Aprendizagem Profunda
(Deep Learning)

Marcondes Ricarte e Aluizio Fausto Ribeiro Araújo

Universidade Federal de Pernambuco

Centro de Informática

Conteúdo

- Motivação
- Por que aprendizagem profunda?
- Redes Neurais Convolucionais
- *Auto-encoder* Profundo
  - *Stacked Auto-encoder*
  - Outros Tipos de *Auto-encoders*
- Ferramentas

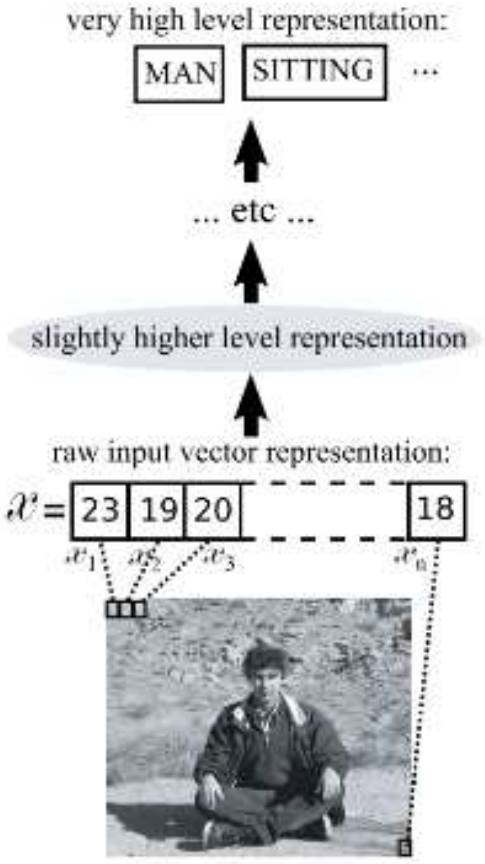
Motivação

- Permitir aos computadores modelar nosso mundo bem o suficiente para exibir o que nós chamamos de inteligência tem sido o foco de pesquisas de mais da metade de um século.
- Para alcançar esse objetivo, é claro que a grande quantidade de informação sobre o nosso mundo deve ser de alguma forma armazenada, explicitamente ou implicitamente, no computador:
  - Para isto se utiliza algoritmos de aprendizagem.

Motivação

- Muito esforço (e progresso!) tem sido feito em entender e melhorar algoritmos de aprendizagem, mas o desafio permanece:
  - Há algoritmos capazes de entender cenas e descrevê-las em linguagem natural?
  - Há algoritmos capazes de inferir conceitos semânticos suficientes a ponto de interagir com humanos?

Motivação

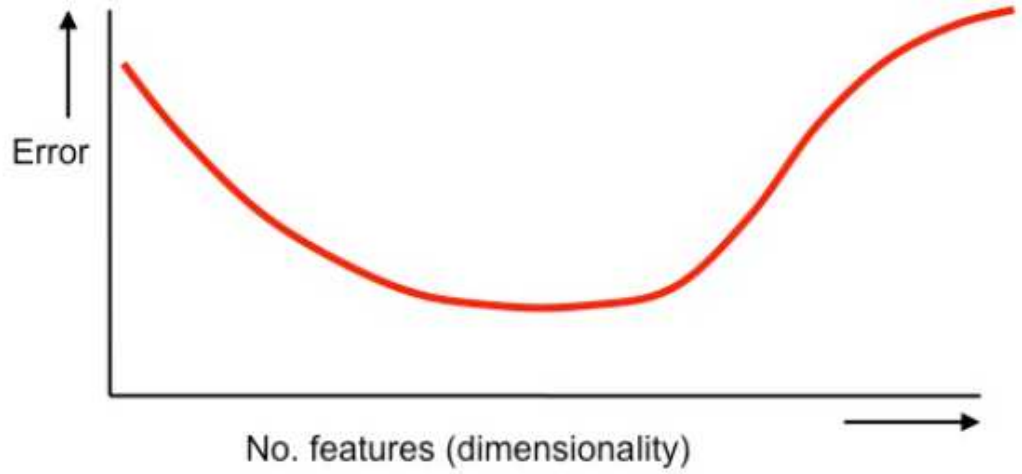

very high level representation:
MAN
SITTING
...
... etc ...
slightly higher level representation
raw input vector representation:
23
19
20
18

Motivação

- Um modo plausível e comum de extrair informação útil a partir de imagens naturais envolve transformar gradualmente os pixels com valores “brutos” em representações mais abstratas:
  - Detecção de borda, detecção de formas locais, identificação de categorias.
- Assim, uma máquina que expresse “inteligência” requer funções matemáticas que expressem variabilidade:
  - Não-lineares em termos de entradas cruas, sendo capazes de apresentar muitas variações.

Motivação

- Redes Neurais Artificiais feed-forward:
  - Maldição da dimensionalidade.



Motivação

- Uma das soluções encontradas para maldição de dimensionalidade é pré-processamento de dados:
  - Redução da dimensionalidade (às vezes por humanos);
  - Desafiante e altamente dependente da tarefa.
- Se pensarmos no cérebro humano, não há indícios de que ele resolva esse problema dessa forma.
- Aprender características automaticamente em múltiplos níveis de abstração permite ao sistema mapear funções complexas sem depender de características intermediárias inteligíveis aos humanos.

Motivação

- As camadas no MLP não aprendem bem:
  - Difusão do gradiente;
  - Treinamento muito lento;
  - Camadas mais baixas tendem a fazer um mapeamento aleatório;
  - Frequentemente há mais dados não rotulados do que dados rotulados;
  - Existência de mínimo locais.

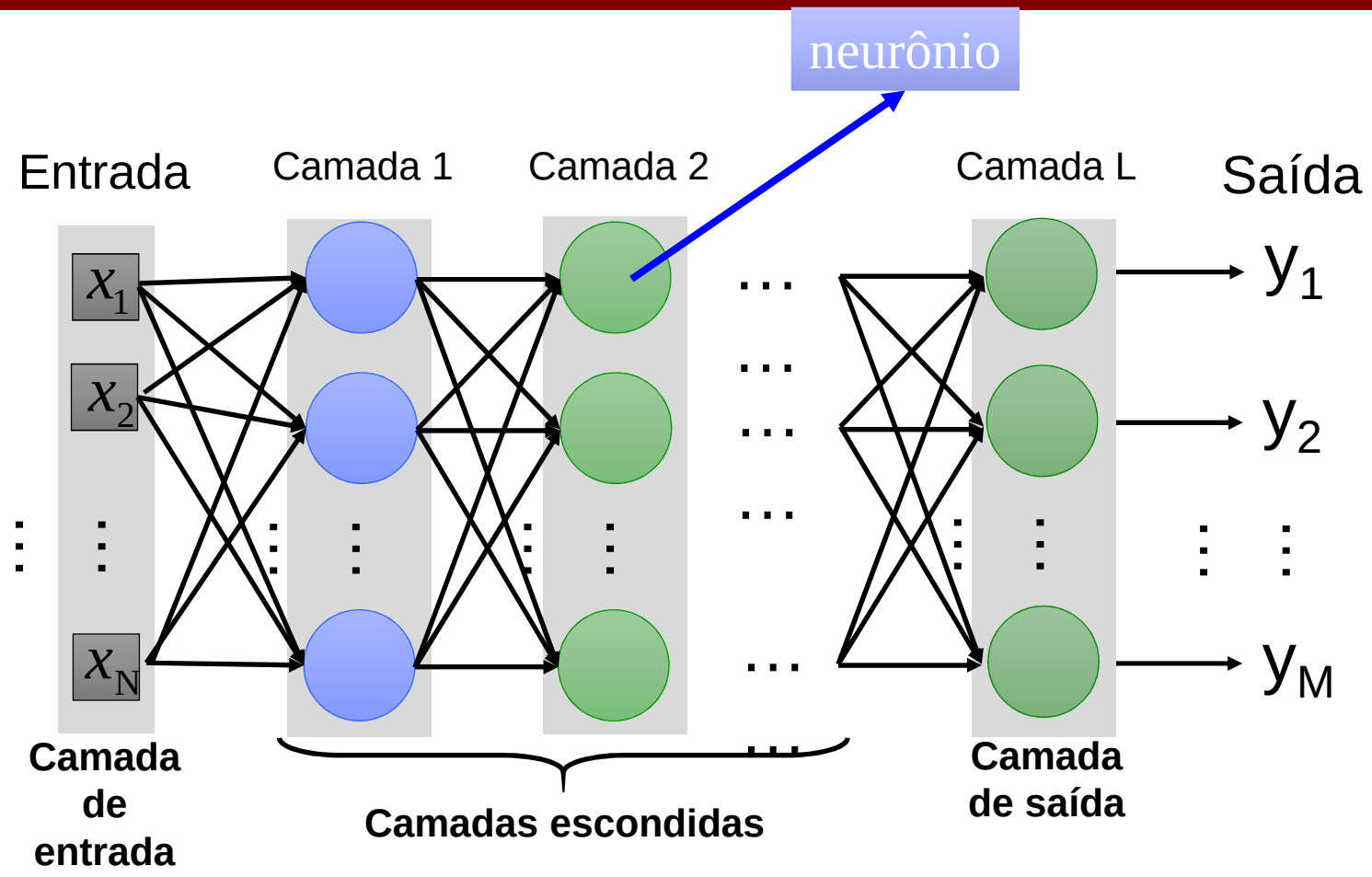
Motivação

- Aprendizagem Profunda aprende em vários níveis de abstração;
  - Representações mais abstratas extraem informações para classificadores ou preditores;
  - Características intermediárias aprendidas podem ser compartilhadas entre diferentes tarefas.
- Por décadas, diversos pesquisadores tentaram, sem sucesso, treinar redes neurais de múltiplas camadas profundas;
  - Inicializadas com pesos aleatórios levavam a mínimos locais.
- Hinton *et al* (2006) melhoraram o desempenho de uma rede neural profunda com etapa de pré-treinamento por aprendizagem não-supervisionada, uma camada após outra a partir da primeira.

Por que Aprendizagem Profunda (*Deep Learning*)?

- Teorema de Kolmogorov-Smirnov: uma única camada escondida pode resolver qualquer função e generalizar:
  - Grande número de nodos pode inviabilizar solução do problema.
- Múltiplas camadas constroem melhor espaço de características:
  - Primeira camada aprende as características de primeira ordem (por exemplo, bordas em imagem);
  - Segunda camada aprende características de maior ordem (por exemplo, combinação de bordas e outras características);
  - Camadas são treinadas por método não-supervisionado e as características alimentam uma camada supervisionada;
    - A rede inteira é então ajustada de modo supervisionado.

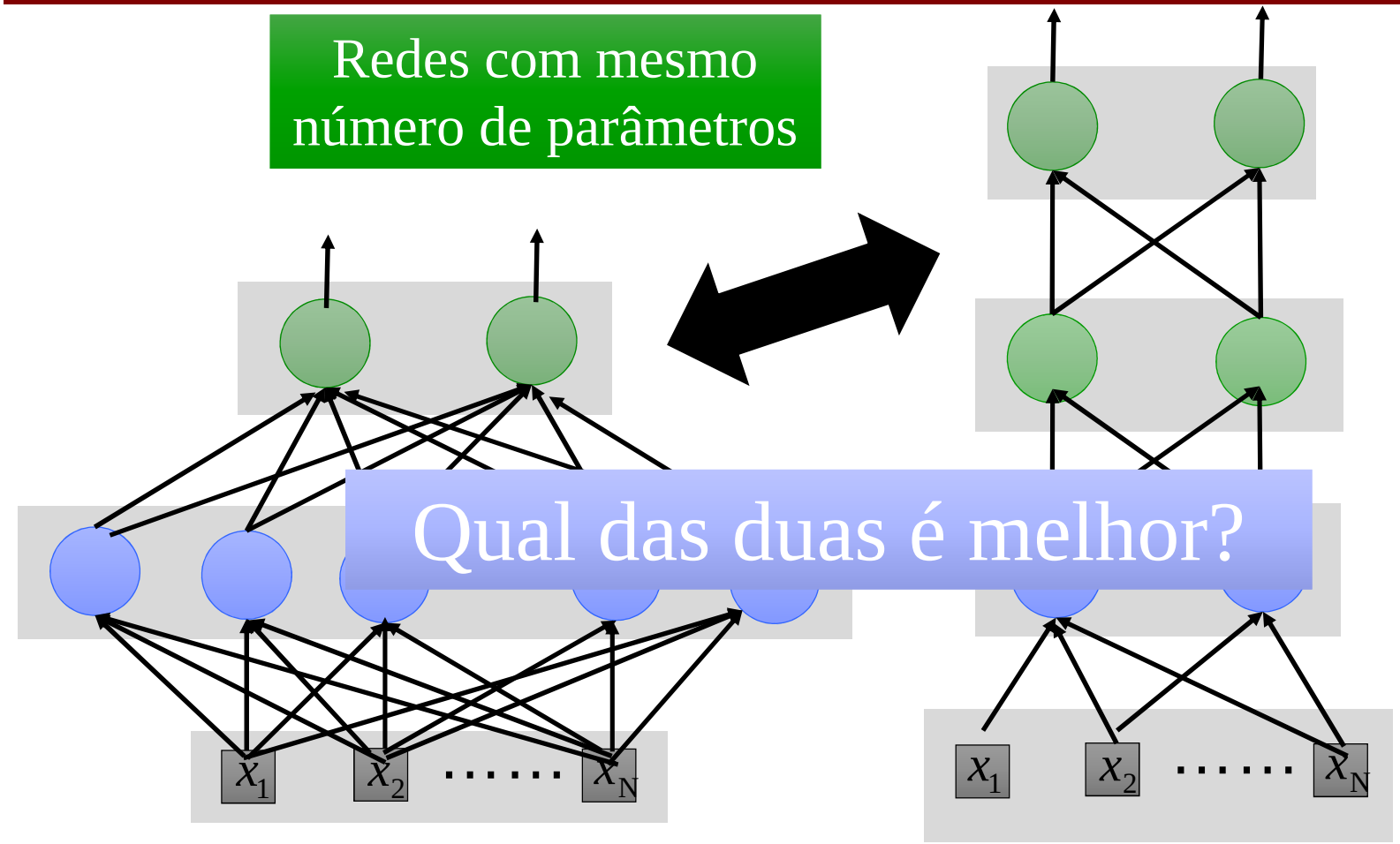
Por que Aprendizagem Profunda (*Deep Learning*)?



Profundo significa muitas camadas

Por que Aprendizagem Profunda (*Deep Learning*)?

Redes com mesmo número de parâmetros



Rasa: gorda e baixa

Profunda: Magra e alta

Por que Aprendizagem Profunda (*Deep Learning*)?

# de camadas X tamanho	Taxa de erro das palavras (%)	# de camadas X tamanho	Taxa de erro das palavras (%)
1 X 2k	24.2		
2 X 2k	20.4		
3 X 2k	18.4		
4 X 2k	17.8		
5 X 2k	17.2	1 X 3772	22.5
7 X 2k	17.1	1 X 4634	22.6
		1 X 16k	22.1

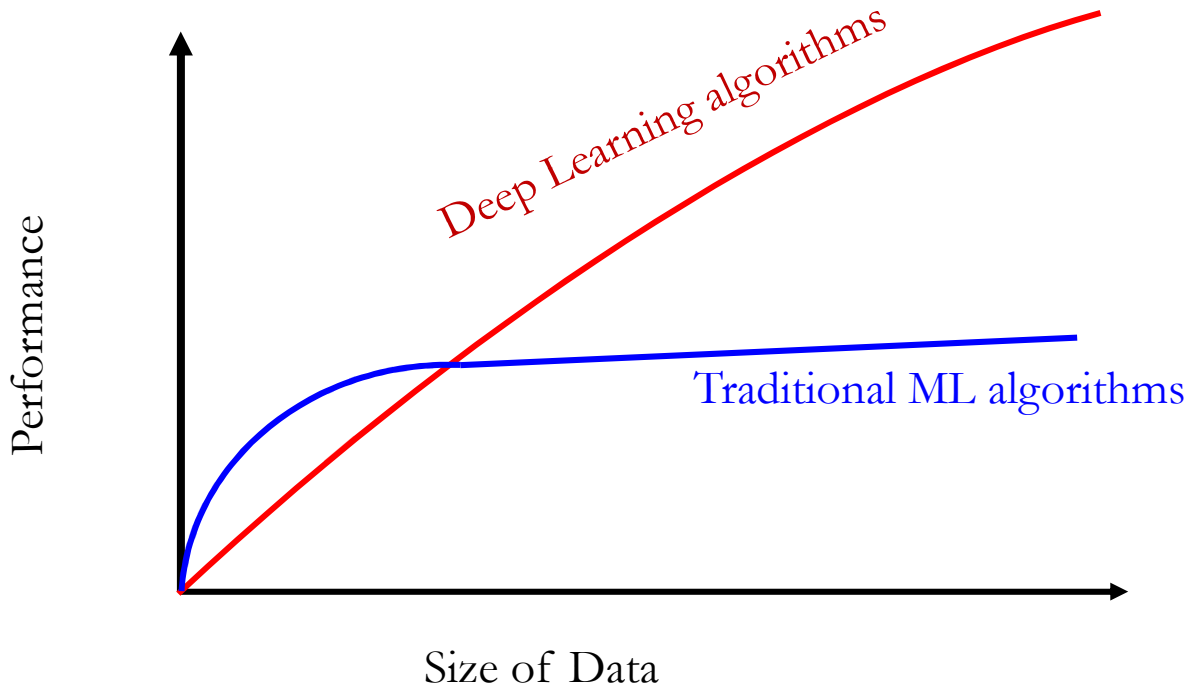
Seide, Frank, Gang Li, and Dong Yu. "Conversational Speech Transcription Using Context-Dependent Deep Neural Networks." *Interspeech*. 2011.

Por que Aprendizagem Profunda (*Deep Learning*)?

- Primeiras MLPs com muitas camadas, constatações do treinamento:
  - Gradiente é muito reduzido com retropropagação dos erros para primeiras camadas, causando lentidão excessiva no treinamento;
  - As camadas superiores costumam aprender bem, reduzindo o erro, portanto, o erro retropropagado para camadas anteriores cai rapidamente, logo as primeiras camadas não conseguem se adaptar para melhorar os resultados, elas fazem o papel de um mapa aleatório de características;
  - Necessidade de um modo efetivo de treinar as camadas iniciais.

Por que Aprendizagem Profunda (*Deep Learning*)?

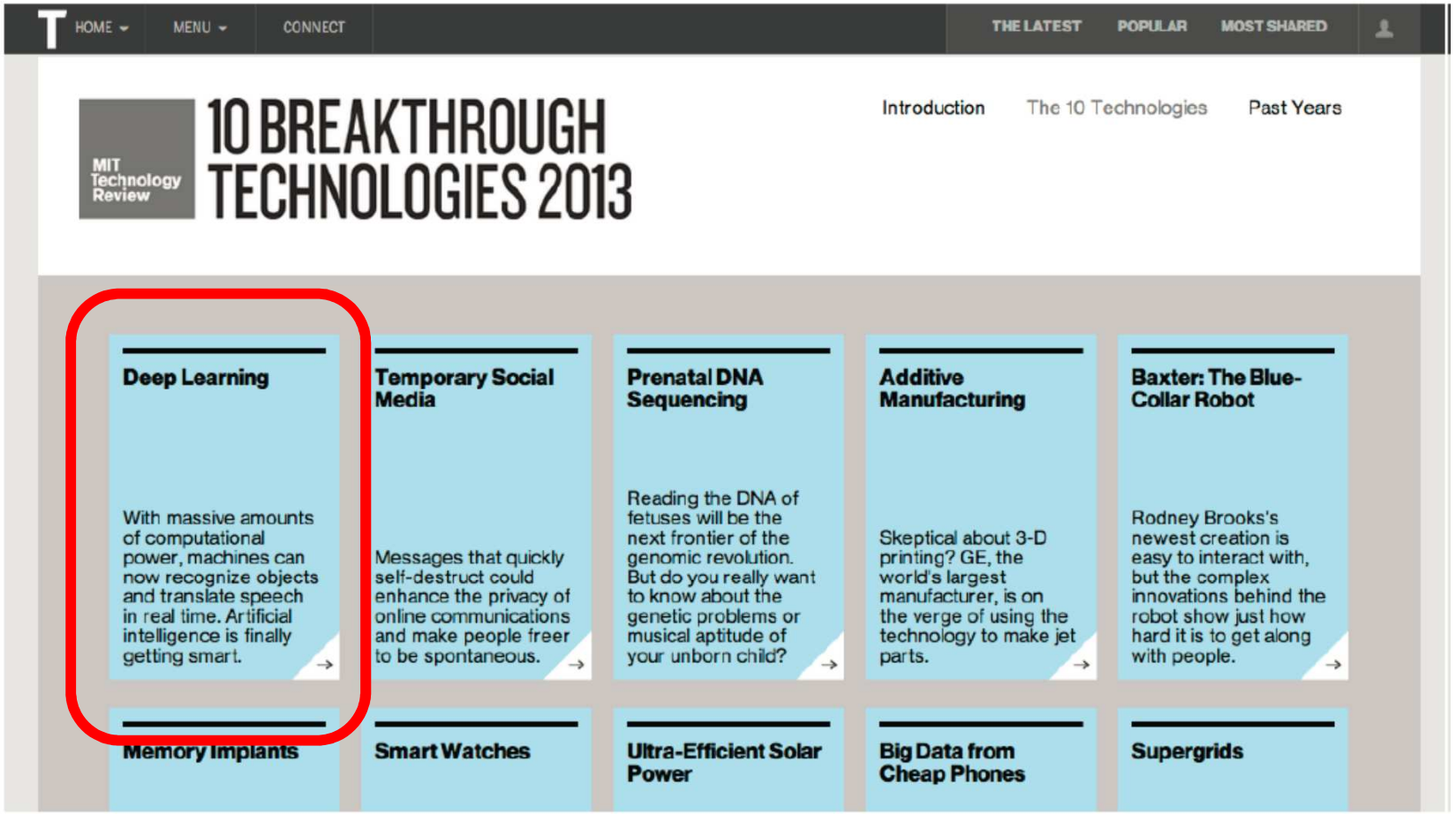
Desempenho vs Tamanho dos dados



Por que Aprendizagem Profunda (*Deep Learning*)?

- Plausibilidade biológica – córtex visual
- Problemas que podem ser representados com um número polinomial de nodos com *k* camadas podem requerer número exponencial de nós com *k*-1 camadas:
  - Funções muito variáveis podem ser eficientemente representadas com arquiteturas profundas.

Por que Aprendizagem Profunda (*Deep Learning*)?



MIT Technology Review, April 23rd, 2013

Por que Aprendizagem Profunda (*Deep Learning*)?

- Alguns modelo importantes:
  - Supervisionado:
    - Convolutional NN (LeCun)
    - Recurrent Neural nets (Schmidhuber)
  - Não-supervisionado:
    - Deep Belief Nets / Stacked RBMs (Hinton)
    - Stacked denoising autoencoders (Bengio)
    - Sparse AutoEncoders (LeCun, A. Ng,)

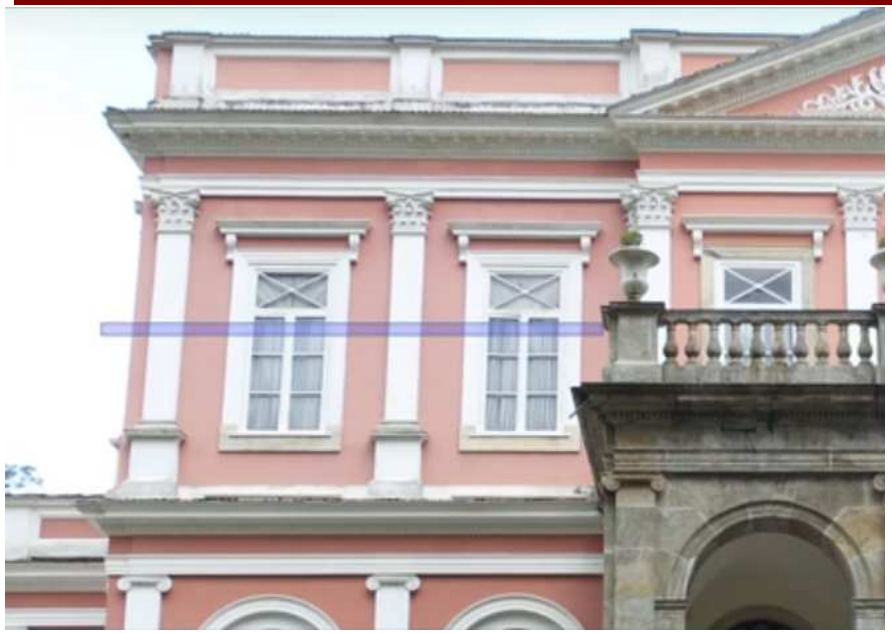
Redes Neurais Convolucionais

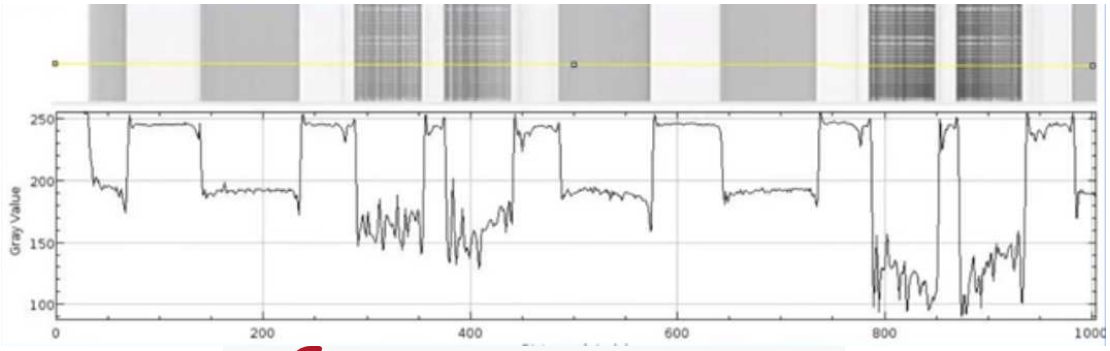
(*Convolutional Neural Networks*)

- Operações básicas empregadas no modelo:
  - Convolução;
  - Inserção de não-linearidade;
  - Sub-amostragem.

Redes Neurais Convolucionais

(*Convolutional Neural Networks*) - Convolução





Redes Neurais Convolucionais

(*Convolutional Neural Networks*) - Convolução

- **Convolução** é um operador linear que toma duas funções no domínio do tempo e produz uma terceira função que calcula a área subentendida pela superposição delas em função do deslocamento existente entre elas.
- Contínua:

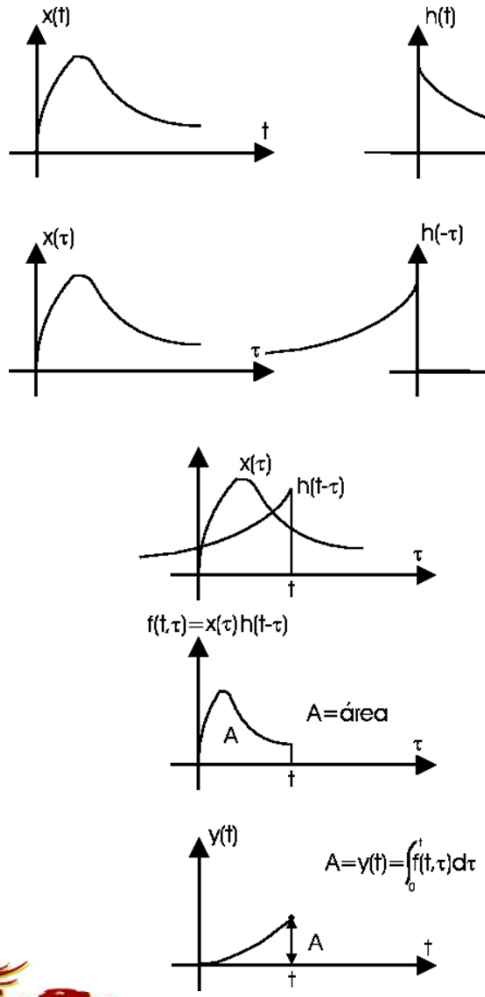
$$(f * g)(x) = h(x) = \int_{-\infty}^{\infty} f(u) \cdot g(x-u)\, du$$

- Discreta:

$$(f * g)(k) = h(k) = \sum_{j=0}^{k} f(j) \cdot g(k-j)$$

Redes Neurais Convolucionais

(*Convolutional Neural Networks*) - Convolução



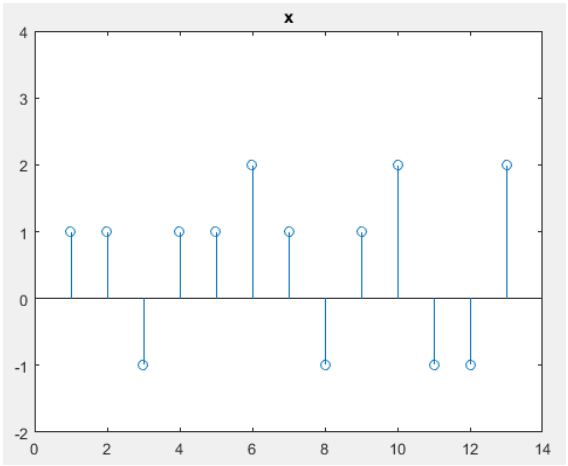
- Convolução é formada pelas operações:
  - Muda-se a variável independente;
  - Escolhe-se uma das funções que é invertida com respeito ao eixo vertical e defasada no tempo;
  - Move-se a 'função móvel' com respeito à "função fixa', integrando-se os intervalos de coexistência das funções.

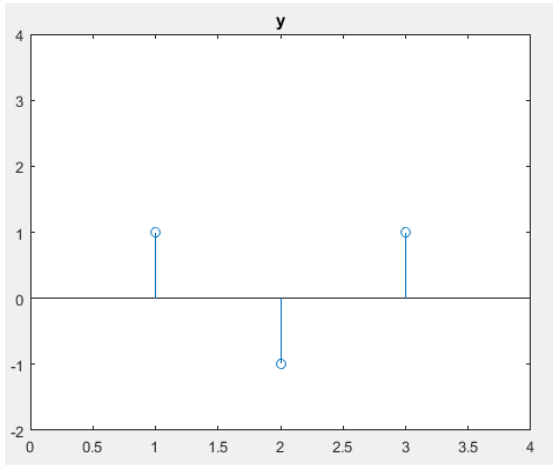
Redes Neurais Convolucionais

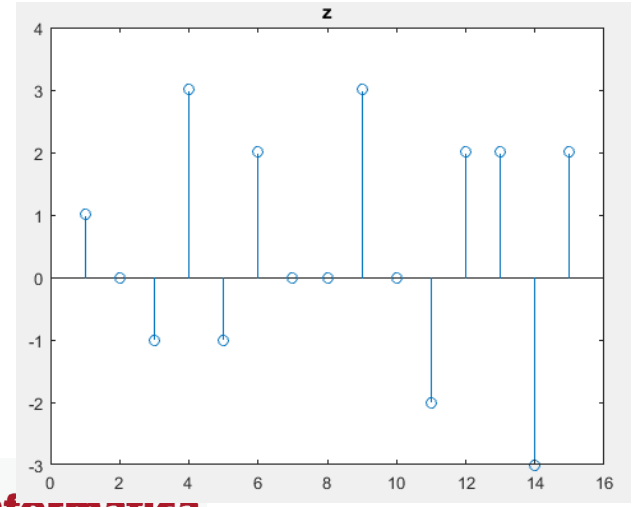
(*Convolutional Neural Networks*) - Convolução

- Convolução discreta:

```
>> x = [1 1 -1 1 1 2 1 -1 1 2 -1 -1 2];
>> y = [1 -1 1];
>> z = conv(x,y)
```



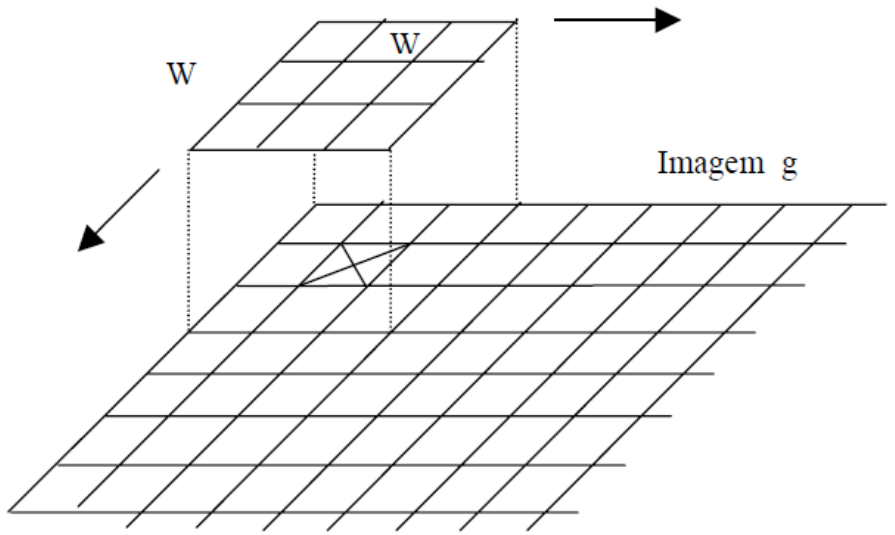




Redes Neurais Convolucionais
(*Convolutional Neural Networks*) - Convolução

- Convolução 2D (para imagens):

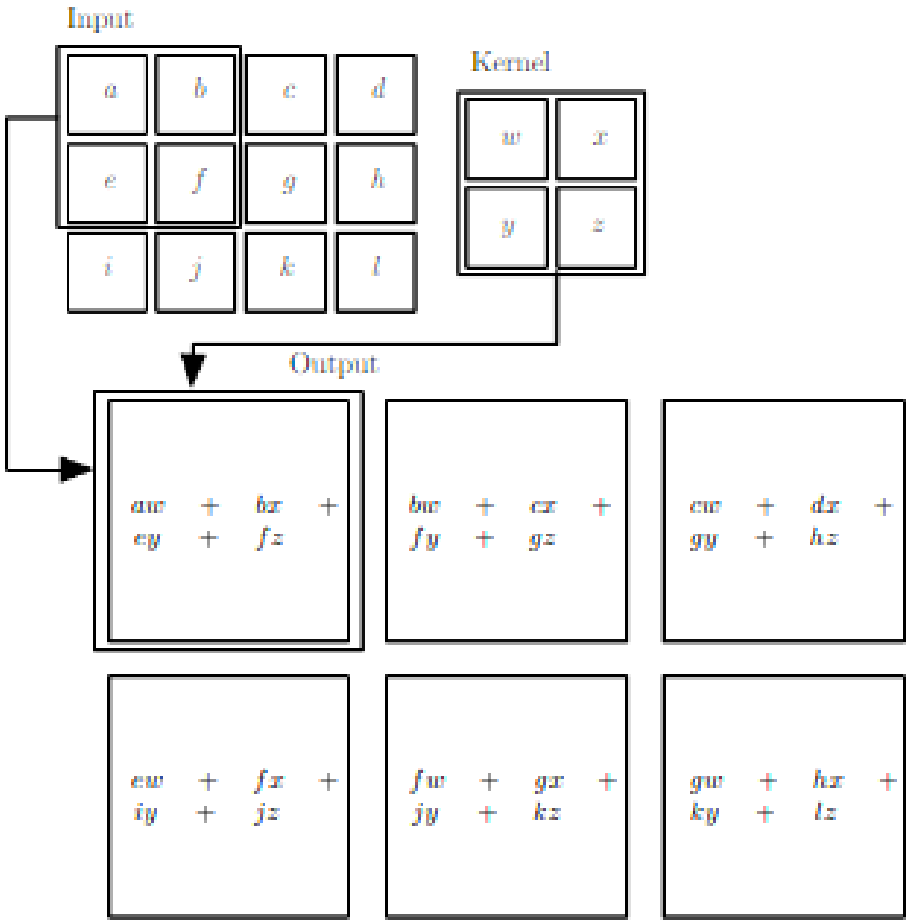
$$g_f(l,p) = \sum_{m=-\text{int}(w/2)}^{\text{int}(w/2)} \sum_{n=-\text{int}(w/2)}^{\text{int}(w/2)} g(l+m, p+n)\, h(m,n)$$



Redes Neurais Convolucionais

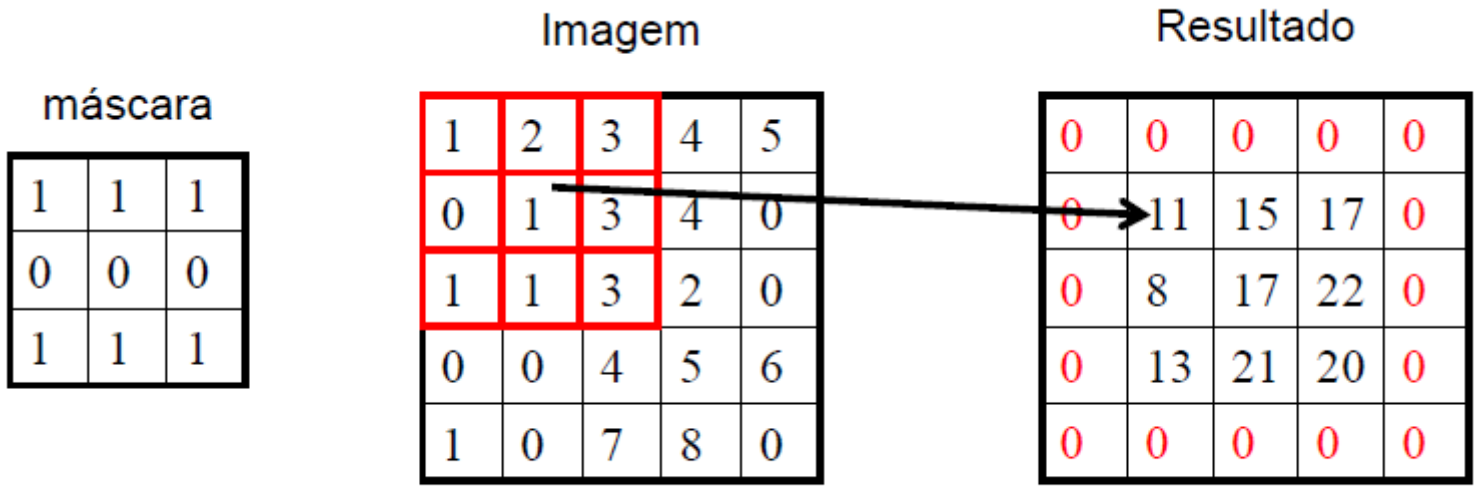
(*Convolutional Neural Networks*) - Convolução

- Convolução 2D (para imagens):



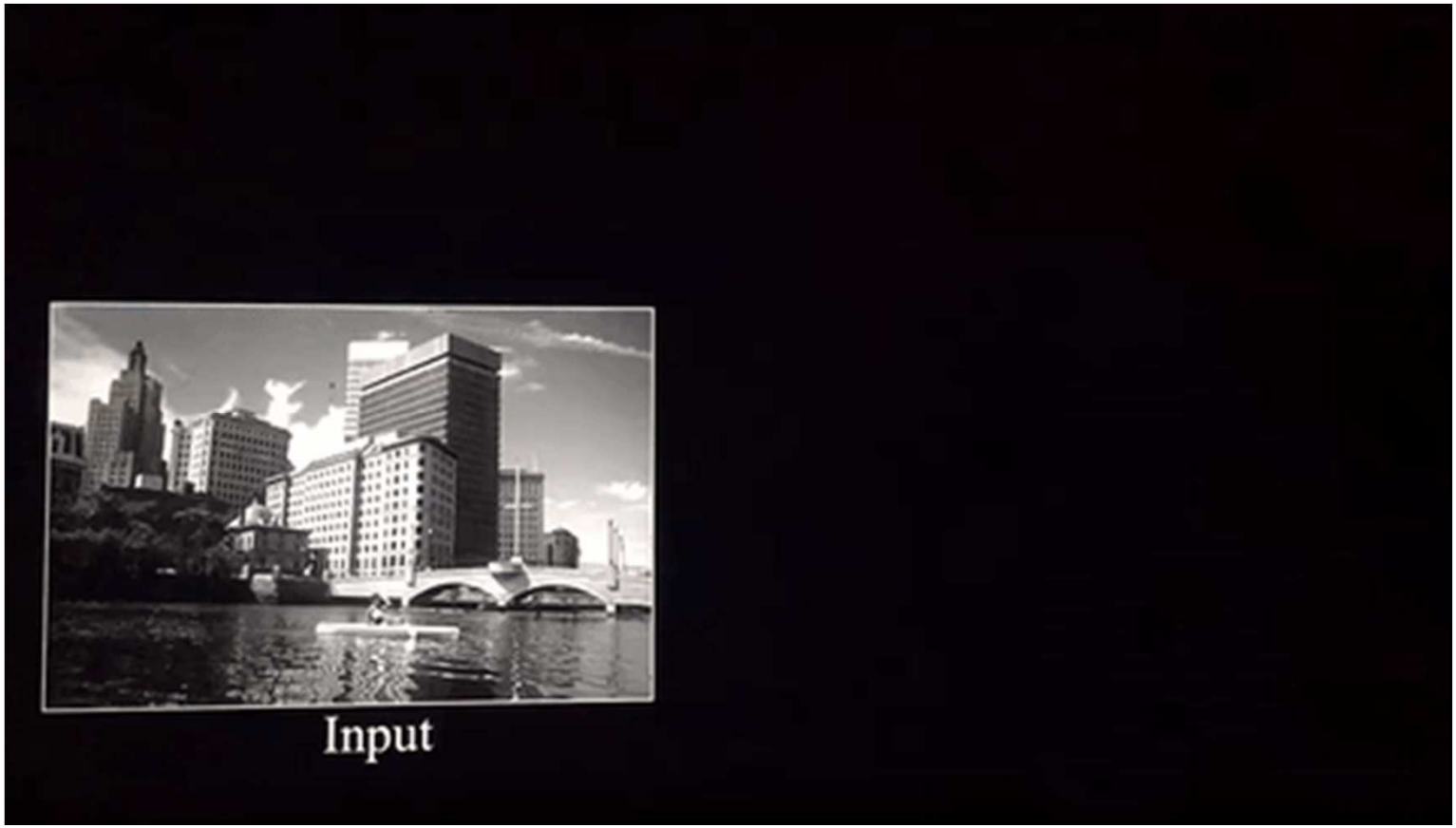
Redes Neurais Convolucionais
(*Convolutional Neural Networks*) - Convolução

- Convolução 2D (para imagens):



Redes Neurais Convolucionais

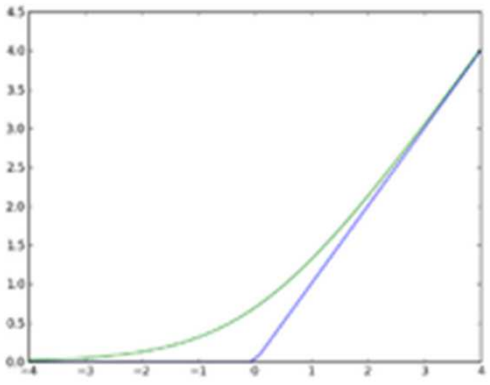
(*Convolutional Neural Networks*) - Convolução

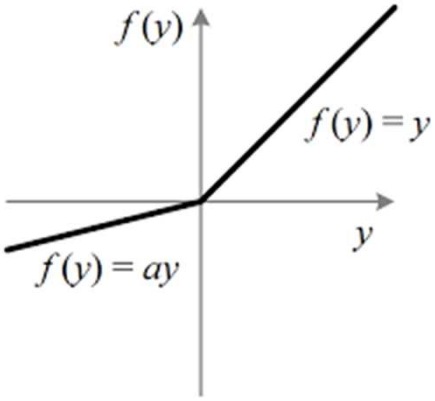


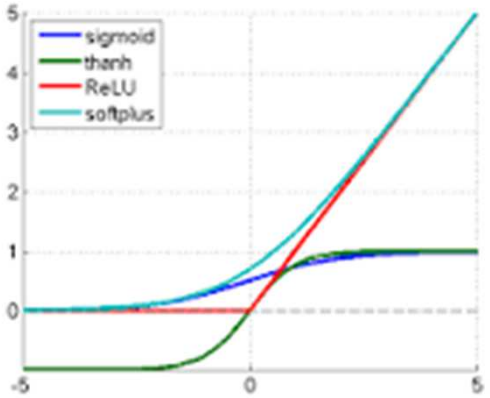
Redes Neurais Convolucionais

(*Convolutional Neural Networks*) - Inserção de não linearidade

- Definições das funções de ativação:
  - ReLU: Rectified Linear Unit: $f(x)$ = Max(0,x), derivada constante ou nula;
  - Leaky ReLU $f(x) = x$ if $x > 0$ else ax para $0 \leq a \leq 1$, assim a derivada fica constante negativa mas não vai para zero;
- Ativação esparsa para produzir um saída, i.e., muitas unidades escondidas não produzem saída;





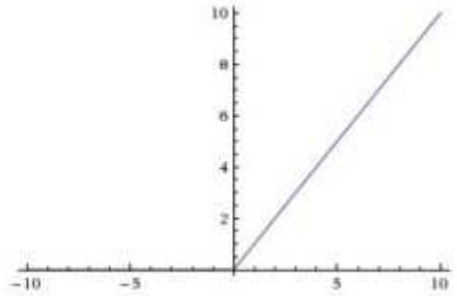


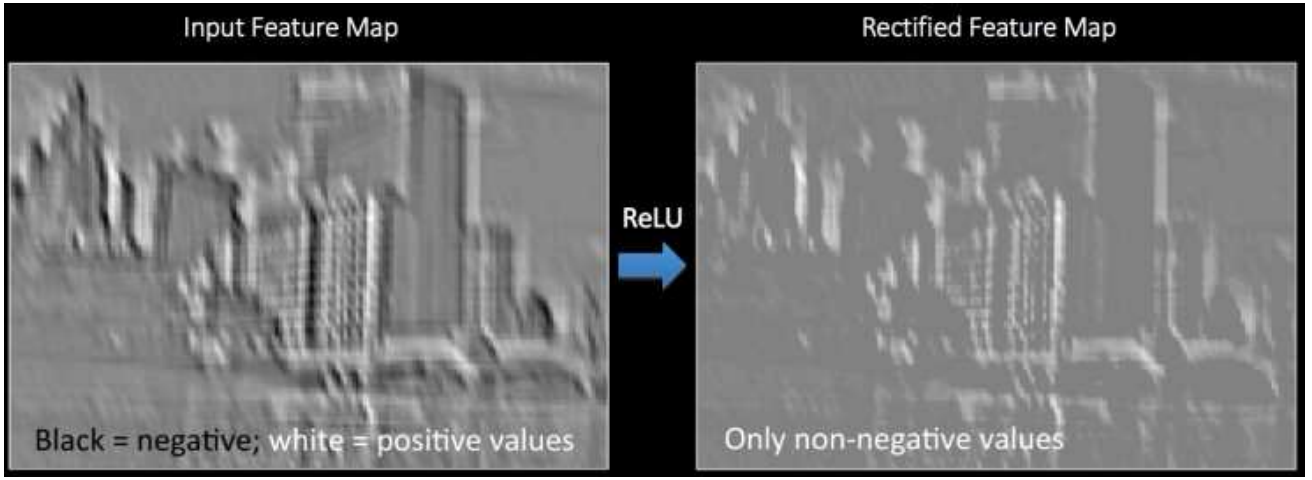
Redes Neurais Convolucionais

(*Convolutional Neural Networks*) - Inserção de não linearidade

- Exemplo com ReLU:

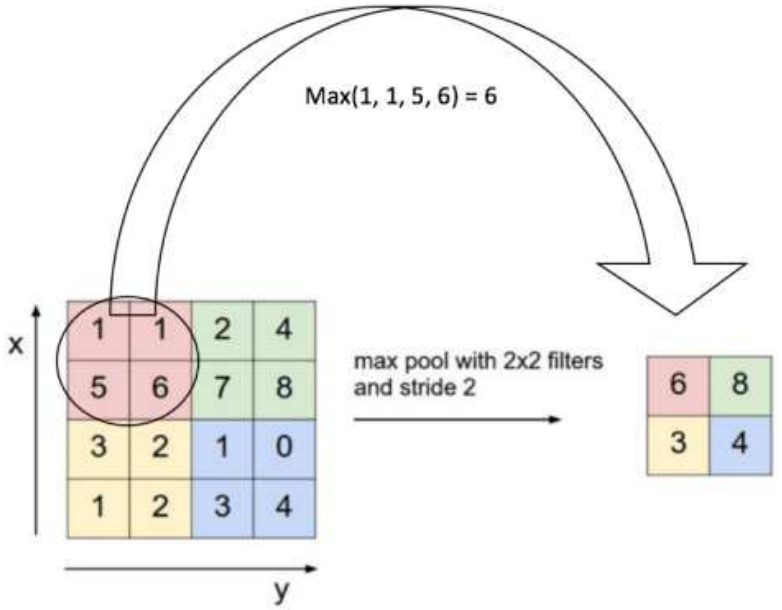
Output = Max(zero, Input)





Redes Neurais Convolucionais
(*Convolutional Neural Networks*) - Subamostragem

- *Pooling* (junção, agrupamento):
  - Esse passo comprime e suaviza os dados;
  - Normalmente toma a média ou o máximo entre trechos disjuntos;
  - Dá robustez a pequenas variações espaciais dos dados.



Redes Neurais Convolucionais

(*Convolutional Neural Networks*) - Evidências

- Hubel e Wiesel em 1968, estudaram o sistema visual de felinos e constataram o papel importante das chamadas *Receptive Cells* que agiam sobre como filtros locais sobre o espaço de entrada e tinham dois comportamentos:
  - *Simple Cells*: Respondem a padrões de bordas na imagem;
  - *Complex Cells*: Possuem campos receptivos grandes e são invariantes à posição do padrão.

Redes Neurais Convolucionais

(*Convolutional Neural Networks*) - Evidências

A bit of history:

Hubel & Wiesel,

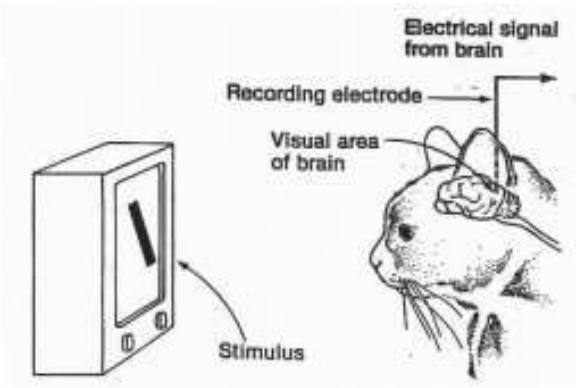
1959

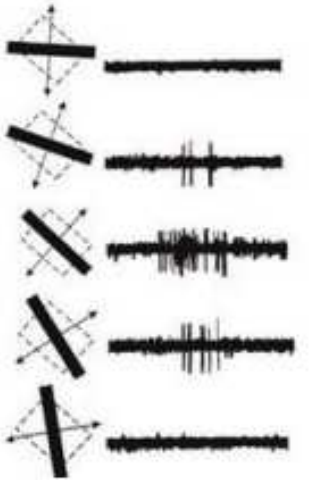
RECEPTIVE FIELDS OF SINGLE NEURONES IN THE CAT'S STRIATE CORTEX

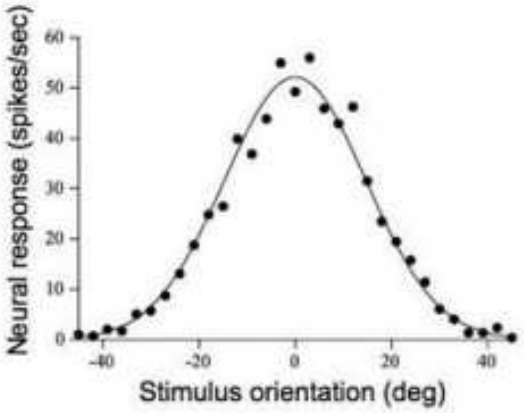
1962

RECEPTIVE FIELDS, BINOCULAR INTERACTION AND FUNCTIONAL ARCHITECTURE IN THE CAT'S VISUAL CORTEX

1968...

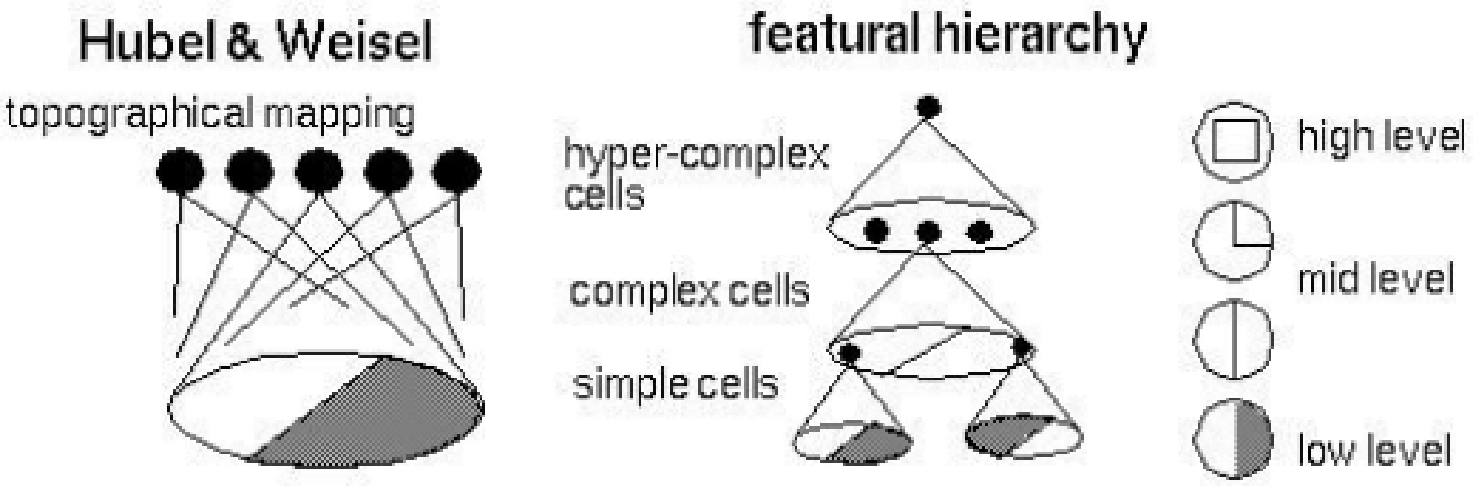






Redes Neurais Convolucionais

(*Convolutional Neural Networks*) - Evidências



Redes Neurais Convolucionais

(*Convolutional Neural Networks*) - História

- Fukushima (1980) – *Neo-Cognitron*;
- MLP (Rumelhart et al., 1986): marco importante;
- Novos modelos relevantes com SVM, final dos 1990s;
- *Convolutional Neural Nets* (LeCun, 1998): Aplicado a imagens, fala;
  - Apresenta similaridades com *Neo-Cognitron*;
- MLP treinada com BP com muitas camadas:
  - As primeiras tentativas tiveram baixo índice de sucesso pois
    - Eram muito lentas;
    - Havia gradiente que ia para zero.

Redes Neurais Convolucionais

(*Convolutional Neural Networks*) - História

- Trabalhos mais recentes mostraram avanços com treinamento lento de MLPs com BP profundas usando máquinas com GPUs;
- *Deep Belief nets* (Hinton) and *Stacked auto-encoders* (Bengio), ambas em 2006: Pré-treinamento não-supervisionado precede treinamento supervisionado;
- Sucessos iniciais em 2012 com aprendizagem supervisionada que conseguiu superar gradientes que desaparecem e se torna mais aplicável.

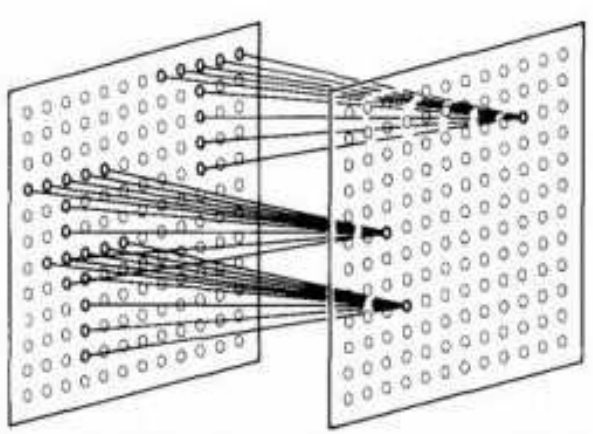
Redes Neurais Convolucionais

(*Convolutional Neural Networks*) - História

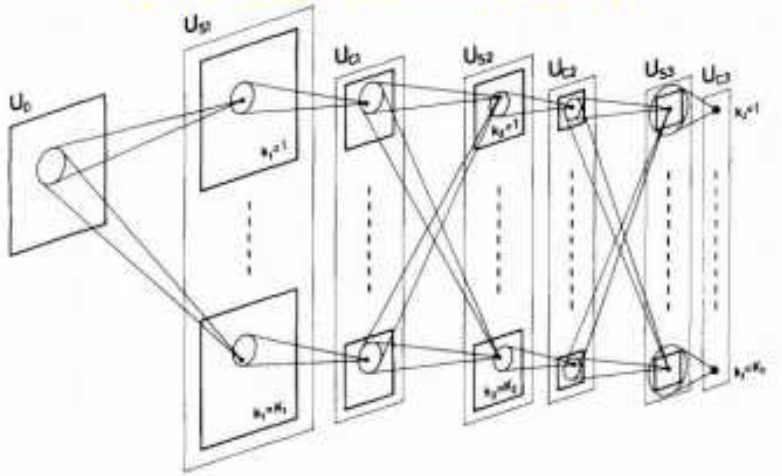
A bit of history:

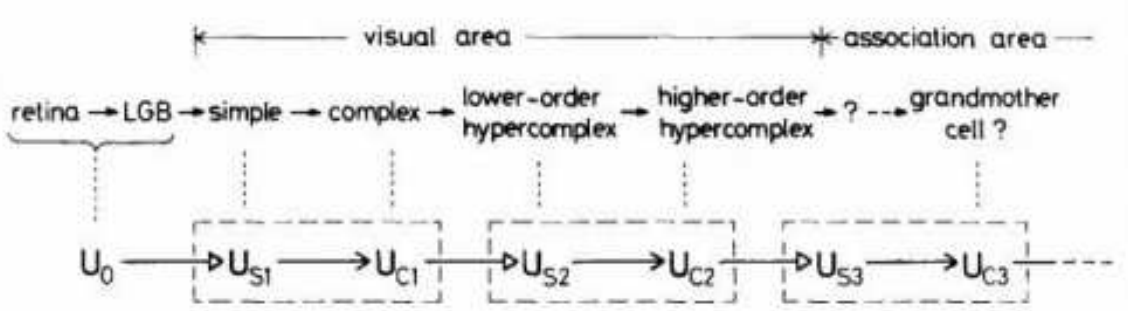
Neurocognitron

[Fukushima 1980]



"sandwich" architecture (SCSCSC...)
simple cells: modifiable parameters
complex cells: perform pooling





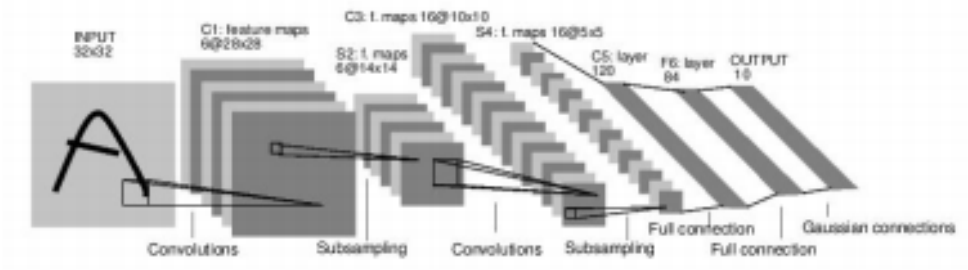
Redes Neurais Convolucionais

(*Convolutional Neural Networks*) - História

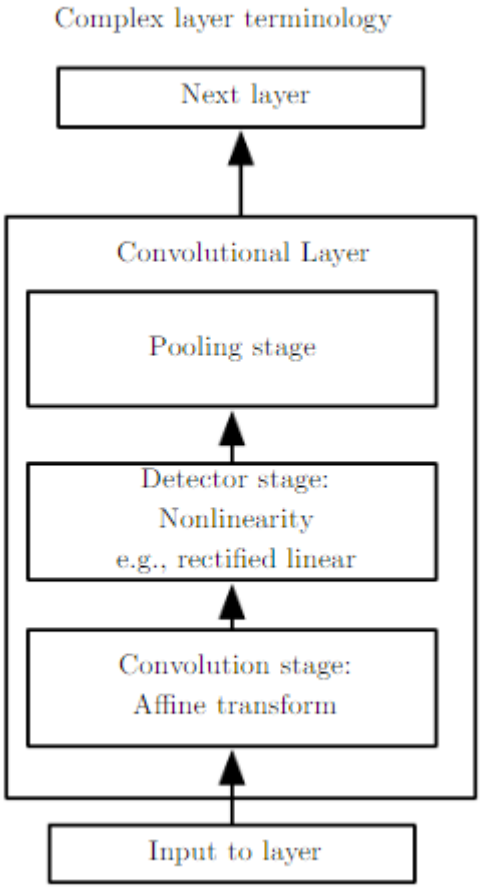
A bit of history:

Gradient-based learning applied to document recognition

[LeCun, Bottou, Bengio, Haffner 1998]

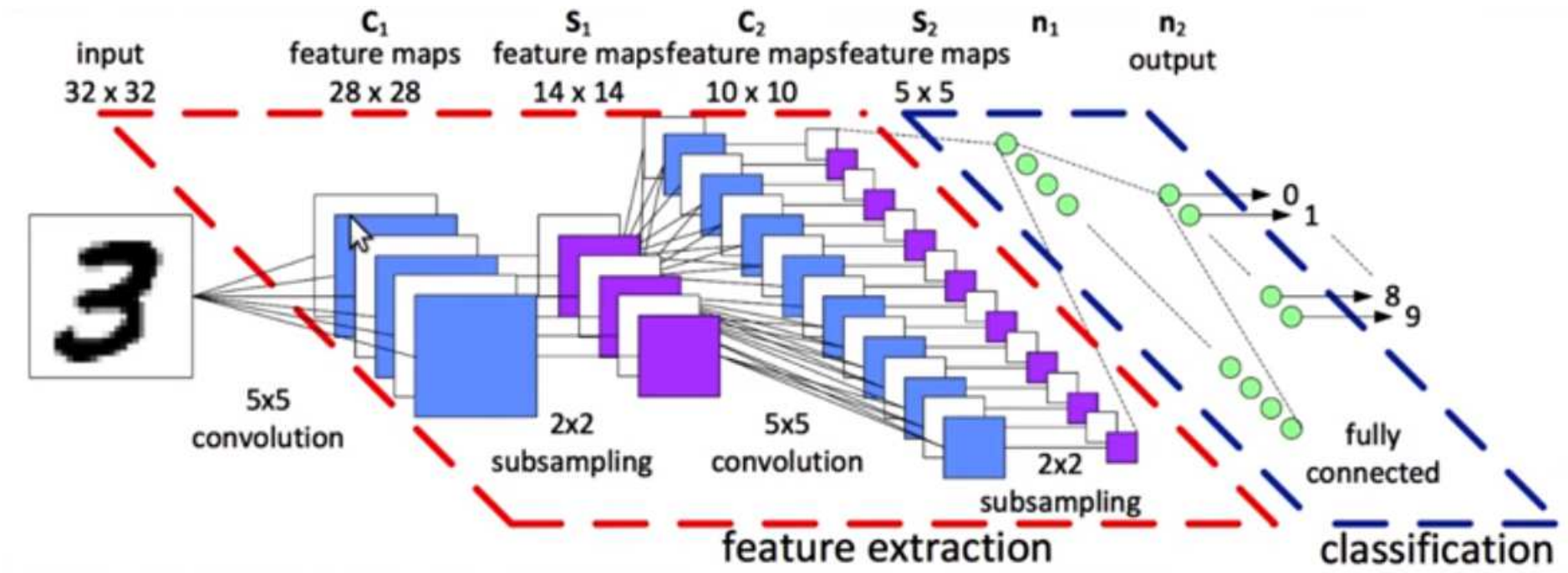


LeNet-5



Redes Neurais Convolucionais

(*Convolutional Neural Networks*)



Redes Neurais Convolucionais

(*Convolutional Neural Networks*)

- Redes Neurais Convolucionais é uma extensão de MLPs tradicionais a partir de 3 ideias:
  - Campos receptivos locais (*local receive fields*);
  - Pesos compartilhados (*shared weights*);
  - Sub-amostragem espaço-temporal (*Spatial / temporal sub-sampling*).

Artigo de LeCun paper (1998) sobre reconhecimento de texto:

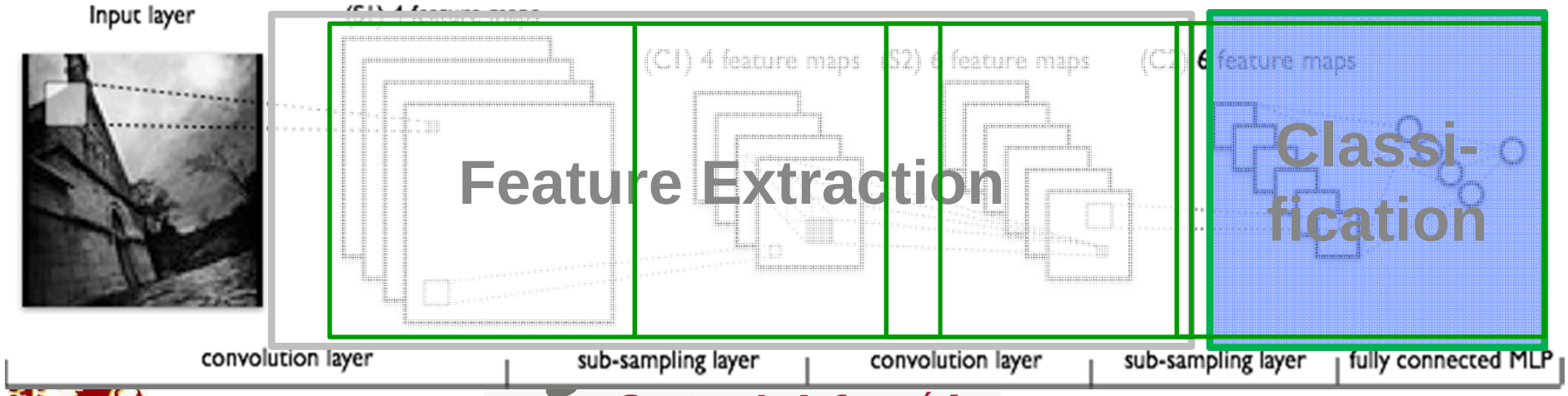
http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

Redes Neurais Convolucionais

(*Convolutional Neural Networks*)

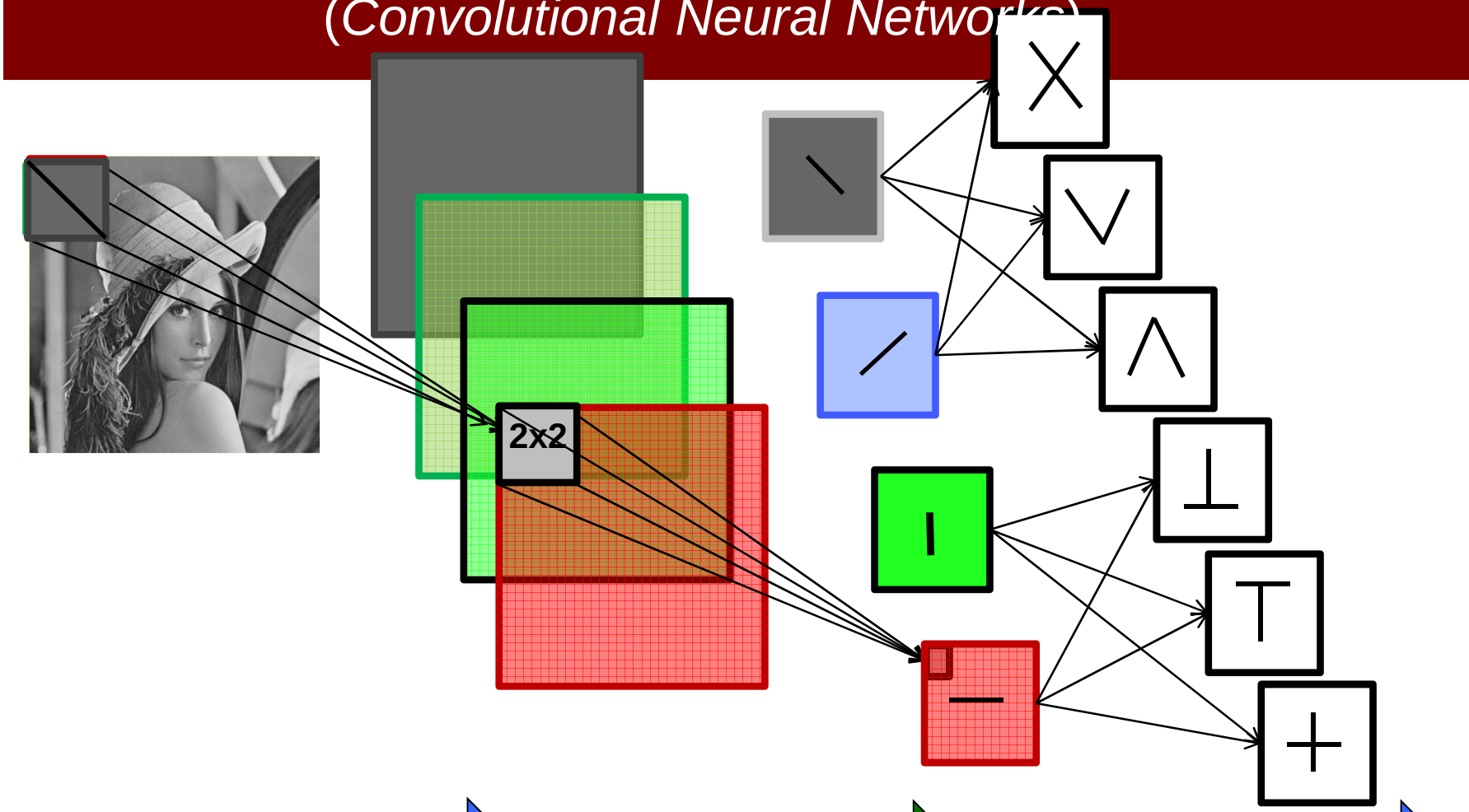
Arquitetura da CNN:

- Camada convolucional + Não-linear (ReLU);
- Camada de subamostragem;
- Camada convolucional + Não-linear (ReLU);
- Linearização da camada + camadas totalmente conectadas de treinamento supervisionado.



Redes Neurais Convolucionais

(Convolutional Neural Networks)



Redes Neurais Convolucionais
(*Convolutional Neural Networks*)

- Parâmetros:
  - Margens: (Ignorar/Replicar/Zerar);
  - Tamanho do *kernel*;
  - Tamanho do passo (*stride*);
  - Quantidade de núcleos;
  - Configuração dos núcleos (aprendidos).

Redes Neurais Convolucionais

(*Convolutional Neural Networks*)

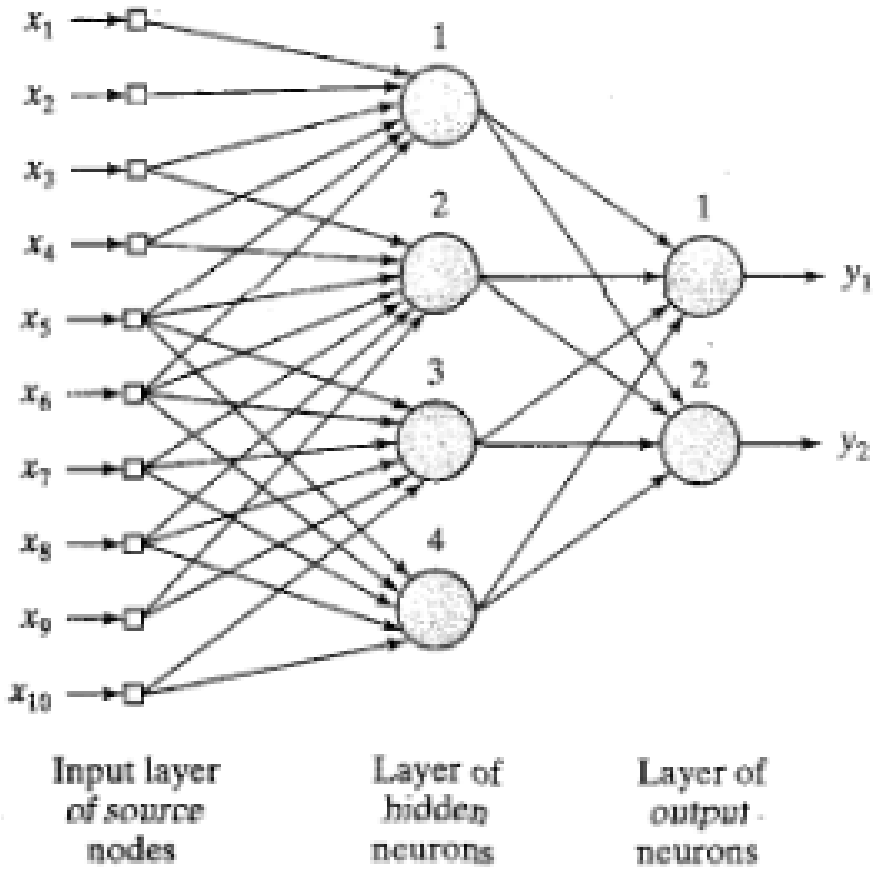
- Connectividade esparsa: CNN explora correlações espaciais focando em conectividade entre unidades de processamento próximas. Os campos receptivos são contíguos.
  - Os nodos são invisíveis as variações fora do seu campo receptivo.
- Analogia (quando a entrada são imagens):
  - Pixels -> Neurônios;
  - Kernel -> Sinapses;
  - Convolução -> Operação básica de um neurônio.

Redes Neurais Convolucionais
(*Convolutional Neural Networks*)

- Procedimento *ad-hoc* para considerar conhecimento prévio no projeto (*design*) de uma RNA:
  - Restringir a arquitetura da rede através do uso de conexões locais conhecidas como campos receptivos;
  - Limitar a escolha dos pesos sinápticos através do uso do compartilhamento de pesos.
- Essas duas técnicas, especialmente a segunda, têm um efeito colateral benéfico:
  - Número de parâmetros livres na rede é reduzido significativamente.

Redes Neurais Convolucionais
(*Convolutional Neural Networks*)

- Seja a rede *feedforward* parcialmente conectada cuja arquitetura é restringida por construção:
  - Por exemplo, os 6 primeiros nodos de entrada constituem o campo receptivo para o neurônio escondido 1.



Redes Neurais Convolucionais

(*Convolutional Neural Networks*)

- Para satisfazer a restrição do compartilhamento de pesos, deve-se usar o mesmo conjunto de pesos sinápticos (a janela de convolução) para cada um dos neurônios da camada escondida.
- Portanto, para seis conexões locais por nodo escondido e quatro nodos escondidos (figura anterior), pode-se expressar o campo local induzido do neurônio escondido *j* como (soma de convolução):

$$v_j = \sum_{i=1}^{6} w_i x_{i+j-1}, \qquad j = 1, 2, 3, 4$$

- Onde $\{w_i\}_{i=1}^{6}$, constitui o mesmo conjunto de pesos compartilhados por todos os quatro nodos escondidos e x_k é o sinal do nó fonte $k=i+j$.

Redes Neurais Convolucionais

(*Convolutional Neural Networks*)

- Uma rede CNN é um MLP projetado para reconhecer formas bidimensionais com um alto grau de invariância para translação, mudança de escala, e outras formas de distorção.
- Esta tarefa difícil é aprendida de maneira supervisionada por uma rede cuja estrutura inclui as seguintes formas de restrições:
  - Extração de características;
  - Mapeamento de características;
  - Sub-amostragem.

Redes Neurais Convolucionais

(*Convolutional Neural Networks*)

- Extração de características: Cada nodo recebe entradas de estímulos em um campo receptivo local, saídas da camada anterior, processando apenas características locais. A posição relativa de cada característica extraída em relação às outras é preservada.
- Mapeamento de características: Cada camada computacional da rede é composta de múltiplos mapas de características. Cada um forma um conjunto no qual os nodos individuais compartilham o mesmo conjunto de pesos sinápticos (janela para convolução).

Redes Neurais Convolucionais
(*Convolutional Neural Networks*)

- Os filtros são aprendidos pelo algoritmo.
- A inicialização dos filtros é aleatória.
- O pesos dos filtros são os mesmo em qualquer uma das regiões do mapa.
- A convolução é linear.
- Facilita o paralelismo do processo.

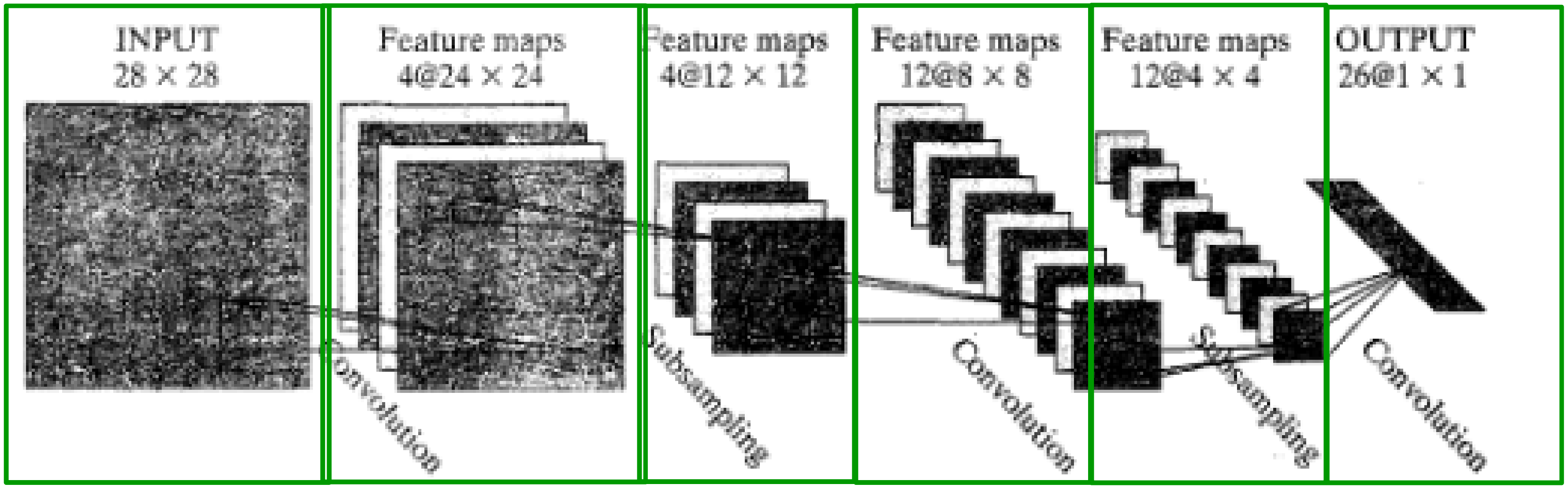
Redes Neurais Convolucionais

(*Convolutional Neural Networks*)

- Sub-amostragem: Cada camada da CNN é seguida por uma camada computacional que realiza cálculo da média local (ou determinação do valor mais alto) e sub-amostragem, onde a resolução do mapa de características é reduzida. Isto busca reduzir a sensibilidade da saída do mapa a deslocamentos e outras formas de distorção.
- Todos os pesos em todas as camadas de uma CNN são aprendidos através do treinamento.
- No entanto, a rede aprende a extrair suas próprias características automaticamente.

Redes Neurais Convolucionais

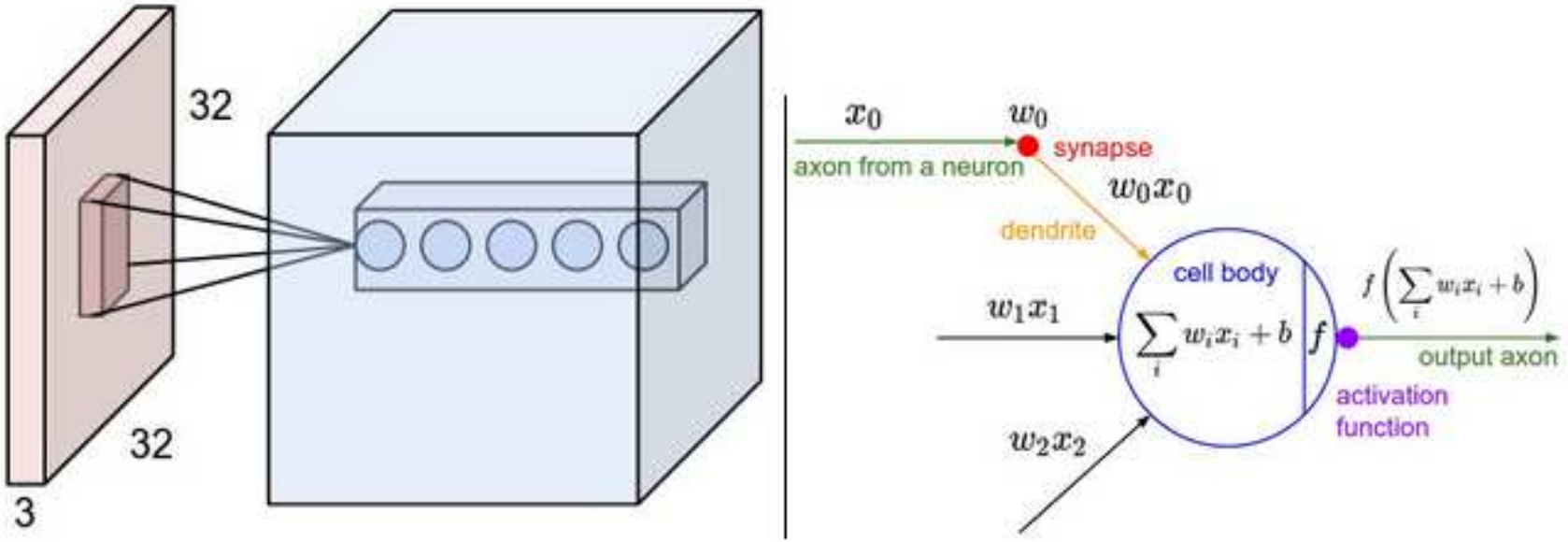
(Convolutional Neural Networks)



- [illegible]

Redes Neurais Convolucionais
(*Convolutional Neural Networks*)

- Camada convolutional:
  - Exemplo ilustrativo com uma imagem com 3 filtros (RGB):
    - Coletar a imagem e trabalhar com o hipervolume;

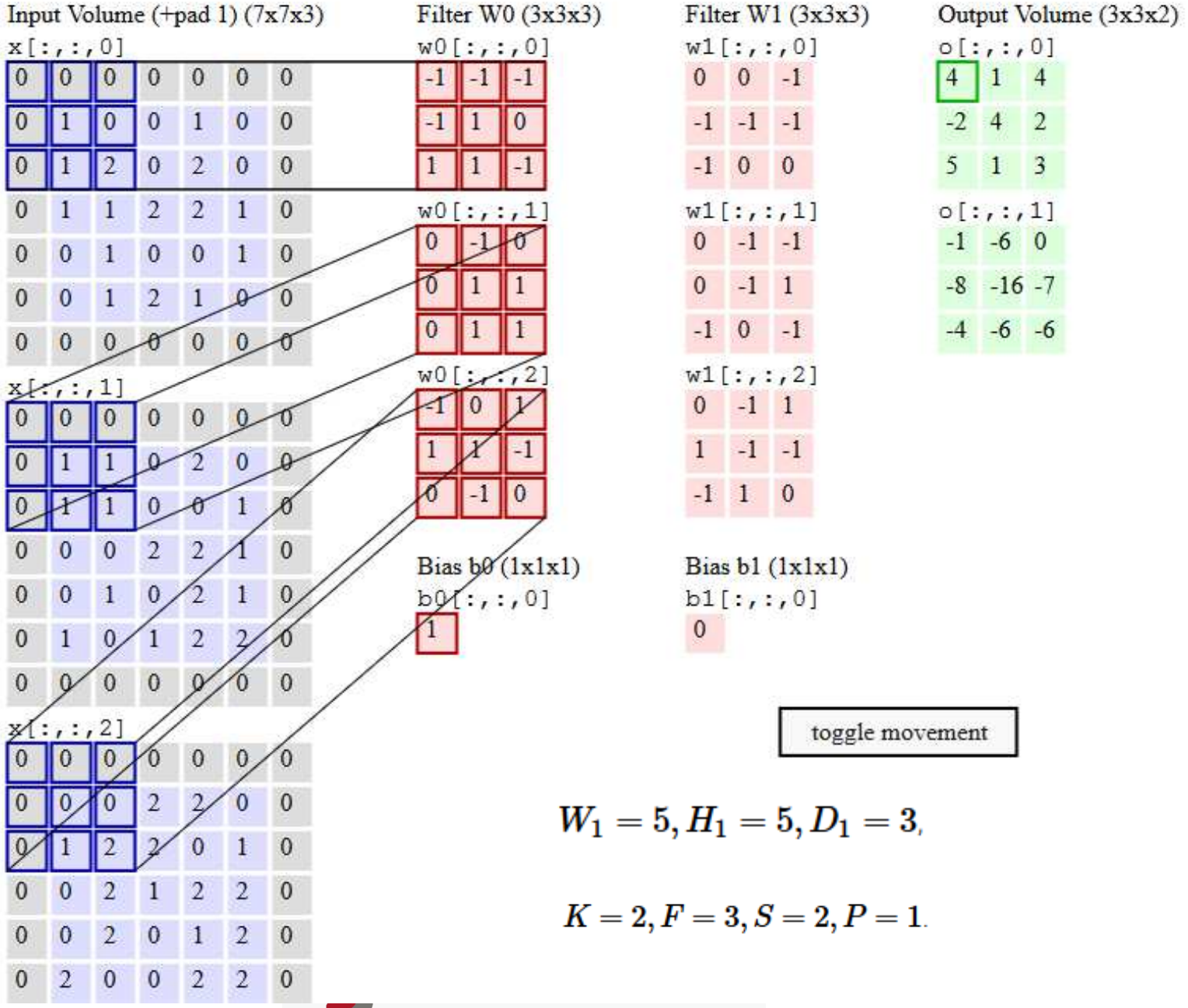


Redes Neurais Convolucionais
(*Convolutional Neural Networks*)

- Camada convolutional:

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.



$$W_1 = 5, H_1 = 5, D_1 = 3,$$

$$K = 2, F = 3, S = 2, P = 1.$$

Redes Neurais Convolucionais
(*Convolutional Neural Networks*)

- Camada de *pooling*:

Accepts a volume of size $W_1 \times H_1 \times D_1$

Requires two hyperparameters:

- their spatial extent F,
- the stride S,

Produces a volume of size $W_2 \times H_2 \times D_2$ where:

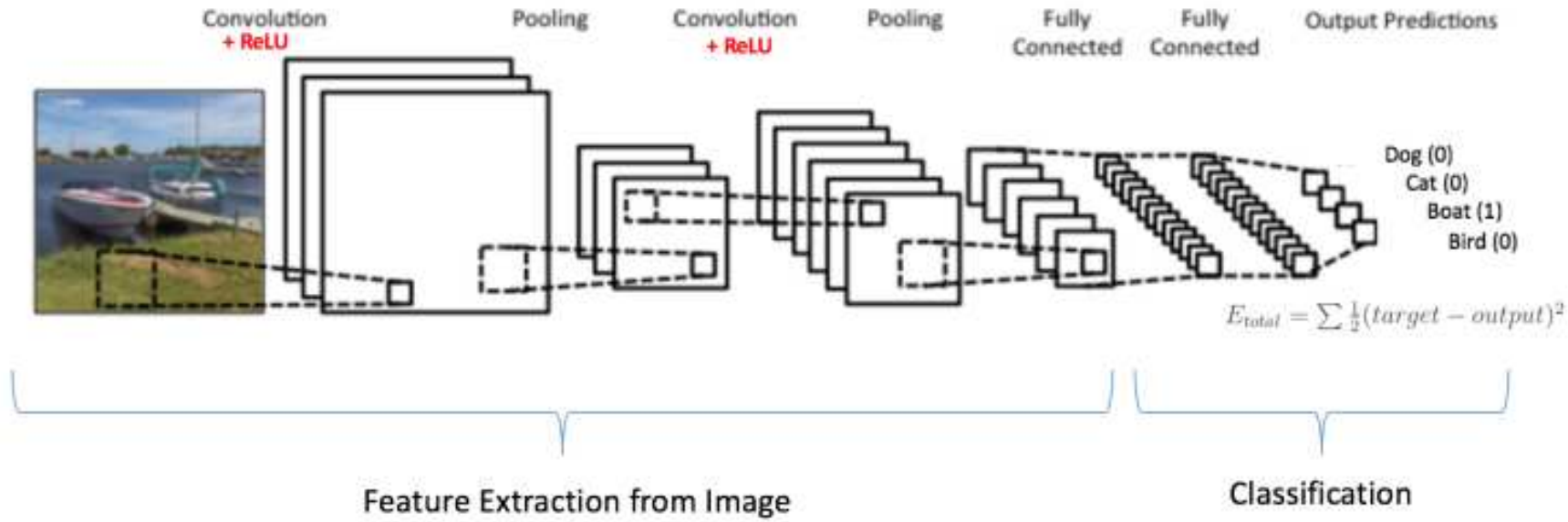
- $W_2 = (W_1 - F)/S + 1$
- $H_2 = (H_1 - F)/S + 1$
- $D_2 = D_1$

Introduces zero parameters since it computes a fixed function of the input

Note that it is not common to use zero-padding for Pooling layers

Redes Neurais Convolucionais
(*Convolutional Neural Networks*)

- |Exemplo: Camada convolucional de retropropagação
  - Input Image = Boat
  - Target Vector = [0, 0, 1, 0]



Redes Neurais Convolucionais

(*Convolutional Neural Networks*)

Passo 1: Inicializamos todos os filtros e parâmetros / pesos com valores aleatórios;

Passo 2: A rede recebe uma imagem de treinamento como entrada, passa pela etapa de propagação direta (convolução, ReLU e operações de agrupamento junto com a propagação direta na camada totalmente conectada) e localiza as probabilidades de saída para cada classe;

Vamos dizer que as probabilidades de saída para a imagem do barco acima são [0.2, 0.4, 0.1, 0.3]

Como os pesos são atribuídos aleatoriamente para o primeiro exemplo de treinamento, as probabilidades de saída também são aleatórias de início.

Passo 3: Calcular o erro total na camada de saída (soma das 4 classes)

Erro total = $\sum$ ½ (probabilidade de destino - probabilidade de saída) ^{2}

Redes Neurais Convolucionais

(*Convolutional Neural Networks*)

Passo 4: Use Backpropagation para calcular os gradientes do erro em relação a todos os pesos na rede e use o gradiente descendente para atualizar todos os valores / pesos de filtro e valores de parâmetros para minimizar o erro de saída. Os pesos são ajustados proporcionalmente à sua contribuição no erro total.

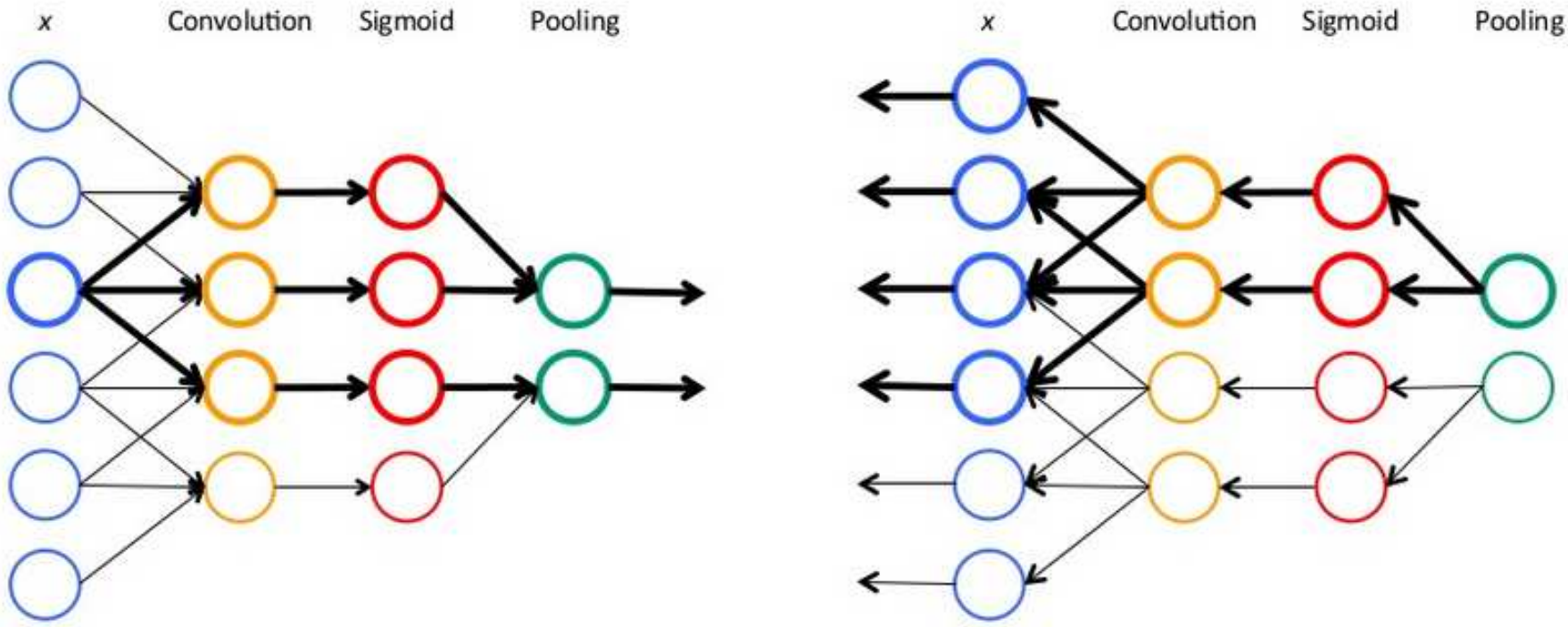
Isso significa que a rede aprendeu a classificar corretamente essa imagem específica ajustando seus pesos / filtros de forma que o erro de saída seja reduzido.

Outros parâmetros (número de filtros, tamanhos de filtros, arquitetura da rede) foram ajustados antes da Etapa 1 e não são alterados durante o treinamento - somente os valores dos pesos de conexão são atualizados.

Passo 5: Repita os passos 2-4 com todas as imagens no conjunto de treinamento.

Redes Neurais Convolucionais

(*Convolutional Neural Networks*)



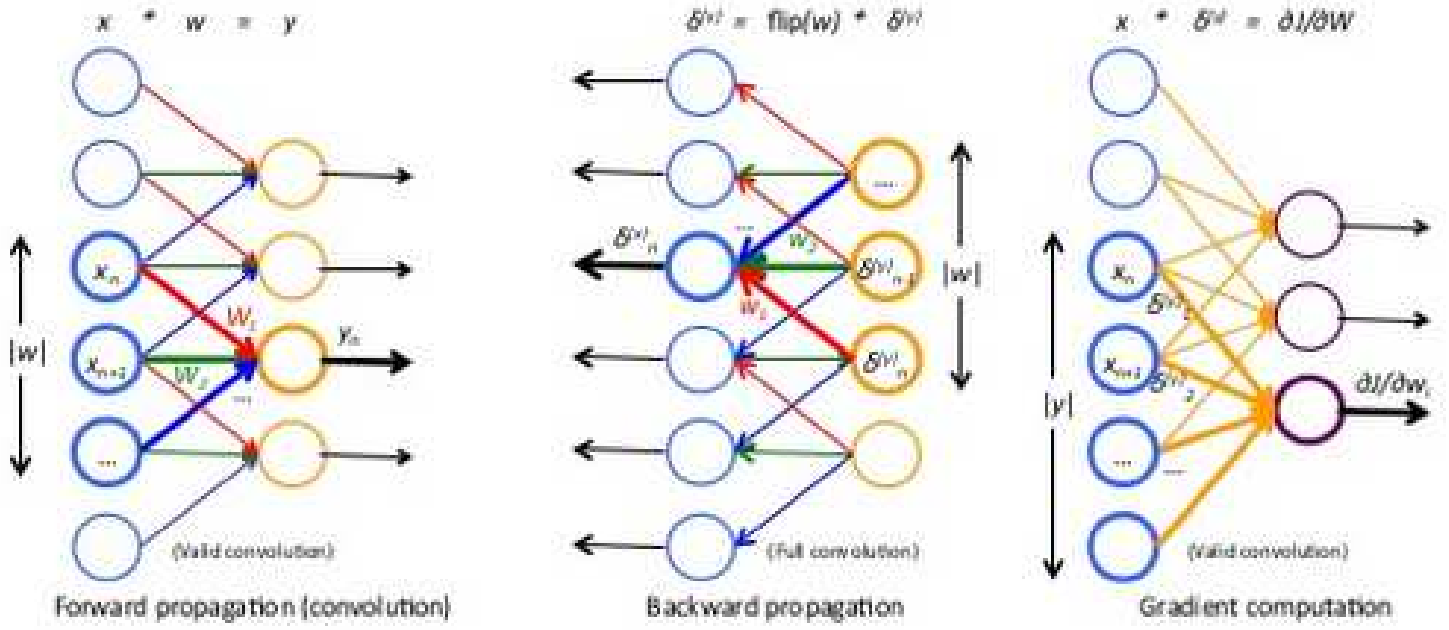
Redes Neurais Convolucionais

(*Convolutional Neural Networks*)

$$\delta_n^{(l)} = \frac{\partial J}{\partial x_n} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial x_n} = \sum_{i=1}^{|w|}\frac{\partial J}{\partial y_{n-i+1}}\frac{\partial y_{n-i+1}}{\partial x_n} = \sum_{i=1}^{|w|}\delta_{n-i+1}^{(l)} w_i = \left(\delta^{(l)} * \text{flip}(w)\right)[n],\ \delta^{(l)} = \left[\delta_n^{(l)}\right] = \delta^{(l)} * \text{flip}(w)$$

↑ Reverse order linear combination

$$\frac{\partial J}{\partial w_i} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial w_i} = \sum_{n=1}^{|x|-|w|+1}\frac{\partial J}{\partial y_n}\frac{\partial y_n}{\partial w_i} = \sum_{n=1}^{|x|-|w|+1}\delta_n^{(l)} x_{n+i-1} = \left(\delta^{(l)} * x\right)[i],\ \frac{\partial J}{\partial w} = \left[\frac{\partial J}{\partial w_i}\right] = \delta^{(l)} * x = x * \delta^{(l)}$$



Redes Neurais Convolucionais
(*Convolutional Neural Networks*)

- Os ajustes dos parâmetros livres são feitos usando uma forma estocásticas (sequencial) do aprendizado *back-propagation*.
- O uso do compartilhamento de pesos torna possível implementar a CNN de forma paralela: outra vantagem sobre a MLP totalmente conectada.

Demo

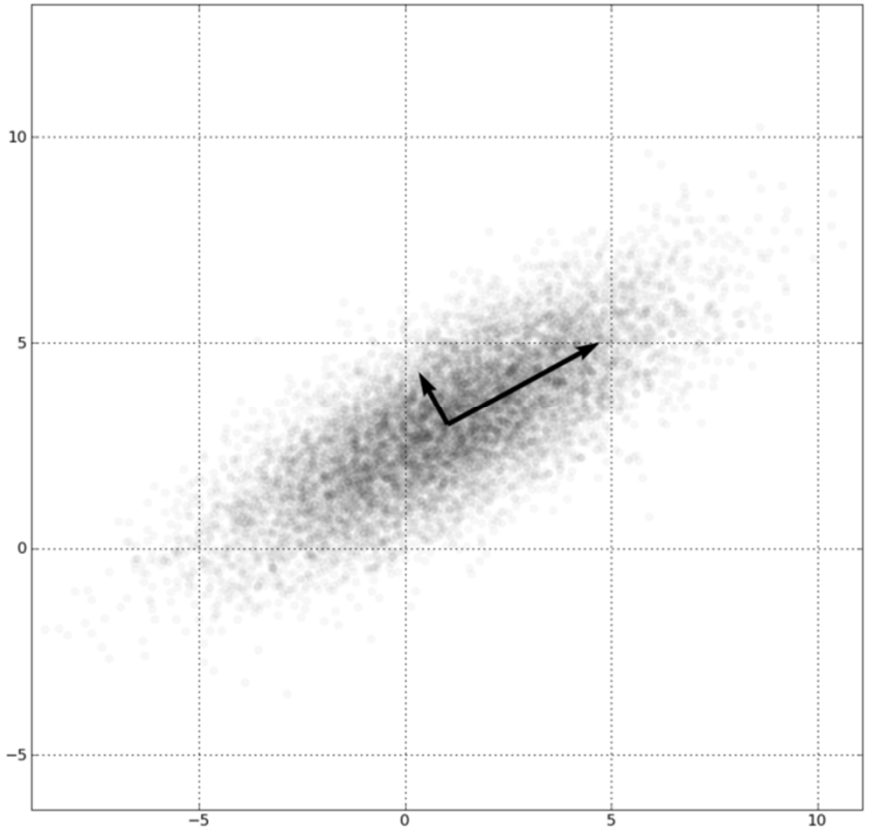
- http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html
- http://www.cs.toronto.edu/~kriz/cifar.html

Auto-encoder Profundo
(*Deep Autoencoder*) - PCA

- Análise de Componentes Principais PCA (*Principal component Analysis*)
  - Originalmente voltado para problemas de redução de dimensionalidade;
  - Objetivo: criar um novo conjunto de características que capture informação essencial do conjunto original de entrada, reduzindo sua dimensão;
  - As novas características são combinações lineares das características originais;
  - Tentam capturar ao máximo a variância original do conjunto de entrada.

Auto-encoder Profundo
(*Deep Autoencoder*) - PCA

- Análise de Componentes Principais:
  - Mudança de eixos.



Auto-encoder Profundo
(*Deep Autoencoder*) - PCA

- A partir de uma matriz de características X, com dimensões $m \times n$, os vetores que representam os componentes principais são os autovetores da matriz $X^T X$, com dimensões $n \times n$, ordenados pela magnitude decrescente dos autovetores.
- Após achar os vetores dos componentes principais, os valores das n colunas originais são reduzidos a k valores, $k \leq n$, através do cálculo das projeções do vetor a partir dos k principais componentes.
- A transformação linear ortogonal captura a variância do conjunto original, de forma que o primeiro componente possui a maior variância, o segundo componente a segunda maior variância, e assim sucessivamente.

Auto-encoder Profundo
(*Deep Autoencoder*) - PCA

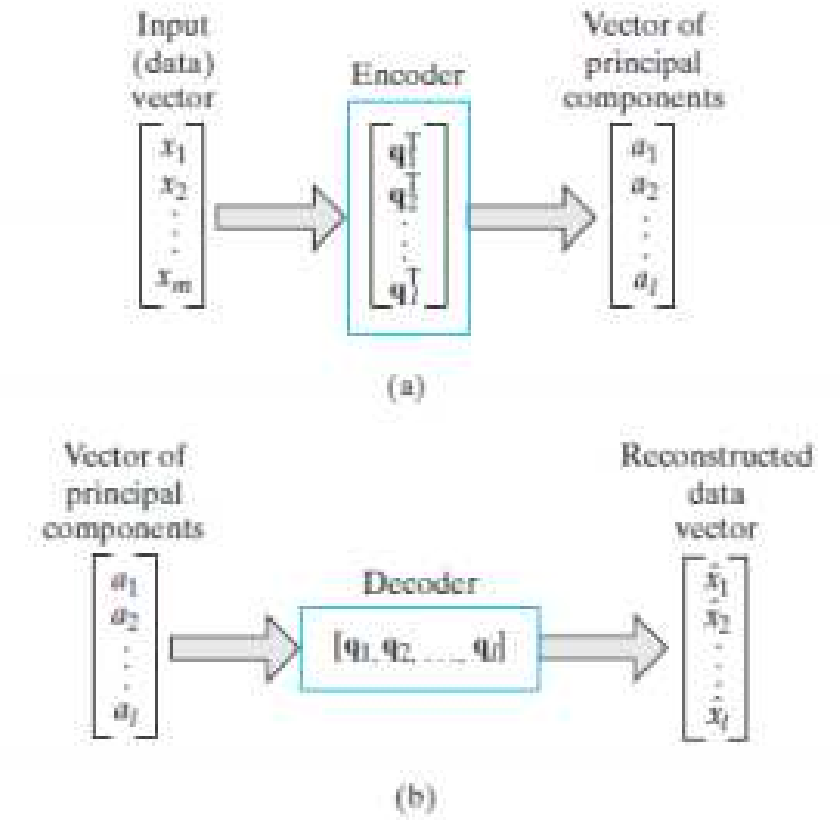
- A transformação PCA que preserva a dimensionalidade é dada por:

$$\begin{aligned}\mathbf{Y}^T &= \mathbf{X}^T\mathbf{W} \\ &= \mathbf{V}\Sigma^T\mathbf{W}^T\mathbf{W} \\ &= \mathbf{V}\Sigma^T\end{aligned}$$

- A transformação da expressão $Y^T = V\Sigma^T$ denota que cada linha de Y^T é uma rotação de linha de X^T, de forma que a primeira coluna de Y^T denota os pesos do primeiro componente principal, a segunda coluna os pesos do segundo componente principal, e assim sucessivamente.

Auto-encoder Profundo
(*Deep Autoencoder*) - PCA

- No processo de redução de dimensionalidade, a projeção linear de R^m para R^L é chamado de codificação (*encoding*).
- O processo inverso, a projeção de R^L para R^m é chamado de decodificação (*decoding*).



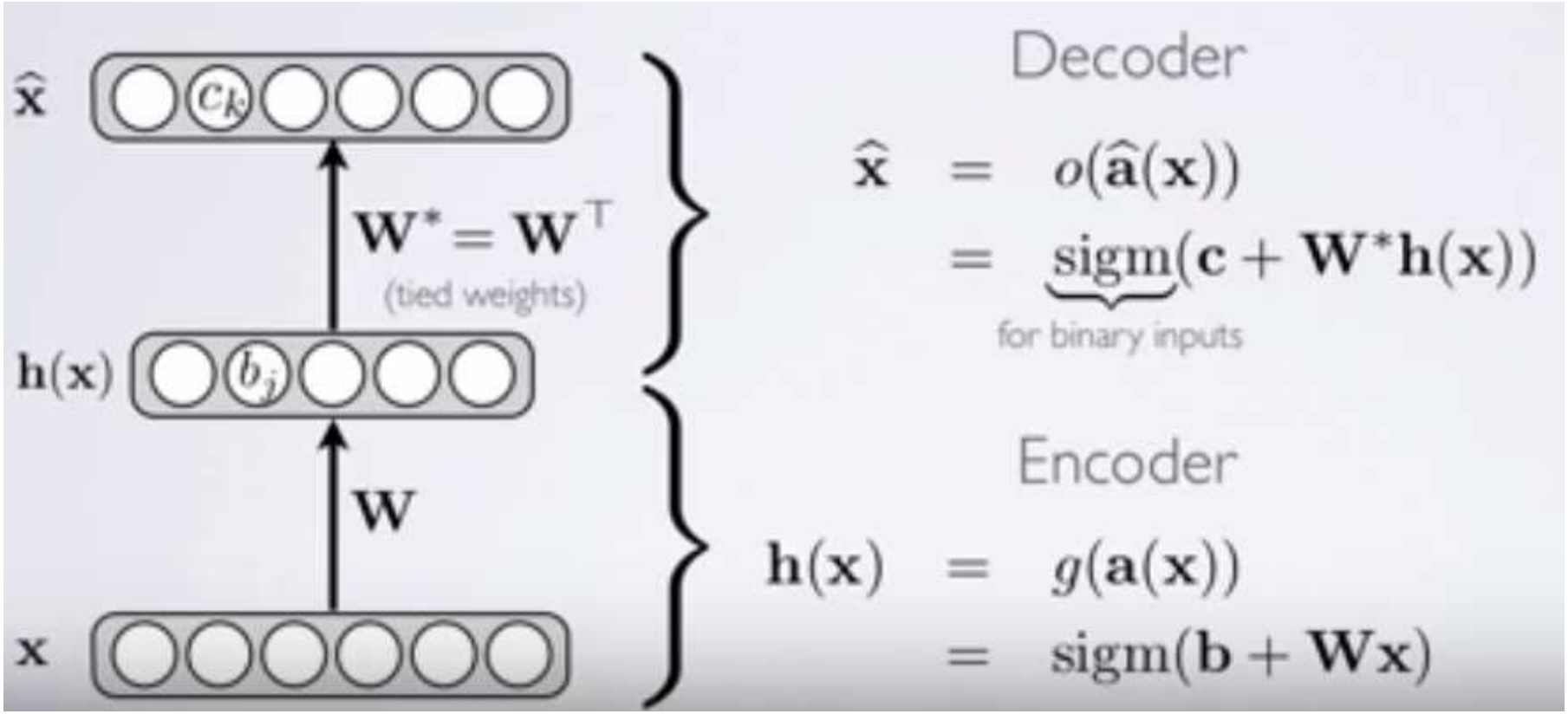
Auto-encoder Profundo
(*Deep Autoencoder*) - PCA

- Limitações:
  - Hipótese assumida: Direções das projeções com maior variância são as que melhor representam as características originais (válido apenas para colunas linearmente correlacionadas);
  - PCA é um método linear de redução de dimensionalidade: Só considera transformações ortogonais, logo, o método não trata de mapeamentos não-lineares.
  - Alterativa ao PCA: Extensões do método com transformações não lineares;
    - Autoencoders: Mapeamentos não lineares ou empilhamento (representação hierárquica).

Autoencoder Empilhado
(*Stacked Autoencoder*)

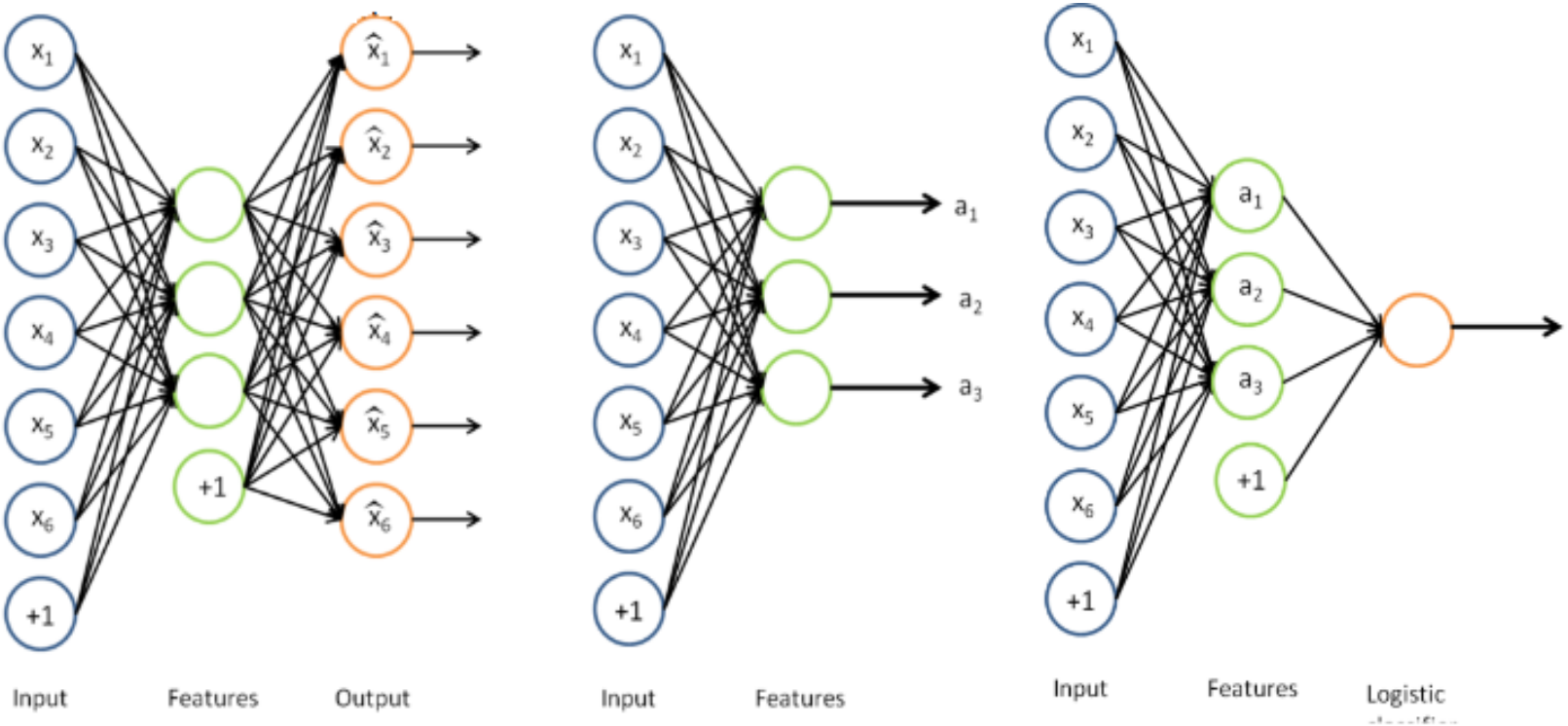
- Um tipo de modelo que tenta descobrir características genéricas dos dados:
  - Aprende a identificar funções através do aprendizado de subcaracterísticas;
  - Compressão: pode usar novas características como um novo conjunto de treinamento.
- Ao colocar uma camada escondida menor que a entrada, a rede é forçada a criar uma representação compacta do espaço de entrada.
- Função de minimização de erro: $L(\boldsymbol{x}, g(f(\boldsymbol{x})))$

Autoencoder Empilhado
(*Stacked Autoencoder*)



Autoencoder Empilhado
(*Stacked Autoencoder*)

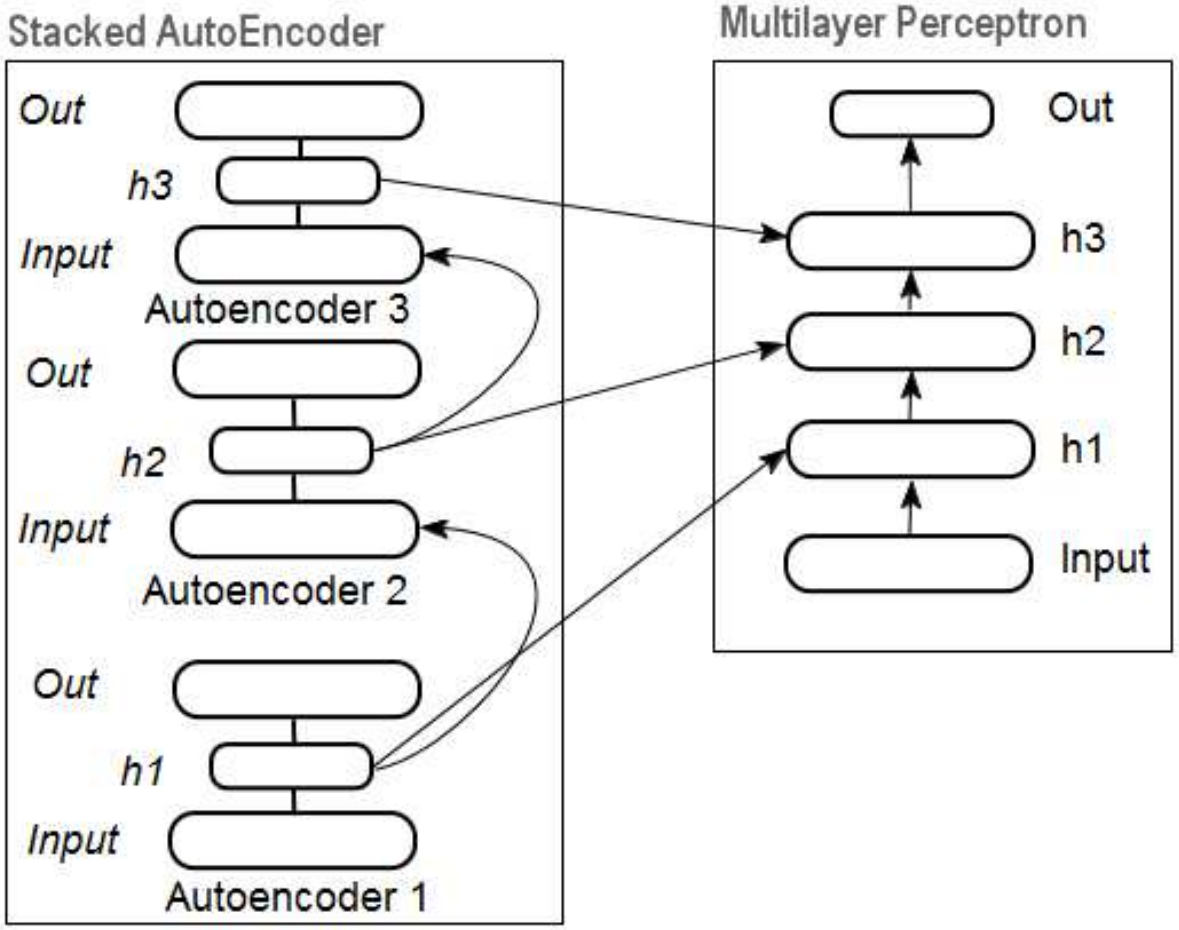
- Exemplo: autoencoder com 6 entradas e 3 nodos escondidos;
  - Exemplos de treinamento possuem 6 bits (cinco 0s e um 1);
  - Representa os 6 exemplos em 3 bits (binario), sem supervisão.



Autoencoder Empilhado
(*Stacked Autoencoder*)

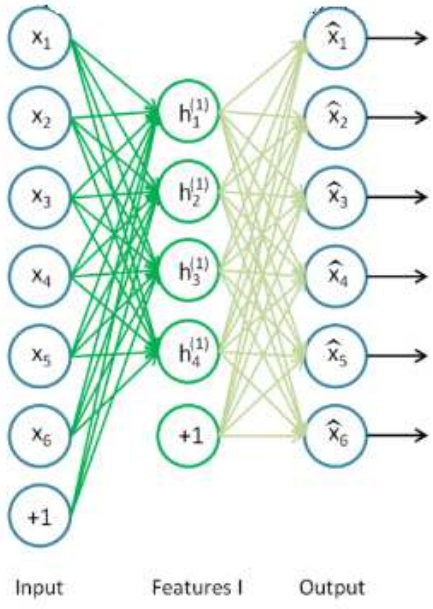
- O número de neurônios na camada oculta deve ser **menor** que a dimensão de entrada para evitar que a rede aprenda a solução trivial, ou seja, simplesmente copiar a entrada;
- Com menos neurônios para codificar a entrada, a rede é forcada a aprender uma representação compacta do espaço de entrada.

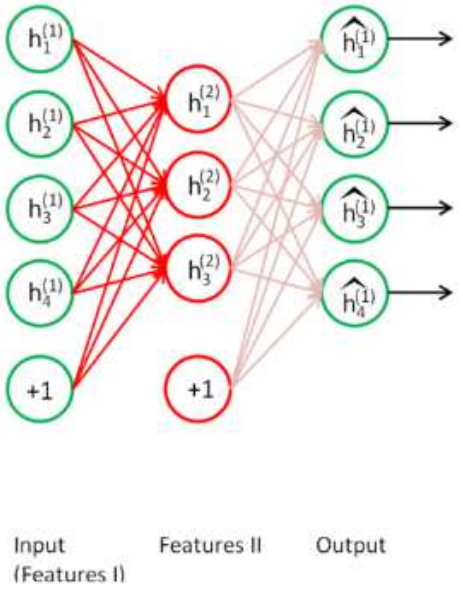
Autoencoder Empilhado
(*Stacked Autoencoder*)



Autoencoder Empilhado
(*Stacked Autoencoder*)

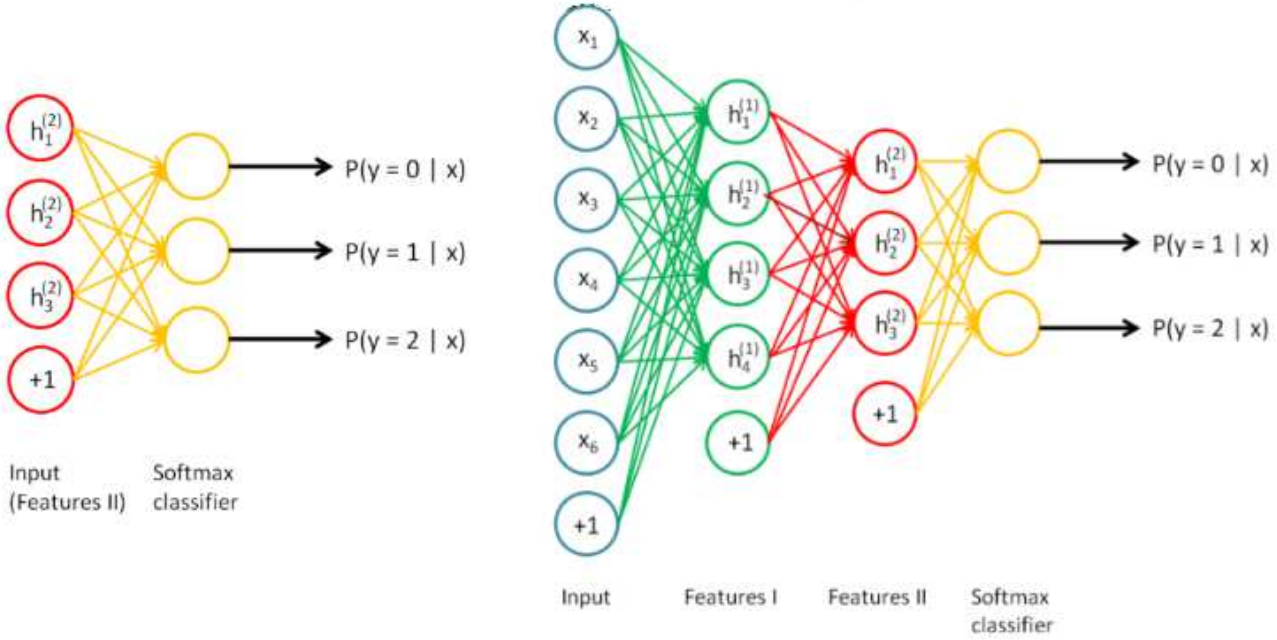
- (Bengio, 2007) Empilhar muitos auto-encoders esparsos e treiná-los de forma gulosa.
- O codigo gerado por um é repassado como entrada para o seguinte.





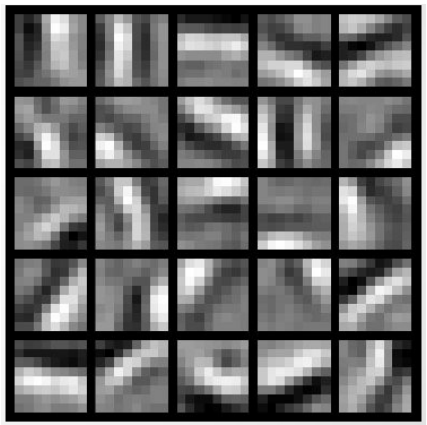
Autoencoder Empilhado
(*Stacked Autoencoder*)

- Realizar treinamento supervisionado na última camada utilizando características finais;
- Treinamento supervisionado da rede como um todo para realizar ajustes finos dos pesos



Autoencoder Empilhado
(*Stacked Autoencoder*)

- Cada camada abstrai um pouco mais a informação da camada anterior, criando representações de alto nível.
- Isso facilita o trabalho de camadas superiores, pois elas passam a trabalhar sobre conceitos de mais alto nível:
  - Ex.: pixels → linhas → formas → objetos.



Autoencoder Empilhado

(*Stacked Autoencoder*) - Treinamento *Greedy Layer-Wise*

1. Treinar primeira camada utilizando dados sem rótulos. Já que não há alvos nesse nível, rótulos não importam;

2. Fixar os parâmetros da primeira camada e começar a treinar a segunda usando a saída da primeira camada como entrada não-supervisionada da segunda camada;

3. Repetir 1 e 2 de acordo com o número de camadas desejada (construindo um mapeamento robusto);

4. Usar a saída da camada final como entrada para uma camada/modelo supervisionada e treinar de modo supervisionado (deixando pesos anteriores fixos);

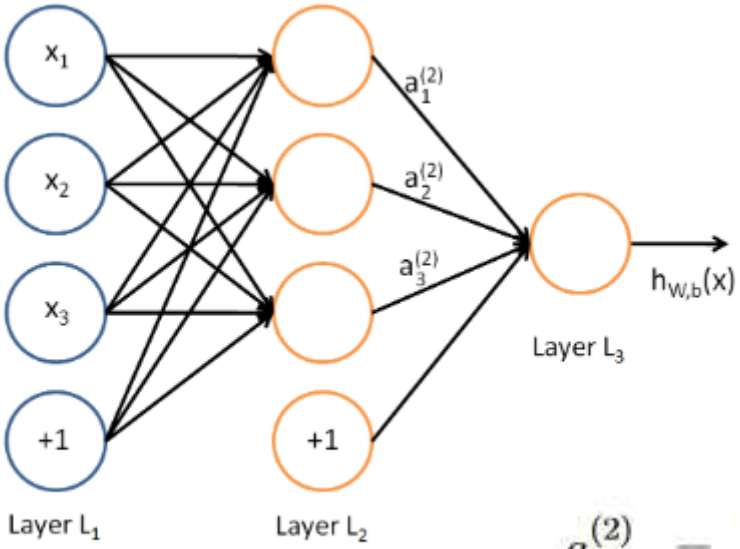
5. Liberar todos os pesos e realizar ajuste fino da rede como um todo utilizando uma abordagem supervisionada

Autoencoder Empilhado
(*Stacked Autoencoder*) - Treinamento *Greedy Layer-Wise*

- Evita muitos dos problemas associados ao treinamento de uma rede profunda de modo totalmente supervisionado.
  - Cada camada foca apenas no processo de aprendizagem;
  - Pode tirar vantagem de dados não rotulados;
  - Quando a rede completa é treinada de modo supervisionado, os pesos já estão minimamente ajustados (evita mínimos locais);

Autoencoder Empilhado

(*Stacked Autoencoder*) - Treinamento *Greedy Layer-Wise*



$$
\begin{aligned}
a_1^{(2)} &= f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)}) \\
a_2^{(2)} &= f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)}) \\
a_3^{(2)} &= f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)}) \\
h_{W,b}(x) &= a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)})
\end{aligned}
$$

Autoencoder Empilhado
(*Stacked Autoencoder*) - Treinamento *Greedy Layer-Wise*

- Função de custo de erro supervisionado:

$$J(W, b; x, y) = \frac{1}{2} \left\| h_{W,b}(x) - y \right\|^2 .$$

- Função de custo de erro não-supervisionado:

$$p(x) = \text{sigm}(c + W\text{sigm}(b + W'x))$$

$$R = -\sum_i x_i \log p_i(x) + (1 - x_i) \log(1 - p_i(x))$$

Autoencoder Empilhado
(*Stacked Autoencoder*) - Treinamento *Greedy Layer-Wise*

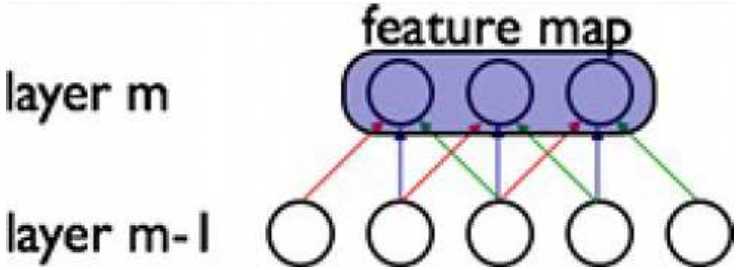
- Se o *Stacked autoencoder* for treinado com retropropagação há grande chance de ocorrer o problema dos gradientes diluídos.
- O treinamento pode se tornar demorado ou mesmo inviável.
- A solução está em treinar cada camada como um *autoencoder* isolado, o que reduz o caminho do gradiente para uma única camada.

Autoencoder Espaçado
(*Spaced Autoencoder*)

- Outra possibilidade de *Autoencoders* está em restringir as ativações das camadas escondidas;
  - Modifica-se a função de custo da rede para incluir um termo que penaliza muitas ativações nos neurônios.
- Como resultado, uma codificação esparsa e aprendida (poucos neurônios ativados por vez), mesmo que uma dada camada escondida seja maior que a dimensionalidade de entrada.
- Ω(h) – penalização por esparsidade.
- Função de minimização de erro: $L(\boldsymbol{x}, g(f(\boldsymbol{x}))) + \Omega(\boldsymbol{h})$

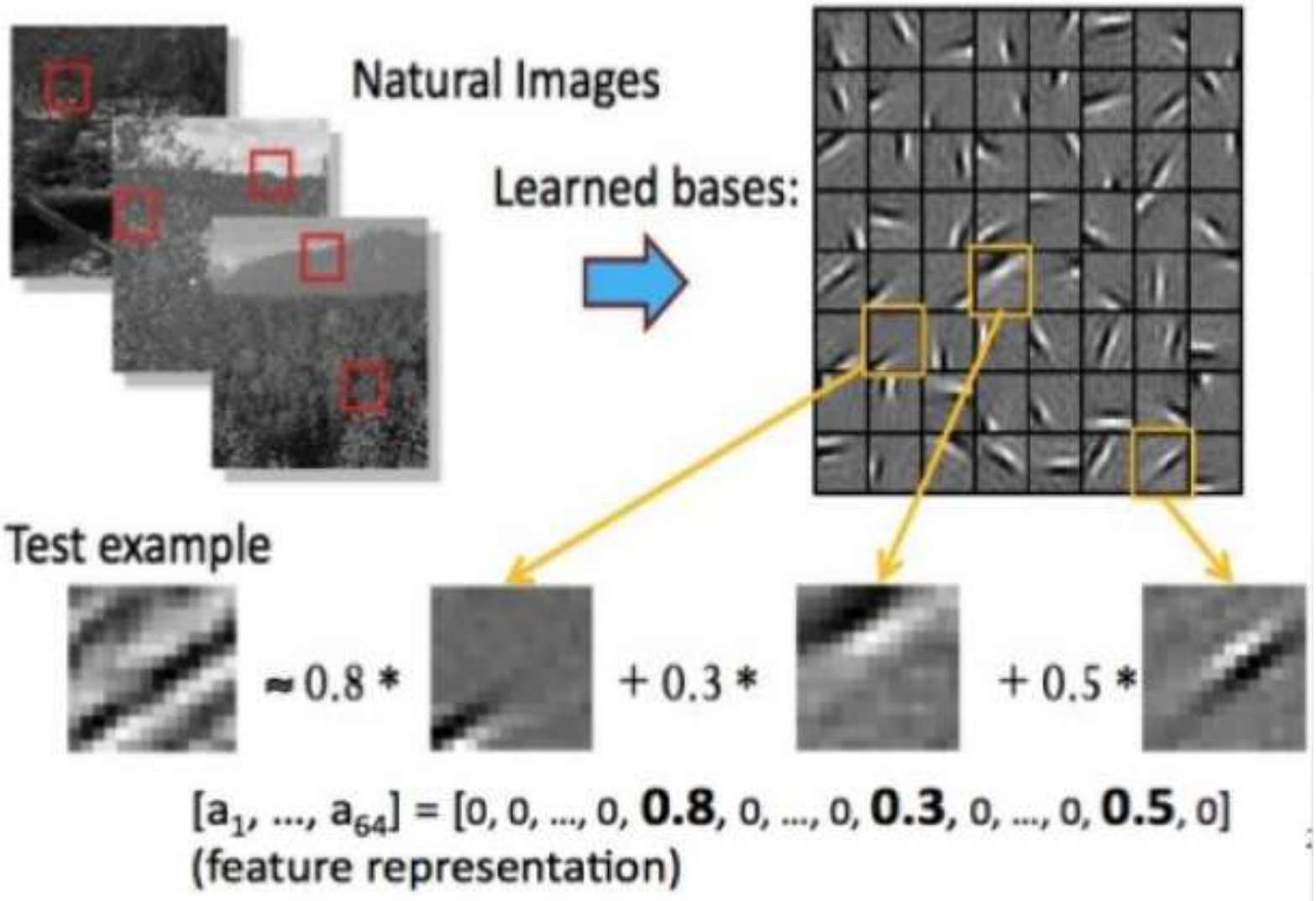
Autoencoder Convolucional
(*Convolutional Autoencoder*)

- Emprega compartilhamento de pesos para reduzir o numero de parâmetros no *autoencoder* e ao mesmo tempo explorar conhecimento prévio dos problemas:
  - Por exemplo, pode-se trabalhar com blocos de tamanho 32x32 utilizando pequenas regiões 3x3 onde todas utilizam os mesmos pesos, efetivamente reduzindo a dimensão de entrada de 1024 para 9.



Autoencoder Convolucional

(*Convolutional Autoencoder*)



Autoencoder Denoising Empilhado
(*Stacked Denoising Autoencoder*)

- O *Denoising Autoencoder* (DAE) é uma versão estocástica do *Autoencoder* que recebe valores corrompidos em sua entrada e é deve gerar uma representação latente dos dados não corrompidos:
  - Este modelo lida com o risco de função de identidade corrompendo aleatoriamente a entrada pela adição de ruído para que o *Autoencoder* reconstrua o sinal ou elimine seu ruído.
- Em Vincent et. al., 2010 é proposta a arquitetura do *Stacked Denoising Autoencoder* (SDA).
- *Denoising Autoencoders* podem ser empilhados através da alimentação da representação latente do DAE da camada anterior, servindo de entrada para a camada atual.
- Função de minimização de erro: $L(x, g(f(\tilde{x})))$

Ferramentas

Software	Creator	Platform	Written in	Interface
Apache MXNet	Apache Software Foundation	Linux, macOS, Windows,AWS, Android,iOS, JavaScript	Small C++core library	C++, Python, Julia, Matlab, JavaScript, Go, R, Scala, Perl
Apache SINGA	Apache Incubator	Linux, macOS, Windows	C++	Python, C++, Java
Caffe	Berkeley Vision and Learning Center	Linux, macOS, Windows	C++	Python, MATLAB, C++
Deeplearning4j	Skymind engineering team; Deeplearning4j community; originally Adam Gibson	Linux, macOS, Windows, Android (Cross-platform)	C++, Java	Java, Scala, Clojure, Python(Keras), Kotlin
Intel Data Analytics Acceleration Library	Intel	Linux, macOS, Windows on Intel CPU	C++, Python, Java	C++, Python, Java[10]
Keras	François Chollet	Linux, macOS, Windows	Python	Python, R
PyTorch	Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan	Linux, macOS	Python, C, CUDA	Python
TensorFlow	Google Brainteam	Linux, macOS, Windows,Android	C++, Python, CUDA	Python (Keras), C/C++, Java, Go, R, Julia
Theano	Université de Montréal	Cross-platform	Python	Python (Keras)
Torch	Ronan Collobert, Koray Kavukcuoglu, Clement Farabet	Linux, macOS, Windows,Android, iOS	C, Lua	Lua, LuaJIT,C, utility library for C++/OpenCL

Referências

- Deep Learning Tutorial. LISA Lab, University of Montreal.
- J. Schmidhuber. Deep Learning in Neural Networks: An Overview. *Neural Networks*, Volume 61, January 2015, Pages 85-117 (DOI: 10.1016/j.neunet.2014.09.003), published online in 2014.
- Bengio, Yoshua, et al. "Greedy layer-wise training of deep networks." *Advances in neural information processing systems* 19 (2007): 153.
- Vincent, Pascal, et al. "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion." *The Journal of Machine Learning Research* 11 (2010): 3371-3408.
- http://www.robots.ox.ac.uk/~vgg/practicals/cnn/