

Machine Learning Aplicada à Agropecuária de Precisão

Dia 04 - Redes Neurais Convolucionais e Métodos Avançados



Giancarlo D. Salton, PhD

Applied Intelligence Research Centre & ADAPT Centre
School of Computing

Chapecó, 10 de maio de 2018

Redes Neurais Convolucionais

Inicialização de Parâmetros

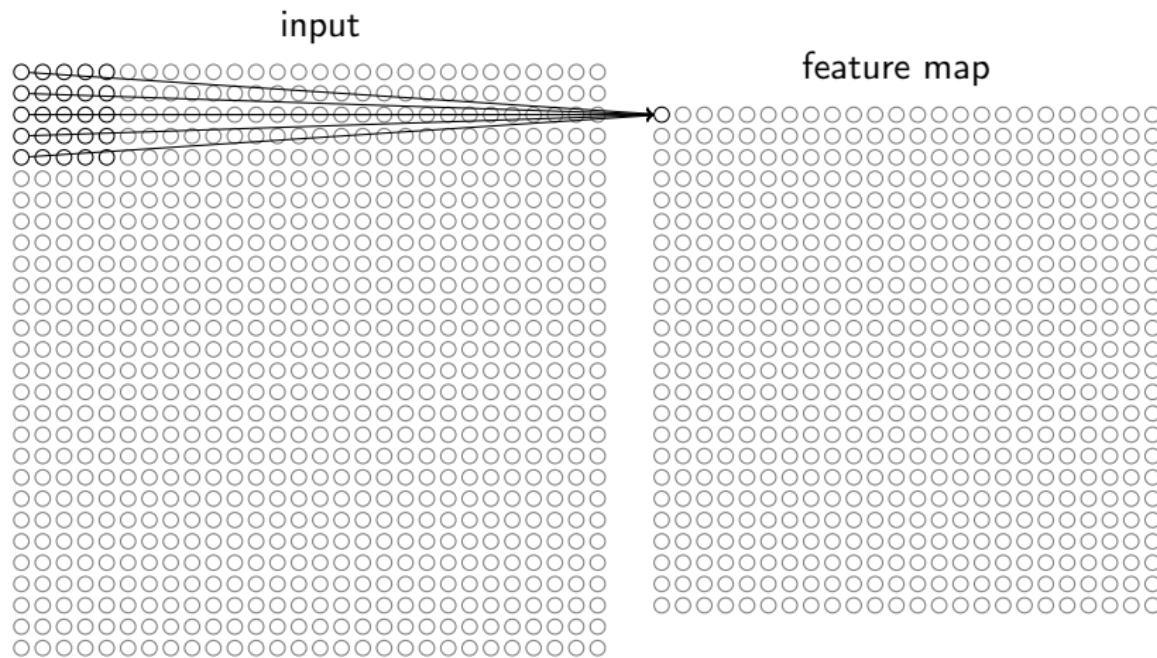
Normalização de Batch

Redes Neurais Residuais

Métodos adaptativos de Gradiente Descendente

Resumo

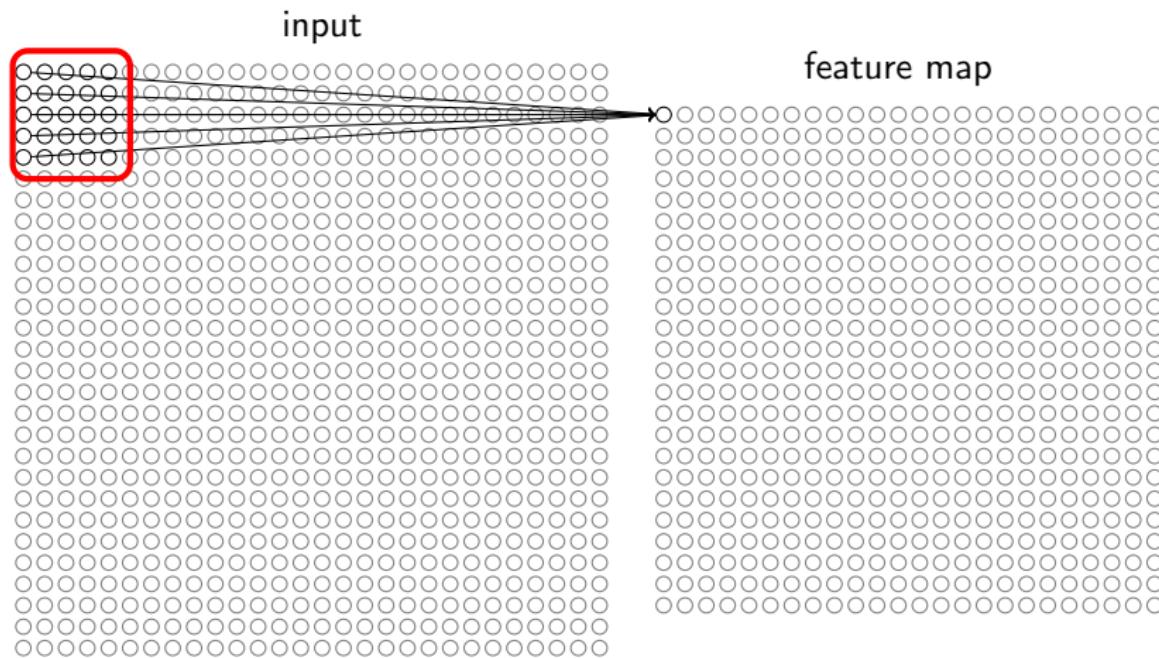
Redes Neurais Convolucionais



$$z = bias(C_{out_j}) + \sum_{k=0}^{C_{in}-1} weight(C_{out_j}, k) \star input(N_i, k)$$

$$y = \max(ReLU(z))$$

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

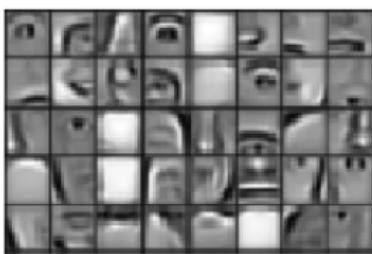
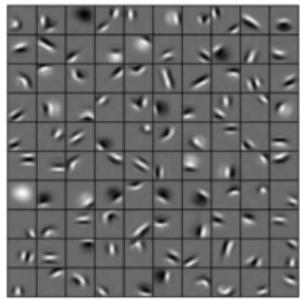


$$z = bias(C_{out_j}) + \sum_{k=0}^{C_{in}-1} weight(C_{out_j}, k) \star input(N_i, k)$$

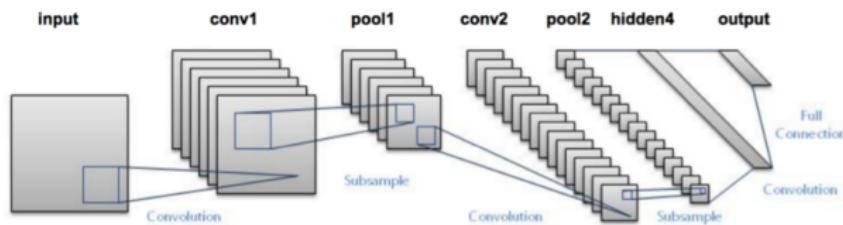
$$y = \max(ReLU(z))$$

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

- ▶ As *conv nets* e as redes *feedforward* são igualmente organizadas em camadas
- ▶ A diferença reside no fato de que as primeiras camadas das *conv nets* são compostas pelas convoluções
 - ▶ As convoluções extraem *features* que servirão de entrada para as camadas finais da rede
 - ▶ As camadas finais, por sua vez, são compostas por 1 ou mais camadas “tradicionais”



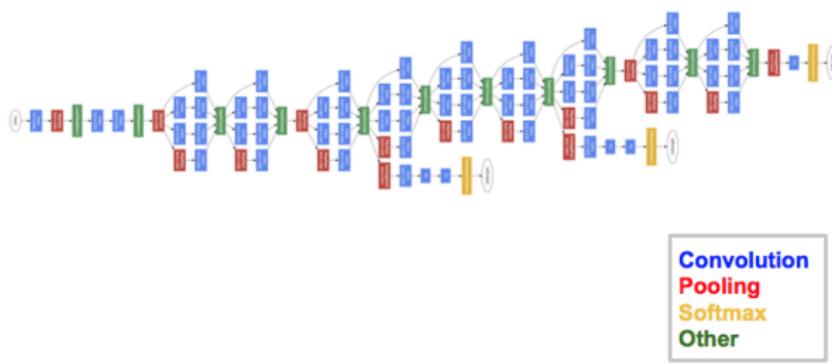
Fonte: <https://devblogs.nvidia.com/deep-learning-nutshell-core-concepts/>



► LeNet-5 (1998)

Fonte:

https://medium.com/@siddharthdas_32104/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5



- ▶ GoogleNet/Inception (2014)

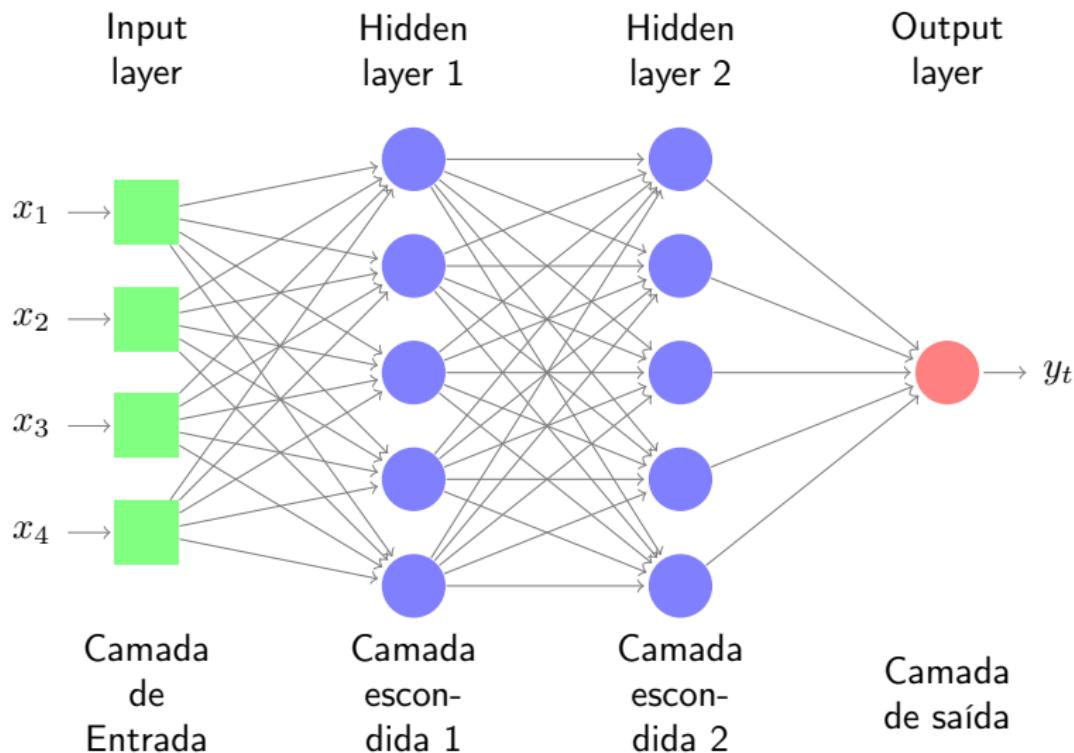
Fonte:

https://medium.com/@siddharthdas_32104/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5

- ▶ Esta hierarquia de *features* é o que torna as redes neurais interessantes
- ▶ Quanto mais longa a hierarquia, maior a chance de se extrair *features* que podem fazer a diferença entre um bom resultado e um resultado ainda melhor
- ▶ No entanto, treinar estes modelos requer um *dataset* grande e um poder computacional maior ainda

- ▶ Por isso, existem iniciativas em competições abertas recomendando que os vencedores compartilhem seus modelos pré-treinados
- ▶ Muitos pesquisadores e empresas na área de visão computacional costumam compartilhar o código fonte e várias versões pré-treinadas de seus modelos
- ▶ Desta forma, podemos utilizar os modelos da forma que estão, adaptá-los ou utilizar parte dos mesmos para criar outros modelos
 - ▶ A utilização de parte dos modelos, especialmente os extratores de *features*, já é prática comum na área

Inicialização de Parâmetros



- ▶ Resumo das equações de backpropagation:

$$\delta^L = \nabla_a \mathcal{C} \odot \sigma'(\mathbf{z}^L) \quad (\text{BP1})$$

$$\delta^l = ((\mathbf{W}^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \quad (\text{BP2})$$

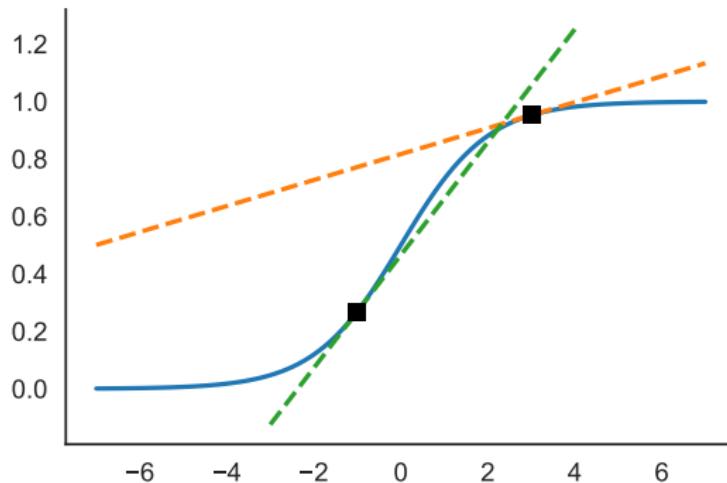
$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \quad (\text{BP3})$$

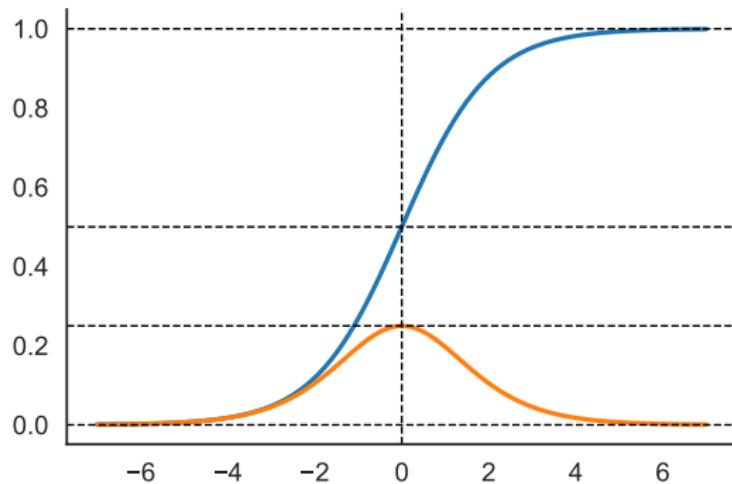
$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \quad (\text{BP4})$$

- ▶ Supondo uma rede neural com 4 camadas e 1 neurônio em cada camada, a cadeia para se atualizar parâmetros na primeira camada é (usando a atualização de bias como exemplo)

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) w_2 \sigma'(z_2) w_3 \sigma'(z_3) w_4 \sigma'(z_4) \frac{\partial C}{\partial a_4}$$

- ▶ A multiplicação por $\sigma'(z_l)$ pode fazer com que o sinal desapareça quase que por completo!
 - ▶ *Vanishing gradients*
 - ▶ Embora menos frequente, há também o problema de *Exploding gradients*, onde o sinal aumenta rapidamente





- ▶ Inicialização “Xavier” (ou “Glorot”)

$$weights = \mathcal{N}(0, std)$$

onde:

$$std = gain \times \sqrt{\frac{2}{n_{in} + n_{out}}}$$

e normalmente

$$gain = 1$$

Normalização de Batch

- ▶ Em geral, a ativação dos neurônios terá um perfil diferente conforme seus parâmetros
 - ▶ ativações podem estar concentradas em intervalos diferentes
 - ▶ exemplos também podem causar essa variação nas ativações!
- ▶ Isso causa instabilidade na rede neural, pois faz com que os gradientes também assumam essa variação
- ▶ Embora os problemas de *vanishing* e *exploding gradients* recebam mais atenção, essa variação atrapalha a otimização das redes neurais
 - ▶ Este problema é “um pouco menos” severo

- ▶ Solução: Normalização de batch (ou *batch normalization*)

$$\mu_{\mathcal{B}} = \frac{1}{m} \sum_{i=1}^m a_i$$

$$\sigma_{\mathcal{B}}^2 = \frac{1}{m} \sum_{i=1}^m (a_i - \mu_{\mathcal{B}})^2$$

$$\hat{a}_i = \frac{a_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

$$a'_i = \gamma \hat{a}_i + \beta$$

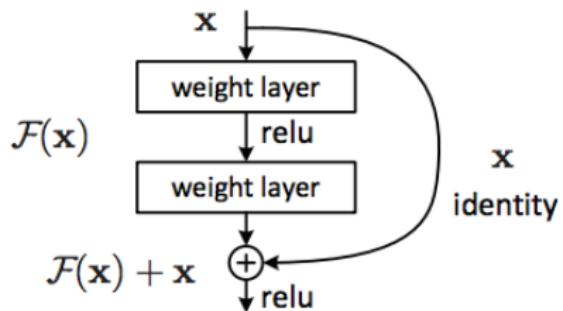
onde a_i é a ativação do neurônio e a'_i é o novo valor da ativação após a normalização de batch; e γ e β são hyperparâmetros que podem ser aprendidos pela rede neural

Ioffe, S. & Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. arXiv. 2015 **Fonte:** <https://arxiv.org/pdf/1502.03167.pdf>

Redes Neurais Residuais

- ▶ A otimização de redes neurais com muitas camadas é bastante difícil
- ▶ Os problemas gerados por *vanishing gradients* são confrontados usando-se uma inicialização inteligente de parâmetros
- ▶ No entanto, outro problema descoberto recentemente tem um efeito parecido durante a etapa de *forward*

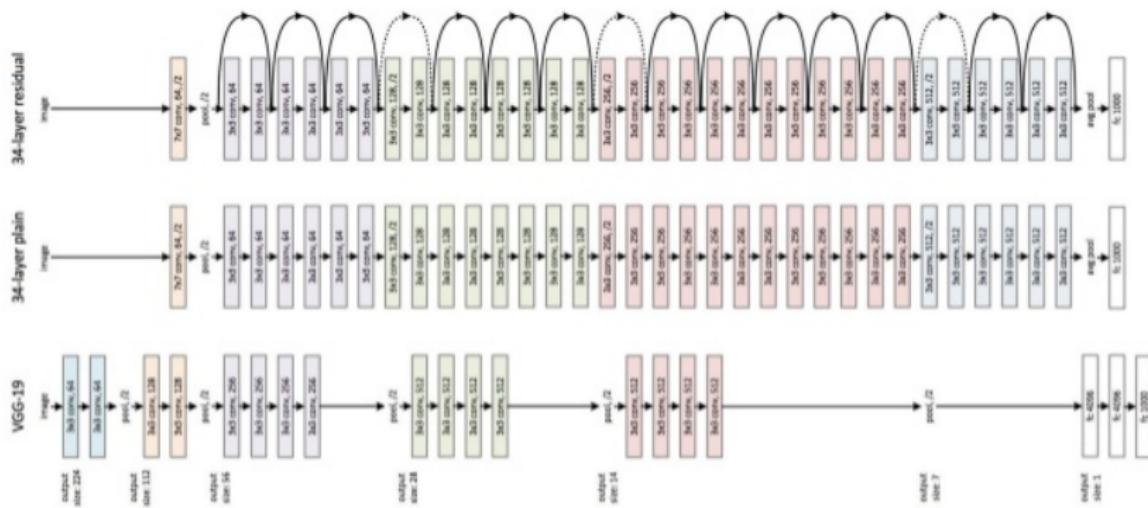
- ▶ Cada camada da rede neural “aprende” uma função independente das outras camadas
- ▶ Isto faz com que nas camadas finais as *features* aprendidas já não tenham mais a mesma robustez em relação àquelas aprendidas nas camadas iniciais
- ▶ Solução: redes neurais residuais (ou *residual networks*)



- ▶ A utilização deste “reforço” no sinal estimula as camadas para que aprendam funções que sejam mais dependentes umas das outras

He et al. Deep Residual Learning for Image Recognition. arXiv. 2015 Fonte:
<https://arxiv.org/pdf/1512.03385.pdf>

ResNet



Somnath Banerjee. LeNet to ResNet: A deep Journey. 2017. Fonte:

<https://www.slideshare.net/SomnathBanerjee17/lenet-to-resnet>

Métodos adaptativos de Gradiente Descendente

- ▶ Embora o método de Gradiente Descendente (SGD) seja o padrão para otimização de redes neurais artificiais, existem outros métodos que estendem sua formulação básica

$$\hat{\theta} = \theta - \alpha \odot \nabla \mathcal{C}$$

▶ Momentum

$$\begin{aligned}v_t &= \gamma v_{t-1} + \alpha \odot \nabla \mathcal{C} \\ \hat{\theta} &= \theta - v_t\end{aligned}$$

- ▶ Nesterov Accelerated Gradient (Nesterov Momentum)

$$\begin{aligned}v_t &= \gamma v_{t-1} + \alpha \odot \nabla \mathcal{C}(\theta - \gamma v_{t-1}) \\ \hat{\theta} &= \theta - v_t\end{aligned}$$

► Adaptive Gradient (AdaGrad)

$$\begin{aligned} g_t &= \nabla \mathcal{C} \\ \theta_{t+1} &= \theta_t - \frac{\alpha}{\sqrt{G_t + \epsilon}} \odot g_t \end{aligned}$$

onde G_t é uma matriz diagonal que contém os gradientes anteriores (primeiro elevados ao quadrado e depois somados)

► RMSprop

$$\begin{aligned}E[g^2]_t &= \gamma E[g^2]_{t-1} + (1 - \gamma) g_t^2 \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t\end{aligned}$$

onde $E[g^2]_t$ é uma matriz diagonal que contém uma “janela” dos gradientes anteriores ao invés de conter todos (mais uma vez, os gradientes são elevados ao quadrado e depois somados)

► Adaptive Learning Rate (AdaDelta)

$$g_t = \nabla \mathcal{C}$$

$$RMS[g]_t = \sqrt{\gamma E[g^2]_{t-1} + (1 - \gamma)g_t^2 + \epsilon}$$

$$RMS[\Delta\theta]_t = \sqrt{\gamma E[\Delta\theta^2]_{t-1} + (1 - \gamma)\Delta\theta_t^2 + \epsilon}$$

$$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

$RMS[g]_t$ é semelhante ao $E[g^2]_t$ definido para o RMSProp; e $RMS[\Delta\theta]_t$ é definida da mesma forma que $RMS[g]_t$ mas usando os valores anteriores de θ

► Adaptive Momentum Estimate (Adam)

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

► Recomendações

- ▶ Se os dados forem esparços (muitos 0s), Adadelta, RMSprop e Adam são as melhores escolhas
- ▶ De maneira geral, SGD provê os melhores resultados
- ▶ contudo, o SGD precisa de mais iterações para atingir tal efeito
- ▶ SGD é menos resistente a inicialização de parâmetros e pode ficar “preso” em “*saddle points*”

Resumo

- ▶ Redes neurais profundas são os modelos mais avançados de IA no momento
- ▶ Ainda é difícil treiná-las e obter bons resultados
- ▶ No entanto é fácil encontrar modelos pré-treinados para adaptar à outras necessidades

- ▶ Entender os métodos mais recentes que foram utilizados para criá-las facilita o processo de adaptação
- ▶ No entanto, o entendimento de como as redes neurais funcionam ainda está na fase da experimentação
 - ▶ Ainda falta uma teoria que explique seu funcionamento e nos guie na criação/otimização de modelos

Laboratório

Redes Neurais Artificiais e Recursos Avançados do Tensorflow