

MÉTODOS E MODELOS AVANÇADOS EM CIÊNCIA DE DADOS

Aula 03 - Redes Neurais Convolucionais
(*Convolutional Neural Networks - CNNs*)

Prof. Rafael G. Mantovani



Universidade Tecnológica Federal do Paraná (UTFPR)
Especialização em Ciência de Dados

Roteiro

- 1 Introdução
- 2 *Deep Learning (DL)*
- 3 *Convolutional Neural Networks (CNNs)*
- 4 Modelagem de CNNs
- 5 Hands on / Exemplos
- 6 Síntese / Próximas Aulas
- 7 Referências

Roteiro

- 1 Introdução
- 2 *Deep Learning (DL)*
- 3 *Convolutional Neural Networks (CNNs)*
- 4 Modelagem de CNNs
- 5 Hands on / Exemplos
- 6 Síntese / Próximas Aulas
- 7 Referências

Introdução

- **Teorema da Aproximação Universal (George Cybenko, 1989)**
 - “... uma RNA com única camada escondida e um número finito de neurônios pode aproximar qualquer função contínua”.

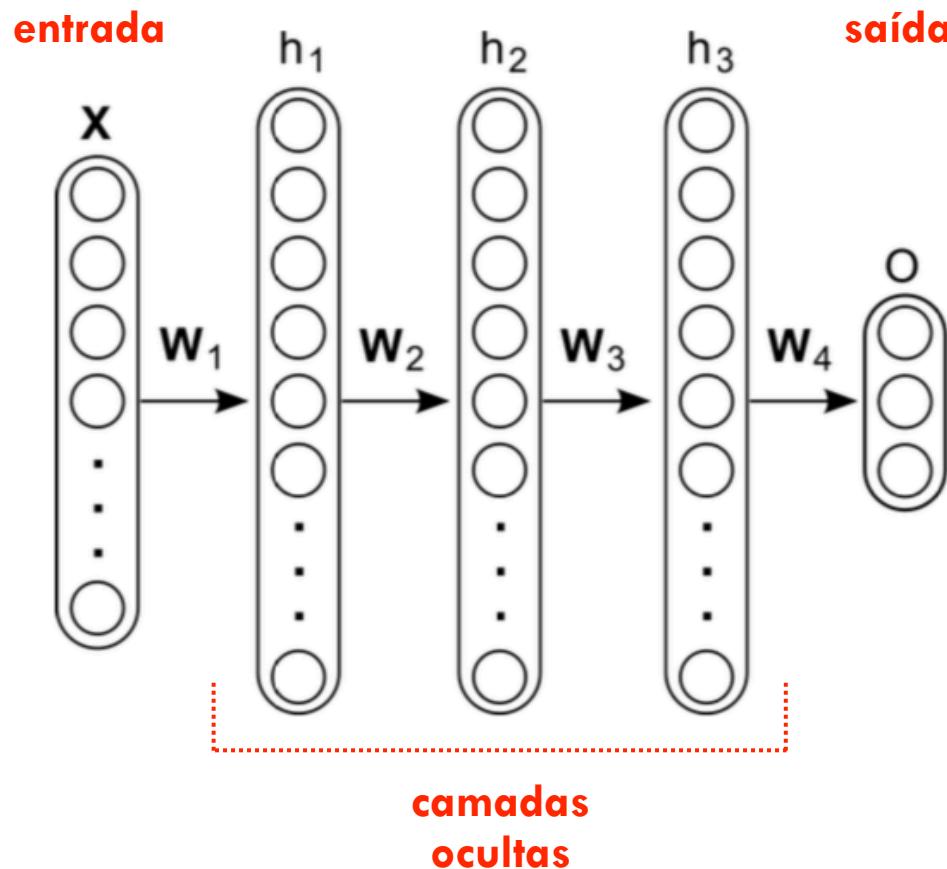
Introdução

- **Teorema da Aproximação Universal (George Cybenko, 1989)**
 - "... uma RNA com única camada escondida e um número finito de neurônios pode aproximar qualquer função contínua".

- **Problemas! Esse teorema não diz nada sobre:**
 - tempo de treinamento
 - facilidade de implementação
 - generalização

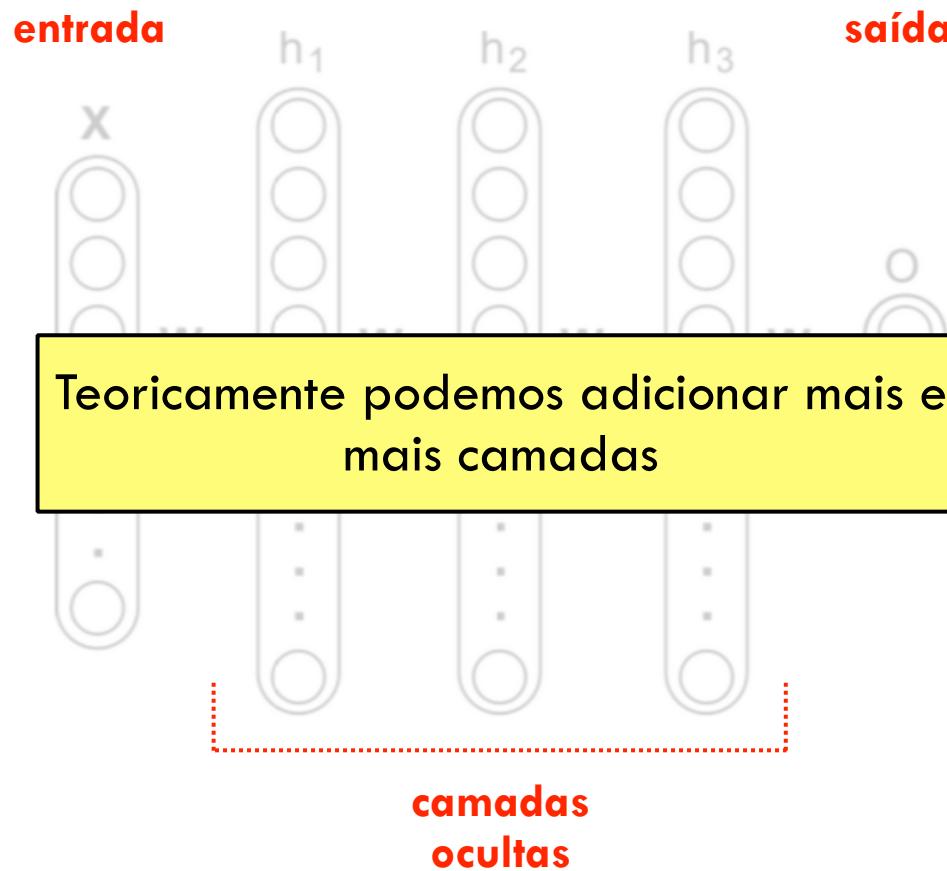
Introdução

- Em uma **MLP** podemos ter **L** camadas ocultas:



Introdução

- Em uma **MLP** podemos ter **L** camadas ocultas:



Introdução

- **Deep Learning**
 - **Objetivo:** aprender modelos com múltiplas camadas de representação. Exs:
 - MLPs, DNNs, DBNs, Deep Autoencoders,
 - CNNs, RBMs, LSTM, etc
 - Cada camada corresponde a uma forma de representação
 - unidade (neurônio) está associada a uma característica da entrada
 - unidades podem ser ativadas simultaneamente

Introdução

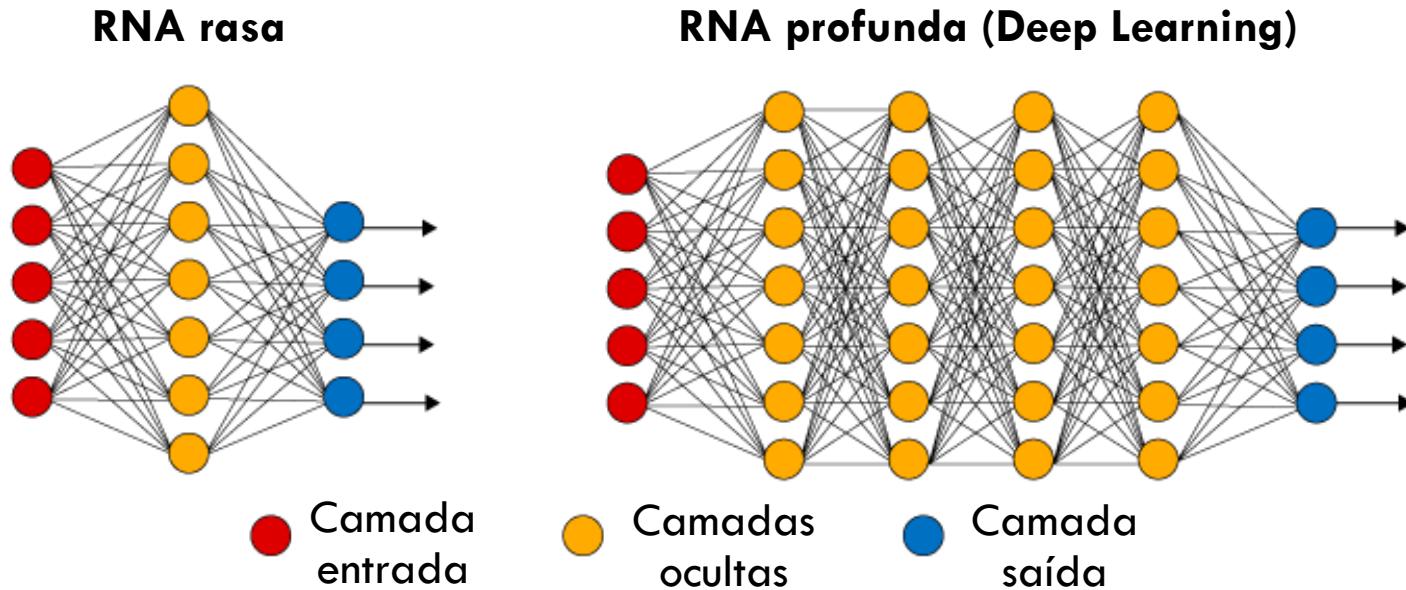
- Algumas implicações de explorarmos arquiteturas profundas (com muitas camadas):
 - podem representar certas funções de maneira mais compacta
 - há funções que podem ser representadas com uma única camada, porém requerem um número exponencial de neurônios
 - porém, pode ser necessário um número polinomial de neurônios, se pudermos aumentar o número de camadas

[Larochelle, et. al. 2009] *Exploring Strategies for Training Deep Neural Networks*

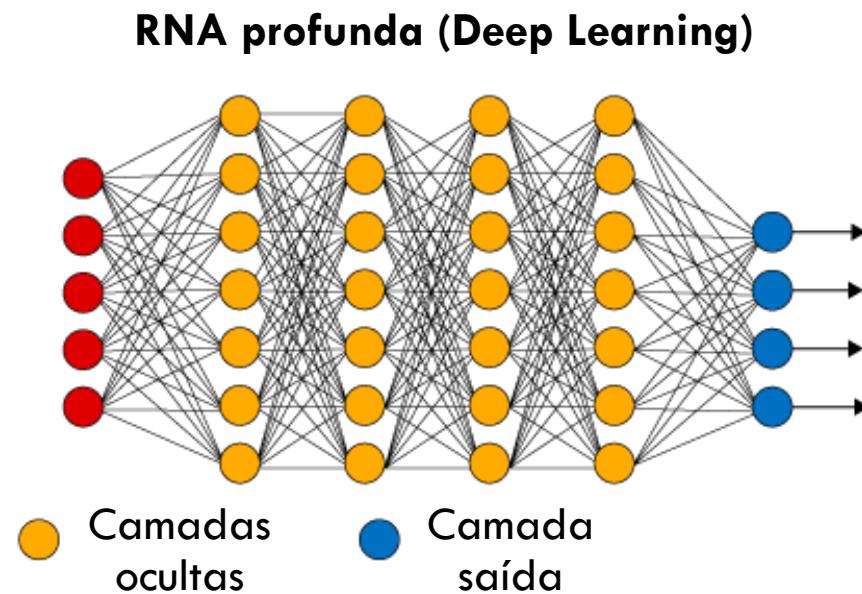
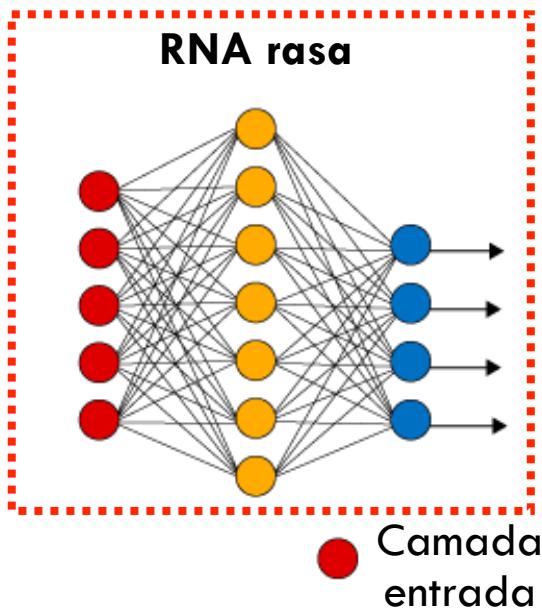
Roteiro

- 1 Introdução
- 2 *Deep Learning (DL)*
- 3 *Convolutional Neural Networks (CNNs)*
- 4 Modelagem de CNNs
- 5 Hands on / Exemplos
- 6 Síntese / Próximas Aulas
- 7 Referências

Deep Learning

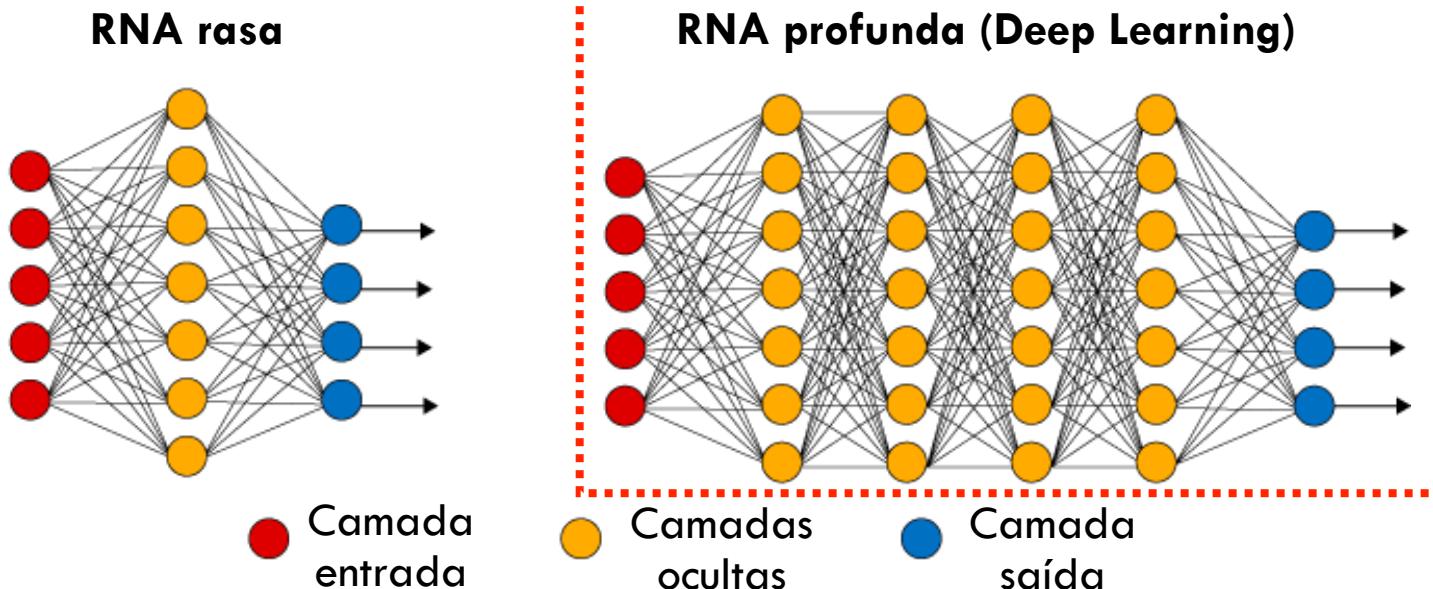


Deep Learning



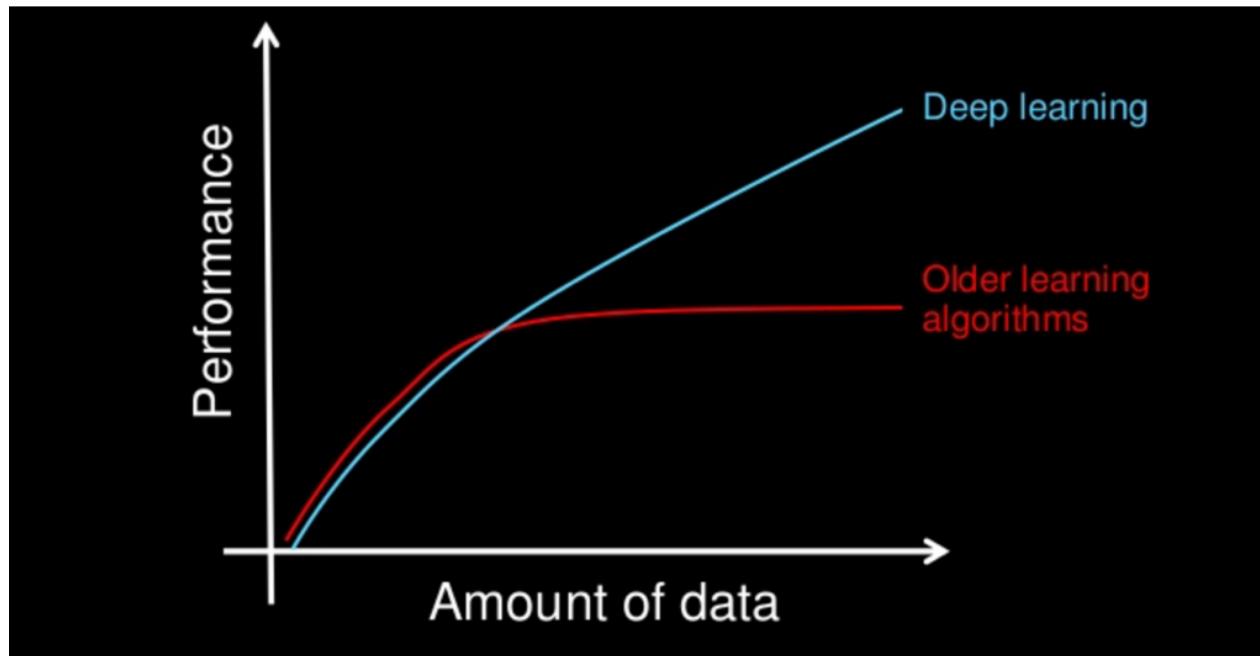
Até agora vimos
modelos "rasos"
(shallow)

Deep Learning



Agora exploraremos
mais camadas
(deep)

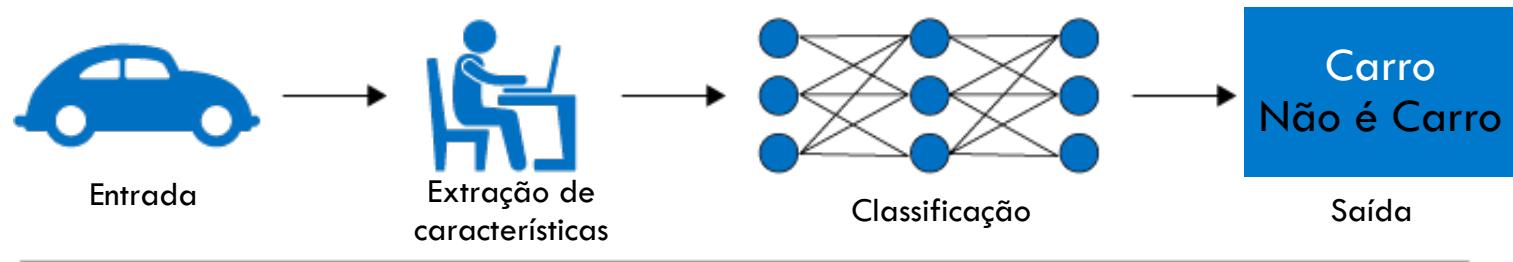
Deep Learning



Teoremas e experimentos indicam que DL tem um desempenho melhor quando manipula uma quantidade maior de dados ...

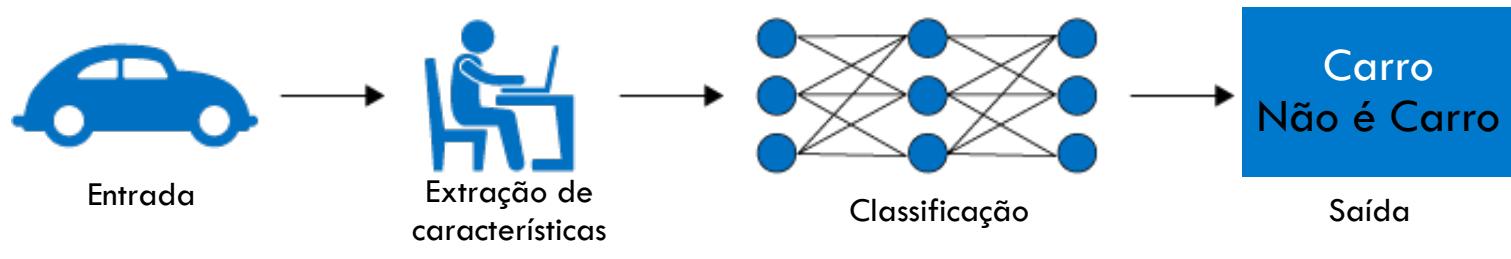
Deep Learning

Aprendizado de Máquina

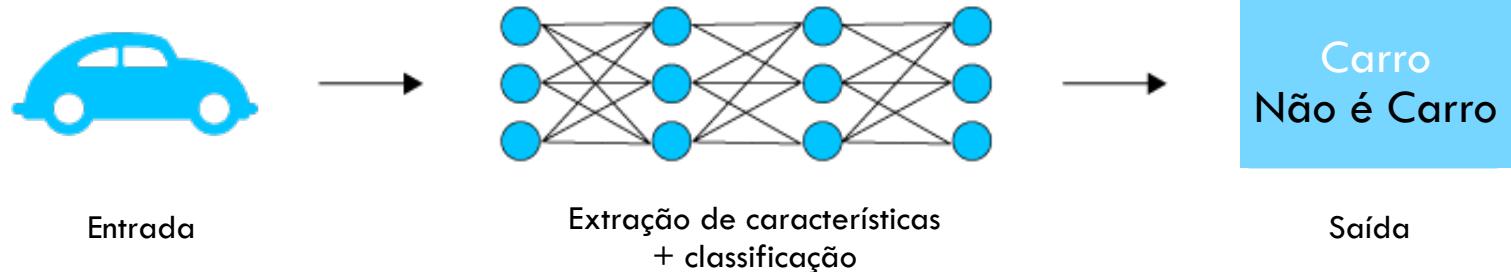


Deep Learning

Aprendizado de Máquina



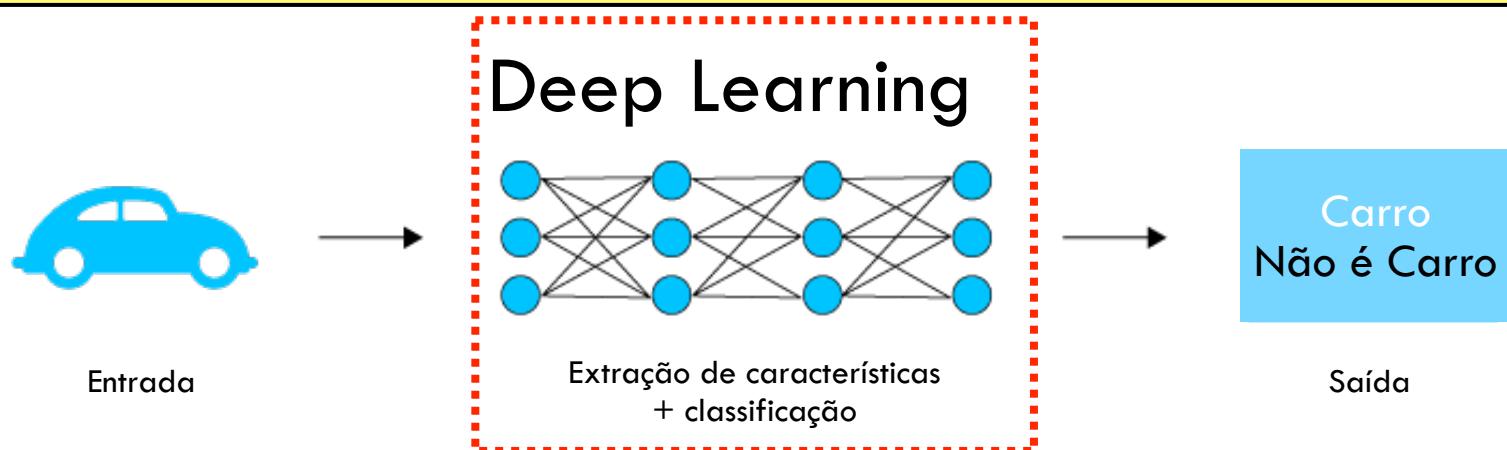
Deep Learning



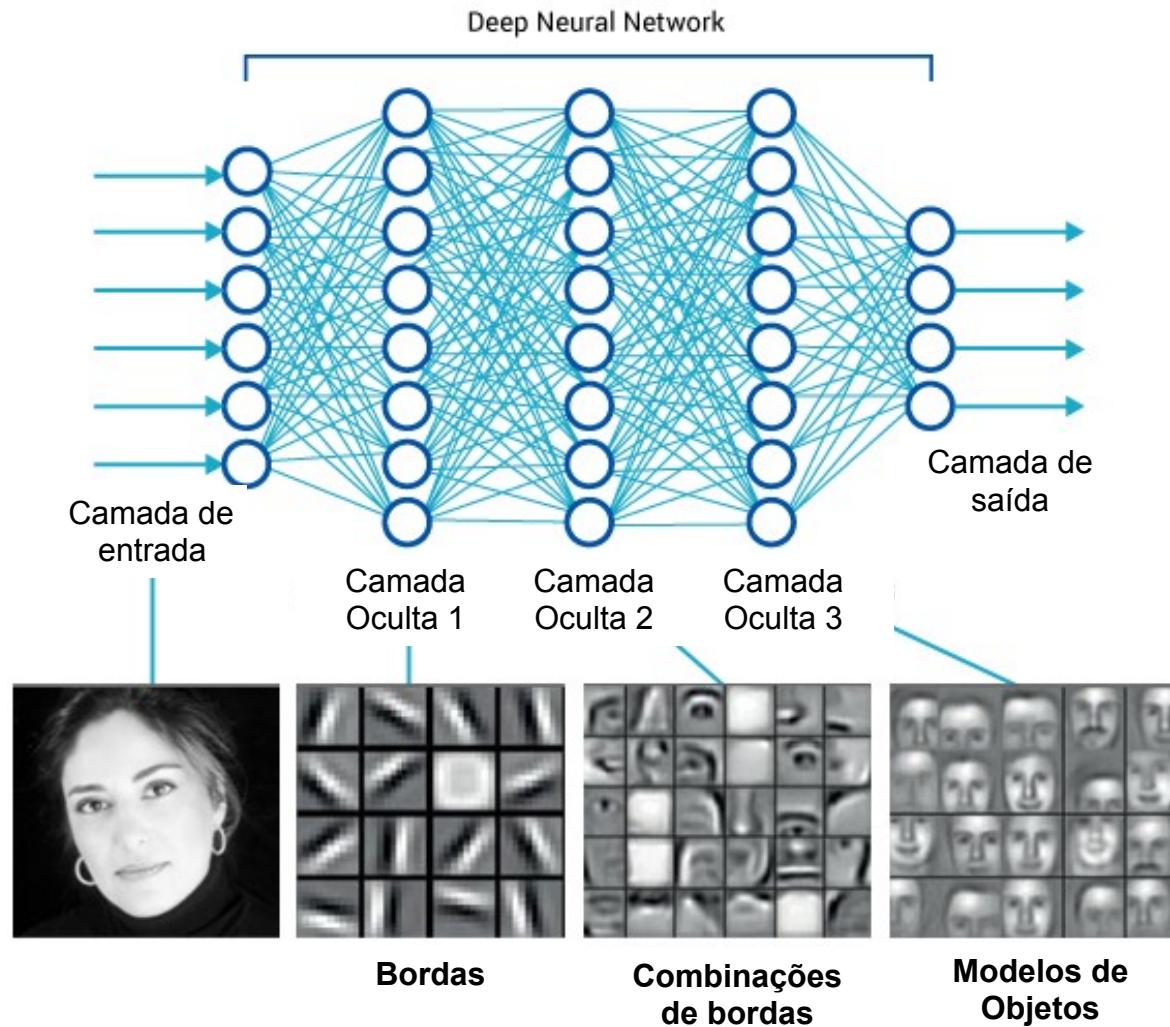
Deep Learning

Tanto a extração das características como a geração do modelo preditivo fazem parte do algoritmo de DL.

Até por isso são comumente referenciados como algoritmos de caixa-preta (“**black-box**”)



Deep Learning



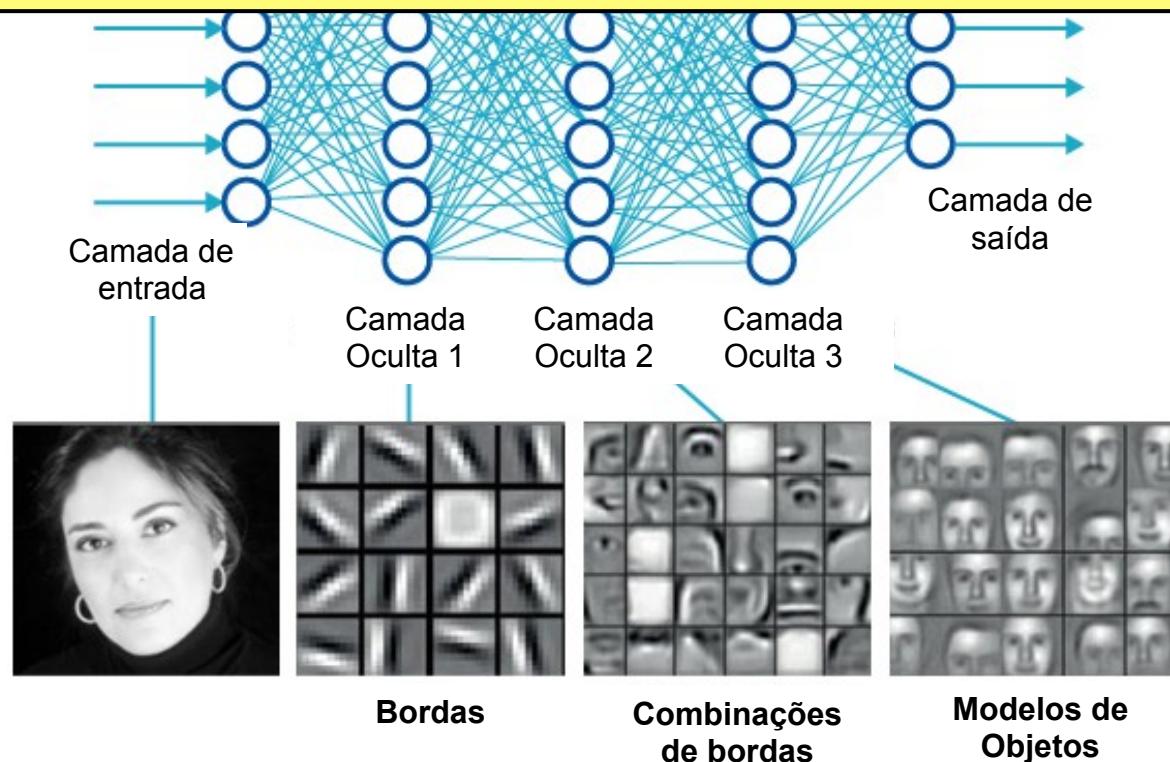
Deep Learning

1a camada extrai **características simples** (bordas)

2a camada combina essas bordas em **formatos mais complexos** (contornos)

3 camada tenta criar padrões **ainda mais complexos** (objetos, etc)

....

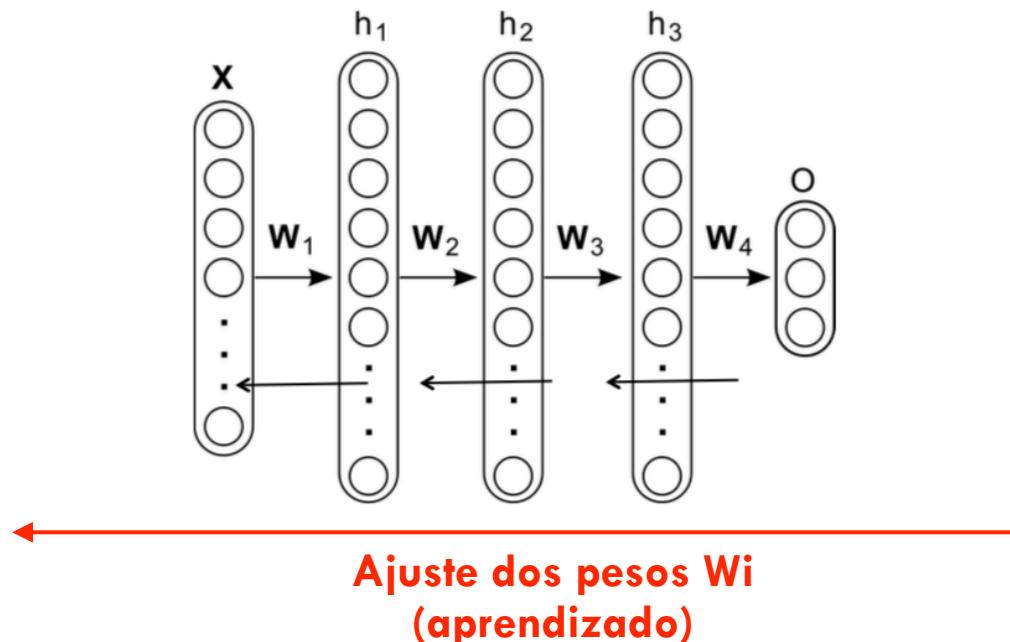


Problemas?

- Problemas → Treinar Deep Learning (DL)

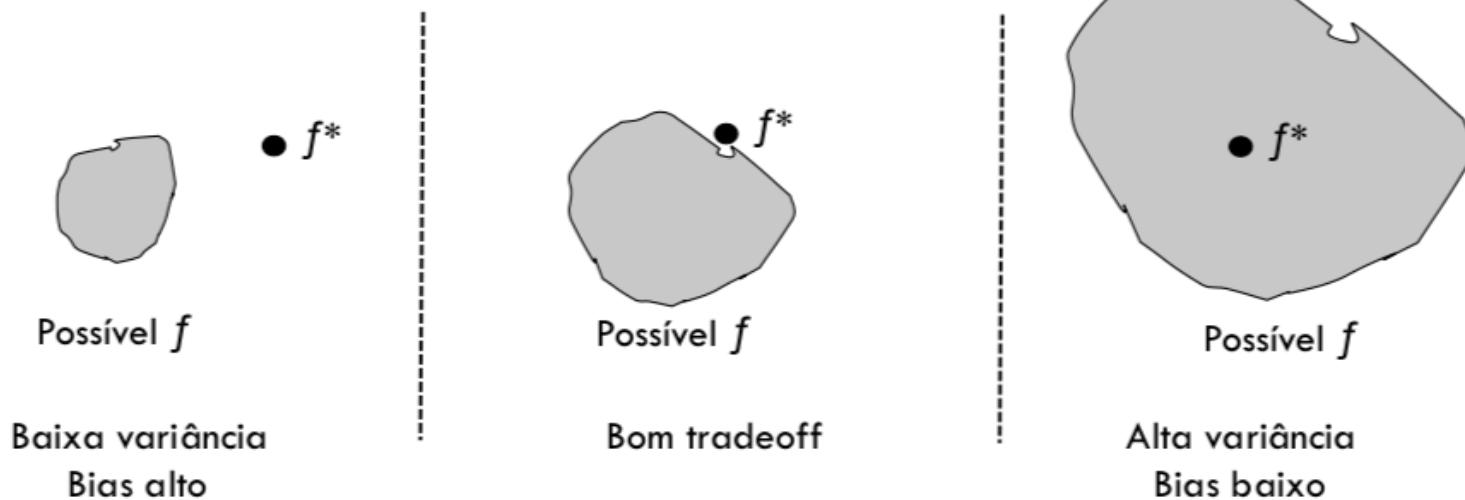
Problemas?

- Problemas → Treinar Deep Learning (DL)
 - *Underfitting* → Gradientes desaparecem ($\Delta w_i = 0$)



Problemas?

- Problemas → Treinar Deep Learning (DL)
 - **Overfitting** → muitos parâmetros, pois explora um espaço muito maior de funções



Soluções

- Usar métodos de regularização melhores:
 1. treinamento não-supervisionado
 2. treinamento com *dropout* estocástico
 3. utilização de outros algoritmos de aprendizado (estimadores)

Soluções

- Usar métodos de regularização melhores:

- 1 treinamento não-supervisionado
- 2 treinamento com *dropout* estocástico
- 3 utilização de outros algoritmos de aprendizado
(estimadores)

1. Pré-treino

- Inicializar as camadas escondidas → aprendizado não-supervisionado
 - força a rede a aprender a estrutura da distribuição dos dados de entrada
 - encoraja as camadas escondidas a codificar essa estrutura

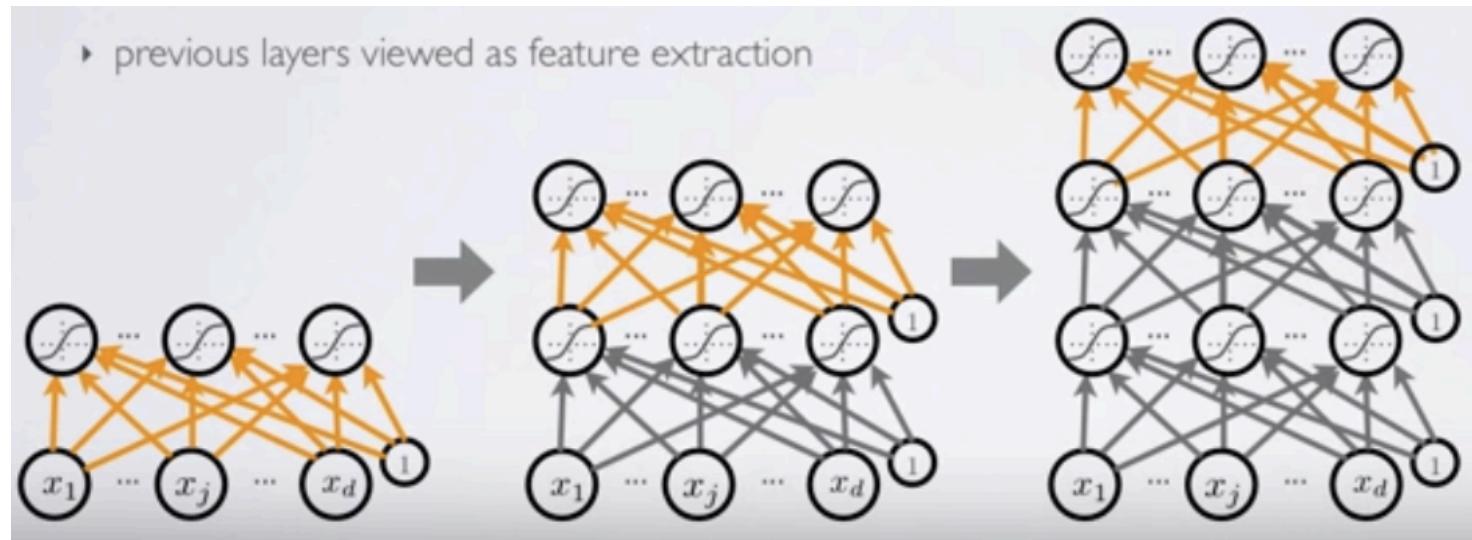
Pré-treino: inicializar os parâmetros de uma região tal que haja menos *overfitting*

1. Pré-treino

- **Estratégia:** procedimento guloso camada a camada
 - treina uma camada por vez, da primeira até a última, utilizando aprendizado não-supervisionado
 - Cada camada ajusta os parâmetros das camadas anteriores
 - Camadas anteriores → extractores de características

1. Pré-treino

- **Estratégia:** procedimento guloso camada a camada



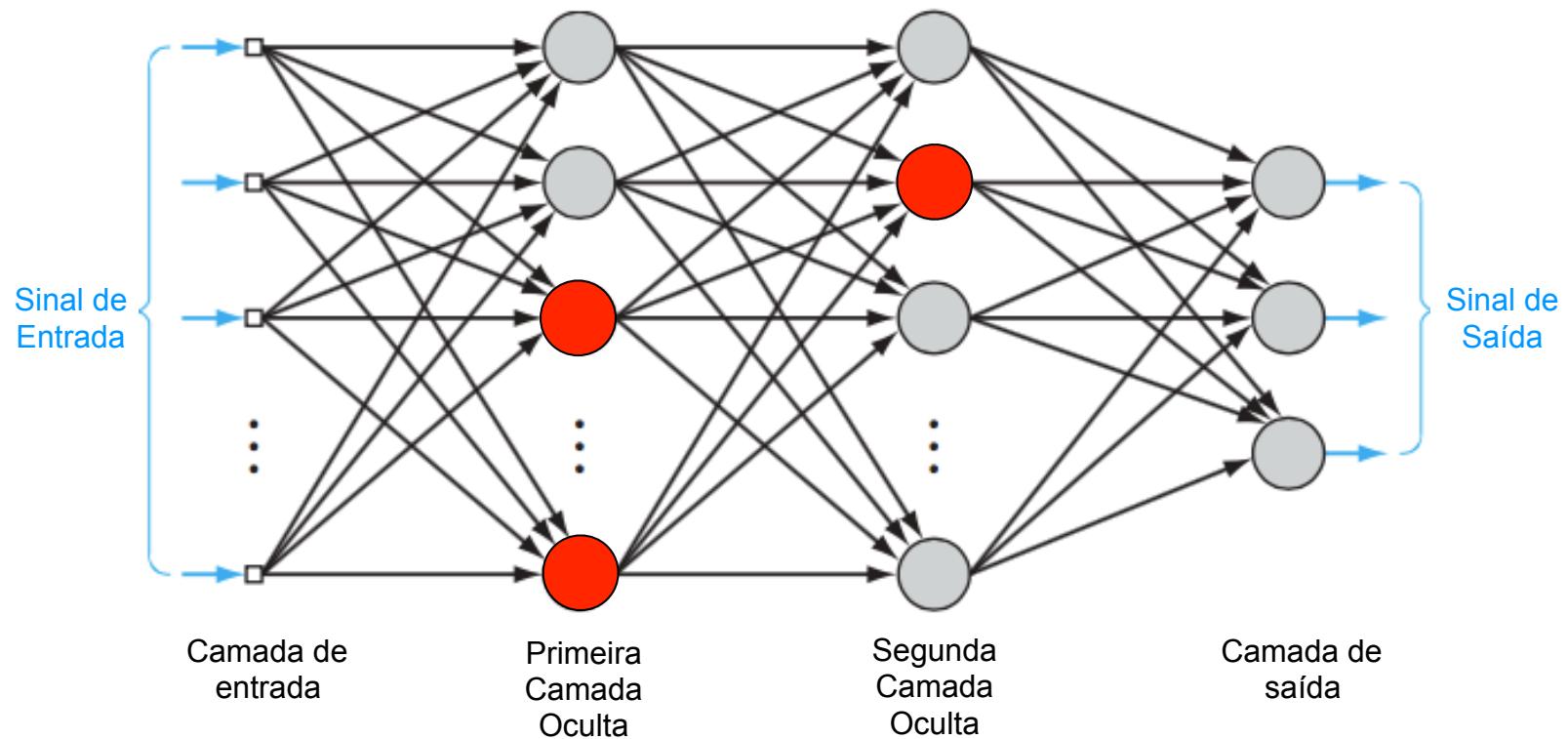
Soluções

- Usar métodos de regularização melhores:
 1. treinamento não-supervisionado
 - 2** treinamento com *dropout* estocástico
 3. utilização de outros algoritmos de aprendizado (estimadores)

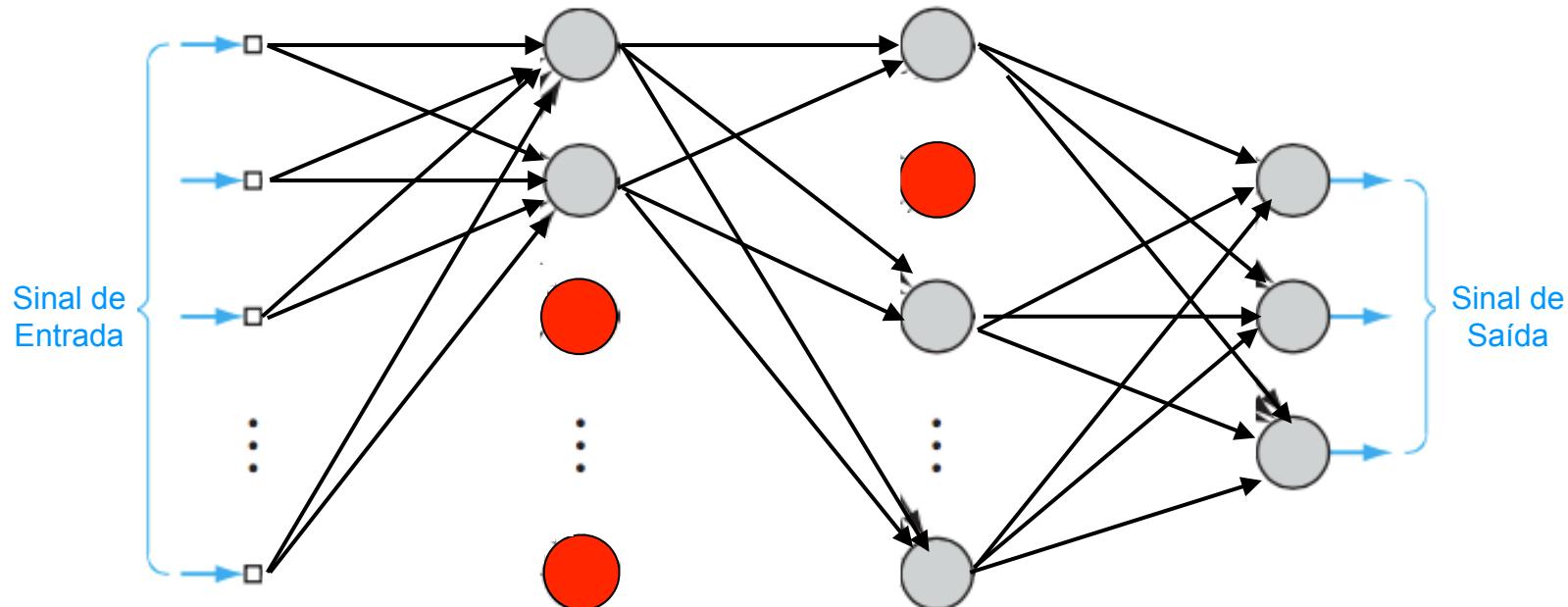
2. Dropout

- **Ideia:** remover aleatoriamente neurônios das camadas ocultas
 - cada neurônio escondido é zerado com probabilidade igual a p ($p = 0.5$)
- **Resultado**
 - neurônios escondidos não podem se coadaptar a outros neurônios escondidos
 - cada neurônio escondido é迫使ado a extrair características mais gerais

2. Dropout



2. Dropout



Neurônios selecionados aleatoriamente (vermelho) são **removidos** da rede

Soluções

- Usar métodos de regularização melhores:
 1. treinamento não-supervisionado
 2. treinamento com *dropout* estocástico
 3. utilização de outros algoritmos de aprendizado (estimadores)

3. Outros estimadores

- **Adagrad: Adaptive Subgradient (Duchi et al, 2011)**
 - ajusta a taxa de aprendizado para cada parâmetro, de acordo com os gradientes das iterações anteriores
 - Divide o gradiente atual na atualização pela soma dos gradientes anteriores
 - Quanto mais atualizações um parâmetro (peso) receber, menor é a atualização

3. Outros estimadores

- **AdaDelta (Zeiler, 2012)**
 - extensão do Adagrad, utiliza uma janela de gradientes ao invés da acumulação de todos os gradientes anteriores
 - aprendizado continua mais do que no Adagrad, mesmo quando muitas atualizações já foram aplicadas
- **Adam (Kingma & Ba, 2014)**
 - Combinação do Adagrad e do RMSProp
 - faz decremento dos ***momentums*** de primeira e segunda ordem

Roteiro

- 1 Introdução
- 2 *Deep Learning (DL)*
- 3 ***Convolutional Neural Networks (CNNs)***
- 4 Modelagem de CNNs
- 5 Hands on / Exemplos
- 6 Síntese / Próximas Aulas
- 7 Referências

CNNs

- **Redes Neurais Convolucionais (CNNs)**
 - surgiram a partir de estudos realizados com o cérebro, na década de 1980 → córtex visual
 - Porém só nos últimos anos tivemos avanços “sobre-humanos” em tarefas visuais complexas → poder/avanço computacional
- Tarefas:
 - reconhecimento/classificação de imagens
 - processamento de linguagem natural
 - reconhecimento de voz

CNNs

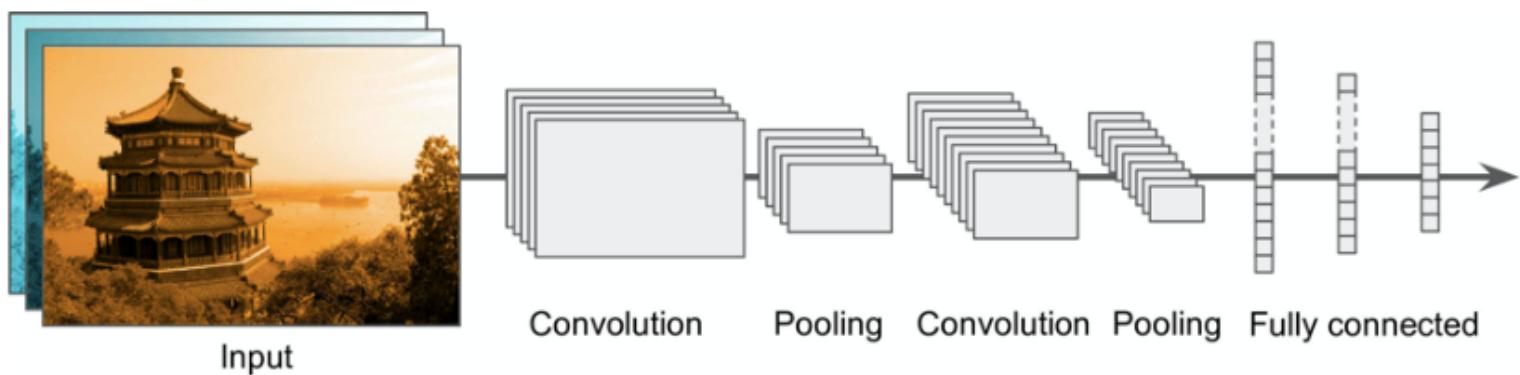


Figura de: Aurélien Gerón (2019)

CNNs

Primeiro Vislumbre. Mas porque tem essa estrutura? Como chegamos até esse tipo de RNAs?

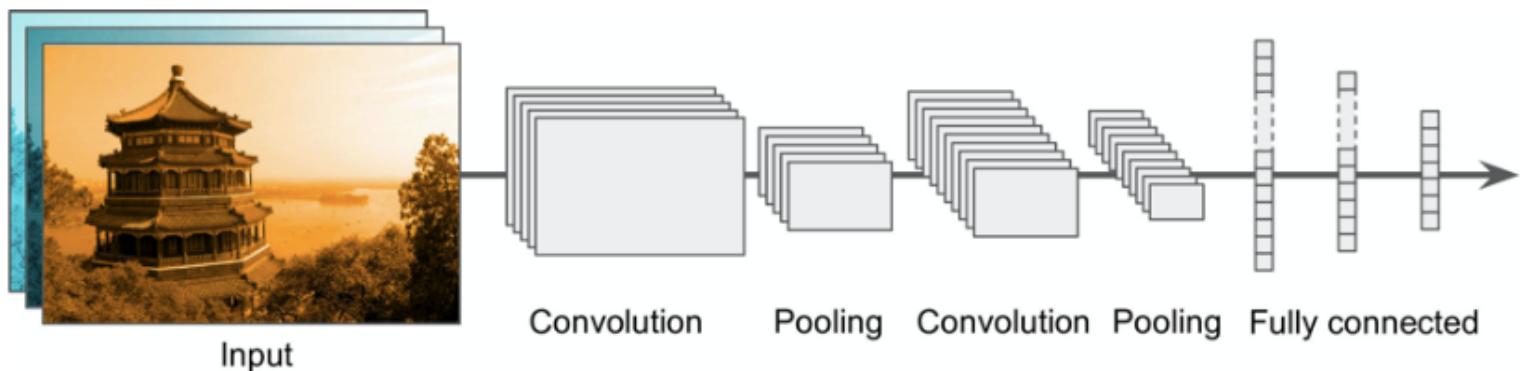


Figura de: Aurélien Gerón (2019)

CNNs

- **Inscrição: córtex visual**



CNNs

□ Inspiração: córtex visual

- trabalhos de *David H Hubel & Torsten Wiesel* (1958-59)
- experimentos com gatos e macacos
 - neurônios possuem um **pequeno campo receptivo local**
 - reagem a **estímulos visuais** localizados em uma **região limitada** do campo visual



CNNs

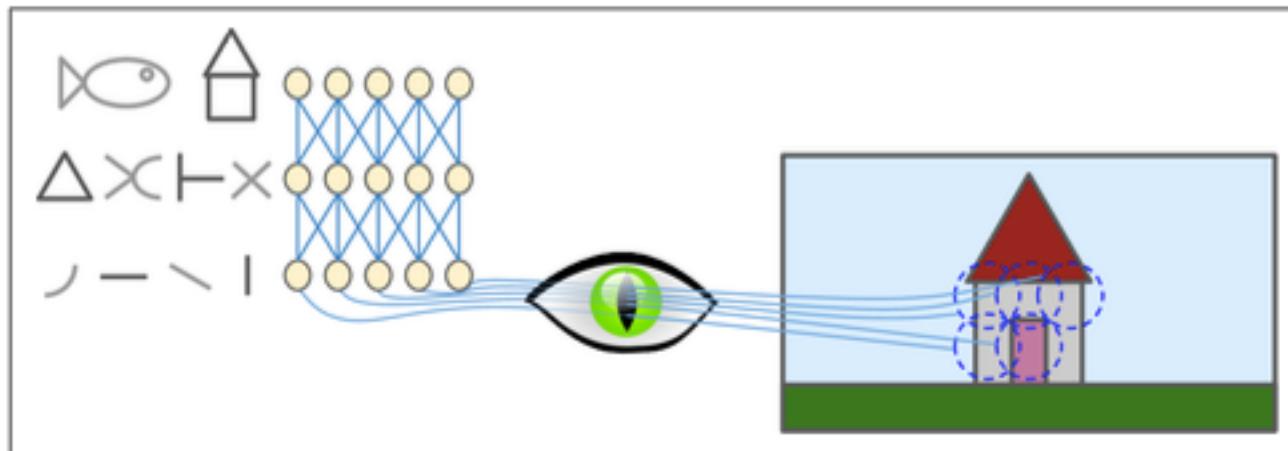


Figura de: Aurélien Gerón (2019)

CNNs

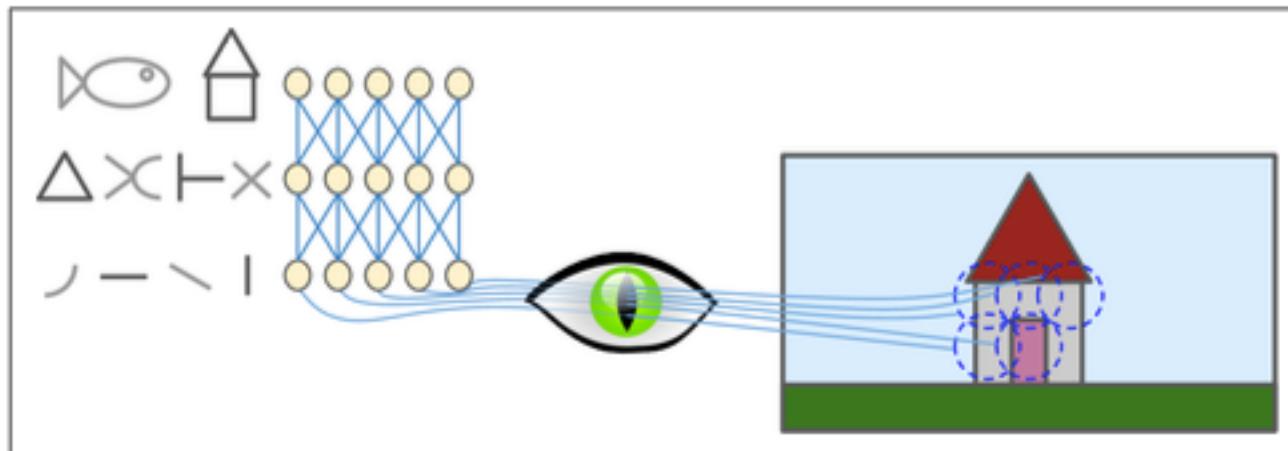


Figura de: Aurélien Gerón (2019)

- alguns neurônios reagem à linhas horizontais/verticais/diagonais, etc
- alguns neurônios tem campo de percepção maior que os outros, e reagem a padrões mais complexos (combinação de padrões simples)
- há uma região de inserção entre o campo de percepção de diferentes neurônios

CNNs

□ Redes Neurais Convolucionais (CNNs)

- 1º modelo é de 1980 (*neocognitron*)
- 1988 → duas novas estruturas
 - camadas convolucionais
 - poolings

Camadas convolucionais

- **Redes Neurais Convolucionais (CNNs)**

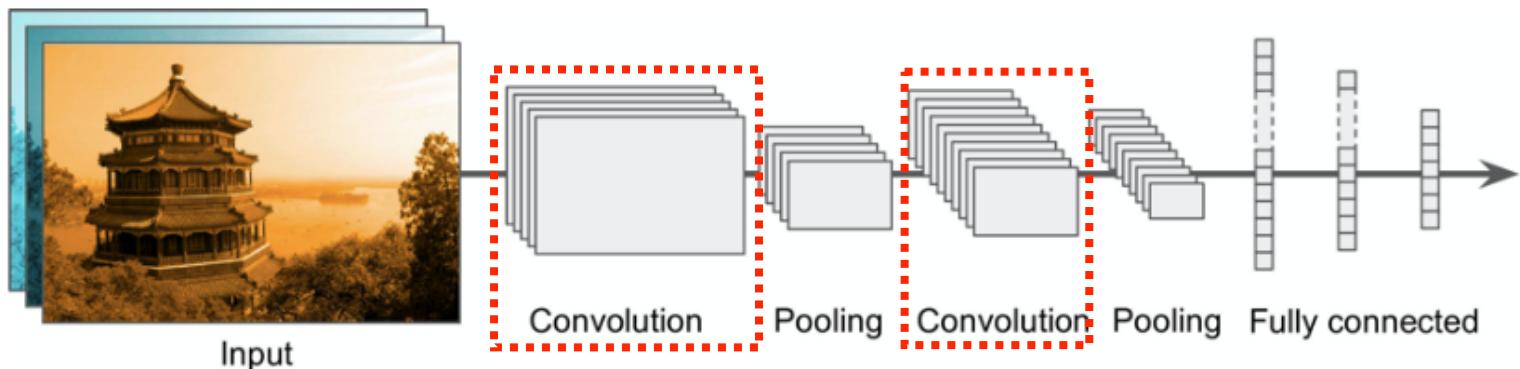


Figura de: Aurélien Géron (2019)

Camadas convolucionais

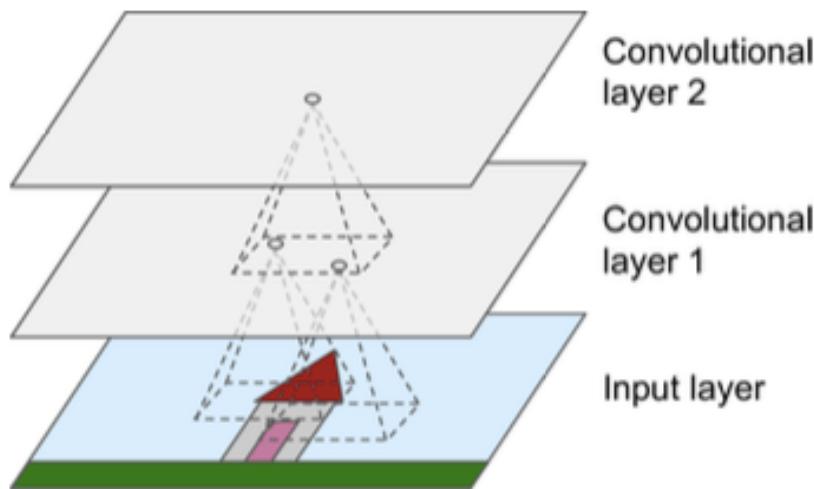
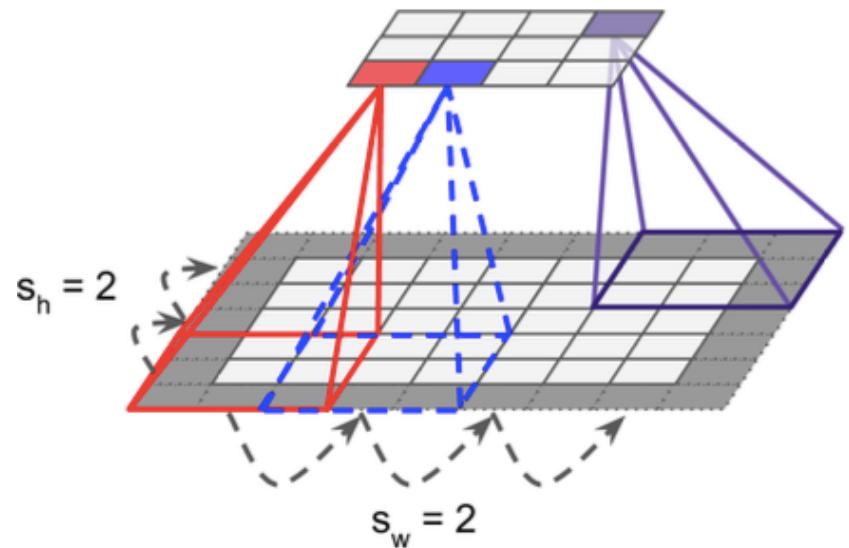
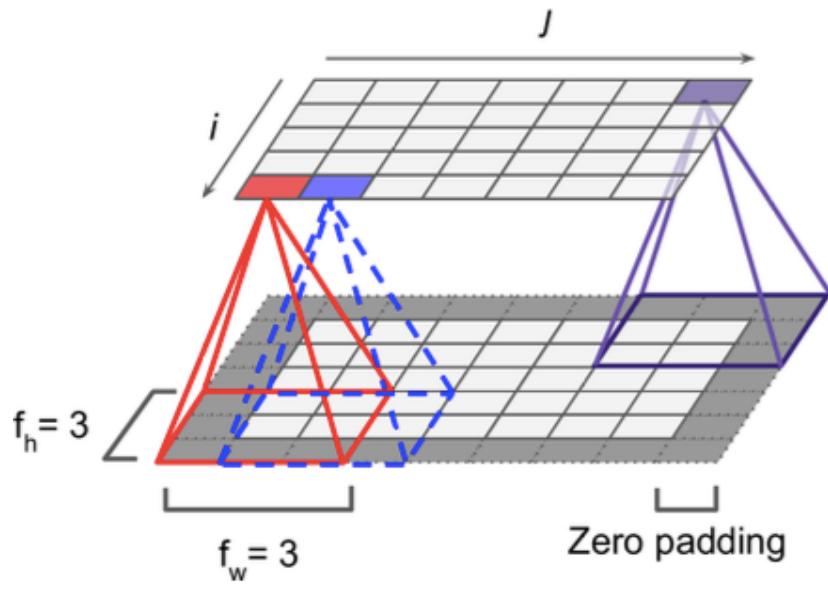


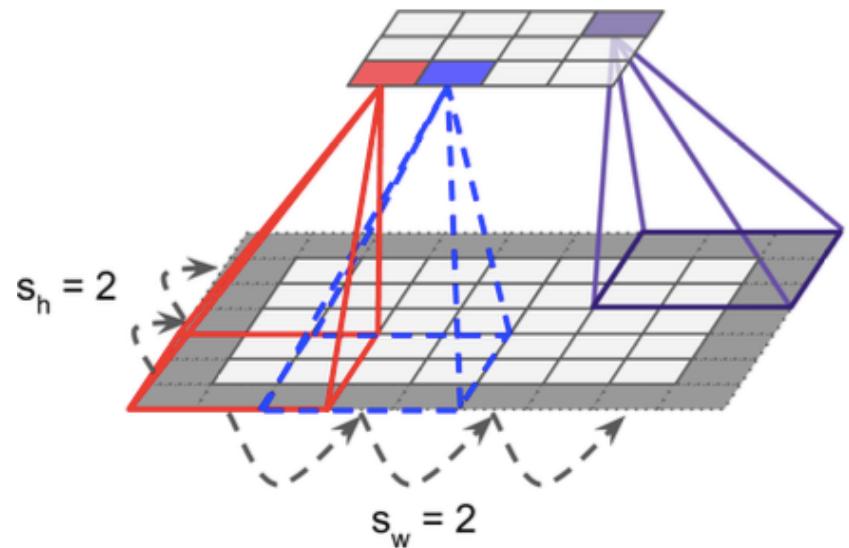
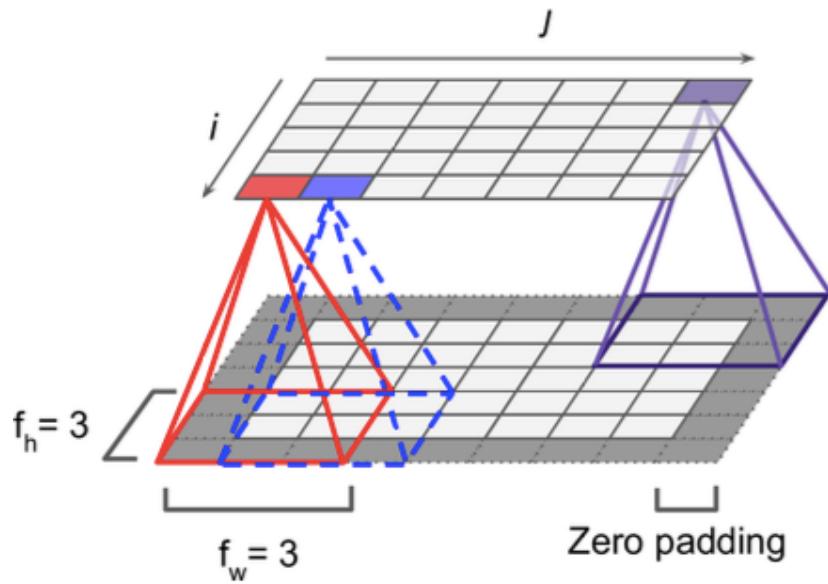
Figura de: Aurélien Gerón (2019)

- neurônios da 1a camada são conectados a alguns pixels de seus campos receptivos
 - não são totalmente conectados (como na MLP)
- neurônios da 2a camada são conectados a poucos neurônios localizados em um pequena região da camada 1
- kernel ($f_w \times f_h$)

Camadas convolucionais

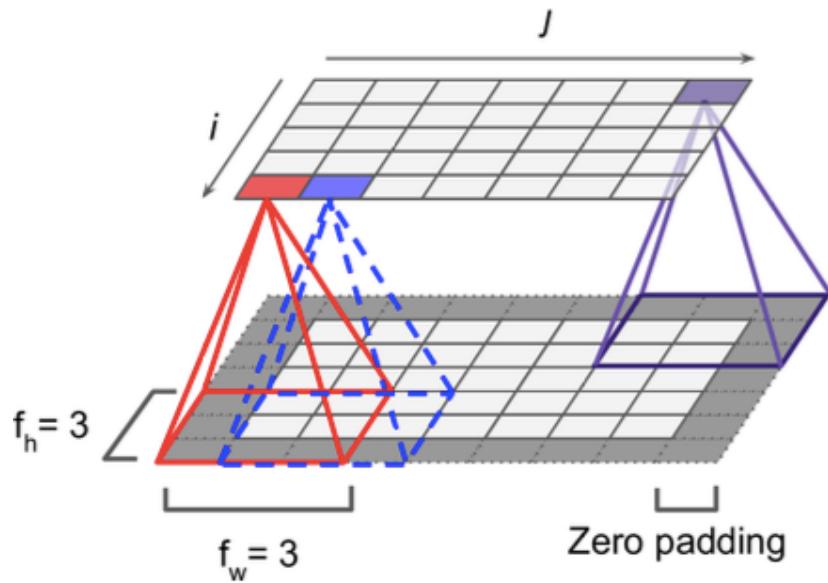


Camadas convolucionais

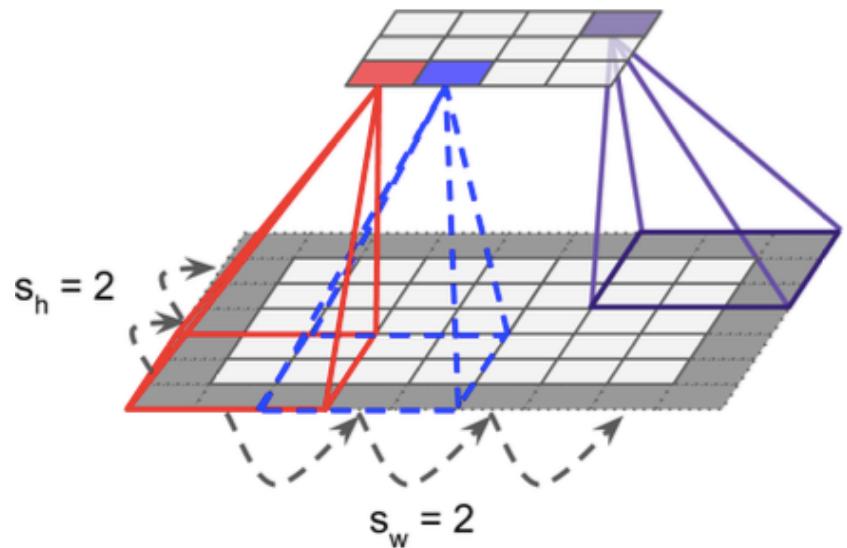


1. neurônio i,j está conectado nos output da camada prévia (*kernel* - 3×3)

Camadas convolucionais



1. neurônio i,j está conectado nos output da camada prévia (*kernel* - 3×3)



2. stride (deslocamento)
Neurônios na borda aplicam **padding**

Filtros

- Pesos dos neurônios formam pequenas imagens do tamanho do campo de percepção
- estas imagens funcionam como **filtros**, detectando padrões simples
- durante o treinamento a camada aprende quais filtros são os mais úteis, e as demais camadas vão combinando estes filtros para encontrar padrões mais complexos

Filtros

imagem com padrões verticais ressaltados

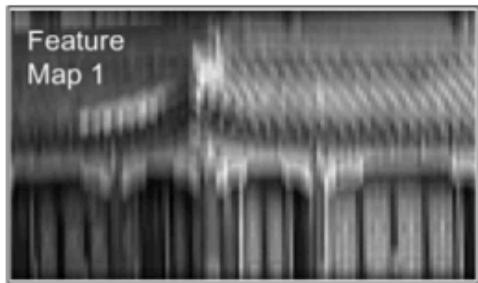
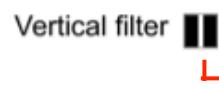


imagem com padrões horizontais ressaltados



Vertical filter



Horizontal filter



filtros
(imagens)



imagem original

Múltiplos filtros

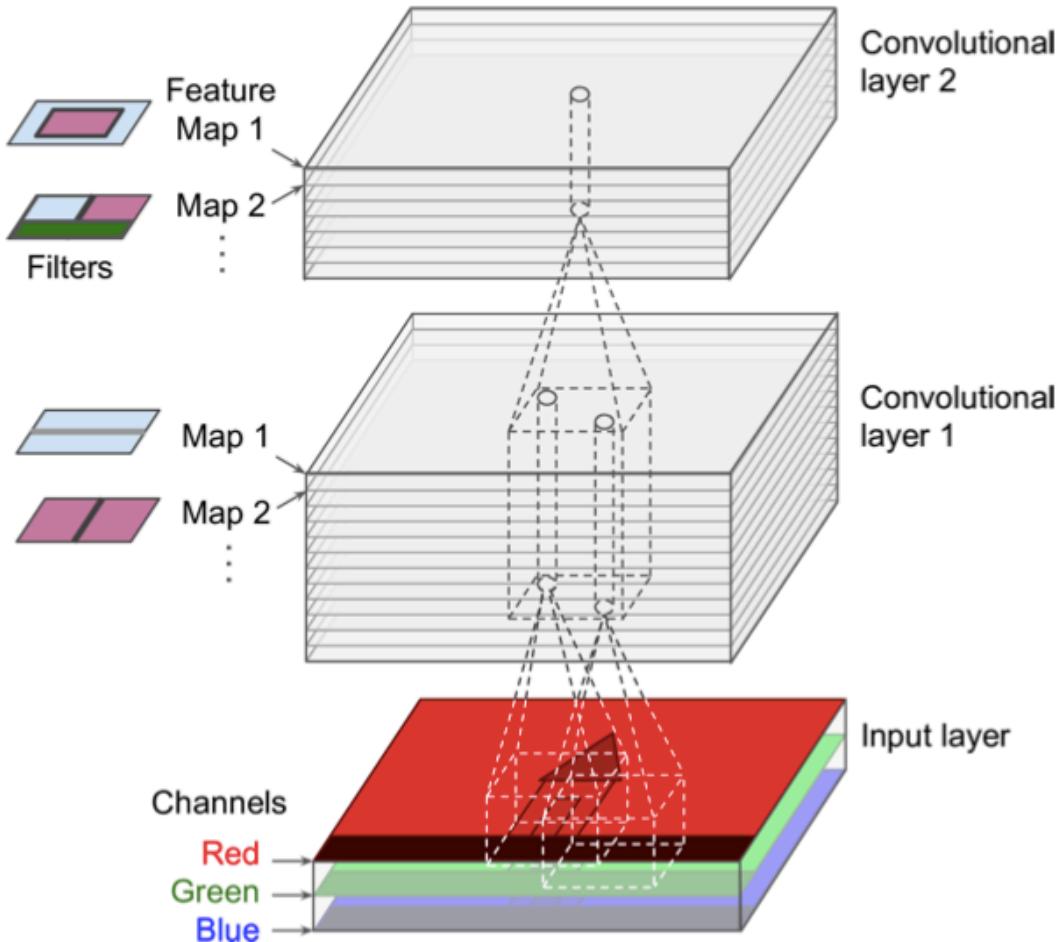
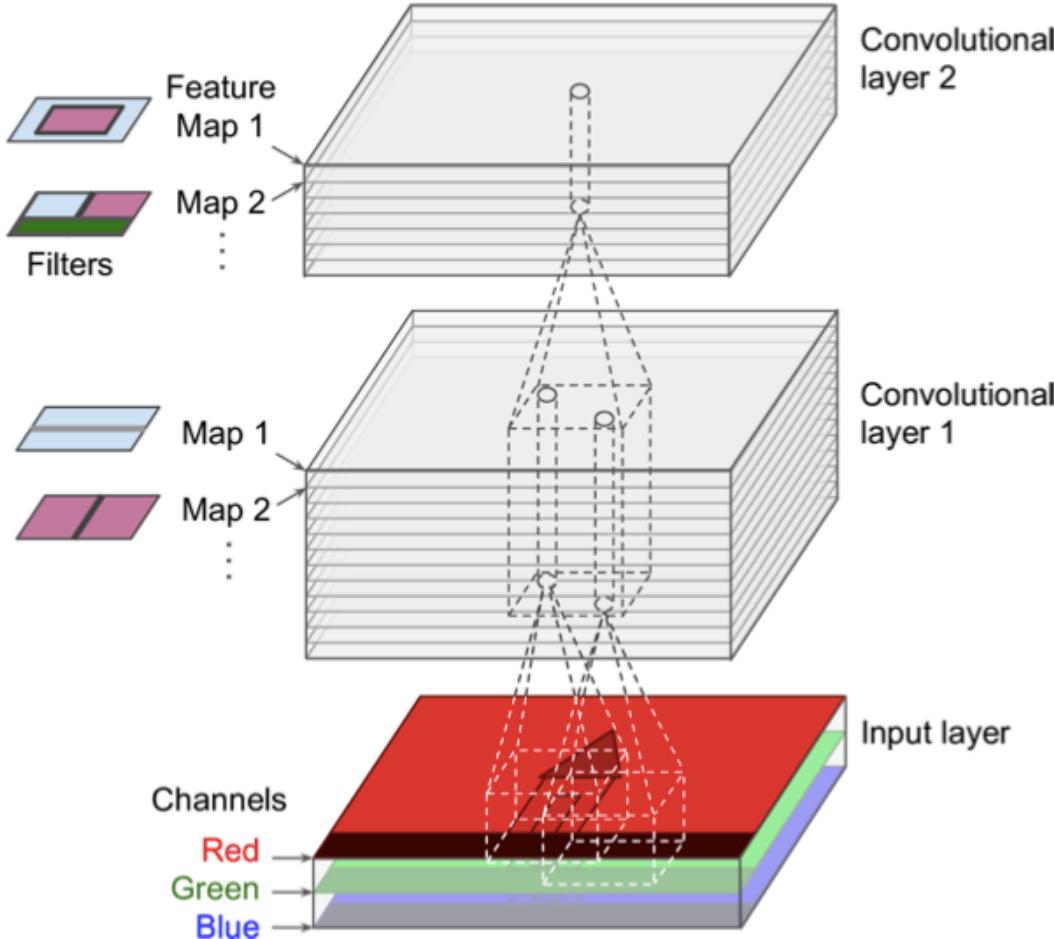


Figura de: Aurélien Gerón (2019)

Múltiplos filtros



- Mas ... saída de cada camada convolucional tem **múltiplos filtros** (matriz 3D)
- 1 neurônios para cada filtro (*feature map*)
- todos os neurônios de um mesmo mapa compartilham os mesmos parâmetros (pesos, bias)
- campo de percepção do neurônio se estende a todos filtros

Figura de: Aurélien Gerón (2019)

Múltiplos filtros

□ Resumindo:

- cada camada convolucional aplica simultaneamente múltiplos filtros treináveis às suas entradas (*inputs*), sendo capaz de identificar múltiplas características em qualquer lugar
- Imagens → RGB (3 canais)
 - cinza = média desses valores
 - $\text{cinza} = (R + G + B)/3$

Camadas de Pooling

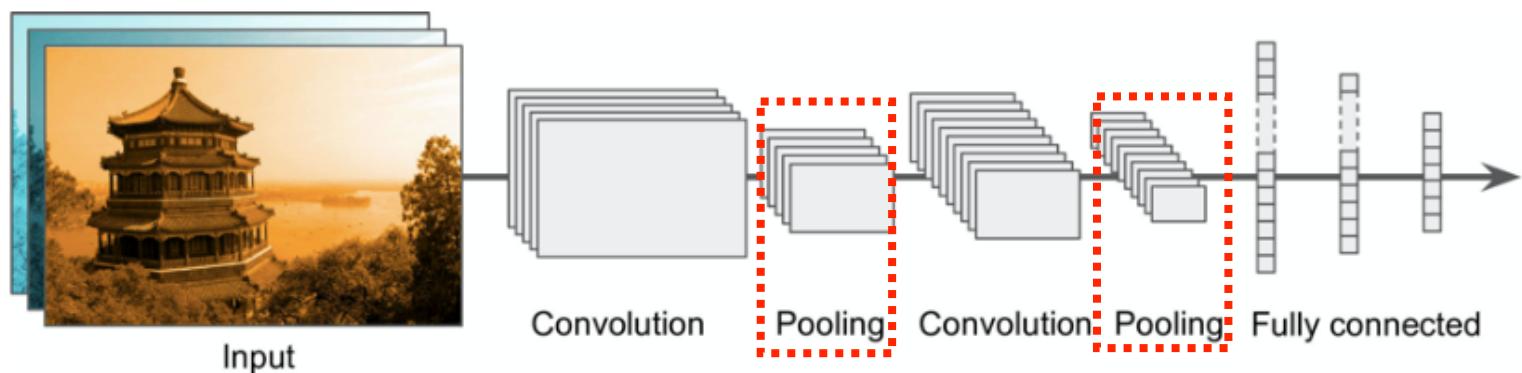


Figura de: Aurélien Gerón (2019)

Camadas de Pooling

- **Objetivo:** reduzir a imagem de entrada na camada (sub-amostragem/*subsample*)
 - Reduz o custo computacional, memória, e parâmetros no modelo (pesos)
 - Cada neurônio é conectado a um campo de visão limitado (kernel)
 - Camada de *pooling* **não tem pesos** !
 - Agregação dos valores (máximo ou média)

Camadas de Pooling

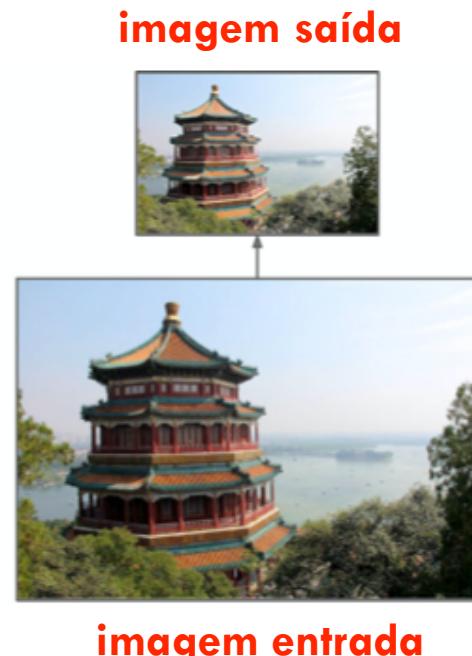
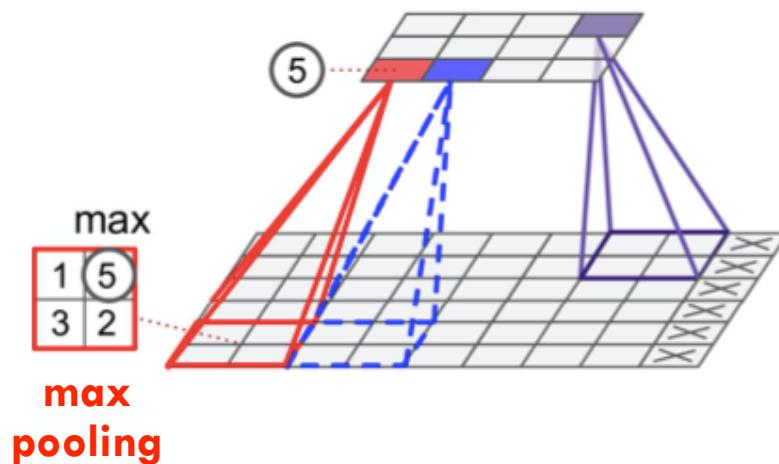


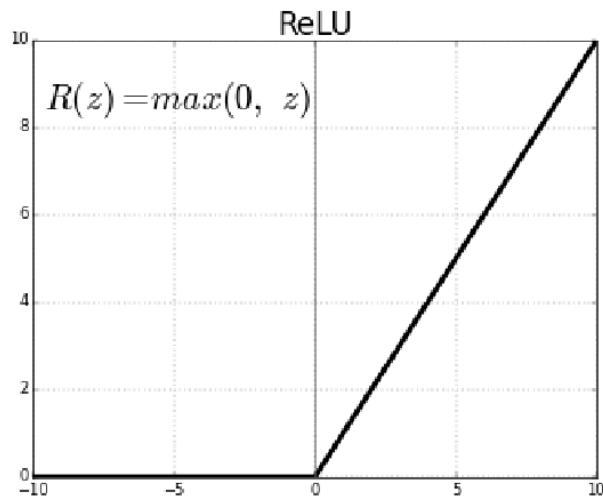
Figura de: Aurélien Gerón (2019)

Roteiro

- 1 Introdução
- 2 *Deep Learning (DL)*
- 3 *Convolutional Neural Networks (CNNs)*
- 4 Modelagem de CNNs
- 5 Hands on / Exemplos
- 6 Síntese / Próximas Aulas
- 7 Referências

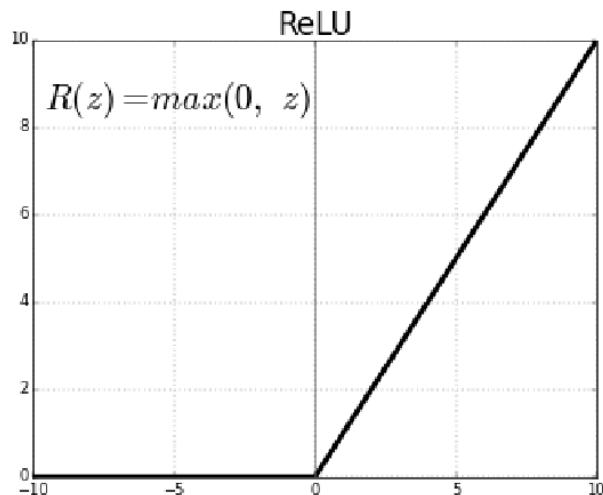
Funções de Ativação

- Rectifier Linear Units (ReLU) → camadas intermediárias



Funções de Ativação

- Rectifier Linear Units (ReLU) → camadas intermediárias



- Acelera a convergência do gradiente
- previne o desaparecimento do gradiente
- adiciona **não-linearidade** no processo de treinamento

$$R(z) = \max(0, z)$$

Arquiteturas

- Arquitetura geral
 - Poucas camadas convolucionais geralmente seguidas de *poolings*
 - mais camadas convolucionais, seguidas *poolings*, etc
 - o final, uma rede *feedforward* totalmente conectada é adicionada, com 2 camadas densas (MLP)
 - camadas intermediarias → ReLU
 - camada final → softmax/sigmoidal (probabilidades das classes)
- Conforme o sinal é propagado a imagem fica menor e menor

Arquiteturas

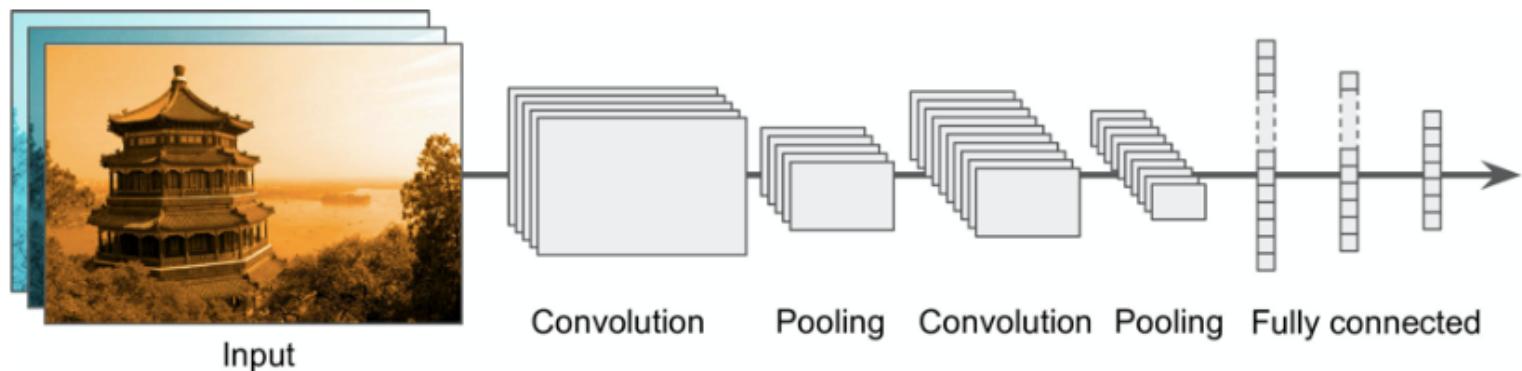


Figura de: Aurélien Gerón (2019)

Arquiteturas

**camada convolucional
seguida de pooling**

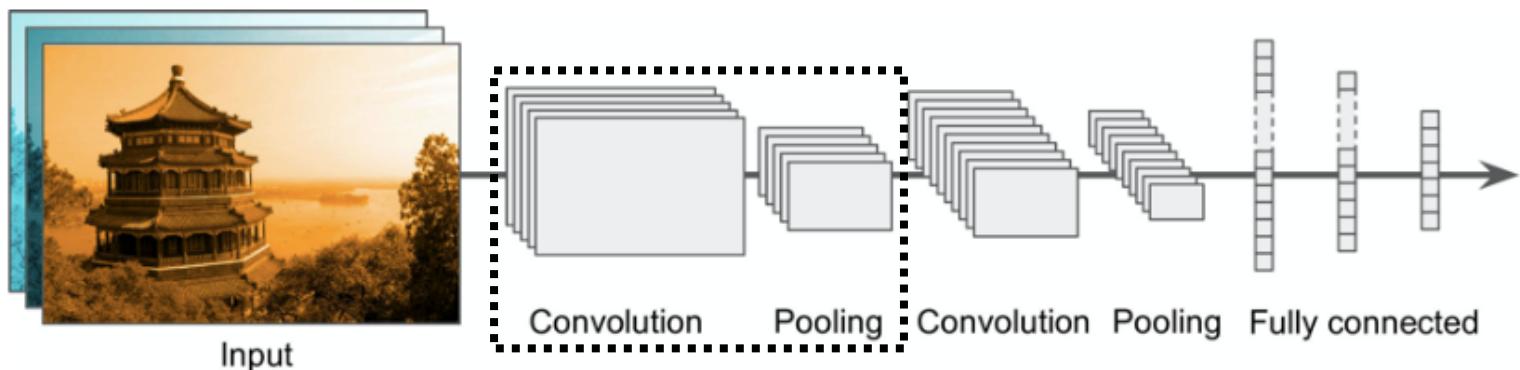


Figura de: Aurélien Gerón (2019)

Arquiteturas

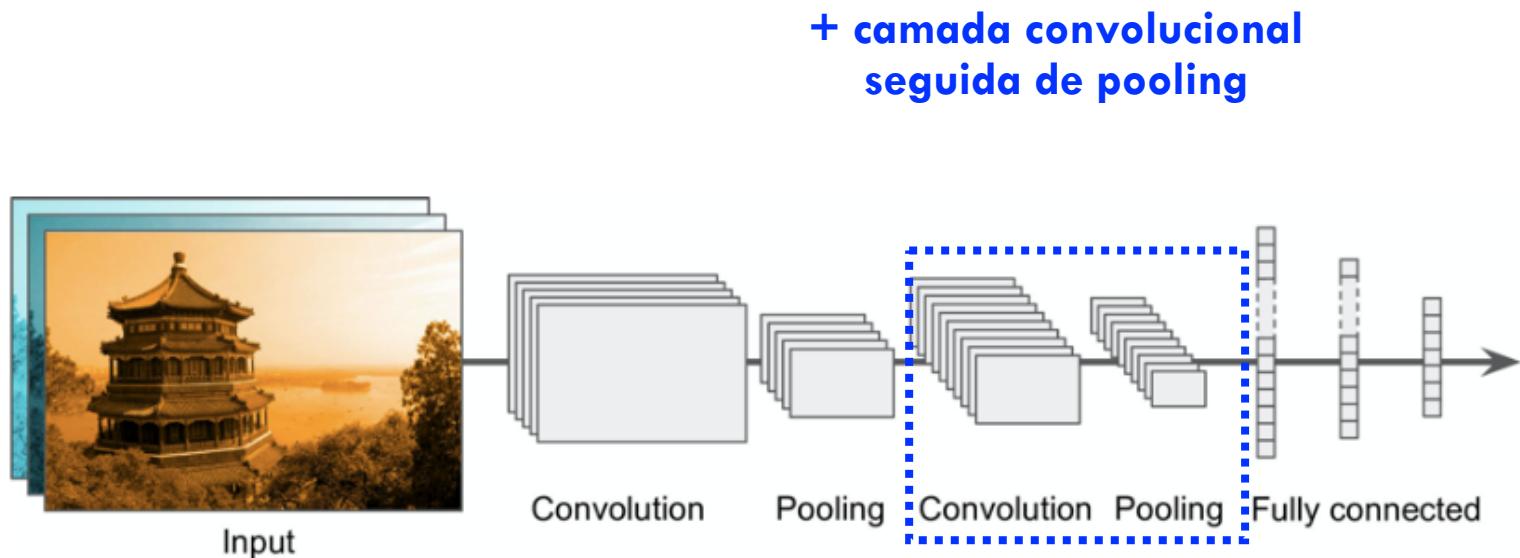


Figura de: Aurélien Gerón (2019)

Arquiteturas

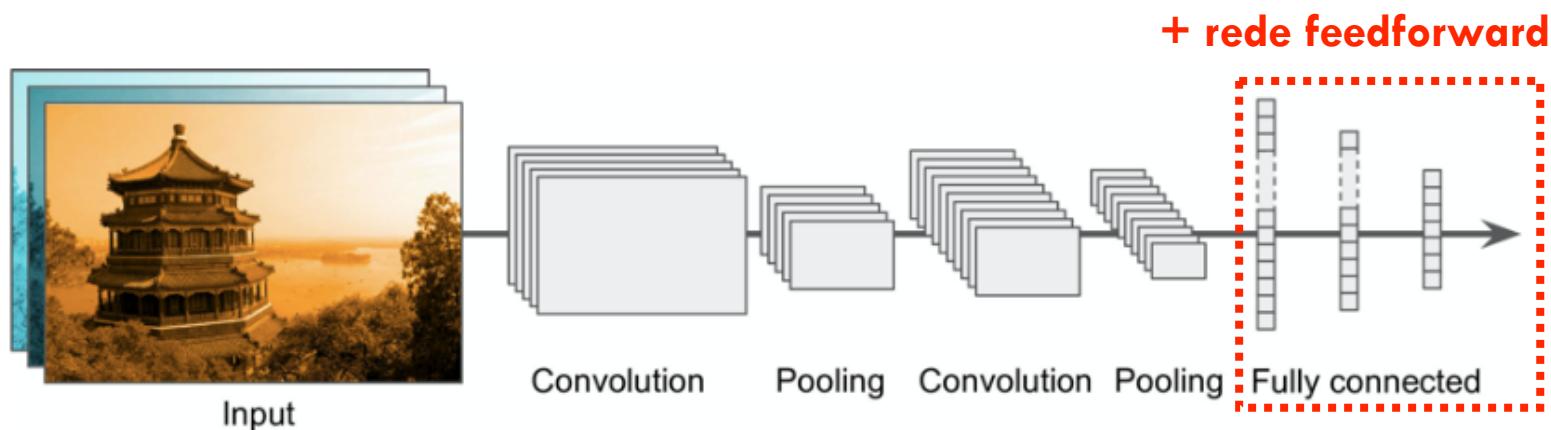


Figura de: Aurélien Gerón (2019)

Arquiteturas

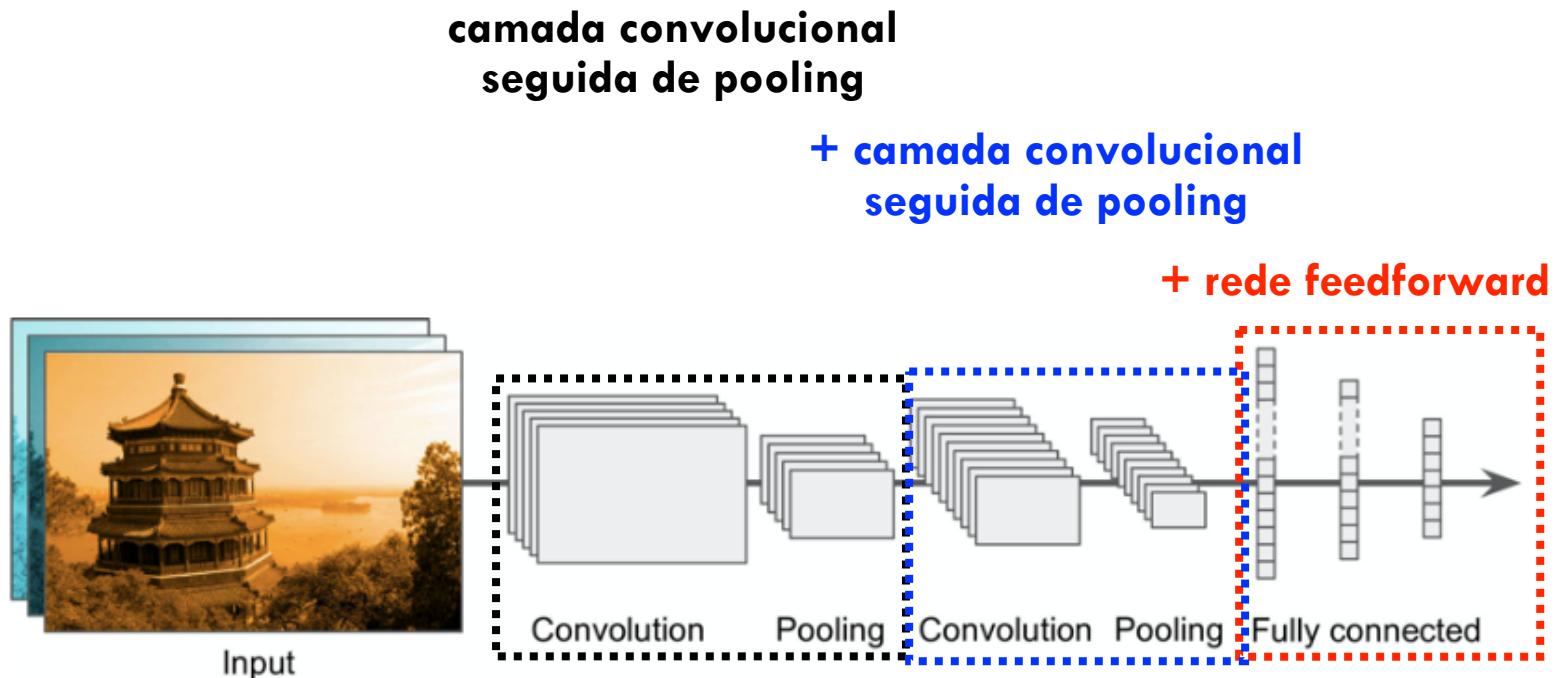


Figura de: Aurélien Gerón (2019)

Arquiteturas

- Ao longo das camadas o número de filtros aumenta
 - poucas formas básicas (usamos menos filtros)
 - mais formas de combinar as formas básicas (usamos mais filtros)
 - **Prática comum:** dobrar o número de filtros depois de *pooling*
- **Flatten:** achatar/condensar o sinal
 - Camadas densas necessitam de sinais 1D (vetores)
 - Convertemos imagens em (2 ou 3 dimensões) em arrays que alimentam a rede feedforward

LeNet-5

- Uma das arquiteturas de CNN mais conhecidas na [literatura](#)
 - criada por Yann LeCun (1988)
 - usada para classificação de dígitos escritos a mão (MNIST)

Layer	Type	Maps	Size	Kernel size	Stride	Activation
Out	Fully Connected	–	10	–	–	RBF
F6	Fully Connected	–	84	–	–	tanh
C5	Convolution	120	1×1	5×5	1	tanh
S4	Avg Pooling	16	5×5	2×2	2	tanh
C3	Convolution	16	10×10	5×5	1	tanh
S2	Avg Pooling	6	14×14	2×2	2	tanh
C1	Convolution	6	28×28	5×5	1	tanh
In	Input	1	32×32	–	–	–

LeNet-5

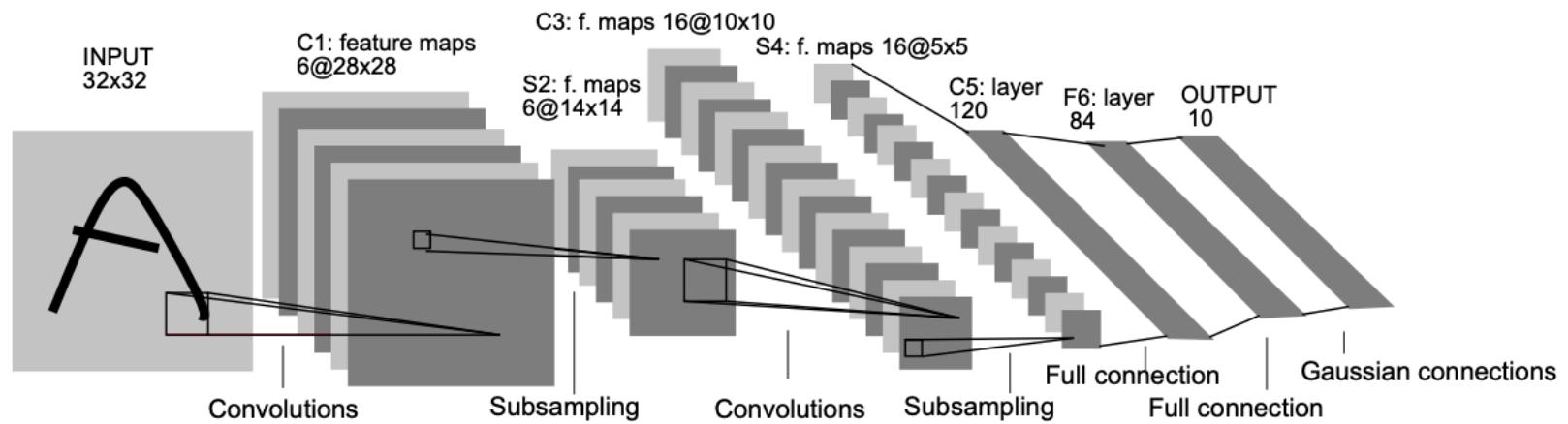


Figura de: [Y. LeCun et al \(1988\)](#)

LeNet-5

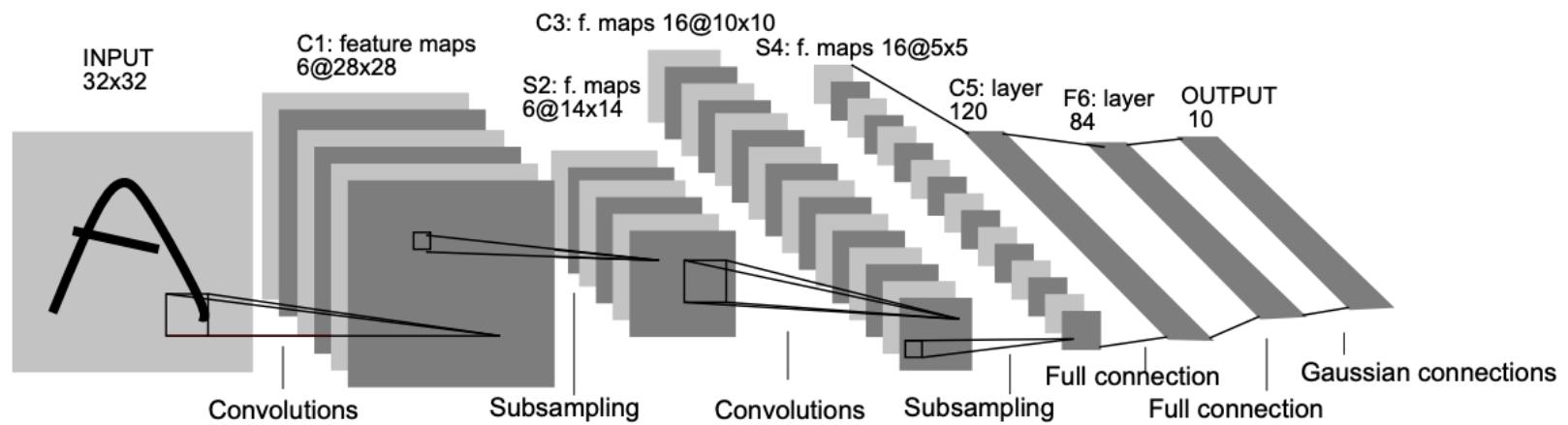


Figura de: [Y. LeCun et al \(1988\)](#)

MNIST: imagens 28×28 , adicionado zero-padded criando imagens 32×32 , posteriormente normalizadas antes de alimentar a rede

Arquiteturas

- Outras arquiteturas famosas

- AlexNet
- GoogLeNet
- VGGNet
- ResNet
- Xception
- ...

Roteiro

- 1 Introdução
- 2 *Deep Learning (DL)*
- 3 *Convolutional Neural Networks (CNNs)*
- 4 Modelagem de CNNs
- 5 Hands on / Exemplos
- 6 Síntese / Próximas Aulas
- 7 Referências

Hands on

CO CNN_Keras_example1_CIFAR10.ipynb ☆

File Edit View Insert Runtime Tools Help Last saved at 1:44 PM

+ Code + Text

```
# e as mostraremos em um grid 5 x 6 - 5 linhas, 6 colunas
par(mfcol = c(5,6), mar = rep(1, 4), oma = rep(0.2, 4))
cifar$train$x[index,,,] %>%
  purrr::array_tree(1) %>%
  purrr::set_names(class_names[cifar$train$y[index] + 1]) %>%
  purrr::map(as.raster, max = 255) %>%
  purrr::iwalk(~{plot(.x); title(.y)})
```

The image shows a 5x6 grid of 30 small square images, each labeled with its corresponding class name below it. The classes represented are: frog, automobile, deer, truck, deer, frog; truck, bird, horse, truck, cat, cat; truck, horse, horse, cat, frog, dog; deer, ship, bird, bird, frog, deer; automobile, cat, truck, frog, bird, airplane. The images are arranged in five rows and six columns.

Seria interessante analisar também qual é a distribuição das classes do problema. Podemos então gerar um histograma para verificar a frequência de cada uma das classes, tanto no conjunto de treinamento (`q1`) como no conjunto de teste (`q2`).

Hands on

CNN_Keras_example1_CIFAR10.ipynb

File Edit View Insert Runtime Tools Help Last saved at 1:44 PM

+ Code + Text

```
# e as mostraremos em um grid 5 x 6 - 5 linhas, 6 colunas
par(mfcol = c(5,6), mar = rep(1, 4), oma = rep(0.2, 4))
cifar$train$x[index,,,] %>%
  purrr::array_tree(1) %>%
  purrr::set_names(class_names[cifar$train$y[index] + 1]) %>%
  purrr::map(as.raster, max = 255) %>%
  purrr::iwalk(~{plot(.x); title(.y)})
```

frog automobile deer truck deer frog

cat dog deer airplane

Vamos exercitar :)

truck

horse

horse

cat

frog

deer

ship

bird

bird

automobile

cat

truck

frog

bird

deer

airplane

Seria interessante analisar também qual é a distribuição das classes do problema. Podemos então gerar um histograma para verificar a frequência de cada uma das classes, tanto no conjunto de treinamento (`q1`) como no conjunto de teste (`q2`).

Roteiro

- 1 Introdução
- 2 *Deep Learning (DL)*
- 3 *Convolutional Neural Networks (CNNs)*
- 4 Modelagem de CNNs
- 5 Hands on / Exemplos
- 6 Síntese / Próximas Aulas
- 7 Referências

Síntese

- **Deep Learning**
 - modelos com muitas (muitas) camadas
 - processamento em vários níveis
 - Treinamento envolve:
 - pré-treino não-supervisionado
 - dropouts
 - diferentes estimadores (algoritmos de aprendizado)
 - aplicações

Síntese

- **CNNs**

- processamento de imagens
- camadas convolucionais e *Pooling*
- ReLu e Softmax
- Flatten + Dropouts, etc ...

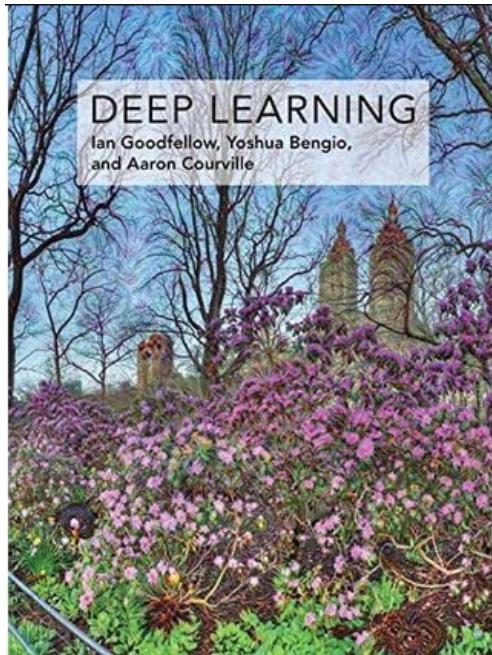
Próxima aula

- Visão geral sobre LSTMs
 - redes recorrentes
 - unidades mais complexas (baseadas em circuitos)
- N aplicações
 - manipulação de séries temporais

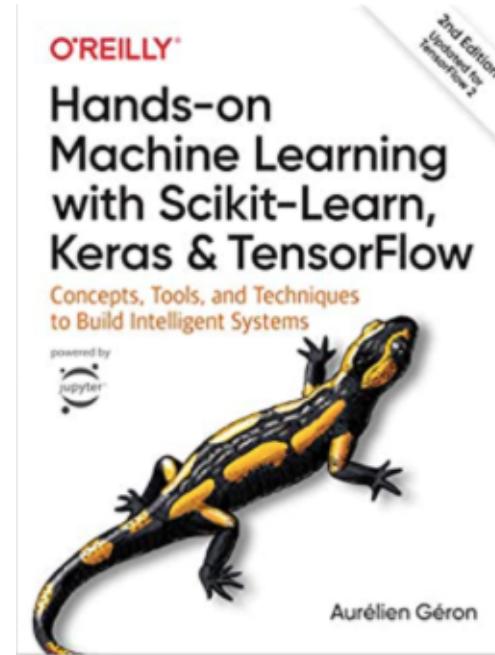
Roteiro

- 1 Introdução
- 2 *Deep Learning (DL)*
- 3 *Convolutional Neural Networks (CNNs)*
- 4 Modelagem de CNNs
- 5 Hands on / Exemplos
- 6 Síntese / Próximas Aulas
- 7 Referências

Literatura Sugerida



(Goodfellow, Bengio, Courville; 2015)



(Géron, 2019)

Literatura Sugerida

- MIT book: <http://www.deeplearningbook.org>
- Deep Learning: <http://deeplearning.net>
- Andrew Ng: <https://www.deeplearning.ai>

Literatura Complementar

- Coursera: <https://www.coursera.org/specializations/deep-learning>
- Google AI: <https://ai.google/education/>
- Keras: <https://keras.io>
- Auto-Keras: <https://autokeras.com>
- h2o: <http://docs.h2o.ai/h2o/latest-stable/h2o-docs/index.html#>

Literatura Complementar

- Aulas de Hugo Larrochelle: <https://www.youtube.com/watch?v=vXMpKYRhpml>
- DL + MK: <https://blog.mgechev.com/2018/10/20/transfer-learning-tensorflow-js-data-augmentation-mobile-net/>
- Hide Screen: <https://github.com/Hironsan/BossSensor>

Perguntas?

Prof. Rafael G. Mantovani

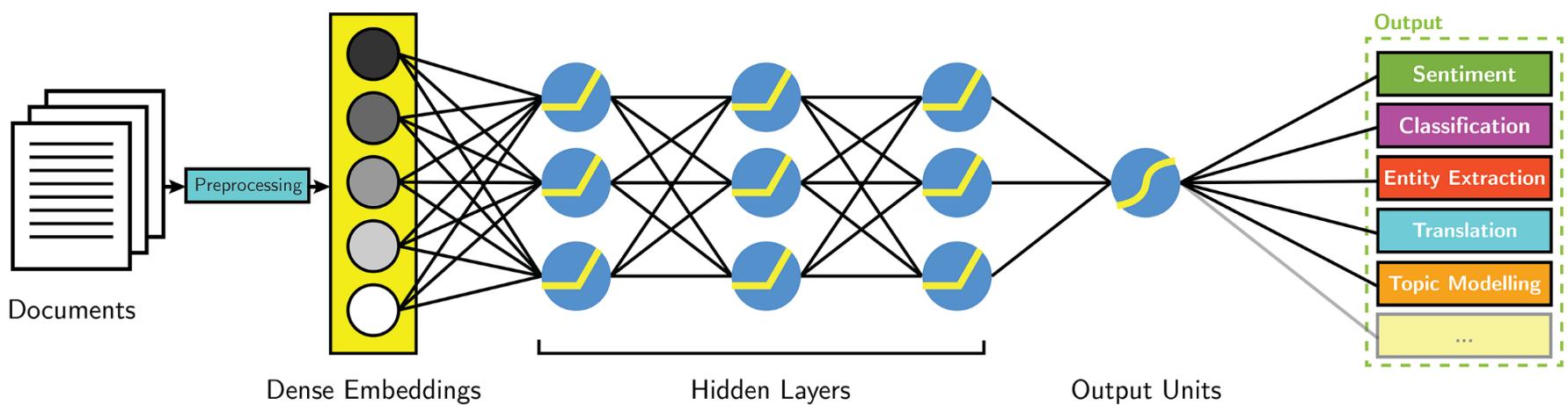
rgmantovani@gmail.com



Material complementar

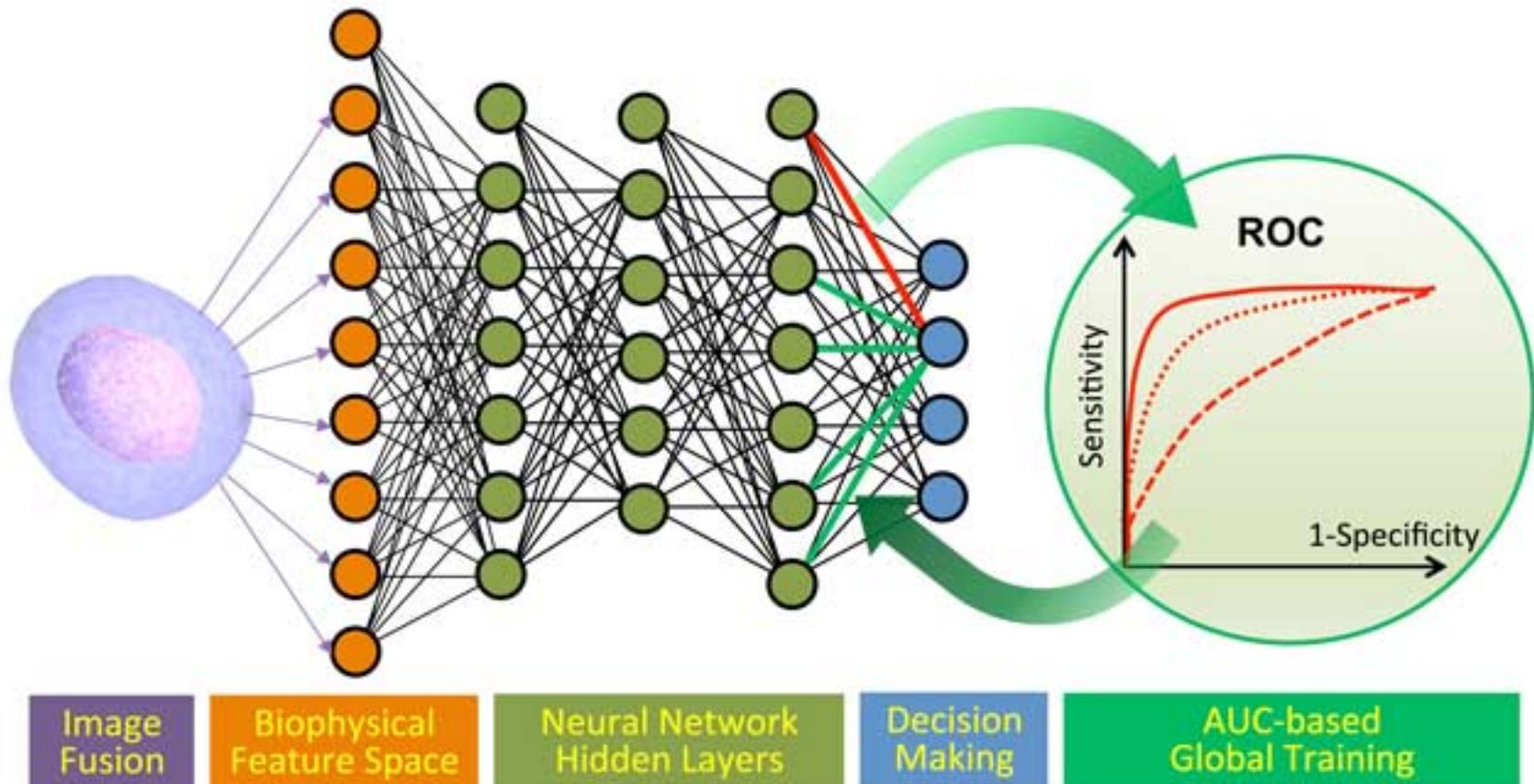
Exemplos de aplicações de DL

- Detecção de sentimentos:



Exemplos de aplicações de DL

- Diagnóstico de cancer:



Exemplos de aplicações de DL

- Redução de dimensionalidade (Autoencoders)

Reducing the Dimensionality of Data with Neural Networks

G. E. Hinton* and R. R. Salakhutdinov

dados originais

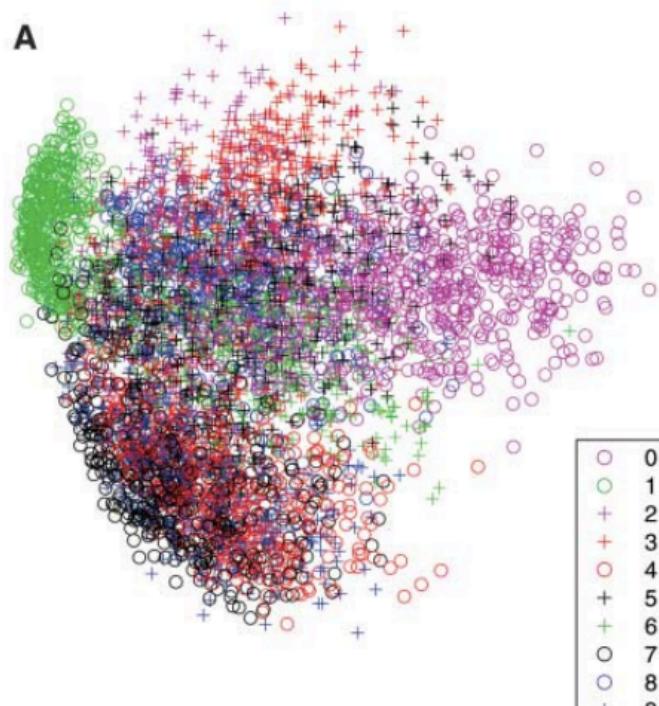


deep autoencoder

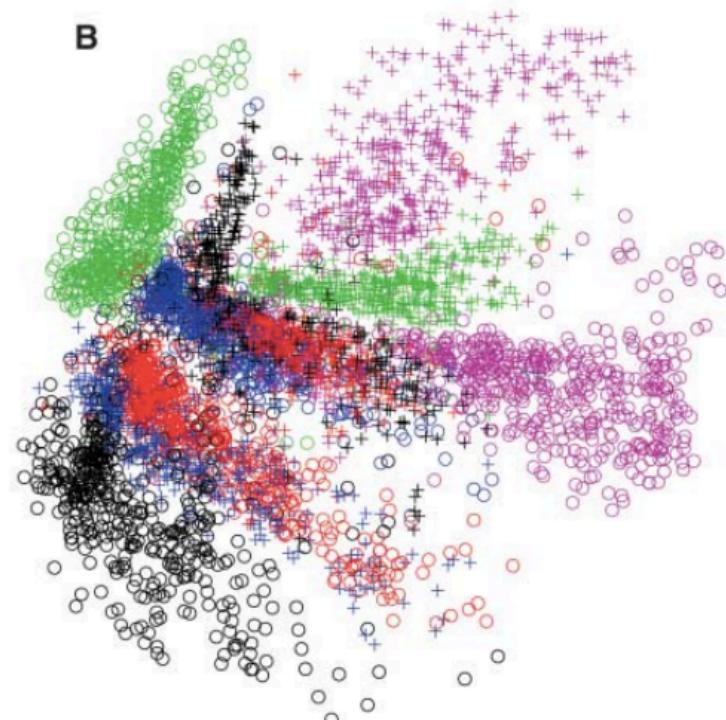
PCA

Exemplos de aplicações de DL

- Reconhecimento de dígitos por imagens



PCA

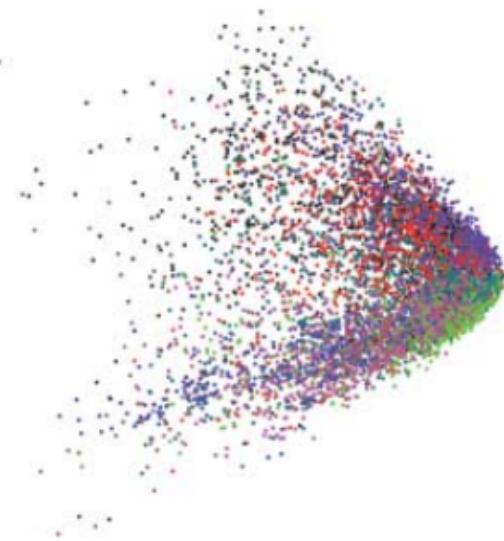


Autoencoder

Exemplos de aplicações de DL

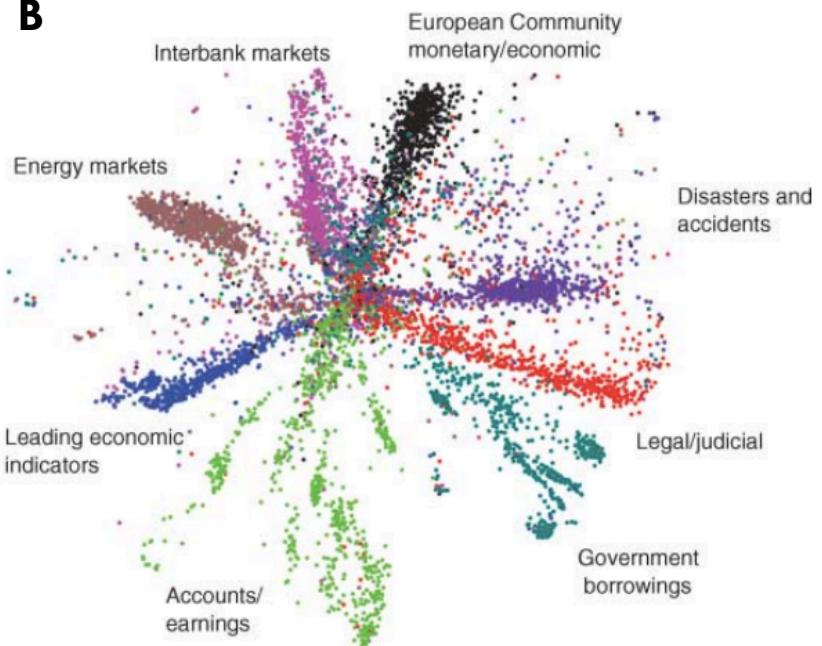
- Classificação de documentos por assunto (tipo)

A



Latent Semantic
Analysis (LSA)

B



Autoencoder

Exemplos de aplicações de DL

BossSensor

Hide your screen when your boss is approaching.

Demo

The boss stands up. He is approaching.



<https://github.com/Hironsan/BossSensor>

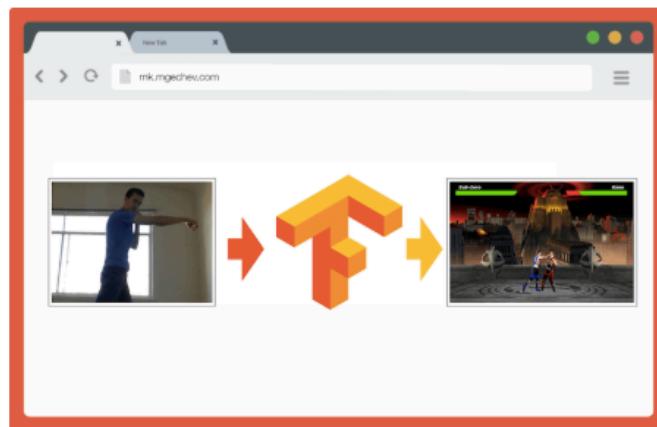
When he is approaching, the program fetches face images and classifies the image.

Exemplos de aplicações de DL

Playing Mortal Kombat with TensorFlow.js. Transfer learning and data augmentation

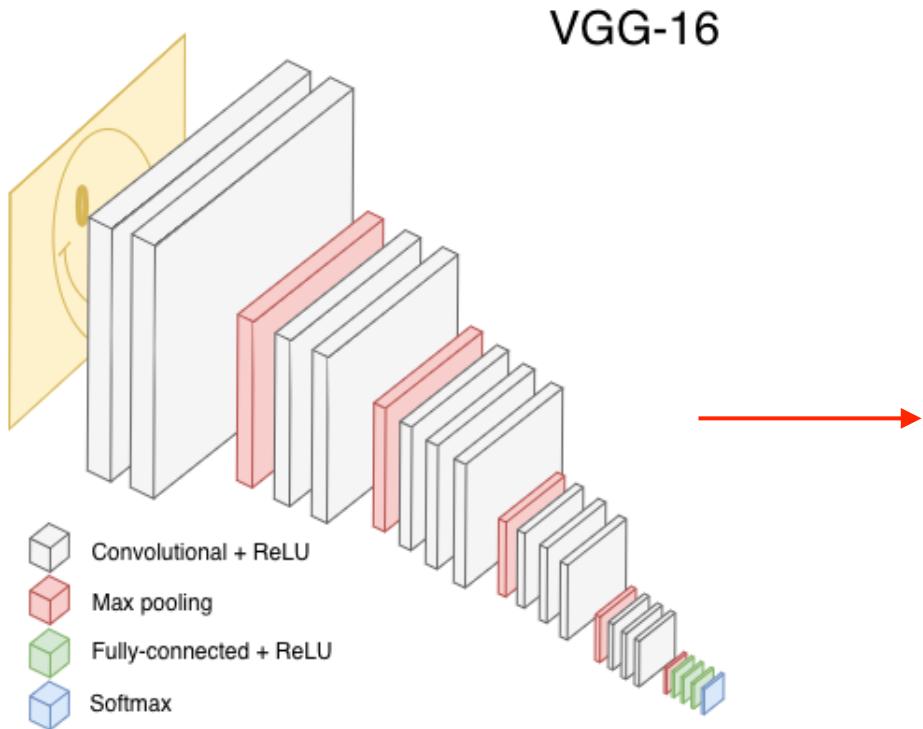
[Edit](#) · Oct 20, 2018 · 25 minutes read · [Follow @mgechev](#) · 10.8K followers
MACHINE LEARNING TENSORFLOW CNN TRANSFER LEARNING DATA AUGMENTATION ML

While experimenting with enhancements of the prediction model of [Guess.js](#), I started looking at deep learning. I've focused mainly on recurrent neural networks (RNNs), specifically LSTM because of their “[unreasonable effectiveness](#)” in the domain of Guess.js. In the same time, I started playing with convolutional neural networks (CNNs), which although less traditionally, are also often used for time series. CNNs are usually used for image classification, recognition, and detection.



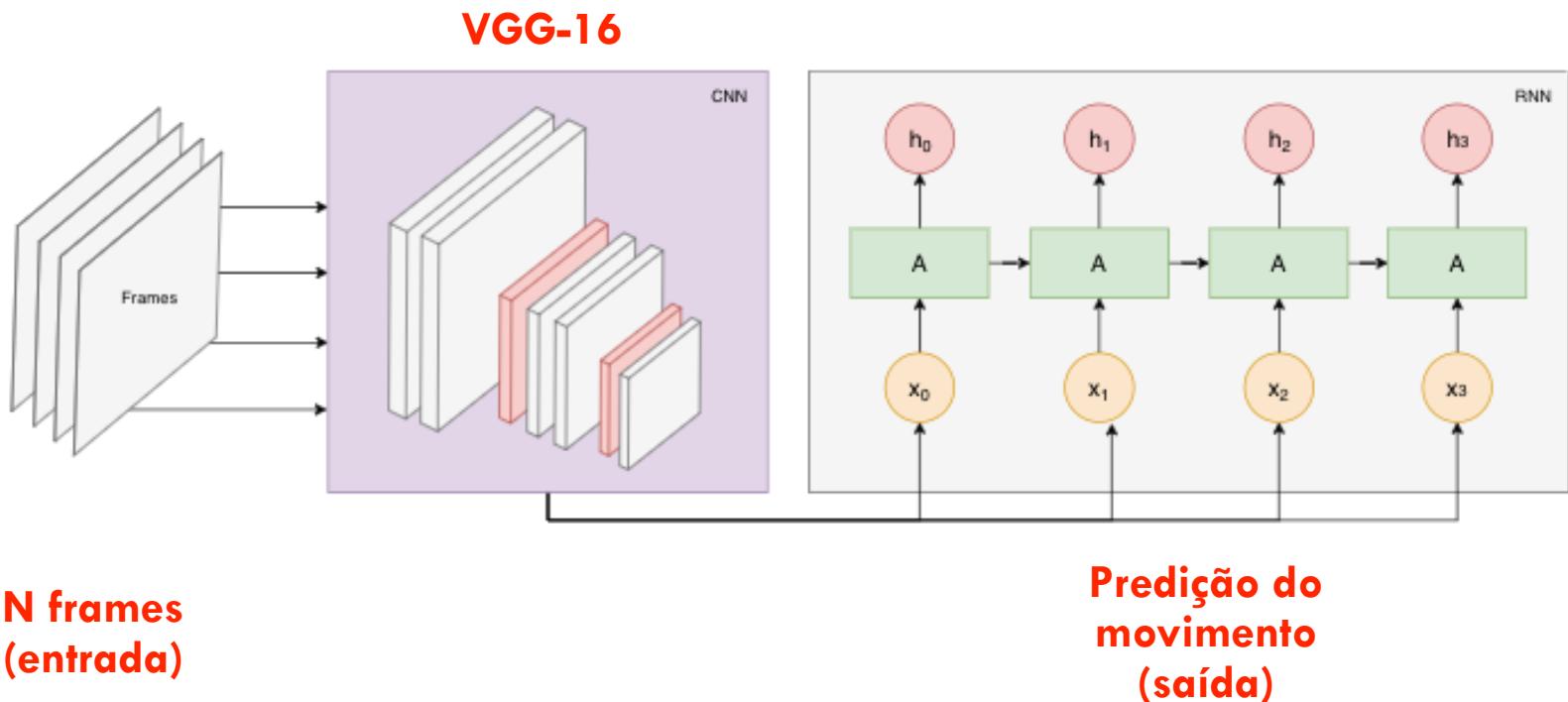
<https://blog.mgechev.com/2018/10/20/transfer-learning-tensorflow-js-data-augmentation-mobile-net/>

Exemplos de aplicações de DL



<https://blog.mgechev.com/2018/10/20/transfer-learning-tensorflow-js-data-augmentation-mobile-net/>

Exemplos de aplicações de DL



<https://blog.mgechev.com/2018/10/20/transfer-learning-tensorflow-js-data-augmentation-mobile-net/>