# Introdução ao Aprendizado Profundo

## Deep Learning 101

**María Fernanda Rodríguez Ruiz**
Prof. Dr. José Mario De Martino

Maio de 2018

Departamento de Engenharia de Computação e Automação Industrial — LCA
Faculdade de Engenharia Eléctrica e de Computação — FEEC
Universidade Estadual de Campinas — UNICAMP

# Agenda

1. Introdução
2. Ambiente de Desenvolvimento
3. Aprendizado de Máquina
4. Aprendizado Profundo
5. Revisão Final

# Introdução

# Introdução

## Objetivo

- Apresentar as **noções básicas** que envolvem o conceito de **Aprendizado Profundo** ou *Deep Learning*.

## Modelos

Apresentar alguns dos **modelos** do aprendizado de máquina:

- Regressão linear,
- Regressão logística,
- Redes neurais artificiais,
- Redes neurais artificiais profundas, e
- Redes neurais convolucionais.

# Introdução

## Implementação

- Breve introdução **teórica** e
- Implementações **práticas** através de um mesmo problema.
  - Os exemplos serão desenvolvidos fazendo uso do **Python** e **Keras/TensorFlow** em **Jupyter**.

## Disciplinas da Inteligência Artificial — IA


**Figura 1:** Diagrama de Venn da relação entre algumas das disciplinas da Inteligência Artificial [Goodfellow et al., 2016].

## Aprendizado de máquina — AM

Disciplina da IA que cria sistemas que **aprendem a partir de dados ou com sua própria experiência** a resolver tarefas específicas sem que estas precisem ser explicitamente programadas [Goodfellow et al., 2016].

- **Aprendizado profundo**
  Técnicas de AM baseada em **múltiplos níveis de aprendizagem**, para análise de grandes volumes de dados [Goodfellow et al., 2016].

## Aplicações do AM

### Dados

- Binário
- Séries Temporais
- Texto
- Sons
- Imagens
- ...

### Áreas

- Agricultura
- Robótica
- Ciência
- Entretenimento
- Finanças
- ...

# Ambiente de Desenvolvimento

# Ambiente de Desenvolvimento

## Ferramentas

- *Frameworks* (aprendizado de máquina e profundo),
- *Python* e
- *Jupyter*.

## *Frameworks* para aprendizado de máquina e aprendizado profundo

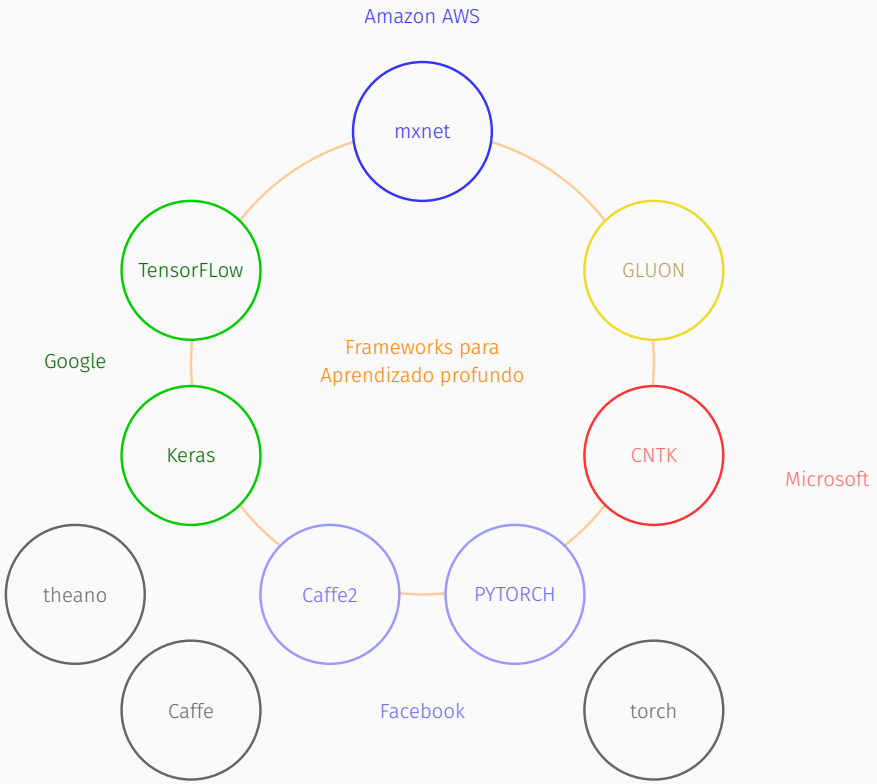



**Figura 2:** *Frameworks*[1] de aprendizado profundo de código aberto.

[1] The 5 Deep Learning Frameworks — https://heartbeat.fritz.ai/

# *Frameworks*

## *Frameworks* para aprendizado de máquina e aprendizado profundo

Operam em 2 níveis de abstrações[2]:

- **Low Level** — operações matemáticas e primitivas de redes neurais são implementadas (**TensorFlow**, Theano, PyTorch.)
- **High Level** — primitivas de baixo nível são usados para implementar abstrações de redes neurais, como modelos e camadas (**Keras**).

[2] The 5 Deep Learning Frameworks — https://heartbeat.fritz.ai/

## TensorFlow[3]


- Biblioteca de software de **código aberto** para computação numérica de alto desempenho, do **Google**.
- **Arquitetura flexível** em várias plataformas CPUs, GPUs, TPUs, e desde *desktops* a dispositivos móveis.
- Forte suporte para **aprendizado de máquina** e **aprendizado profundo**.

[3] https://www.tensorflow.org/

## Keras[4]


- API de redes neurais de **alto nível**, escrita em Python.
- Pode ser executado em cima do **TensorFlow**, CNTK ou Theano
- Foi desenvolvida para permitir a **experimentação rápida**.
- **Suporta** redes convolucionais e redes recorrentes.
- **Funciona** para CPU e GPU.

[4] https://keras.io/

## Python[5]


- Linguagem de programação interpretada de **alto nível**.
- Pacotes:
  - **NumPy**[6]: suporta arrays e matrizes multidimensionais.
  - **Matplotlib**[7]: biblioteca para gerar gráficos a partir de dados contidos em listas ou matrizes.

[5] https://www.python.org/
[6] http://www.numpy.org/
[7] https://matplotlib.org/

## Jupyter notebook[8]


- **Linguagens** (*kernels*):
  - **Ju**lia, **Pyt**hon, **R**, MATLAB/Octave, Javascript, C, ....
- **Aplicação Web**:
  - Ambiente de computação interativo que permite criar (escrever e executar) documentos/código no navegador.
  - Markdown[9], HTML, LaTeX, PNG, SVG, PDF
- **Documento interativo**:
  - Sequência linear de células

[8] http://jupyter.org/
[9] https://www.markdownguide.org/

## Fluxo de trabalho Keras

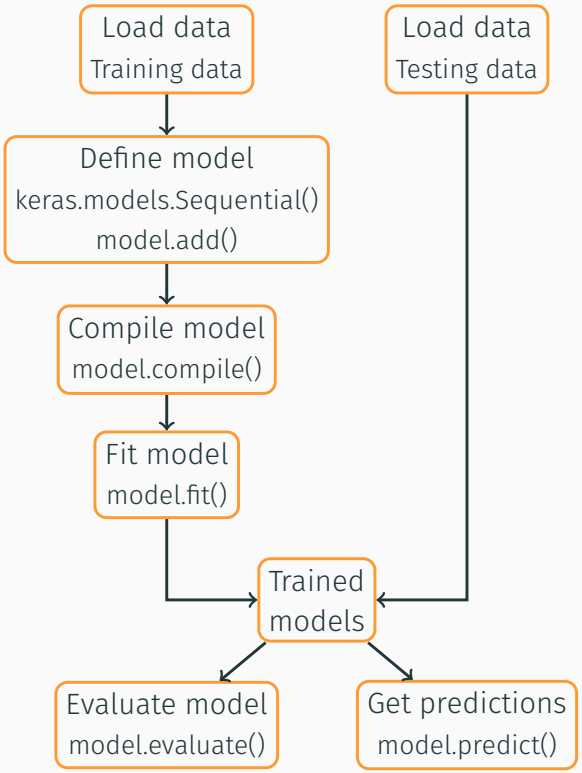



Figura 3: *Workflow* Keras

**Treinamento e avaliação** dos modelos.

1. Load Data
2. Define Model
3. Compile Model
4. Fit Model
5. Evaluate Model

# Fluxo de trabalho Keras

## 1. Load data

- Ler dados,
  - Conjunto de **treinamento** e
  - Conjunto de **teste**.
- Visualizar dados,
- Pre-processar.

```
X_train, y_train = load_data_train()
X_test, y_test = load_data_test()
```

# Fluxo de trabalho Keras

## 2. Define model

Dois modelos: **Sequential** e **Functional API**.

**Sequential**[10] usado para empilhar camadas (layers),
**model.add()**[11] usada para adicionar as camadas,
**input_shape=()** especificar a forma de entrada.

```
model = keras.models.Sequential()
model.add(layer1 ..., input_shape=(nFeatures))
model.add(layer2 ... )
```

[10] https://keras.io/models
[11] https://keras.io/layers

# Fluxo de trabalho Keras

## 3. Compile model

Configurar o **processo de aprendizado** especificando:

**Otimizador**[12] que determina como os pesos são atualizados,
**Função de custo**[13] ou função de perda,
**Métricas**[14] para avaliar durante o treinamento e o teste.

```
model.compile(optimizer='SGD',
              loss='mse',
              metrics=['accuracy'])
```

[12] https://keras.io/optimizers
[13] https://keras.io/losses
[14] https://keras.io/metrics

# Fluxo de trabalho Keras

## 4. Fit model

Iniciar o **processo de treinamento**.

**batch_size**[15]: dividir o conjunto de dados em número de lotes.
**epochs**: número de vezes que é treinado o conjunto de dados completo.

```
model.fit(X_train, y_train,
          batch_size=500,
          epochs=1)
```

[15] Epoch vs Batch Size vs Iterations — https://towardsdatascience.com

# Fluxo de trabalho Keras

## 5. Evaluate model

Avaliar o **desempenho** do modelo.

**model.evaluate()** encontra a perda e as métricas especificadas.
Fornece uma medida **quantitativa** da precisão.
**model.predict()** encontra a saída para os dados de teste fornecidos e é útil para verificar as saídas **qualitativamente**.

```
history = model.evaluate(X_test, y_test)
y_pred = model.predict(X_test)
```

# Aprendizado de Máquina

# Noções básicas

Para entender o **aprendizado profundo**, é preciso ter uma sólida compreensão dos princípios básicos do aprendizado de máquina [Goodfellow et al., 2016].

**Aprendizado de Máquina**

É tudo sobre a criação de um **algoritmo que pode aprender** com os dados para fazer uma previsão [Goodfellow et al., 2016].

## Noções matemáticas

Quatro grandes **disciplinas matemáticas**[16] compõem o aprendizado de máquina:

- **Estatística** é um núcleo de tudo, nos diz qual é o nosso objetivo.
- **Cálculo** nos diz como aprender e otimizar nosso modelo.
- **Álgebra linear** torna a execução dos algoritmos viáveis em conjuntos de dados massivos.
- **Probabilidade** ajuda a prever a possibilidade de um evento ocorrer.

[16] https://github.com/llSourcell/math_of_machine_learning

## Aprendizado de Máquina

Pode ser classificada em [Goodfellow et al., 2016]:

- **Supervisionado**: conjunto de dados de características, associada a um rótulo ou alvo,
- **Não supervisionado**: conjunto de dados de muitas características que aprendem propriedades úteis da estrutura desse conjunto de dados, e
- **Por reforço**: algoritmos interagem com um ambiente, portanto há um ciclo de feedback entre o sistema de aprendizado e suas experiências.

## Regressão linear

- **Algoritmo simples** de aprendizado de máquina, resolve um problema de **regressão** [Goodfellow et al., 2016]:
  - Na **estatística**, é uma técnica para modelar a relação entre uma variável dependente $y$, e uma ou mais variáveis independentes $x$.
- A saída é uma **função linear** da entrada,

$$\hat{y} = Wx + b \tag{1}$$

  - **Hipótese** $\hat{y}$ o valor que o modelo prevê,
  - **Parâmetros** $W$ que determinam como cada característica afeta a previsão, e $b$ que controla o deslocamento fixo da previsão [Goodfellow et al., 2016].

# Regressão linear

## Parâmetros

- **Parâmetros** são valores que controlam o comportamento do sistema.
  - Objetivo de **encontrar** o "melhor" conjunto de parâmetros possível $W$ e $b$, para descrever os dados.
- Primeiro precisamos definir o **erro/custo**.

## Função de custo

- Definir uma **medida de desempenho** do modelo.
- Uma maneira de medir o desempenho do modelo é calcular o **erro quadrático médio** (MSE) no conjunto de teste.
  - Na **estatística**, o MSE mede a média dos quadrados dos erros ou desvios [Goodfellow et al., 2016].

$$L = MSE = \frac{1}{n}\sum_{i}(y_i - \hat{y}_i)^2 \tag{2}$$

- **Diferença** entre a hipótese $\hat{y}$ a partir do valor real $y$.

## Problema de otimização

- Precisamos **minimizar o custo** da hipótese $\hat{y}$, em função dos parametros do modelo $W$ e $b$:

$$\min_{W,b} L = \min_{W,b} \frac{1}{N} \sum_{i} (y_i - \hat{y}_i)^2 \tag{3}$$

## Possíveis soluções

- Analítica
- Numérica:
  - **Algoritmos de otimização** que iteram sobre o conjunto de dados.

## Gradiente descendente

- Em **cálculo**, é um algoritmo de otimização iterativa de primeira ordem para encontrar o **mínimo de uma função** [Goodfellow et al., 2016].
    - **Gradiente** é uma operação que assume uma função de múltiplas variáveis, e retorna um **vetor na direção da inclinação** máxima no gráfico da função original.

$$\nabla L = \begin{bmatrix} \frac{\partial L}{\partial W} \\ \frac{\partial L}{\partial b} \end{bmatrix} \tag{4}$$

- Se quisermos **descer**, o que temos a fazer é **andar na direção oposta** ao gradiente.
- Esta seria a **estratégia para minimizar** as funções de custo.

## Taxa de Aprendizagem $\epsilon$

- Escalar positivo que determina o **tamanho do passo** [Goodfellow et al., 2016].

$$\begin{bmatrix} W' \\ b' \end{bmatrix} = \begin{bmatrix} W \\ b \end{bmatrix} - \epsilon \nabla L \tag{5}$$

$$= \begin{bmatrix} W \\ b \end{bmatrix} - \epsilon \begin{bmatrix} \frac{\partial L}{\partial W} \\ \frac{\partial L}{\partial b} \end{bmatrix} \tag{6}$$

- Poderia ser um problema de duas maneiras:
  - Se o passo é **muito pequeno**, nos moveremos lentamente para o mínimo,
  - Se o passo é **muito grande**, podemos acabar pulando além do mínimo.

## Back-propagation

Método para **calcular** o gradiente.

Enquanto **outro** algoritmo, como gradiente descendente, é usado para realizar a **aprendizagem** (otimização) usando esse gradiente [Goodfellow et al., 2016].

Duas fases:

- Forward-propagation
- Back-propagation

## Forward-propagation[17]

Processo de alimentação dos valores.

A entrada $x$ fornece a informação inicial que se **propaga**, e finalmente produz $\hat{y}$ [Goodfellow et al., 2016].

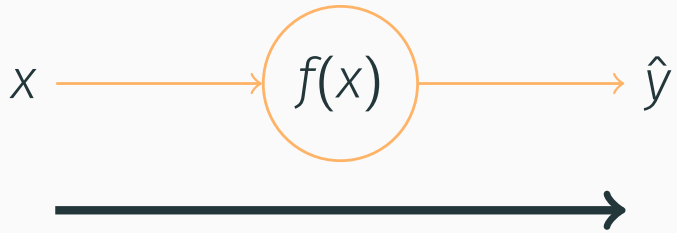



**Figura 4:** Forward-propagation.

[17] Everything you need know about neural networks — https://hackernoon.com/

## Back-propagation[18]

Usamos o valor da função de custo para calcular o erro. O valor do erro é propagado para atrás afim de calcular o gradiente com relação aos pesos.

Os gradientes são calculados usando a regra da cadeia.


**Figura 5:** Back-propagation.

[18] Everything you need know about neural networks — https://hackernoon.com/

## Algoritmo Back-propagation[19]

- Após Forward-propagation, obtemos um valor de saída que é o valor previsto.
- Usamos uma função de perda $L$ para calcular o valor do erro.
- Calculamos o gradiente do erro em relação a cada peso,
- Subtraímos o valor do gradiente do valor do peso.

Desta forma, nos aproximamos ao mínimo local.

[19] Everything you need know about neural networks — https://hackernoon.com/

## Problemas com gradiente descendente

O método de **gradiente descedente tradicional** calculará o gradiente de **todo o conjunto de dados**, mas executará apenas uma atualização.

Por isso pode ser muito lento e difícil de controlar para conjuntos de dados muito grandes e que não se encaixam na memória.

## Otimizadores[20]

Existem **extensões** (modificações ao método original) que tentam solucionar os problemas do gradiente descendente.

- Básico [Goodfellow et al., 2016]:
  - **SGD**: Stochastic Gradient Descent.
- Taxas de aprendizado adaptativas [Goodfellow et al., 2016]:
  - **AdaGrad** [Duchi et al., 2011].
  - **RMSprop** [Hinton, 2012].
  - **Adam** [Kingma and Ba, 2014].

[20] Types of Optimization Algorithms — https://towardsdatascience.com/

# Regressão linear:
# Código GitHub

# Regressão linear

## Resumo parcial

- Regressão linear define a relação entre duas variáveis.
- Como **encontrar** a "melhor" combinação dos parâmetros?
  - **Função de custo**:
    - Erro quadrático médio
  - **Gradiente descendente**
    - Back-propagation para calcular o gradiente
    - Otimizadores (extensões do gradiente descedente)
- O processo de encontrar a melhor combinação de parâmetros (otimização) é chamado de **treinamento**.

## Regressão logística

- Na **estatística**, é um modelo de regressão em que a variável dependente é **categórica**.
  - Que a variável dependente seja categórica implica que estamos tratando um **problema de classificação**.
- A saída deve estar sempre entre 0 e 1 [Goodfellow et al., 2016].
  - A saída indica a probabilidade de uma entrada pertencer a uma classe.
- A **probabilidade** é modelada usando a relação

$$\hat{y} = \sigma(Wx + b) \tag{7}$$

- onde $\sigma$ é chamada de **função de ativação** e limita a saída para o intervalo (0, 1).

## Função de ativação[21] $\sigma$

Deve ser uma função **sigmóide** para mapear qualquer número real da hipótese para o intervalo (0, 1) [Goodfellow et al., 2016]:

**logistic**

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{8}$$

**tanh**

$$\sigma(z) = \frac{2}{1 + e^{-2z}} - 1 \tag{10}$$

**softmax**

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{9}$$

O nome sigmóide vem da forma em **S** do seu gráfico.

[21] Activation functions — https://towardsdatascience.com/

## Representação gráfica

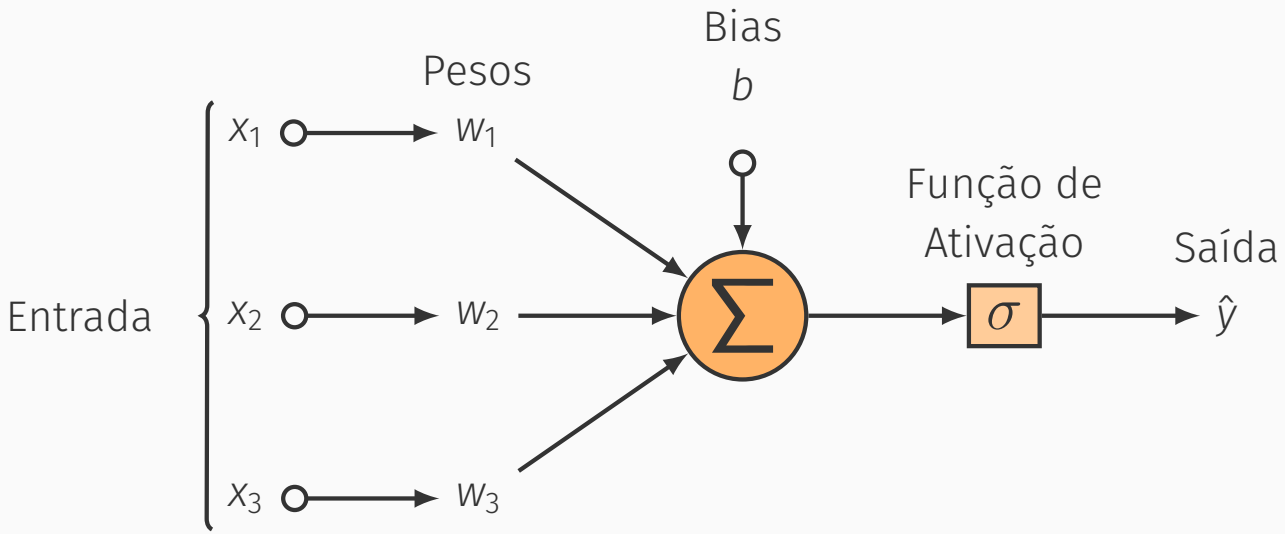



**Figura 6:** Representação gráfica da regressão logística $\hat{y} = \sigma(Wx + b)$.

## Função de custo[22]

- Para classificação pode-se usar ***cross entropy cost*** [Goodfellow et al., 2016]:

$$E(y, \hat{y}) = -\sum_{i=1}^{K} y_i log(\hat{y}) \tag{11}$$

  - **binary-cross-entropy**: problema de classificação binária,
  - **categorical-cross-entropy**: problema de classificação multi-classe.

[22] Everything you need to know about Neural Networks — https://hackernoon.com/

# Regressão logística:
# Código GitHub

# Regressão logística

## Resumo parcial

**Tabela 1:** Regressão linear e regressão logística.

| Método | Alvo | Hipótese | Custo |
|---|---|---|---|
| Regressão linear | Contínua | $\hat{y} = Wx + b$ | MSE |
| Regressão logística | Categórica | $\hat{y} = \sigma(Wx + b)$ | Cross Entropy |

# Aprendizado supervisionado

A maioria dos problemas do mundo real **não são linearmente separáveis**.

Para calcular **hipóteses não-lineares**, uma das formas computacionalmente mais eficientes é **conectar pequenas unidades** que fazem "regressão logística"[23].

[23] Playing with machine learning — https://medium.com/rocknnull

## Redes Neurais Artificiais — RNA

São chamadas de redes porque elas são representadas pela composição de **várias funções** diferentes.

A informação **flui** através de funções conectadas em cadeia [Goodfellow et al., 2016]:

$$\hat{y} = f(x) \tag{12}$$
$$= f^{(2)}(f^{(1)}(x)) \tag{13}$$

- $x$: Camada de entrada
- $f^{(1)}$: Camada oculta
- $f^{(2)}$: Camada de saída

## Camadas

$$f^{(i)}(h) = \sigma_i(W_i h + b_i) \tag{14}$$

- $\sigma_i$: função de ativação
- $W_i$: Matriz de pesos
- $b_i$: Vetor de bias

ou seja,

$$\hat{y} = \sigma_2(W_2\sigma_1(W_1 x + b_1) + b_2) \tag{15}$$

## Representação gráfica

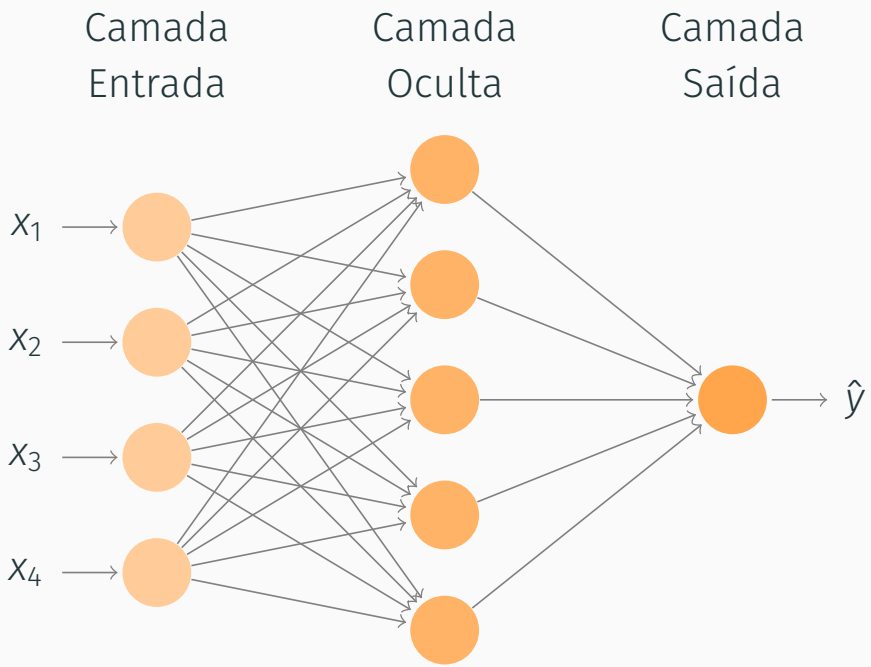
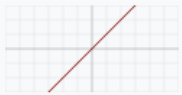
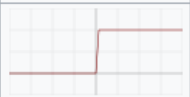
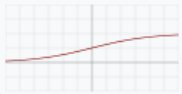
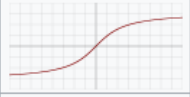
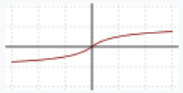



**Figura 7:** Representação gráfica de uma redes neurais artificiais com uma camada oculta $\hat{y} = \sigma_2(W_2\sigma_1(W_1x + b_1) + b_2)$.

## Funções de ativação[24]

Introduzem **não linearidade** nas redes neurais artificiais.

| | | |
|---|---|---|
| Identity |  | $f(x) = x$ |
| Binary step |  | $f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$ |
| Logistic (a.k.a. Sigmoid or Soft step) |  | $f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$ [1] |
| TanH |  | $f(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$ |
| ArcTan |  | $f(x) = \tan^{-1}(x)$ |
| Softsign [7][8] |  | $f(x) = \frac{x}{1+\lvert x\rvert}$ |

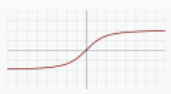
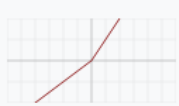
| | | |
|---|---|---|
| Inverse square root unit (ISRU)[9] |  | $f(x) = \frac{x}{\sqrt{1+\alpha x^2}}$ |
| Rectified linear unit (ReLU)[10] |  | $f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ |
| Leaky rectified linear unit (Leaky ReLU)[11] |  | $f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ |
| Parameteric rectified linear unit (PReLU)[12] |  | $f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ |
| Randomized leaky rectified linear unit (RReLU)[13] |  | $f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ [2] |

**Figura 8:** Algumas funções de activação.

[24] Activation Functions: Neural Networks — https://towardsdatascience.com/

## Funções de ativação[25]

- Para problemas de **regressão** $\sigma_2$: função **identidade**.

$$\sigma(z) = z \tag{16}$$

- Para problemas de **classificação** $\sigma_2$: função **sigmóide**.
  - logistic,
  - tanh,
  - softmax,
  - etc.

[25] Activation Functions: Neural Networks — https://towardsdatascience.com/

## Função de custo

- Para problemas de **regressão**: minimização do **MSE**.

$$MSE = \frac{1}{n}\sum_{i}(y_i - \hat{y}_i)^2 \tag{17}$$

- Para problemas de **classificação**: minimização do **cross entropy**.

$$E(y, \hat{y}) = -\sum_{i=1}^{K} y_i log(\hat{y}) \tag{18}$$

# Redes neurais artificiais:
# Código GitHub

## Resumo parcial

**Tabela 2:** Regressão linear, regressão logística e RNA.

| Método | Alvo | Hipótese | Custo |
|---|---|---|---|
| R. Linear | Contínua | $\hat{y} = Wx + b$ | MSE |
| R. Logística | Categórica | $\hat{y} = \sigma(Wx + b)$ | Cross Entropy |
| RNA | Contínua, Categórica | $\hat{y} = \sigma_2(W_2\sigma_1(W_1x + b_1) + b_2)$ | MSE, Cross Entropy |

# Aprendizado Profundo

# Noções básicas

Rede neural artificial + Volume dados + Poder computacional

$\Downarrow$

Aprendizado Profundo

## Aprendizado profundo

Abordagem do AM, que creceu em sua popularidade e utilidade, como resultado de **computadores poderosos**, **conjuntos de dados** e **técnicas maiores para treinar** redes mais profundas [Goodfellow et al., 2016].

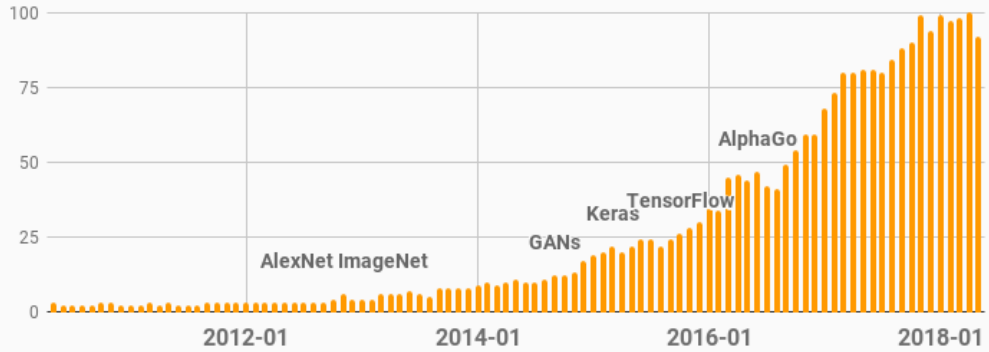


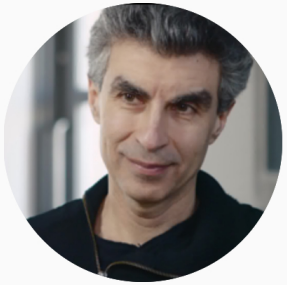

**Figura 9:** Interesse do aprendizado profundo nos ultimos 6 anos[26].

[26] https://trends.google.com

## Pesquisadores destacados

(a) Geoff Hinton
Backpropag.

(b) Yann LeCun
CNN.



(c) Yoshua Bengio
RNA, GANs.

(d) Ian Goodfellow
GANs.

(e) Andrew Ng
Google Brain

**Figura 10:** Pesquisadores destacados de aprendizado profundo.

## Redes neurais artificiais profundas

Representadas pela composição de **várias camadas ocultas** [Goodfellow et al., 2016].

Pesquisadores concordam que o aprendizado profundo envolve mais que duas transformações da entrada para a saída (**CAP > 2**) [Schmidhuber, 2014].

$$\hat{y} = f(x) \tag{19}$$
$$= f^{(n)}(\ldots f^{(2)}(f^{(1)}(x))) \tag{20}$$
$$= \sigma_n(W_n \ldots \sigma_2(W_2\sigma_1(W_1x + b_1) + b_2) \cdots + b_n) \tag{21}$$

- $f^{(1)}$: Camada de entrada
- $f^{(i)}\ \forall i = [2, n-1]$: C. ocultas
- $f^{(n)}$: Camada de saída
- $\sigma_i$: função de ativação
- $W_i$: Matriz de pesos
- $b_i$: Vetor de bias

## Dificuldades no treinamento

Quanto mais profunda a RNA:

- Pode ser mais robusta,
- Porém, pode ser **mais difícil** de treiná-la.

## Problema do gradiente descendente[27]

Dada uma RNA Profunda com quatro camadas:

$$\hat{y} = f_4(f_3(f_2(f_1(x, z_1), z_2), z_3), z_4), \text{sendo } z_n : \text{parâmetros camada } n \quad (22)$$

usamos **back-propagation** para atualizar, $z_1 = z_1 - \epsilon \frac{\partial L}{\partial z_1}$:

$$\frac{\partial L}{\partial z_1} = \frac{\partial L}{\partial z_4} \frac{\partial z_4}{\partial z_3} \frac{\partial z_3}{\partial z_2} \frac{\partial z_2}{\partial z_1} \quad (23)$$

- Se os gradientes forem **maiores do que zero**, o produto **explode** para um número muito grande,
- Se os gradientes forem **menores do que zero**, o produto **desvanece** para um número muito próximo de zero.

[27] https://matheusfacure.github.io/2017/07/10/problemas-treinamento/

## Função de ativação — ReLU[28]

Para evitar tais complicações e melhorar a precisão, pode ser usada a função de ativação baseadas em **ReLU** (*Rectified Linear Unit*) no treinamento da RNA Profunda.

$$\sigma(z) = \begin{cases} 0 & \text{para } z < 0 \\ z & \text{para } z \geq 0 \end{cases} \rightarrow \frac{d\sigma}{dz} = \begin{cases} 0 & \text{para } z < 0 \\ 1 & \text{para } z \geq 0 \end{cases} \tag{24}$$

As derivadas não vão mais desaparecer ou explodir, porque a derivada da função de ativação é limitada a os valores 0 e 1.[29]

[28] The vanishing gradient problem — https://medium.com/@anishsingh20/
[29] The vanishing gradient problem — https://ayearofai.com/

# Redes neurais artificiais profundas: Código GitHub

## Resumo parcial

**Tabela 3:** Regressão linear, regressão logística, RNA e RNA profundas.

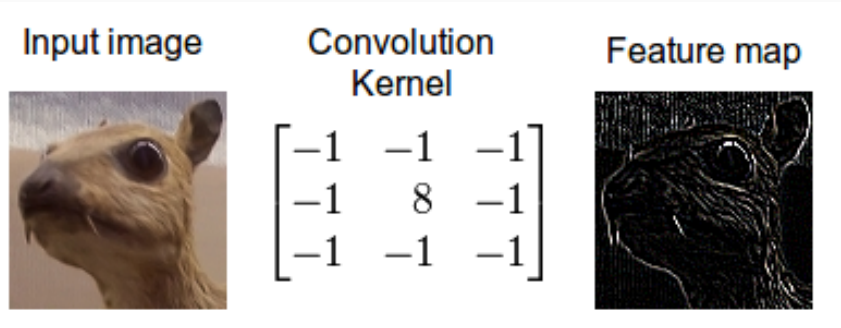
| Método | Alvo | Hipótese | Custo |
|---|---|---|---|
| R. Linear | Continua | $\hat{y} = Wx + b$ | MSE |
| R. Logística | Categórica | $\hat{y} = \sigma(Wx + b)$ | Cross Entropy |
| RNA | Continua, Categórica | $\hat{y} = f^{(2)}(f^{(1)}(x))$ | MSE, Cross Entropy |
| RNA Profundas | Continua, Categórica | $\hat{y} = f^{(n)}(\ldots f^{(2)}(f^{(1)}(x)))$ | MSE, Cross Entropy |

## Redes neuronais convolucionais — Redes convolucionais

Redes neurais que usam a **convolução** no lugar da multiplicação geral da matriz, em **pelo menos uma de suas camadas** [Goodfellow et al., 2016].

- Convolução é uma **operação matemática** que descreve uma regra de como misturar duas funções ou partes da informação[30]:

- Mapa de características $I$,
- Kernel de convolução $K$, e
- Mapa de características transformadas $S(i, j)$.




$$S(i, j) = (I * K)(i, j) \tag{25}$$

[30] https://devblogs.nvidia.com/deep-learning-nutshell-core-concepts/

## Parâmetros

As **camadas convolucionais** têm **parâmetros** que são aprendidos para que esses filtros sejam ajustados automaticamente para **extrair as informações** mais úteis para a tarefa em questão[31].

- Entrada é uma matriz multidimensional de dados,
- Kernel é uma matriz multidimensional de parâmetros,
- Essas matrizes multidimensionais são **tensores** [Goodfellow et al., 2016].
  - Séries temporais: grade 1D intervalos de tempo regulares,
  - Dados de imagem: grade 2D de pixels.

[31] https://devblogs.nvidia.com/deep-learning-nutshell-core-concepts/

## Camadas[32]

- **Convolution**: extrair características da imagem,
- **Pooling**: reduzir a dimensão da entrada, e
- **Dense/Fully connected**: conectar as camadas.

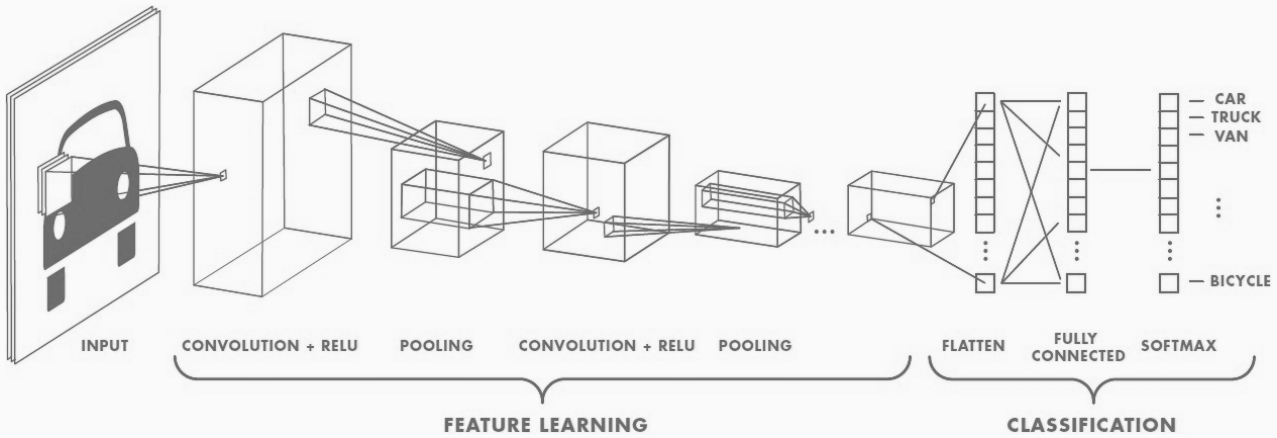



**Figura 11:** Exemplo de uma rede com varias camadas convolucionais[33].

[32] A deeper understanding of CNNs — https://towardsdatascience.com/
[33] https://ch.mathworks.com/fr/discovery/convolutional-neural-network.html

## Importância[34]

Ganharam popularidade através de **competições** como o **ImageNet** e, mais recentemente, eles são usados para **NLP** (Natural Language Processing) e **reconhecimento de fala** também.

[34] A deeper understanding of CNNs — https://towardsdatascience.com/

# Redes neuronais convolucionais: Código GitHub

## Resumo parcial

**Tabela 4:** Regressão linear, regressão logística, RNA e RNA profundas.

| Método | Alvo | Hipótese | Custo |
|---|---|---|---|
| R. Linear | Continua | $\hat{y} = Wx + b$ | MSE |
| R. Logística | Categórica | $\hat{y} = \sigma(Wx + b)$ | Cross Entropy |
| RNA | Continua, Categórica | $\hat{y} = f^{(2)}(f^{(1)}(x))$ | MSE, Cross Entropy |
| RNA Profundas | Continua, Categórica | $\hat{y} = f^{(n)}(\ldots f^{(2)}(f^{(1)}(x)))$ | MSE, Cross Entropy |
| RNA Convolucional | Continua, Categórica | $\hat{y} = f^{(n)}(\ldots f^{(2)}_{(*)}(f^{(1)}_{(*)}(x)))$ | MSE, Cross Entropy |

# Revisão Final

# Resumo

## Resultados MNIST

- **Épocas**: 50
- **Batch size**: 256
- **Otimizador**: RMSProp
- **Camada de saída**: 10 unidades softmax

| Modelo | Arquitetura | Ativação | Parâmetros | Precisão [%] |
|---|---|---|---|---|
| LOGREG | [] | [] | 7.850 | 92.79 |
| NN | [32] | [sigmoide] | 25.450 | 96.27 |
| DNN | [128, 64] | [relu, relu] | 109.386 | 97.90 |
| CDNN | [32*, 64*, 128] | [relu, relu, relu] | 4.738.826 | **98.84** |

*: tamanho do *kernel* convolucional ($3 \times 3$)

## Regularização[35]

Usado para superar o problema de *underfitting* e *overfitting*. Na regularização penalizamos a perda adicionando uma norma **L1** (LASSO) ou **L2** (Ridge) no vetor de peso *W*. Essas penalidades são incorporadas na **função de perda** que a rede otimiza[36].

- L1: soma do valor absoluto dos coeficientes.
- L2: soma do valor ao quadrado dos coeficientes.
- **Dropout**: configura aleatoriamente uma fração de unidades da entrada para 0 a cada atualização durante o tempo de treinamento.

[35] Everything you need to know about Neural Networks — https://hackernoon.com/
[36] https://keras.io/regularizers/

# Tópicos de interesse

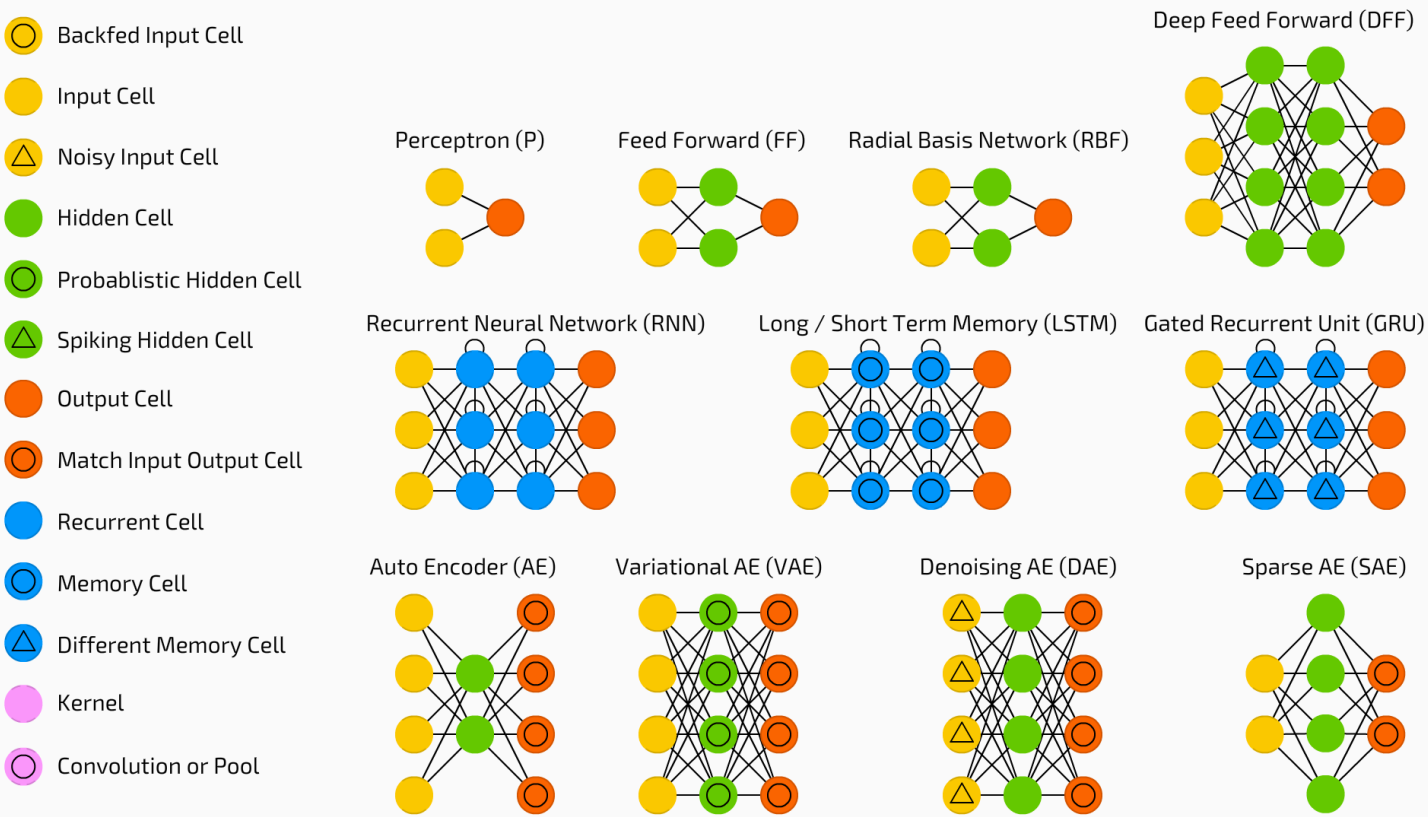



**Figura 12:** The neural network zoo[37]

[37] http://www.asimovinstitute.org/neural-network-zoo/

# Tópicos de interesse

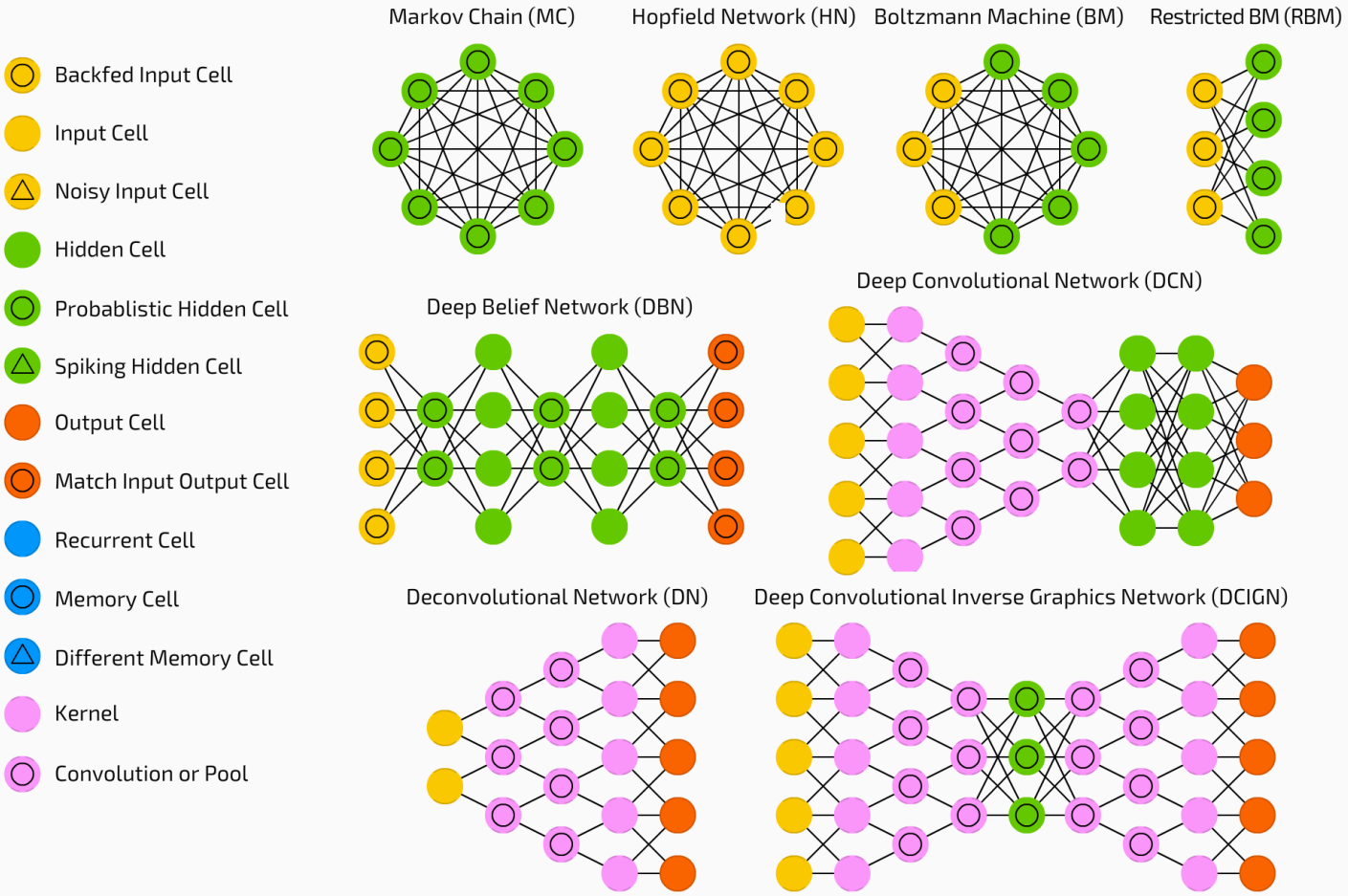



**Figura 13:** The neural network zoo[38]

[38] http://www.asimovinstitute.org/neural-network-zoo/

# Tópicos de interesse

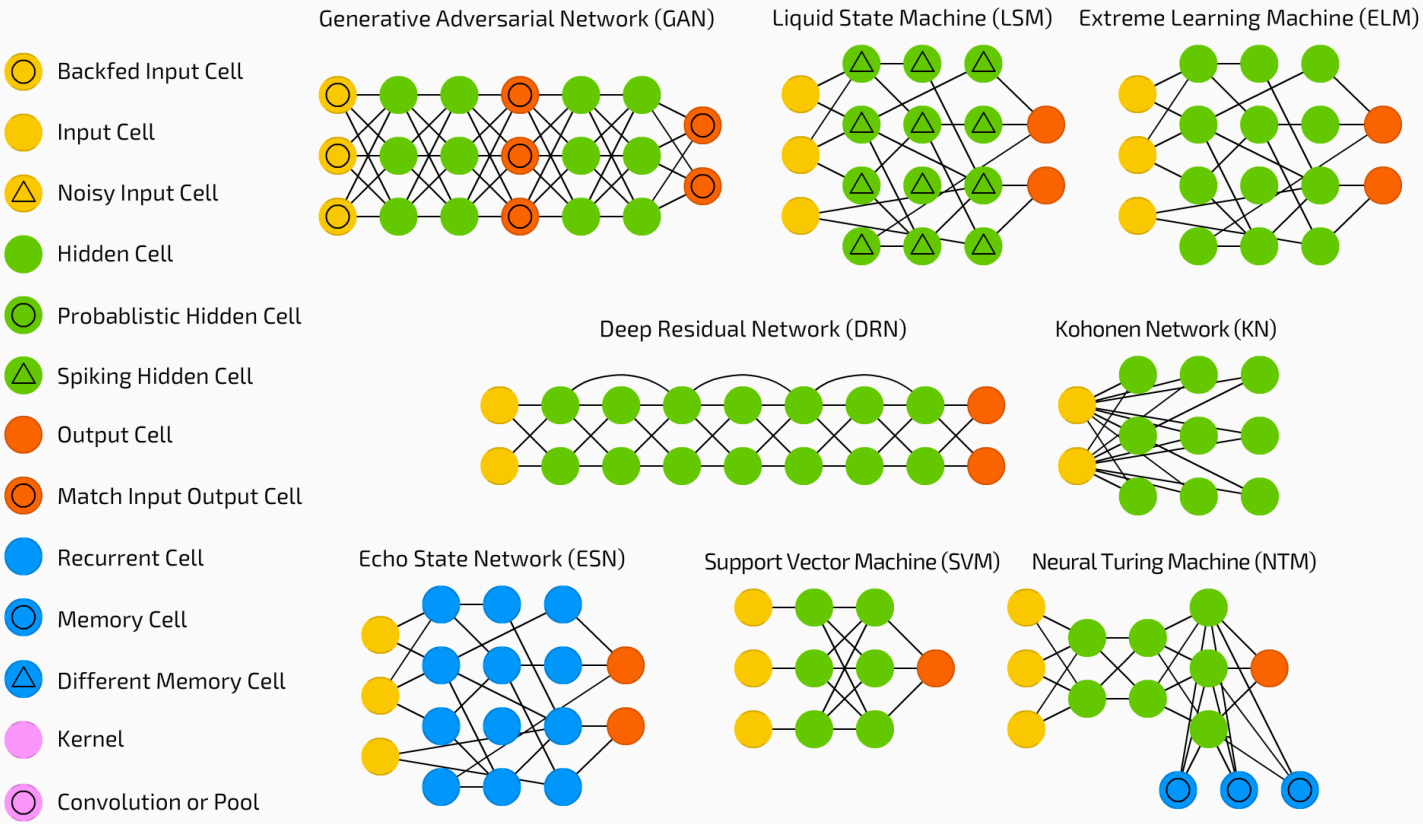



**Figura 14:** The neural network zoo[39]

[39] http://www.asimovinstitute.org/neural-network-zoo/

Obrigada

## Códigos

https://github.com/mafda/deep_learning_101

# Referências i



Duchi, J., Hazan, E., and Singer, Y. (2011).
**Adaptive subgradient methods for online learning and stochastic optimization.**
*Journal of Machine Learning Research*, 12(Jul):2121–2159.


Goodfellow, I., Bengio, Y., and Courville, A. (2016).
***Deep Learning*.**
MIT Press.


Hinton, G. (2012).
**Neural networks for machine learning.**
Coursera, video lectures.


Kingma, D. P. and Ba, J. (2014).
**Adam: A method for stochastic optimization.**
*CoRR*, abs/1412.6980.

# Referências ii


Schmidhuber, J. (2014).
**Deep learning in neural networks: An overview.**
*CoRR*, abs/1404.7828.