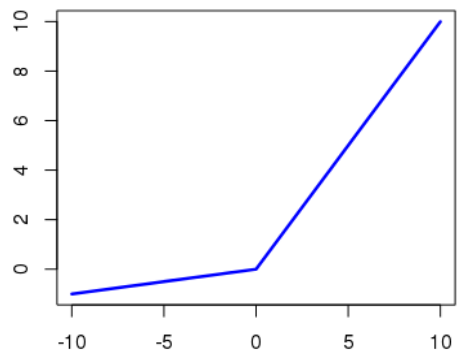
ReLU

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



Leaky ReLU

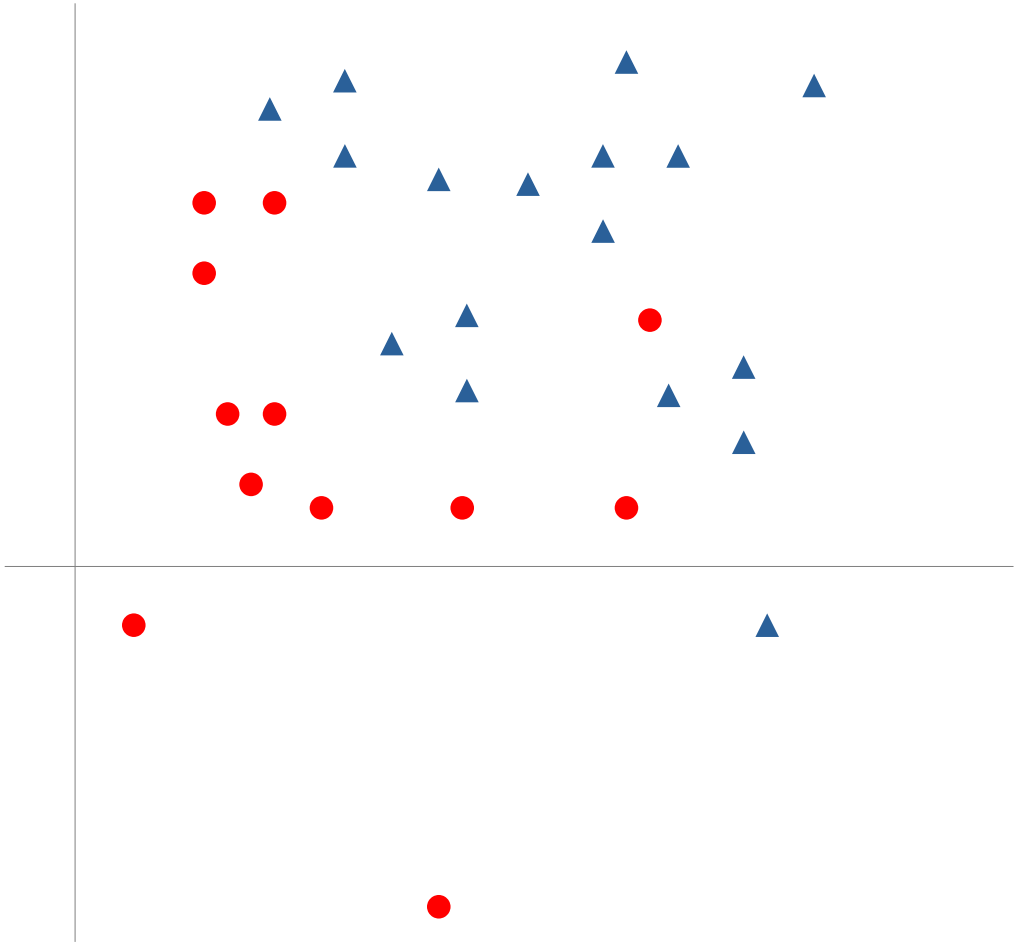
$$f(x) = \max(0.1x, x)$$



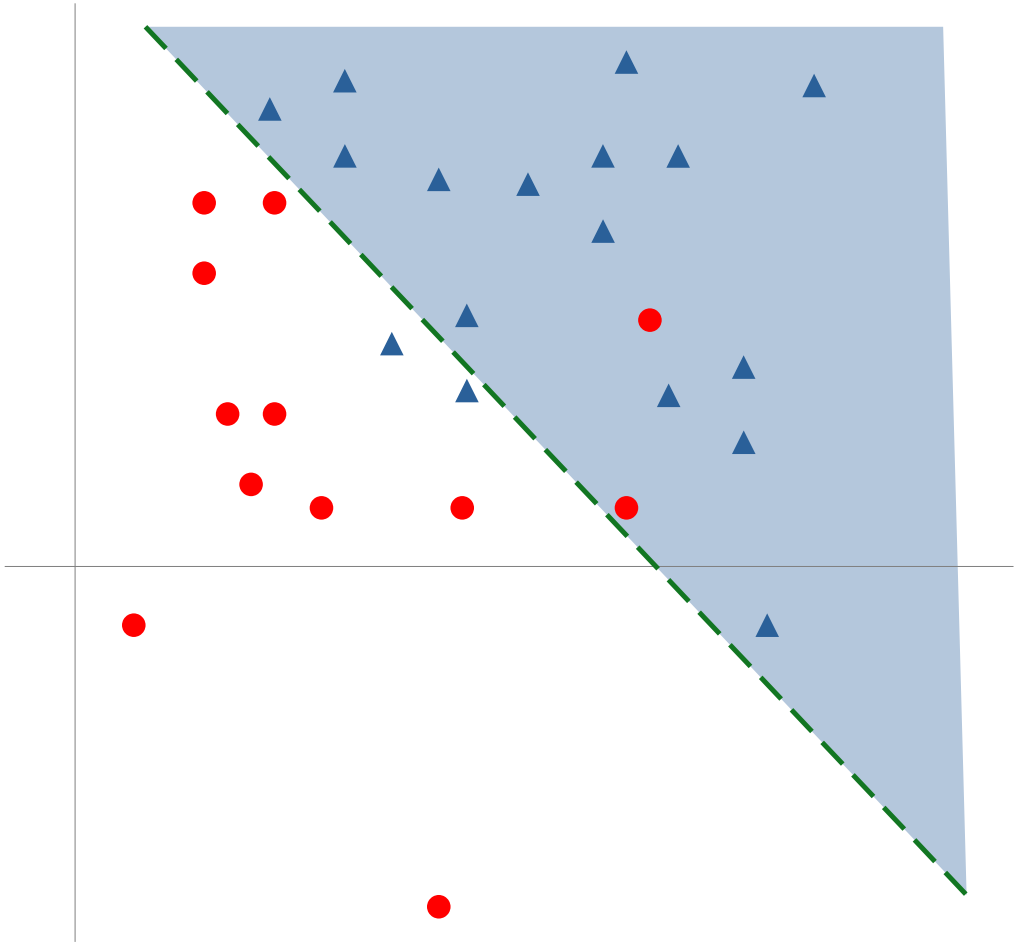
Rede neural Feed-forward

- Recursivamente observa exemplos de entrada e adapta os pesos para todos os parâmetros da rede neural.
- **Feed forward**: todos os neurônios processam a entrada, e computamos a perda
- **Backpropagation**: algoritmo que usa a formulação da regra da cadeia para calcular o gradiente da função de perda, e propaga esse gradiente pelas camadas e neurônios

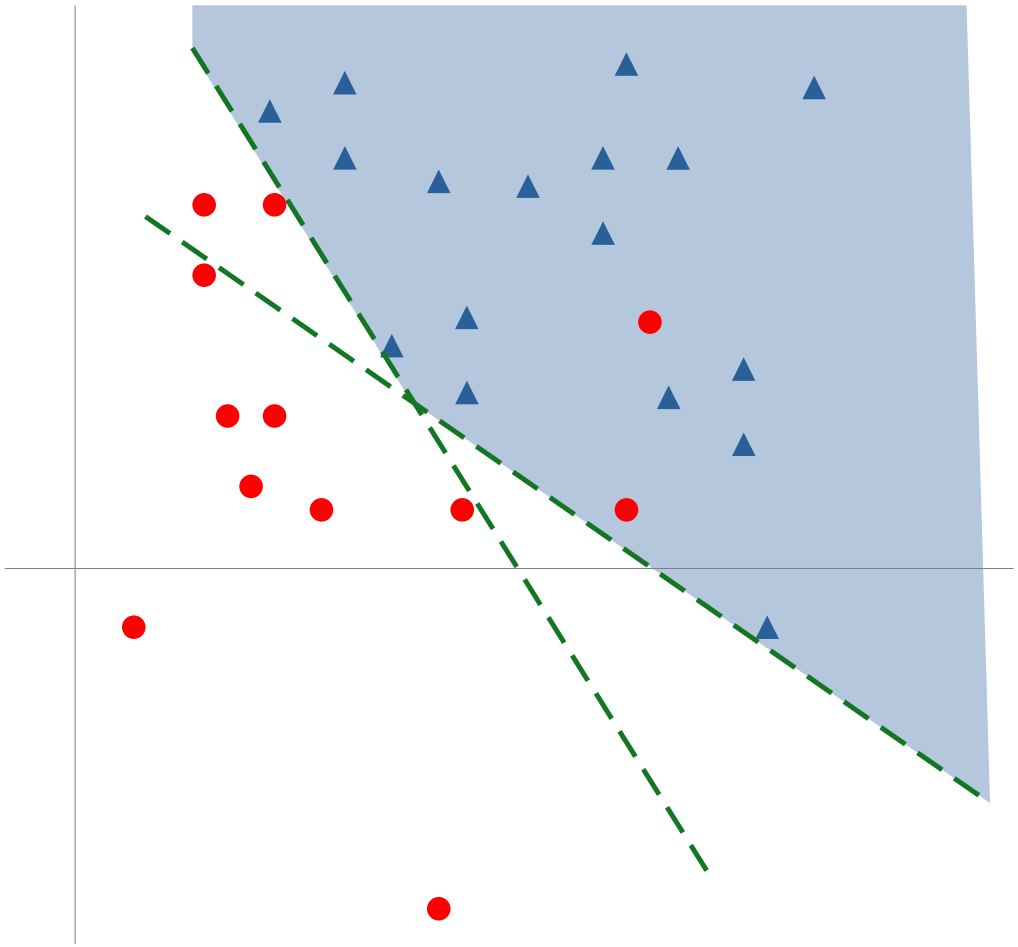
Perceptron Multicamadas: número de neurônios



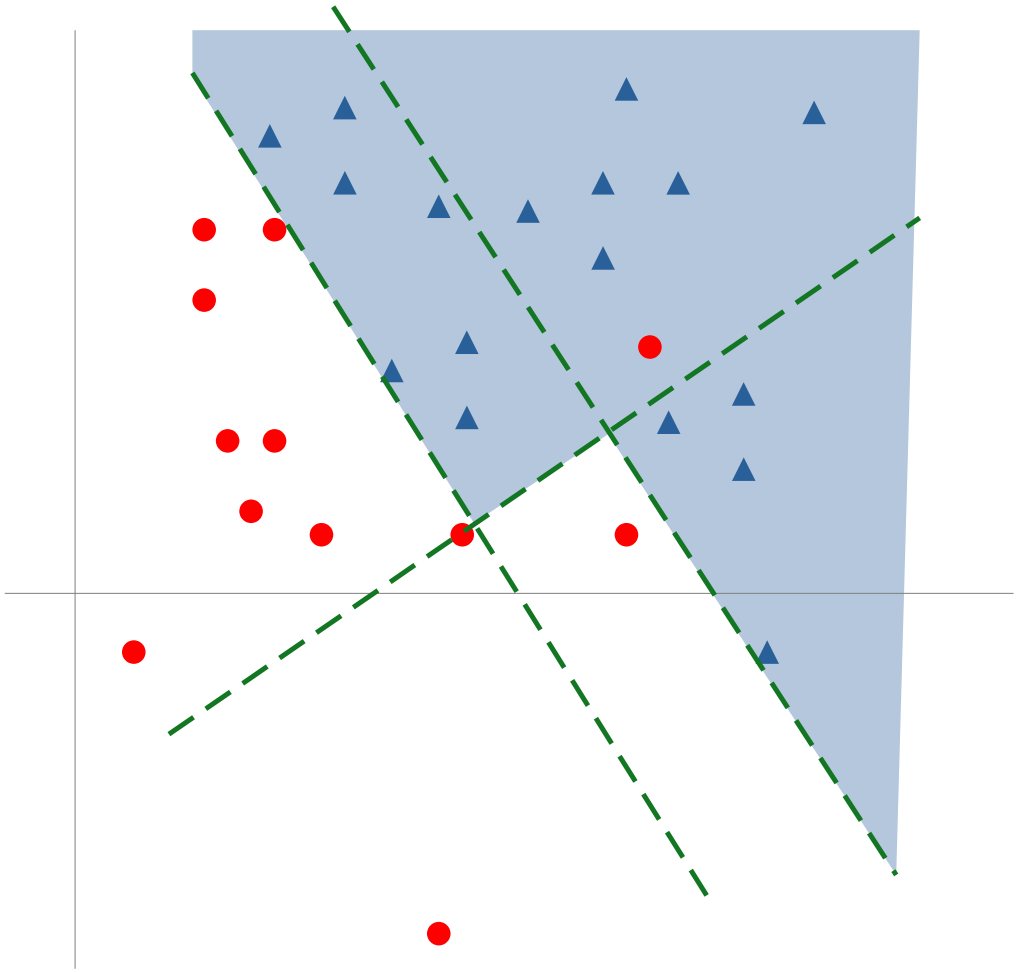
Perceptron Multicamadas: número de neurônios



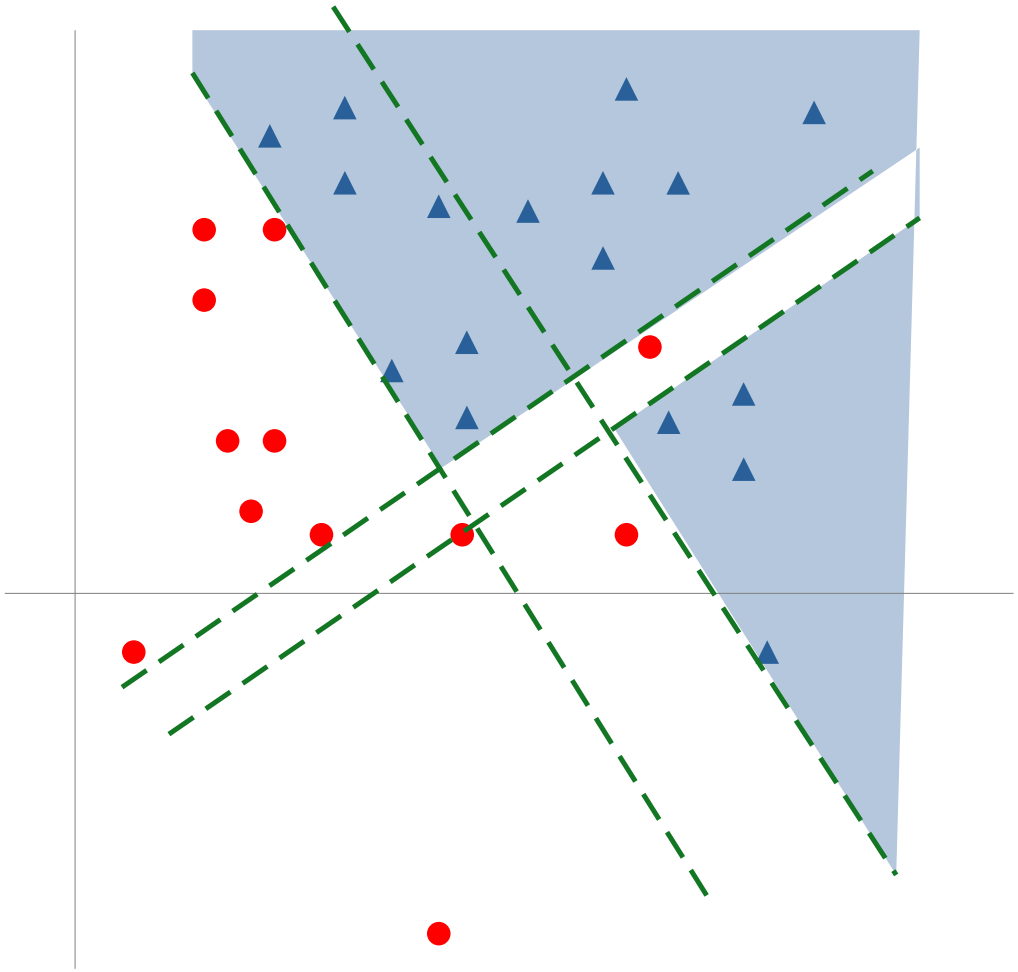
Perceptron Multicamadas: número de neurônios



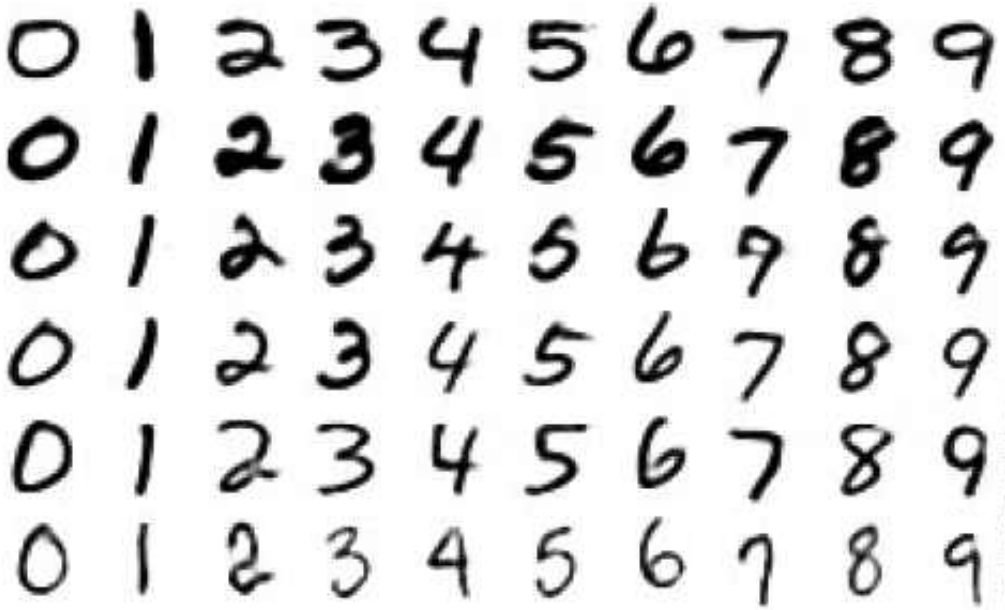
Perceptron Multicamadas: número de neurônios



Perceptron Multicamadas: número de neurônios

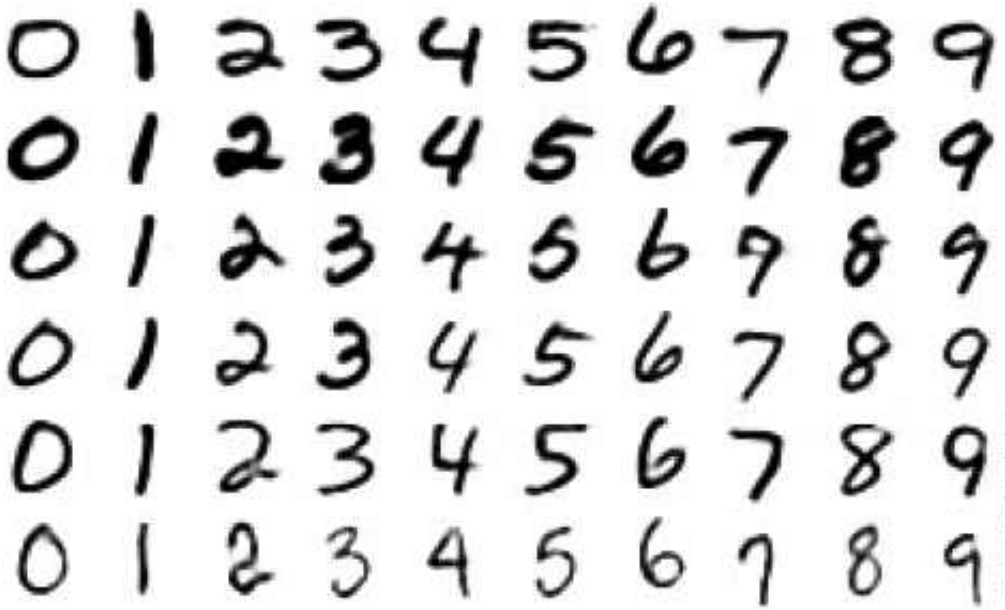


Exemplo de problema: classificação de dígitos



- Imagens com $28 \times 28 = 784$ pixels,

Exemplo de problema: classificação de dígitos



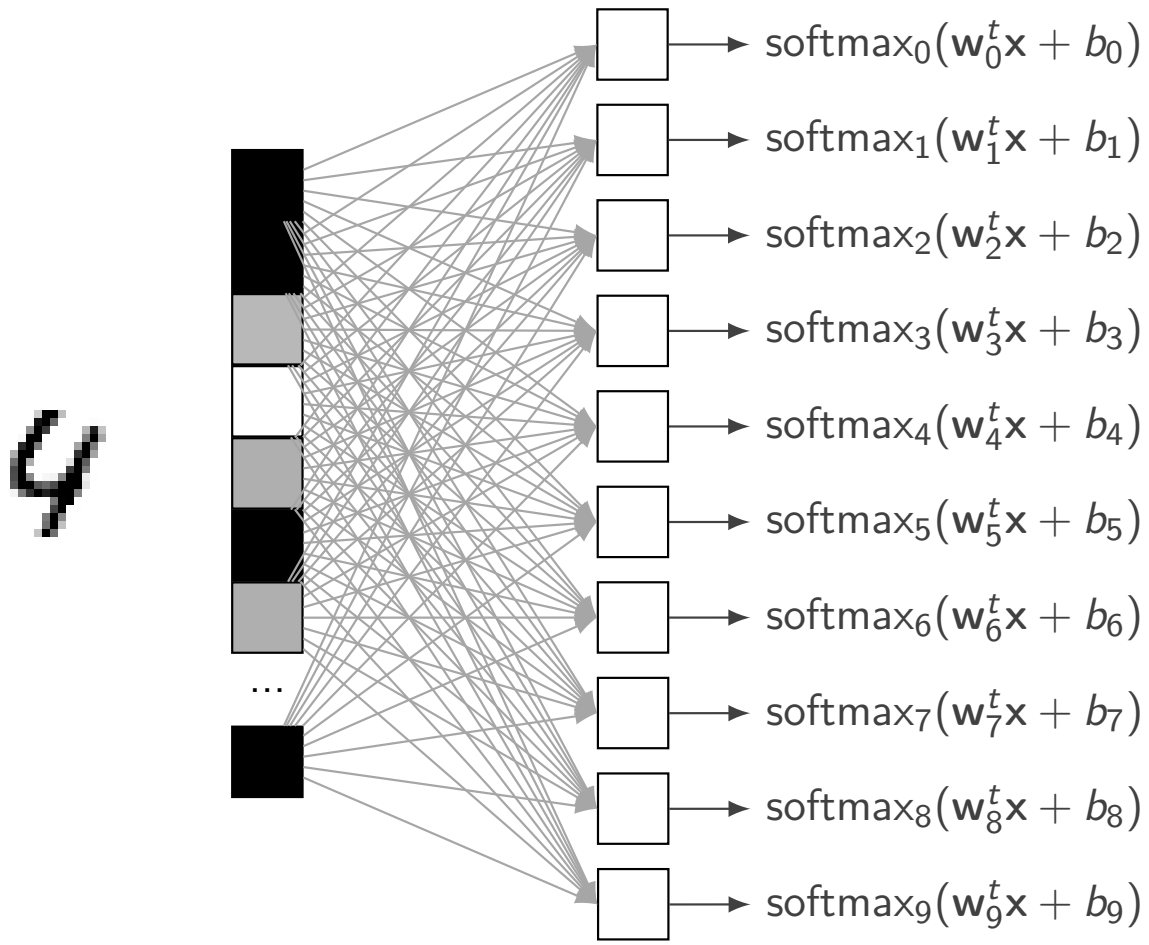
- Imagens com $28 \times 28 = 784$ pixels,
- Redes do tipo Perceptron,
- Algoritmo SGD com 32 imagens no batch,
- Camada de saída normalizada de forma a somar 1: softmax.

"Classificador" Softmax

- Normalizar saída de forma a somar 1
- Permite interpretar cada valor como sendo uma probabilidade
- Na saída, temos a distribuição de probabilidades das classes

$$\text{softmax}(\hat{\mathbf{y}}) = \frac{\hat{y}}{\sum_j \hat{y}_j}$$

Rede neural rasa, com uma única camada



Formulação da rede neural

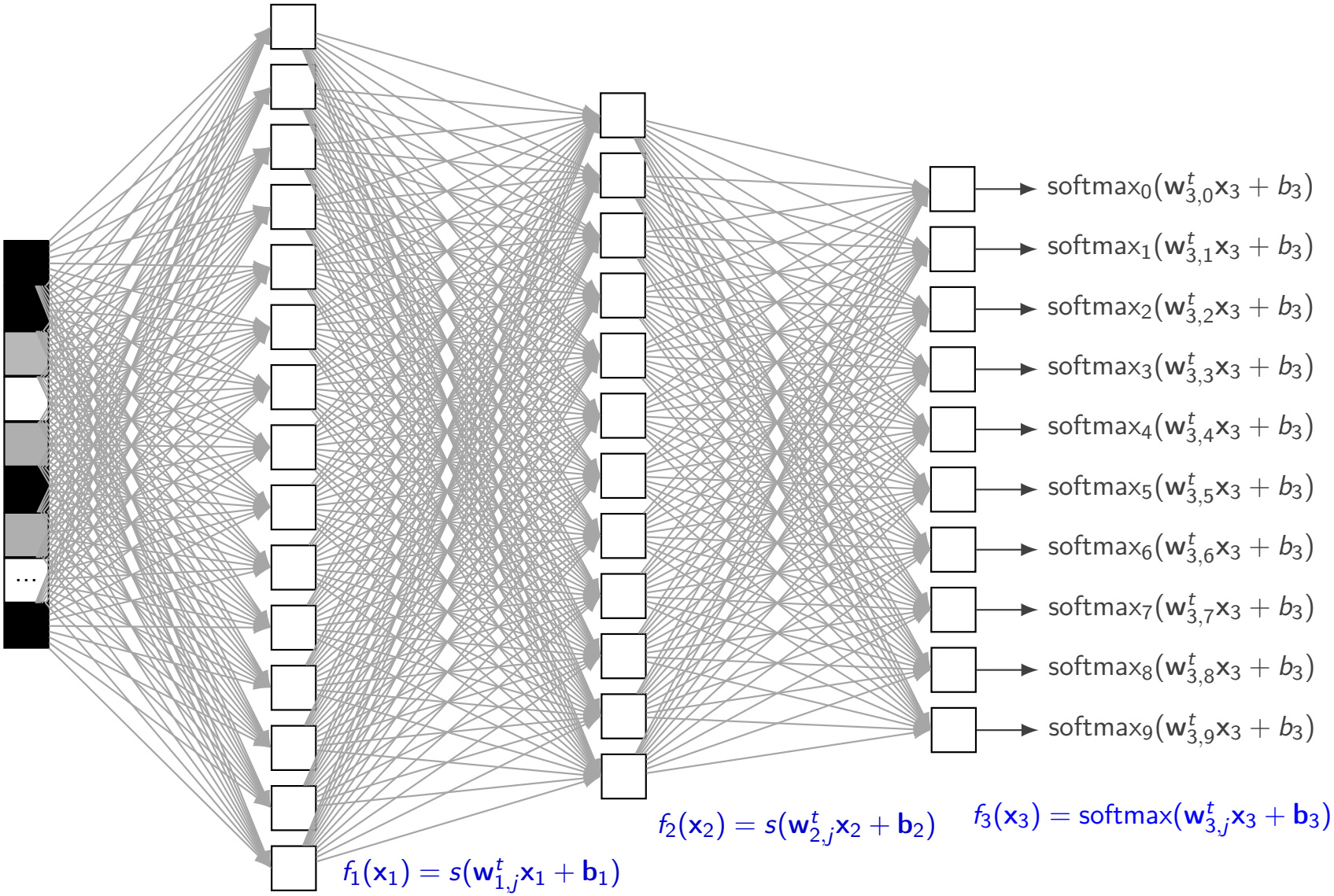
10 classes, batch-size 32, e 784 características (pixels) por imagem

$$\begin{bmatrix} x_{0,0} & x_{0,1} & x_{0,2} & \cdots & x_{0,783} \\ x_{1,0} & x_{0,1} & x_{1,2} & \cdots & x_{0,783} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{31,0} & x_{31,1} & x_{31,2} & \cdots & x_{31,783} \end{bmatrix} \begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,9} \\ w_{1,0} & w_{1,1} & \cdots & w_{1,9} \\ w_{2,0} & w_{2,1} & \cdots & w_{2,9} \\ \vdots & \vdots & \ddots & \vdots \\ w_{783,0} & w_{783,1} & \cdots & w_{783,9} \end{bmatrix} + \begin{bmatrix} b_0 & b_1 & b_2 & \ldots & b_9 \end{bmatrix}$$

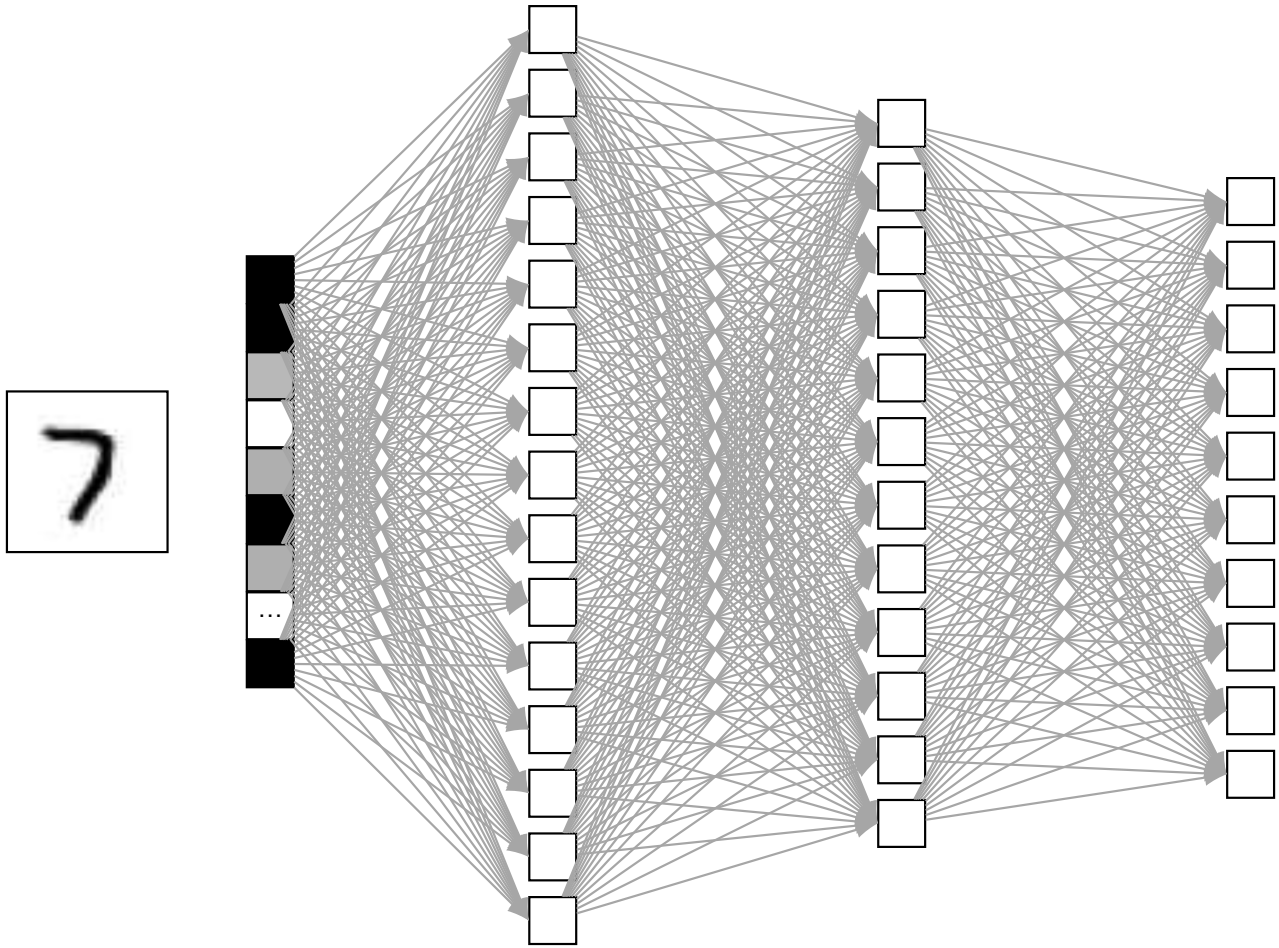
$$\mathbf{Y} = \text{softmax}(\mathbf{X} \cdot \mathbf{W} + \mathbf{b})$$

$$\mathbf{Y} = \begin{bmatrix} y_{0,0} & y_{0,1} & y_{0,2} & \cdots & y_{0,9} \\ y_{1,0} & y_{1,1} & y_{1,2} & \cdots & y_{1,9} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ y_{31,0} & y_{31,1} & y_{31,2} & \cdots & y_{31,9} \end{bmatrix}$$

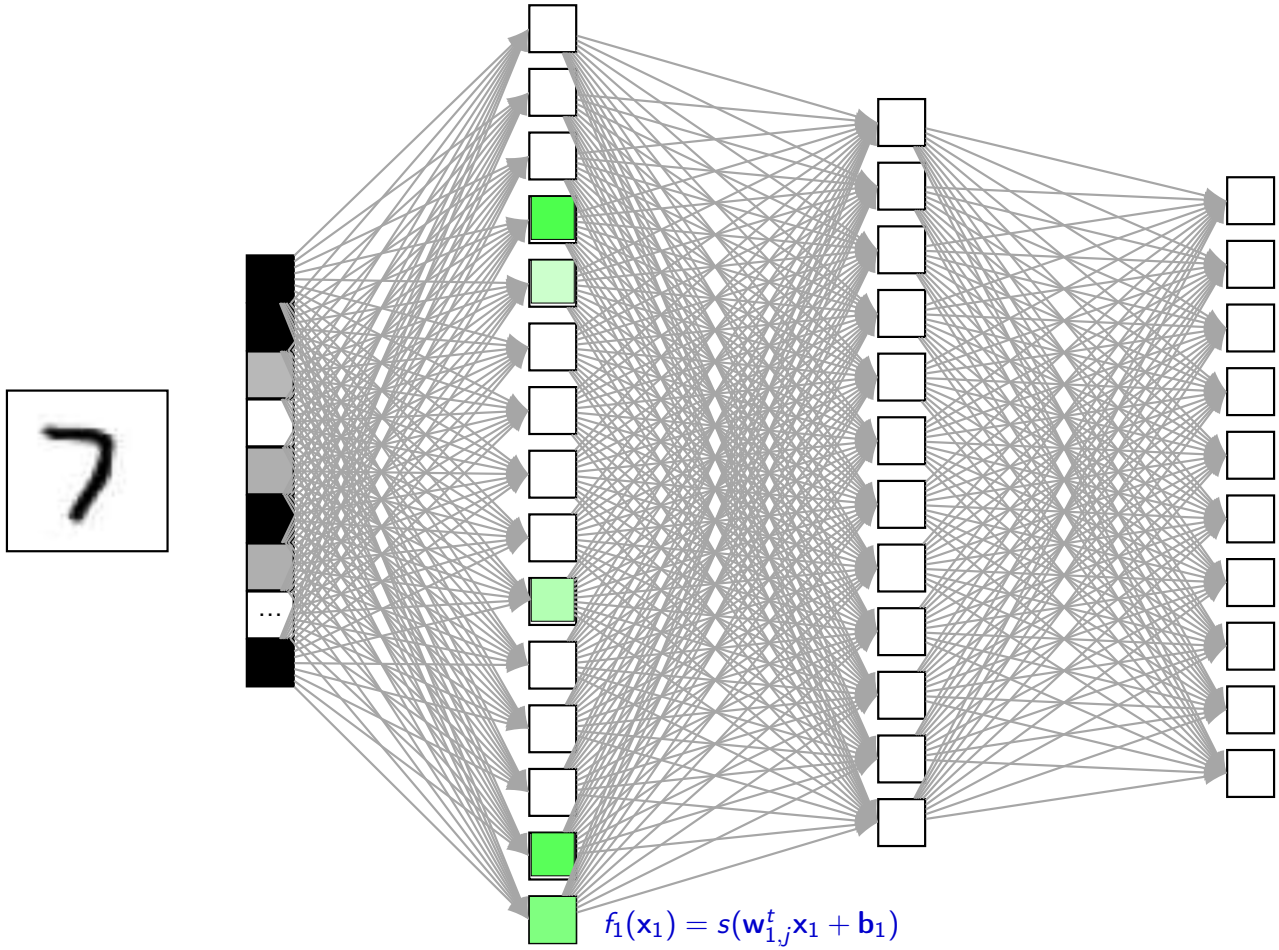
Rede MLP "profunda" com 2 camadas ocultas



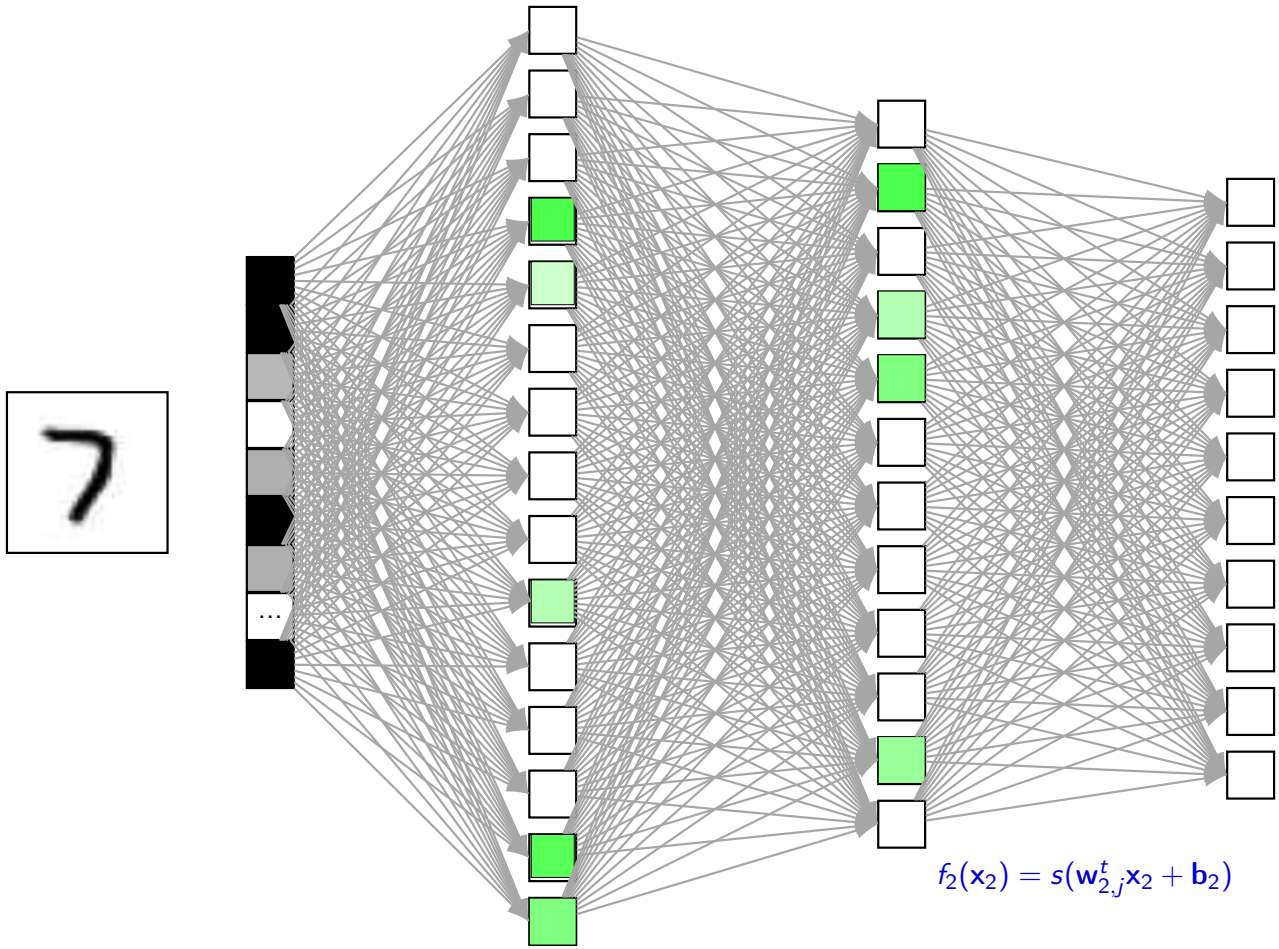
Rede MLP "profunda" com 2 camadas ocultas : Input



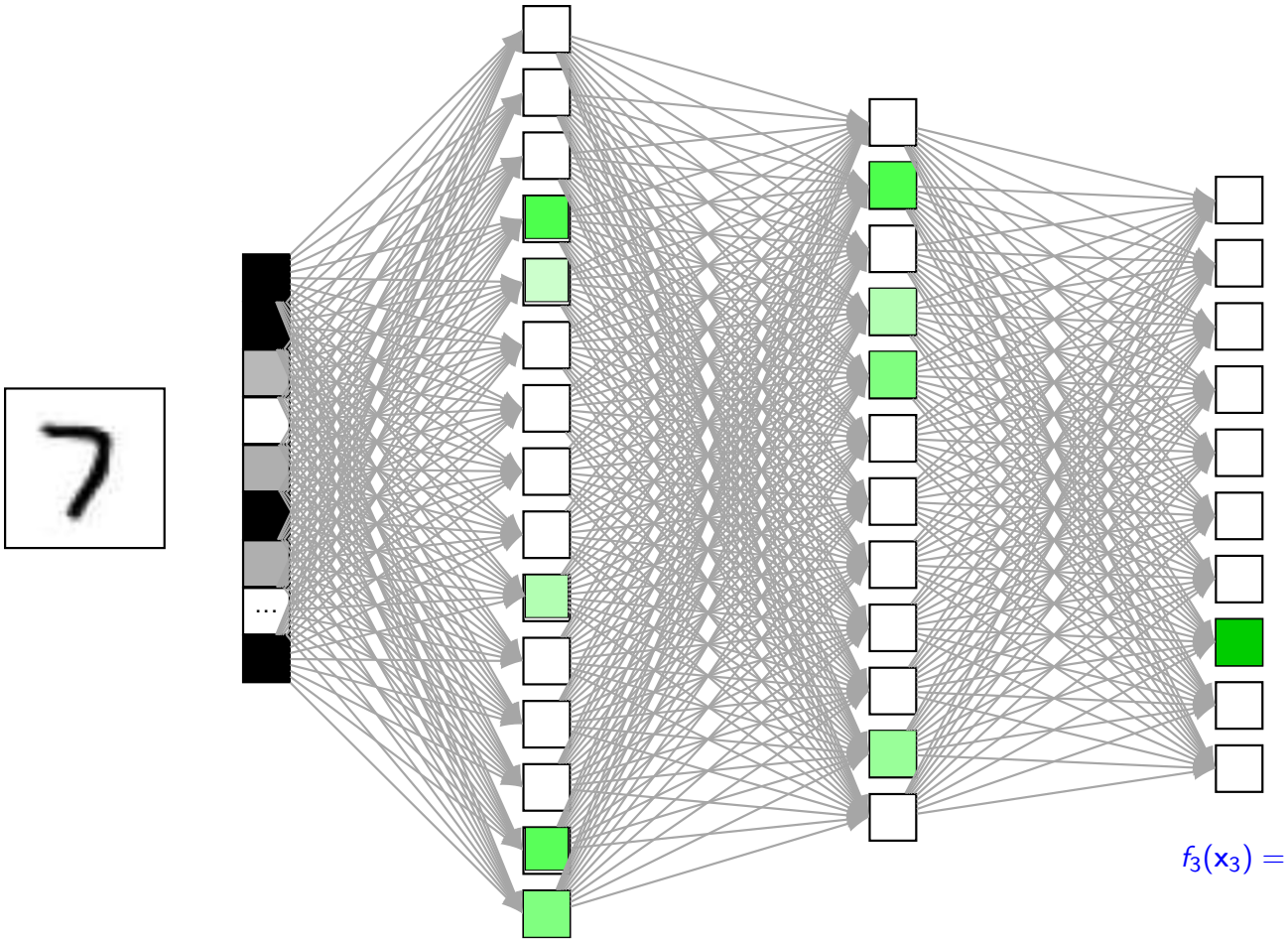
Rede MLP "profunda" com 2 camadas ocultas : Hidden layer 1



Rede MLP "profunda" com 2 camadas ocultas : Hidden layer 2



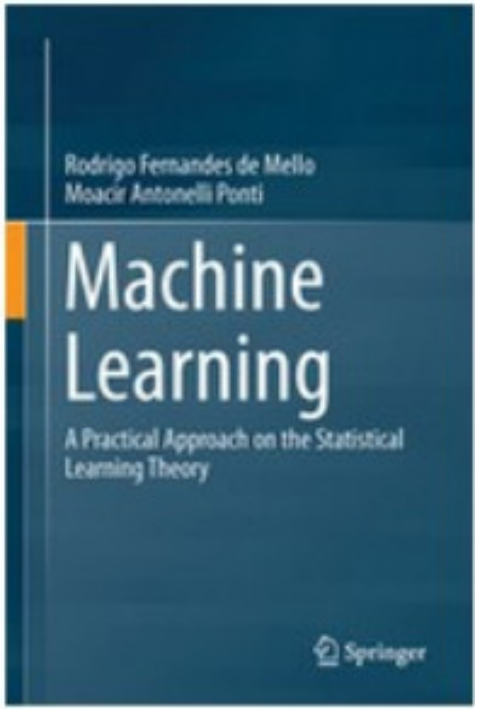
Rede MLP "profunda" com 2 camadas ocultas : output



$f_3(\mathbf{x}_3) = \text{softmax}(\mathbf{w}_{3,j}^t \mathbf{x}_3 + \mathbf{b}_3)$

Bibliografia I

Rodrigo Mello, Moacir A. Ponti. **Machine Learning: a practical approach on the statistical learning theory** Springer, 2018.



Bibliografia II

Moacir A. Ponti, Gabriel Paranhos da Costa. **Como funciona o Deep Learning**
SBC, 2017. Book chapter.
`https://arxiv.org/abs/1806.07908`

Moacir A. Ponti, Leo Ribeiro, Tiago Nazaré, Tu Bui, John Collomosse. **Everything You Wanted to Know About Deep Learning for Computer Vision but were Afraid to Ask**.
SIBGRAPI-T, 2017. Tutorial.