Aprendizado de Máquina

Do Perceptron a CNN

Luiz Eduardo S. Oliveira

Universidade Federal do Paraná
Departamento de Informática
web.inf.ufpr.br/luizoliveira

Introdução

- Um dos maiores desafios em aprendizagem de máquina e reconhecimento de padrões é a definição de um conjunto de características com alto poder de discriminação.
- Isso não é uma tarefa trivial.
- Alguns argumentam que o ponto fraco nos métodos de aprendizagem de máquina está exatamente na definição de características.

Introdução

- Um dos maiores desafios em aprendizagem de máquina e reconhecimento de padrões é a definição de um conjunto de características com alto poder de discriminação.
- Isso não é uma tarefa trivial.
- Alguns argumentam que o ponto fraco nos métodos de aprendizagem de máquina está exatamente na definição de características.

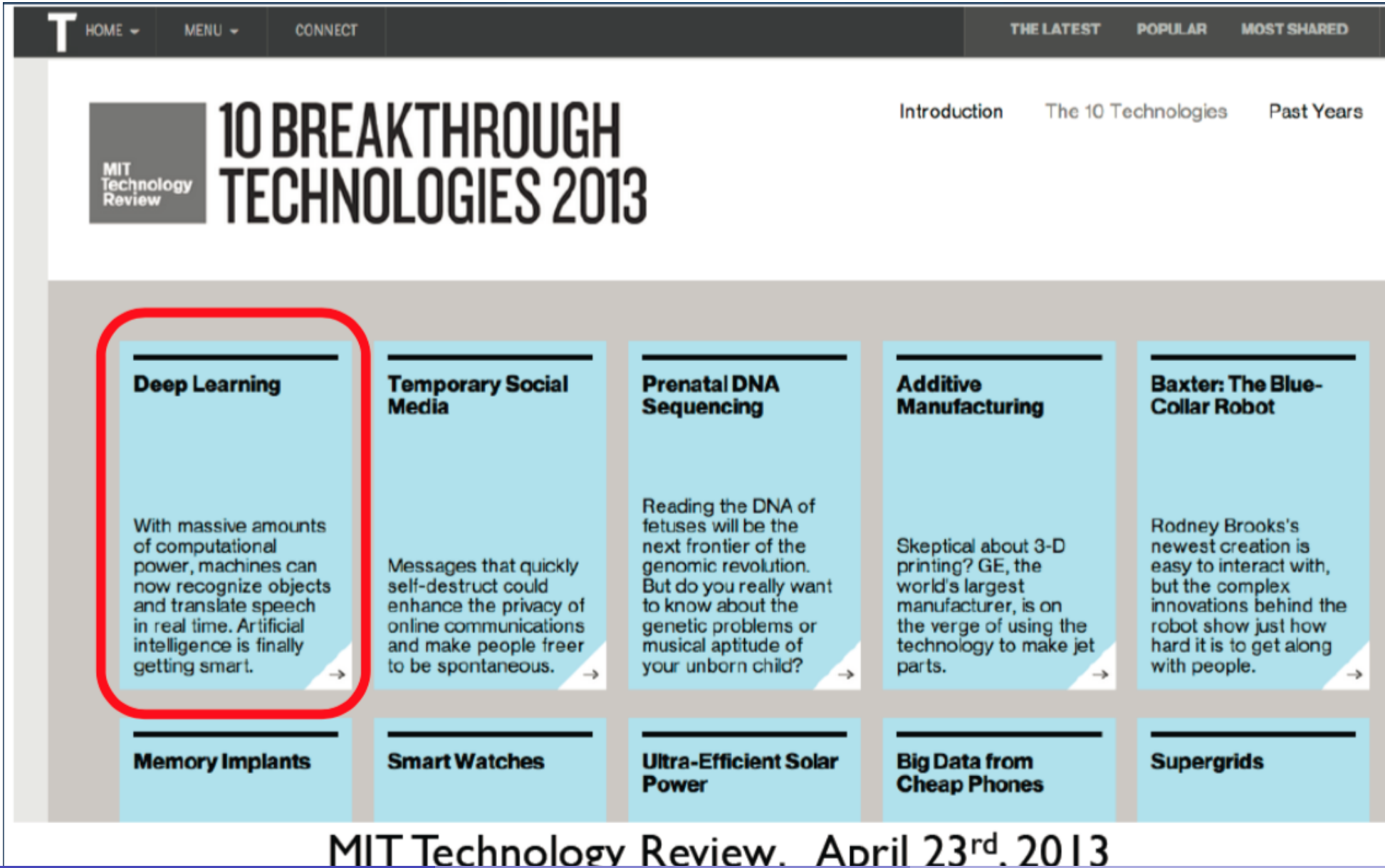
Alternativa

Aprender as características a partir dos dados!

Introdução

- A inspiração vem da arquitetura do cérebro humano.
- Por décadas, os pesquisadores tentaram treinar arquiteturas profundas de redes neurais (com diversas camadas)
- Um experimento somente teve sucesso (CNN, Yann LeCun em 1998)
- Alto custo computacional e muitos dados são necessários para a aprendizagem dos modelos
- Anos 90 e 2000 foram dominados pelo SVM
- Os métodos começaram a se popularizar por volta de 2010 (paralelismo, GPU)

Introdução



MIT Technology Review, April 23rd, 2013

Um breve histórico

- 1949, Donald Hebb
  - Definiu que a informação da rede fica armazenada nos pesos.
  - Propõe uma lei de aprendizagem especifica para as sinapses dos neurônios
    - ★ Quando ocorre o aprendizado os pesos são atualizados
- 1953, McCulloch e Pitts
  - Estudaram o comportamento do neurônio biológico com o objetivo de criar um modelo matemático para este.
  - Sugeriram a construção de uma máquina baseada ou inspirada no cérebro humano.

Um breve histórico

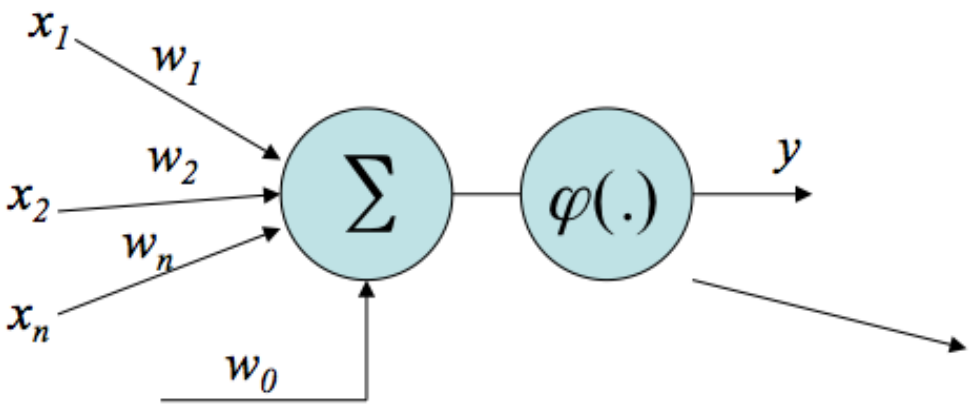
- 1957/1958, Frank Rosenblatt
  - Estudos aprofundados
  - Pai da neuro-computação
  - Perceptron
  - Criados do primeiro neuro-computador a obter sucesso
    - ★ Mark I

Um breve histórico

- 1958-1967
  - Várias pesquisas mal sucedidas.
- 1967-1982
  - Pesquisas silenciosas.
- 1986
  - Livro "Parallel Distributed Processing".
  - Algoritmo eficaz de aprendizagem
- 1987
  - Primeira conferência IEEE IJCNN (International Joint Conference on Neural Networks)

Perceptron

- Primeira e mais primitiva estrutura de rede neural



$$y = \varphi\left(\sum w_i \times x_i + w_0\right)$$

A função de ativação normalmente utilizada no perceptron é a hardlim (threshold)

$$f(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$$

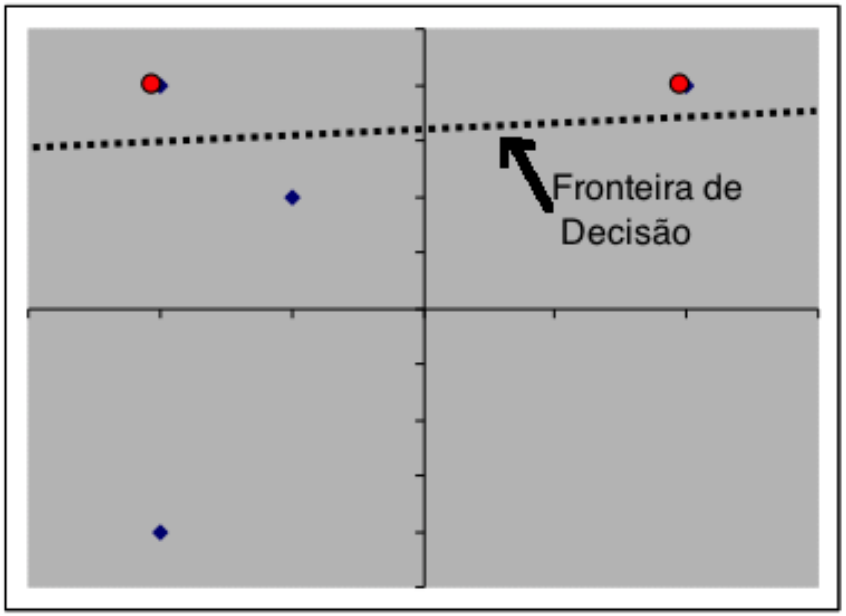
Algoritmo de Aprendizagem

- Iniciar os pesos e bias com valores pequenos, geralmente no intervalo [0.3, 08]
- Enquanto critério de parada não for alcançado
  - Aplicar um padrão de entrada com seu respectivo valor desejado de saída (t_j) e verificar a saída y da rede.
  - Calcular o erro da saída ($e = t_j - a$)
  - Se $e \neq 0$
    - ★ Atualizar pesos ($w_i = w_i^{old} + e \times x_i$)
    - ★ Atualizar bias ($b = b^{old} + e$)

Perceptron

- Considere o seguinte exemplo

Características (X)		Label (t)
2	2	0
-2	-2	1
-2	2	0
-1	1	1



Perceptron

Nesse exemplo, vamos inicializar os pesos e bias com 0, ou seja, $w = (0, 0)$ e $b = 0$

Apresentando o primeiro padrão (x_1) a rede

- $y = \text{hardlim}([0, 0][2, 2]^t + 0) = \text{hardlim}(0) = 1$
- Erro: $e = t_j - y = 0 - 1 = -1$
- Como o erro $\neq 0$, atualizam-se os pesos e bias
  - $W = W^{old} + e \times x_i = [0, 0] + (-1[2, 2]) = [-2, -2]$
  - $b = b^{old} + e = 0 + (-1) = -1$

Perceptron

Apresentando o segundo padrão (x_2) a rede

- $y = \text{hardlim}([-2, -2][-2, -2]^t + (-1)) = \text{hardlim}(7) = 1$
- Erro: $e = t_j - y = 1 - 1 = 0$
- Como o erro $= 0$, pesos e bias não precisam ser atualizados

Apresentando o terceiro padrão (x_3) a rede

- $y = \text{hardlim}([-2, -2][-2, 2]^t + (-1)) = \text{hardlim}(-1) = 0$
- Erro: $e = t_j - y = 0 - 0 = 0$
- Como o erro $= 0$, pesos e bias não precisam ser atualizados

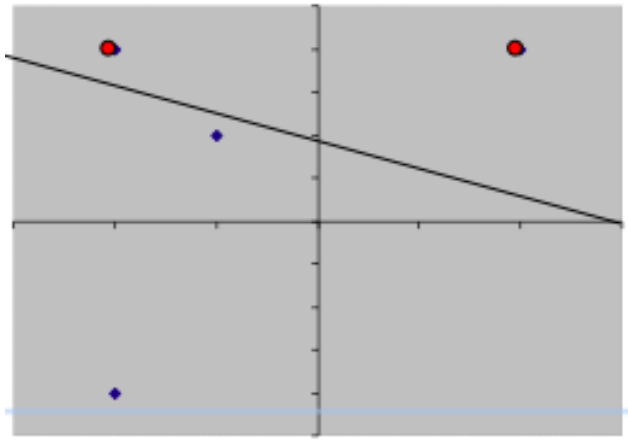
Perceptron

Apresentando o quarto padrão (x_4) a rede

- $y = \text{hardlim}([-2,-2][-1,1]^t + (-1)) = \text{hardlim}(-1) = 0$
- Erro: $e = t_j - y = 1 - 0 = 1$
- Como o erro $\neq 0$, atualizam-se os pesos e bias
  - $W = W^{old} + e \times x_i = [-2,-2] + (1[-1,1]) = [-3,-1]$
  - $b = b^{old} + e = -1 + 1 = 0$

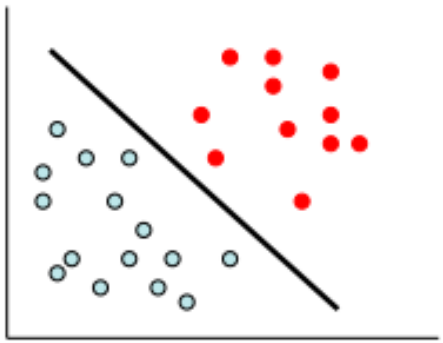
Perceptron

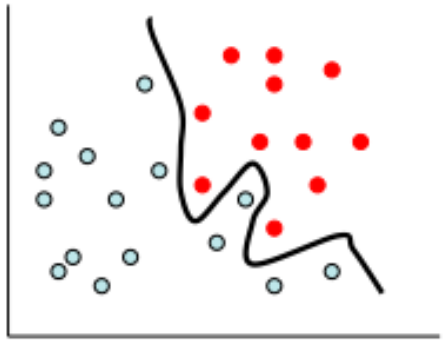
- Como todos os padrões foram apresentados a rede, o processo começa novamente, utilizando os pesos encontrados no último passo da primeira época (iteração).
- O algoritmos acaba quando o erro for zero (todos os padrões forem classificados corretamente) ou quando um certo número de épocas for alcançado.
- Para esse exemplo, os pesos finais são $w = [-1, -3]$ e $b = 2$.



Perceptron

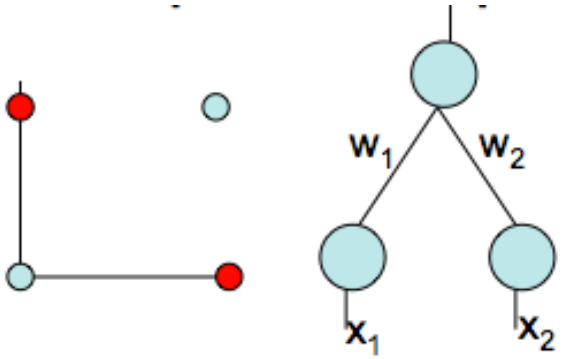
- Resolve problemas linearmente separáveis
- Entretanto, nem sempre os problemas são linearmente separáveis.





Problemas não Linearmente Separáveis

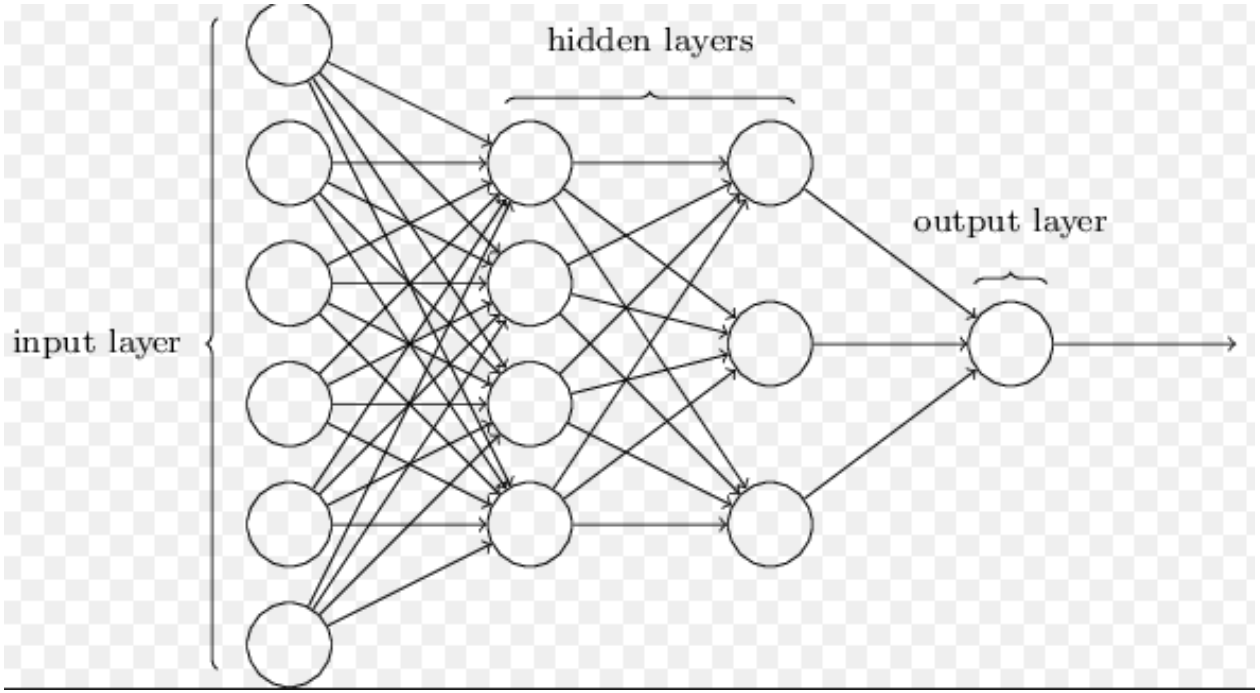
- Uma solução para resolver um problema não linearmente separável consiste em adicionar novas características ao vetor.
- Considere o problema XOR, o qual não pode ser resolvido com um perceptron.
- Nesse caso, a característica adicionada (x_3) é a operação AND entre x_1 e x_2
- O fato de adicionarmos essa característica faz com que o problema torne-se linearmente separável



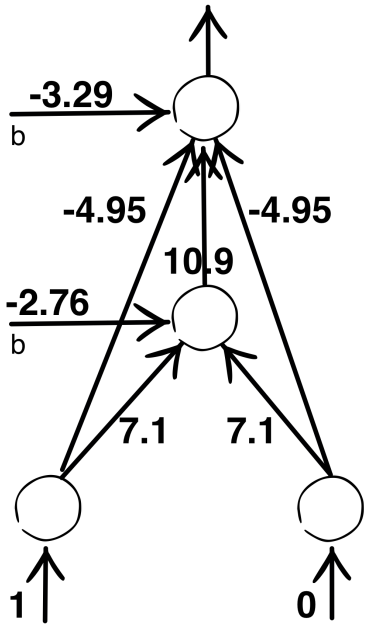
X1	X2	X3	Output
1	1	1	0
1	0	0	1
0	1	0	1
0	0	0	0

MLP - Multi-Layer Perceptron

- Outra maneira de resolver esse problema consiste em adicionar uma camada extra (camada escondida - hidden layer) entre as camadas de entrada e saída.



MLP - Multi-Layer Perceptron



- Considere o problema XOR e a rede abaixo já treinada.
- A saída é calculada de maneira similar ao perceptron
- A função de ativação comumente utilizada é a sigmoid $v = \frac{1}{1+e^{-s}}$

- Para classificar o padrão de entrada [1,0], primeiro devemos calcular o valor do neurônio da camada escondida
  - $s = 1 \times 7.1 + 1 \times -2.76 + 0 \times 7.1 = 4.34$
  - $v = 0.98$ (após passar pela sigmoid (função de ativação))
- A saída então utiliza o valor (característica) da camada escondida
  - $s = 1 \times -4.95 + 0 \times -4.95 + 0.98 \times 10.9 + 1 \times -3.29 = 2.52$
  - Após passar pela sigmoid, a saída será 0.91 (próximo de 1)

MLP - Multi-Layer Perceptron

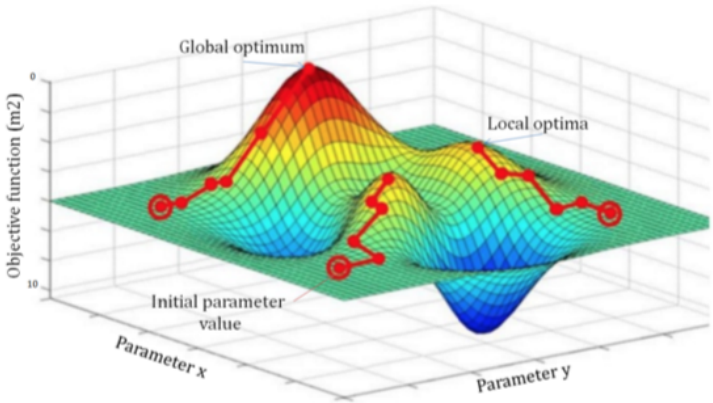
Problema de Atribuição de Créditos

- Quando temos uma camada escondida, surge o problema da atribuição de créditos ao neurônio desta camada.
  - Não existem "targets"como na camada de saída.
  - A solução foi encontrada em 1986 por David Rumelhart com o algoritmo conhecido como Backpropagation.

MLP - Multi-Layer Perceptron

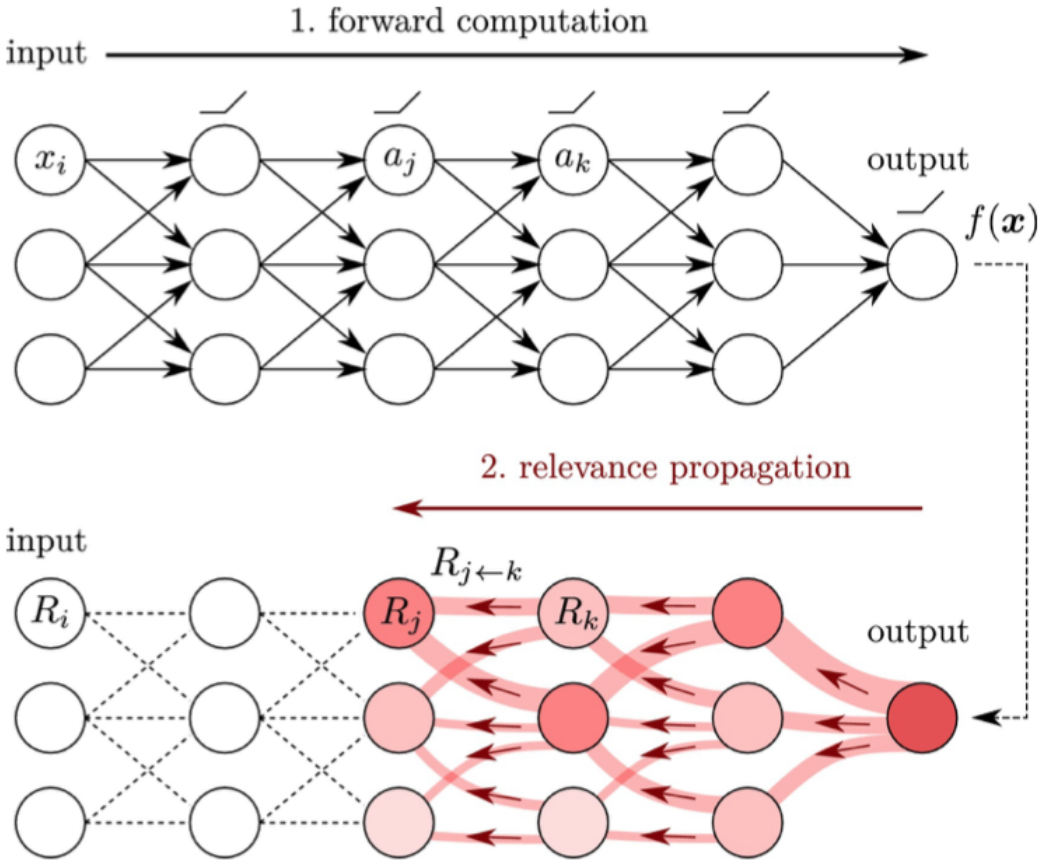
Backpropagation

- Minimizar uma função de error, a qual define o quanto a saída do modelo difere da saída desejada.
- Uma função bastante utilizada é a soma dos erros quadrados (E).
- BackProp encontra o conjunto de pesos que minimiza o error através de uma técnica conhecida como a Descida do Gradiente (Gradient Descent)



MLP - Multi-Layer Perceptron

Backpropagation



MLP - Multi-Layer Perceptron

Backpropagation - Algoritmo

1. Iniciar os valores dos pesos aleatoriamente
2. Apresentar um padrão a camada de entrada da rede
3. Encontrar os valores para as camadas escondidas e camada de saída
4. Encontrar o erro na camada de saída
5. Retro-propagar os erro para ajustar os pesos (tornar o erro menor a cada época)
6. Encontrar o erro na camada escondida
7. Ajustar os pesos

MLP - Multi-Layer Perceptron

Backpropagation - Um exemplo

- Seja o_j o valor de ativação para o neurônio j
- Seja f uma função de ativação e w_{ij} o peso entre os neurônios i e j.
- Seja net_j a entrada para o neurônio, a qual é dada por

$$net_j \sum_{i=1}^{n} w_{ij} o_i$$

- em que n é o número de unidades ligadas ao neurônio j e $o_i = f(net_j)$
- O valor corrente de ativação de um neurônio k é o_k e o label será y_k
- Após realizar os passos 1, 2, e 3, o passo 4 consiste em calcular o erro, o qual é dado por

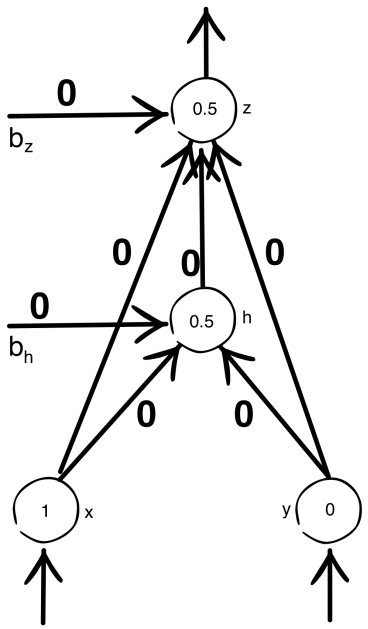
$$\delta_k = (y_k - o_k) \times o_k(1 - o_k)$$

- e os pesos são atualizados da seguinte maneira

$$w_{jk} = w_{jk} + \eta \delta_j o_k \text{ sendo } \eta \text{ a taxa de aprendizagem}$$

MLP - Multi-Layer Perceptron

Considere a rede iniciada da seguinte forma:



- O erro na saida δ_z para o vetor de entrada $x = [1, 0], y = 1$ é dado por

$$\delta_z = (1 - 0.5) \times 0.5(1 - 0.5) = 0.125$$

- Nesse exemplo, vamos usar $\eta = 0.1$
- Atualizando os pesos em função do erro, temos
- $w_{zx} = 0 + 0.1 \times 0.125 \times 1 = 0.125$
- $w_{zy} = 0 + 0.1 \times 0.125 \times 0 = 0$
- $w_{zh} = 0 + 0.1 \times 0.125 \times 0.5 = 0.00625$
- $w_{zb_z} = 0 + 0.1 \times 0.125 \times 1 = 0.125$

MLP - Multi-Layer Perceptron

- O erro dos neurônios da camada escondida é dado por

$$\delta_j = o_j(1 - o_j) \sum_k \delta_j w_{kj}$$

- Como no nosso exemplo temos um único neurônio, podemos simplificar para

$$\delta_h = o_h(1 - o_h)\delta_z w_{zh}$$

- ou seja, $\delta_h = 0.5(1 - 0.5) \times 0.125 \times 0.00625 = 0.000195313$
- Os pesos então podem ser atualizados
  - $w_{hx} = 0 + 0.1 \times 0.000195313 \times 1 = 0.0000195313$
  - $w_{hy} = 0 + 0.1 \times 0.000195313 \times 0 = 0$
  - $w_{hb_h} = 0 + 0.1 \times 0.000195313 \times 1 = 0.0000195313$

MLP - Multi-Layer Perceptron

- Com nos novos pesos calculados, a saída da rede seria 0.507031
- Após uma época (todos os padrões aplicados na rede), temos o seguinte:

x_1	x_2	y_i	z_i
1	0	1	0.499830
0	0	0	0.499830
0	1	1	0.499830
1	1	0	0.499768

- Usando $\eta = 0.1$ o algoritmo levará 20000 épocas para convergir
- Com $\eta = 2$, o número de épocas diminui para 480.

MLP - Multi-Layer Perceptron

- No fim do treinamento temos algo similar à tabela abaixo

x_1	x_2	y_i	z_i
1	0	1	0.994
0	0	0	0.009
0	1	1	0.994
1	1	0	1.0

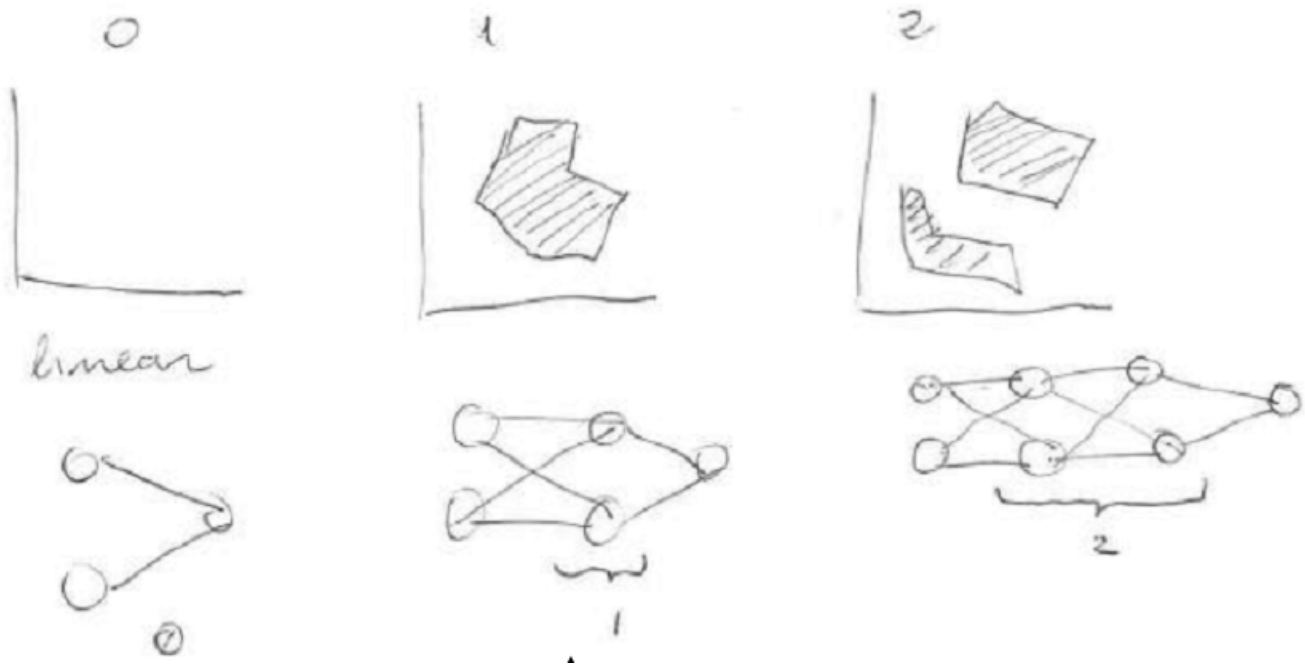
MLP - Multi-Layer Perceptron

Camadas vs Fronteiras

- Dado uma quantidade suficiente de neurônios na camada escondida, é possível resolver qualquer tipo de problema.
- Claro que isso depende do poder de discriminação do vetor de características
- Essa camada pode ser vista como um extrator de características, ou seja, a grosso modo, o neurônio escondido seria uma característica a mais

MLP - Multi-Layer Perceptron

Camadas vs Fronteiras



- Geralmente uma camada escondida resolve a maioria dos problemas de classificação.

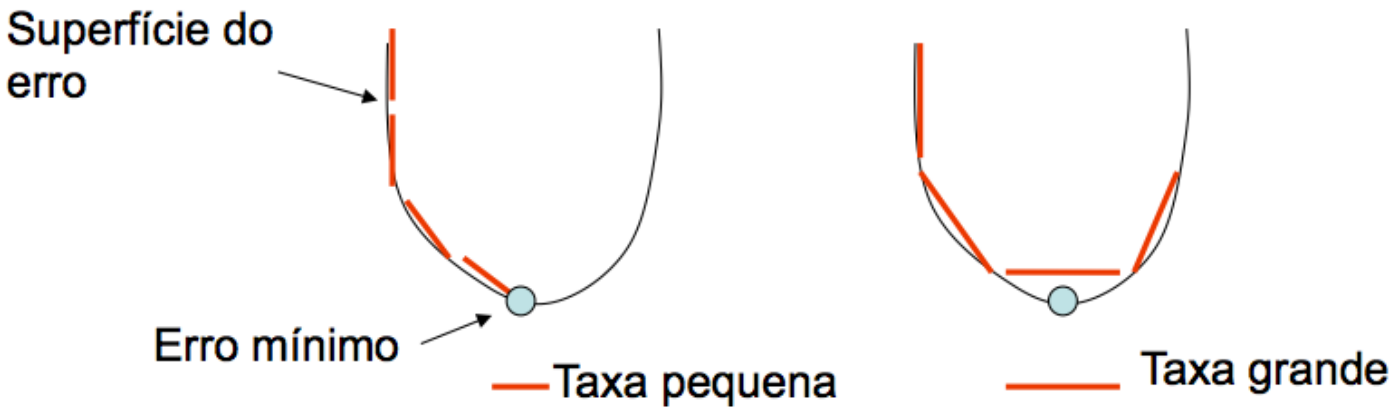
MLP - Multi-Layer Perceptron

Aspectos Práticos

- Alguns pontos devem ser considerados na utilização de redes neurais MLP.
- Leitura recomendada: "Efficient Backprog", Y. LeCun et al., 1988.
  - Taxa de aprendizagem
  - Momentum
  - Shuffle
  - Normalização
  - Generalização

Taxa de Aprendizagem (Learning Rate)

- Taxas muito pequenas tornam o processo mais lento.
- Taxas muito grandes tornam o processo mais rápido.
  - Entretanto, podem não trazer os resultados ideais
  - Busca aleatória.

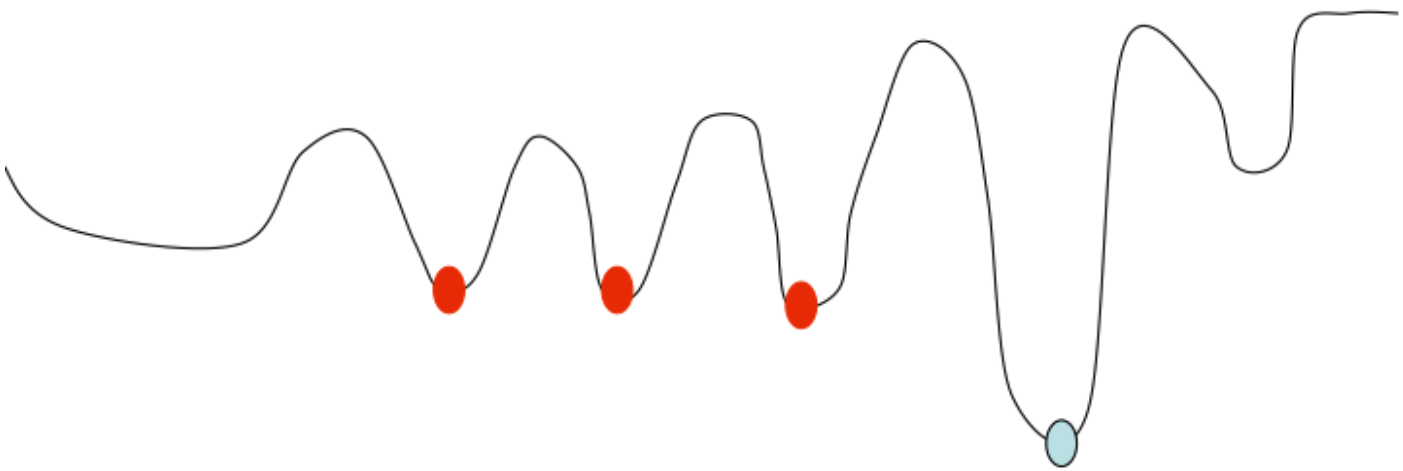


Taxa de Aprendizagem (Learning Rate)

- O ideal é começar com uma taxa grande e reduzir durante as iterações
- Permite a exploração global no início (exploration) a local (exploitation) quando o algoritmo estiver próximo do ótimo global.
- Geralmente valores entre 0.05 e 0.75 fornecem bons resultados

Momentum

- É uma estratégia usada para evitar mínimos locais. Considere a seguinte superfície
- Constante que determina o efeito das mudanças passadas dos pesos na direção atual do movimento no espaço de pesos.
- Desta forma, o termo momentum leva em consideração o efeito de mudanças anteriores de pesos na direção do movimento atual no espaço de pesos. O termo momentum torna-se útil em espaços de erro que contenham longas gargantas, com curvas acentuadas ou vales com descidas suaves



Shuffle

- Redes neurais aprendem melhor quando diferentes exemplos de diferentes classes são apresentados a rede
- Uma prática muito comum consiste em apresentar um exemplo de cada classe a rede
  - Isso garante que os pesos serão atualizados levando-se em consideração todas as classes
- Se apresentarmos à rede todos os exemplos de uma classe, e assim por diante, os pesos finais tenderão para a última classe.
  - Isso garante que os pesos serão atualizados levando-se em consideração todas as classes.
  - Esse fenômeno é conhecido como "catastrophic forgetting"

Normalização

- A normalização é interessante quando existem características em diversas unidades dentro do vetor de características.
- Nesses casos, valores muito altos podem saturar a função de ativação.
- Uma maneira bastante simples de normalizar os dados consiste em somar todas as características e dividir pela soma
- Outra normalização bastante usada é a normalização Z-score.
- Para redes neurais MLP, geralmente é interessante ter as características com média próxima de zero

$$Z = \frac{X - \mu}{\sigma}$$

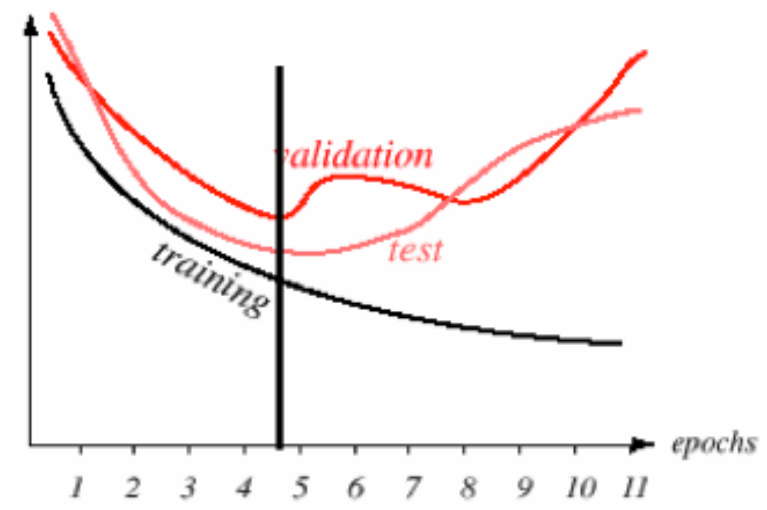
- Melhora o tempo de convergência durante a aprendizagem

Normalização

- As características devem ser não correlacionadas se possível
- Quando temos poucas características podemos verificar isso facilmente.
- Com várias características, o problema se torna muito mais complexo.
- Métodos de seleção de características

Generalização

- Um aspecto bastante importante quando treinamos um classificador é garantir que o mesmo tenha um bom poder de generalização.
  - Evitar "overfitting'
- A maneira clássica de se garantir uma boa generalização consiste em reservar uma parte da base para validar a generalização.
- A cada iteração, devemos monitorar o desempenho na base de validação.
- Não é raro observar o seguinte desempenho



Tamanho da Rede

- Geralmente uma camada escondida é suficiente.
- Em poucos casos você vai precisar adicionar uma segunda camada escondida.
- Não existe uma formula matemática para se encontrar o número de neurônios (processo empírico)
- Dica prática: Comece com uma rede pequena, pois a aprendizagem vai ser mais rápida

Aprendendo a Representação

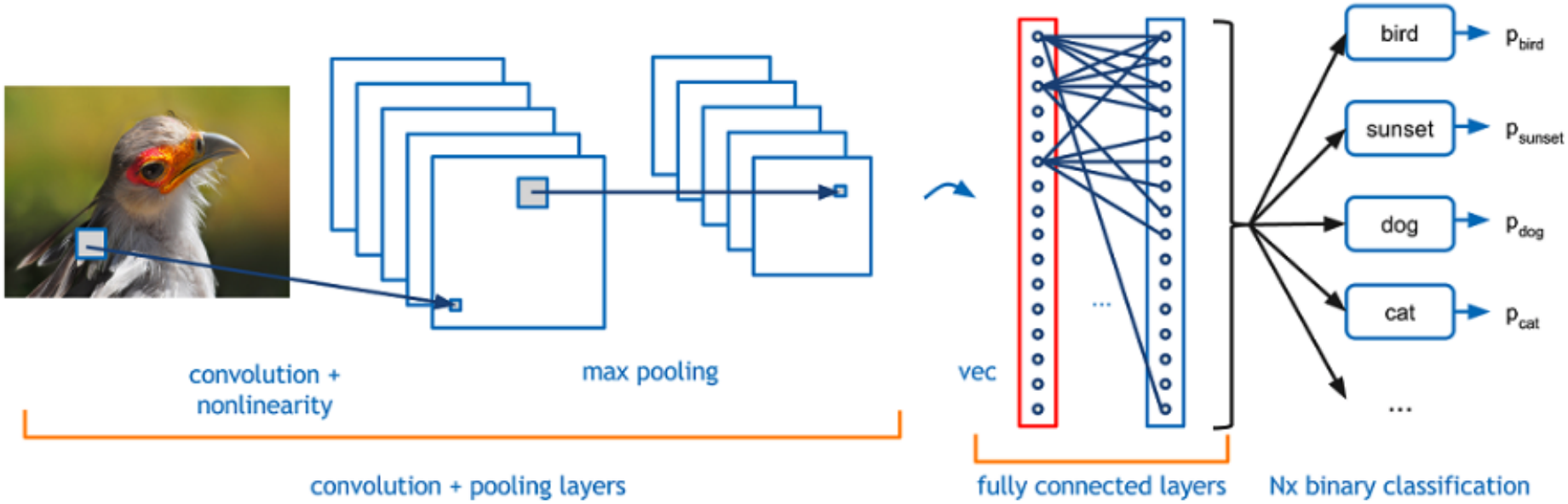
- O desempenho de qualquer método de aprendizagem de máquina depende da escolha da representação (características).
- Em geral, tal representação é definida empiricamente por um especialista humano.
- O ideal seria tornar os métodos de aprendizagem de máquina menos dependente do especialista humano.

Representation Learning

Aprender representações discriminantes a partir dos dados.

- Feature Learning ou Deep Learning.
- Uma técnica bastante utilizada nesse contexto é a Convolutional Neural Network (CNN).

Convolutional Neural Networks (CNN)

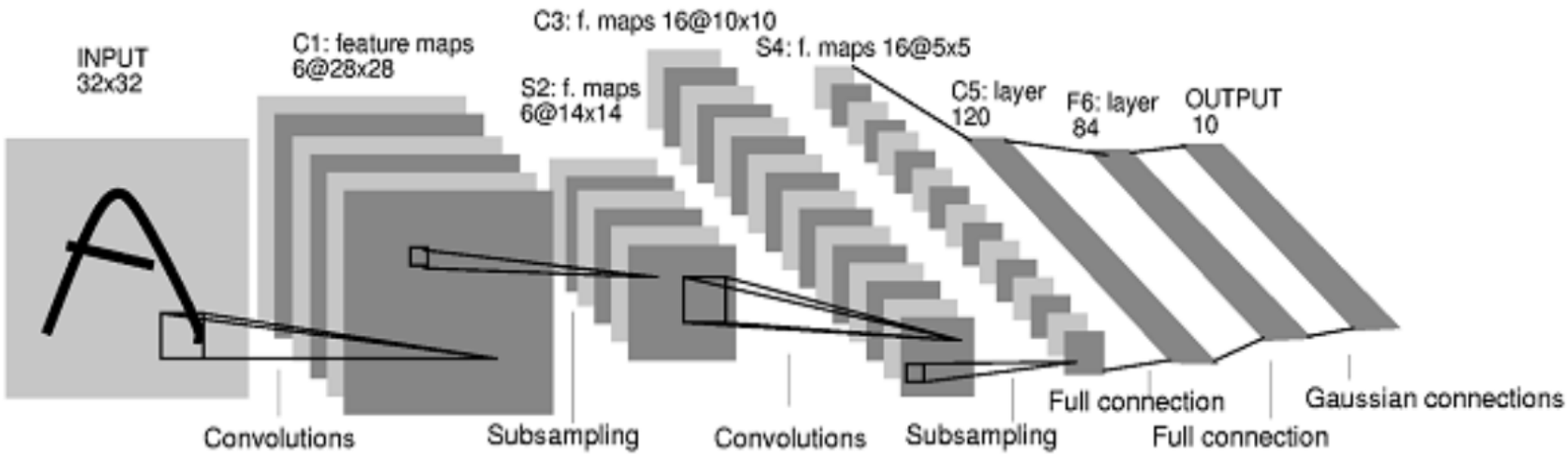


Existem quatro tipos básicos de camadas em uma CNN:

- Convolução
- Camada de não linearidade (ReLU)
- Pooling (Agregação)
- Classificação (Fully Connected Layer)

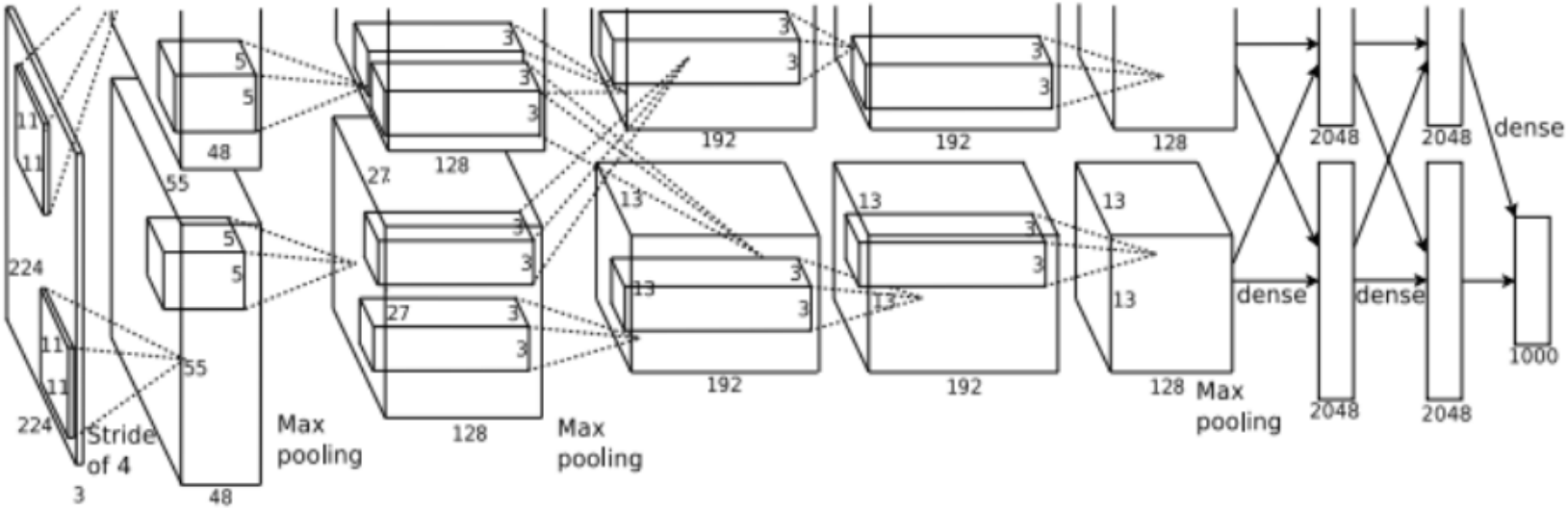
Convolutional Neural Networks

- Conceito introduzido por Fukushima em 1980 (Neocognitron).
- Popularizado por Yann LeCun (1998) com a LeNet 5, uma CNN com 7 camadas para reconhecer dígitos manuscritos.
  - Alto custo computacional não possibilitou a popularização na época.



Convolutional Neural Networks

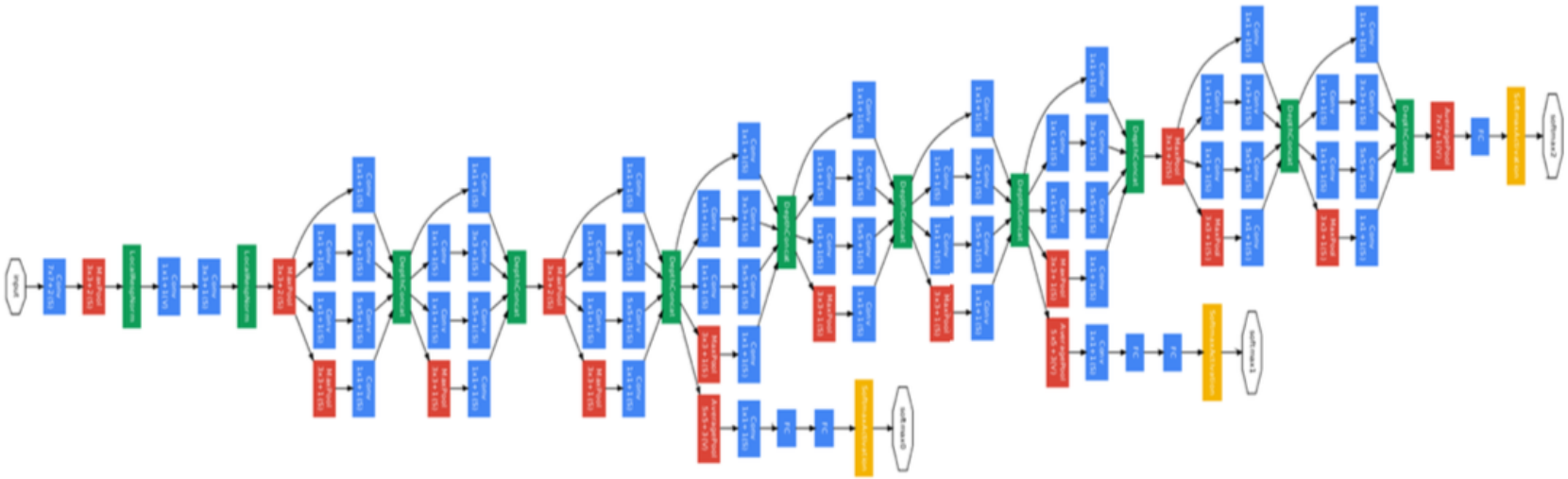
- Em 2012, Alex Krizhevsky, propôs a utilização de GPU para o treinamento da CNN.
  - Venceu a competição ImageNet baixando o erro de 26% para 15%.
  - Despertou o interesse da comunidade científica e empresas para diversas aplicações.



Convolutional Neural Networks

GoogleNet

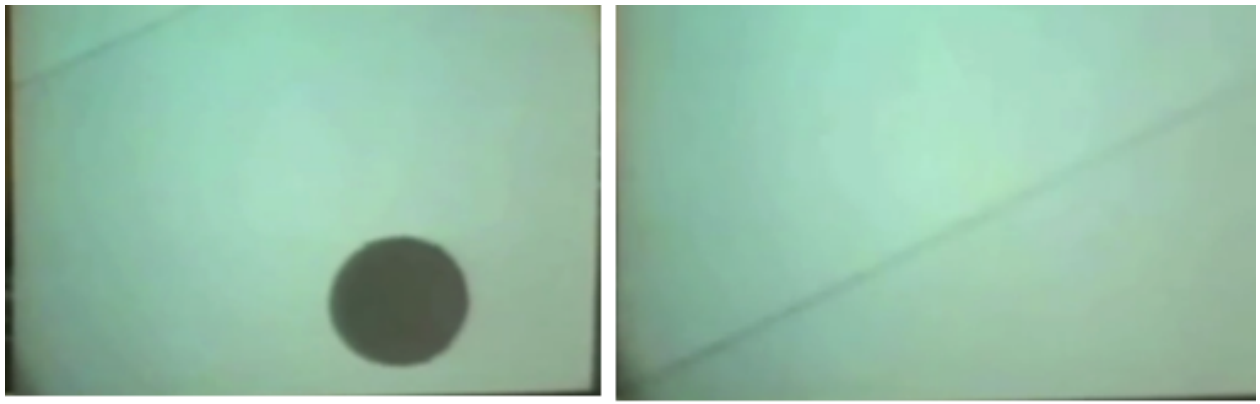
- Melhor desempenho na ImageNet em 2014.



Convolutional Neural Networks

Fundamentação Biológica

- Experimento conduzido por Hubel e Wiesel em 1962 mostrou "por acidente" que certos neurônios do cérebro respondiam somente na presença de bordas com certas orientações ((▸ Video Youtube)).

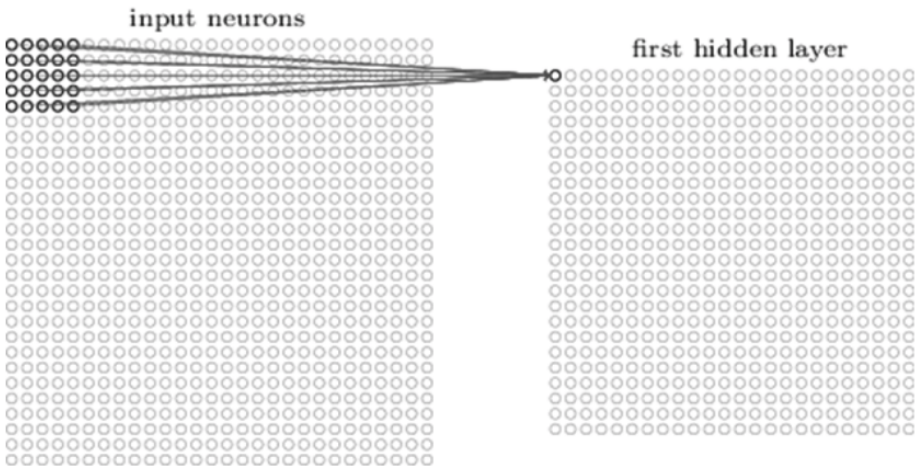


- Classificar imagens olhando somente para características de baixo nível como bordas e curvas.
  - Essa é a ideia da CNN. Como isso acontece ?

Convolutional Neural Networks

Primeira Camada

- A primeira camada é sempre uma camada do tipo "Convolucional".
- Um filtro (também conhecido como "kernel') com um conjunto de pesos é deslizado pela imagem e o resultado da convolução é atualizado no mapa de ativação ou "feature map".
- No exemplo abaixo, o kernel 5×5 aplicado na imagem 32×32 produz um feature map de tamanho 28×28.



Visualization of 5 x 5 filter convolving around an input volume and producing an activation map

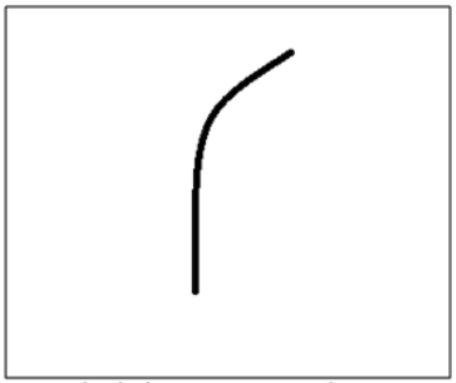
Convolutional Neural Networks

O que acontece nessa camada?

- Cada filtro desse pode ser visto como um detector de características, como por exemplo, bordas, linhas curvas, cores.
- Considere o filtro abaixo de tamanho 7×7 projetado para ser um detector de curva.

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

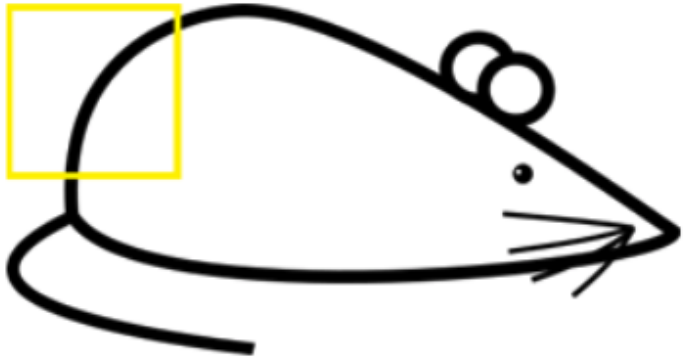


Visualization of a curve detector filter

- A ideia é que quando tiver algum traço parecido com o filtro na imagem de entrada, a convolução entre o filtro e imagem vai retornar um valor alto, caso contrário, um valor baixo será retornado.

Convolutional Neural Networks

Original image



Visualization of the filter on the image

Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

- Esse valor alto significa que é provável que esse tipo de curva exista na imagem original

Convolutional Neural Networks

Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

$*$

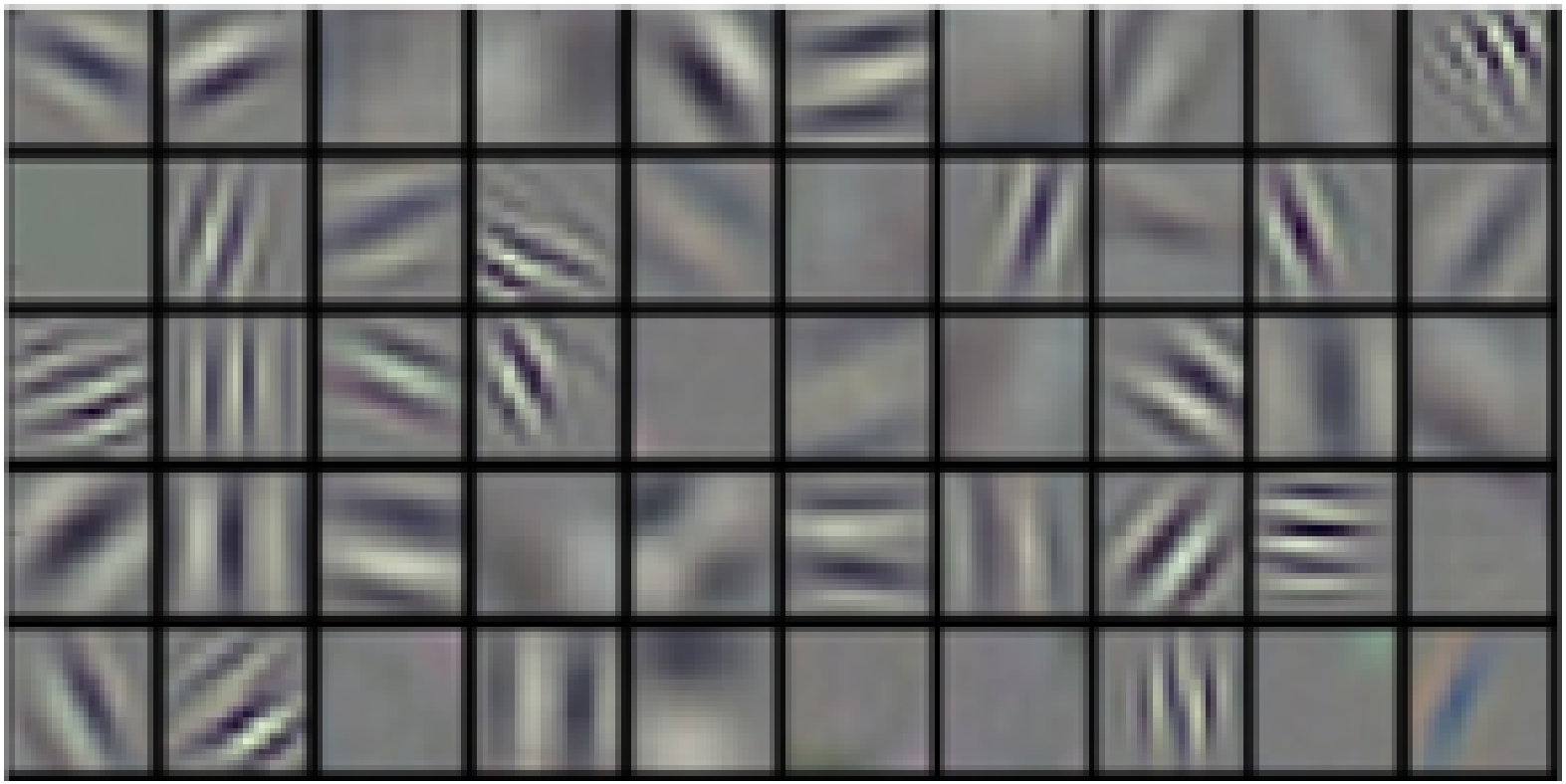
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = 0

Convolutional Neural Networks

Filtros extraídos da primeira camada de convolução de uma CNN.



- Na prática, a CNN aprende o valor desse filtros durante o processo de aprendizagem.
- Precisamos especificar, porém, parâmetros como o número de filtros, tamanho, arquitetura da rede, etc.

Convolutional Neural Networks

O tamanho do Feature Map depende basicamente de três parâmetros:

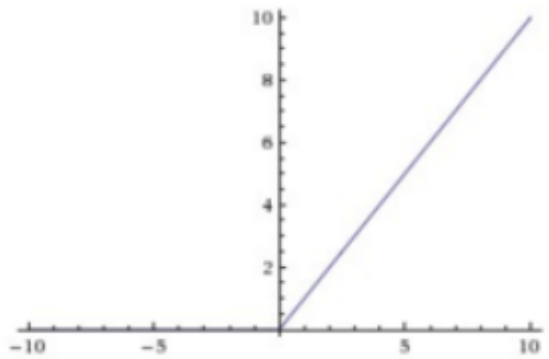
- Número de filtros
- Stride: Deslocamento do filtro, por exemplo, stride=2 significa que o filtro se desloca de 2 e 2 pixels a cada iteração.
- Tamanho do kernel (Kernel size)

Convolutional Neural Networks

ReLU (Rectified Linear Unit)

- Operação aplicada no pixel (element wise) que basicamente substitui todos os valores negativos do Feature Map por zero.
- O objetivo da ReLU é introduzir não linearidade na rede, uma vez que a maioria dos dados no mundo real é não linear (e a operação de convolução é linear).
- Além disso, a ReLU resolve o problema do desaparecimento do gradiente (Vanish Gradient) e convergência da rede quando outras funções não lineares como a Sigmoid [0,1] e Tangh [-1,1] são utilizadas.
- Diferentemente da Sigmoid e Tangh, a ReLU não satura.

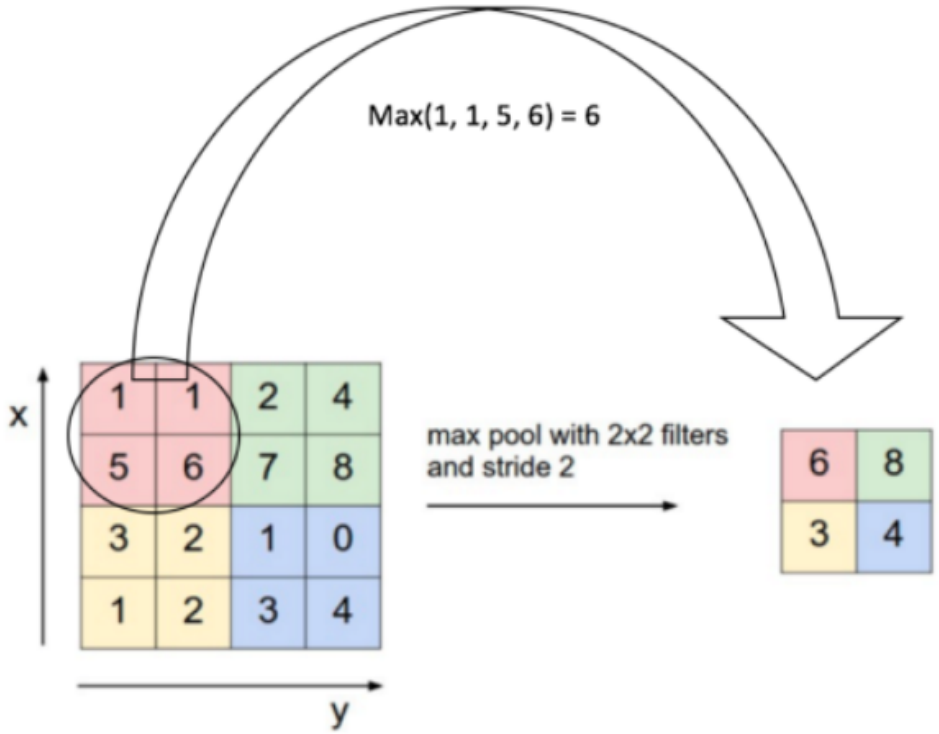
$$\text{Output} = \text{Max}(\text{zero}, \text{Input})$$



Convolutional Neural Networks

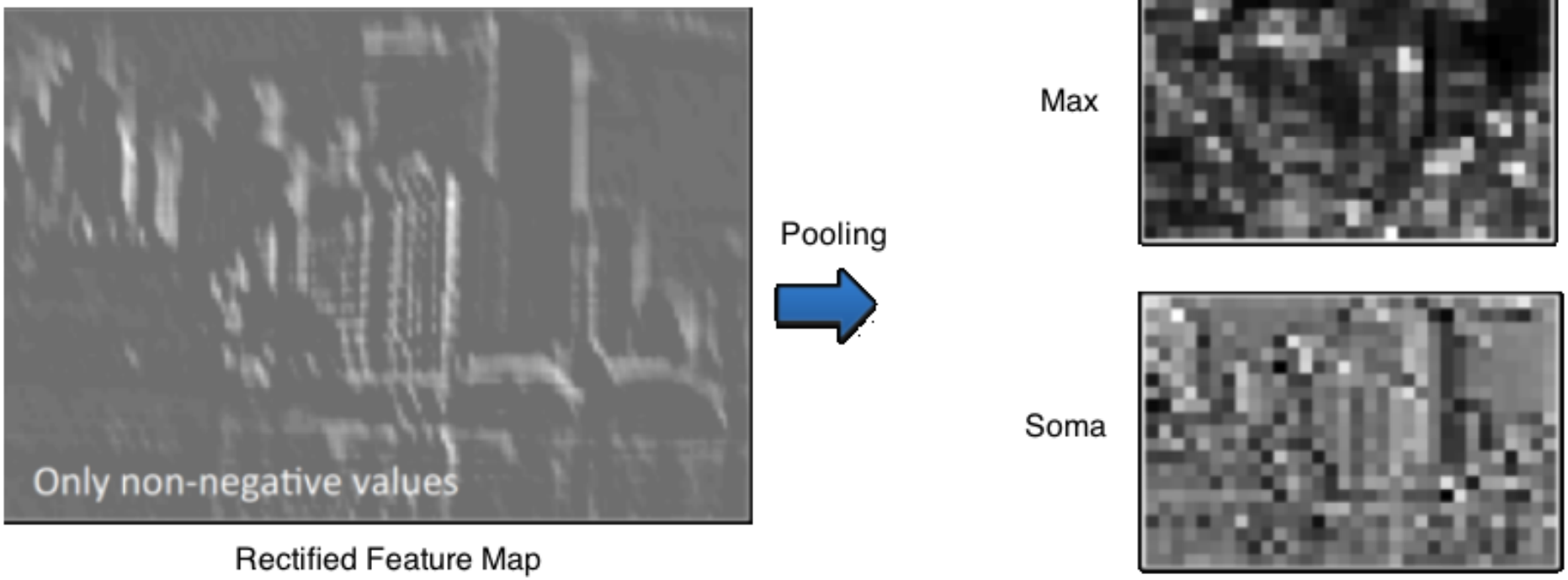
Pooling (Agregação)

- Reduz a dimensionalidade de cada Feature Map tentando conservar as informações mais relevantes.
- Pode ser implementado de diversas maneiras, como por exemplo, Max, Média, Soma, etc.



Convolutional Neural Networks

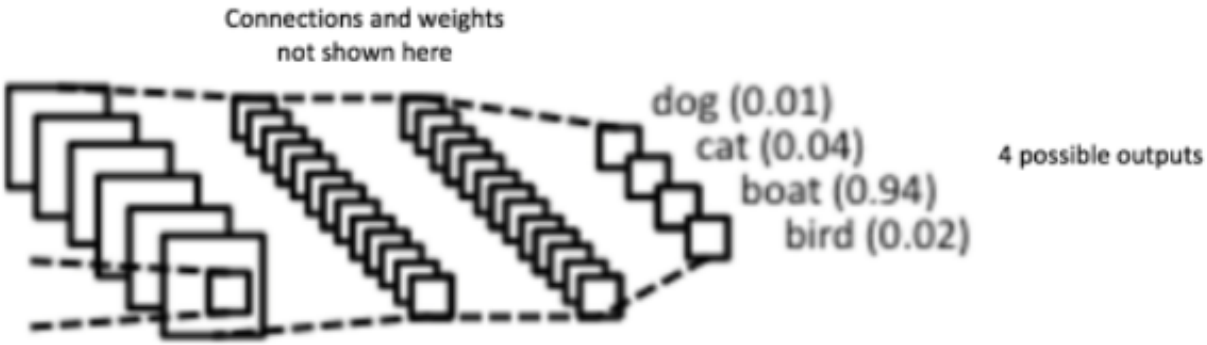
Exemplo de Pooling



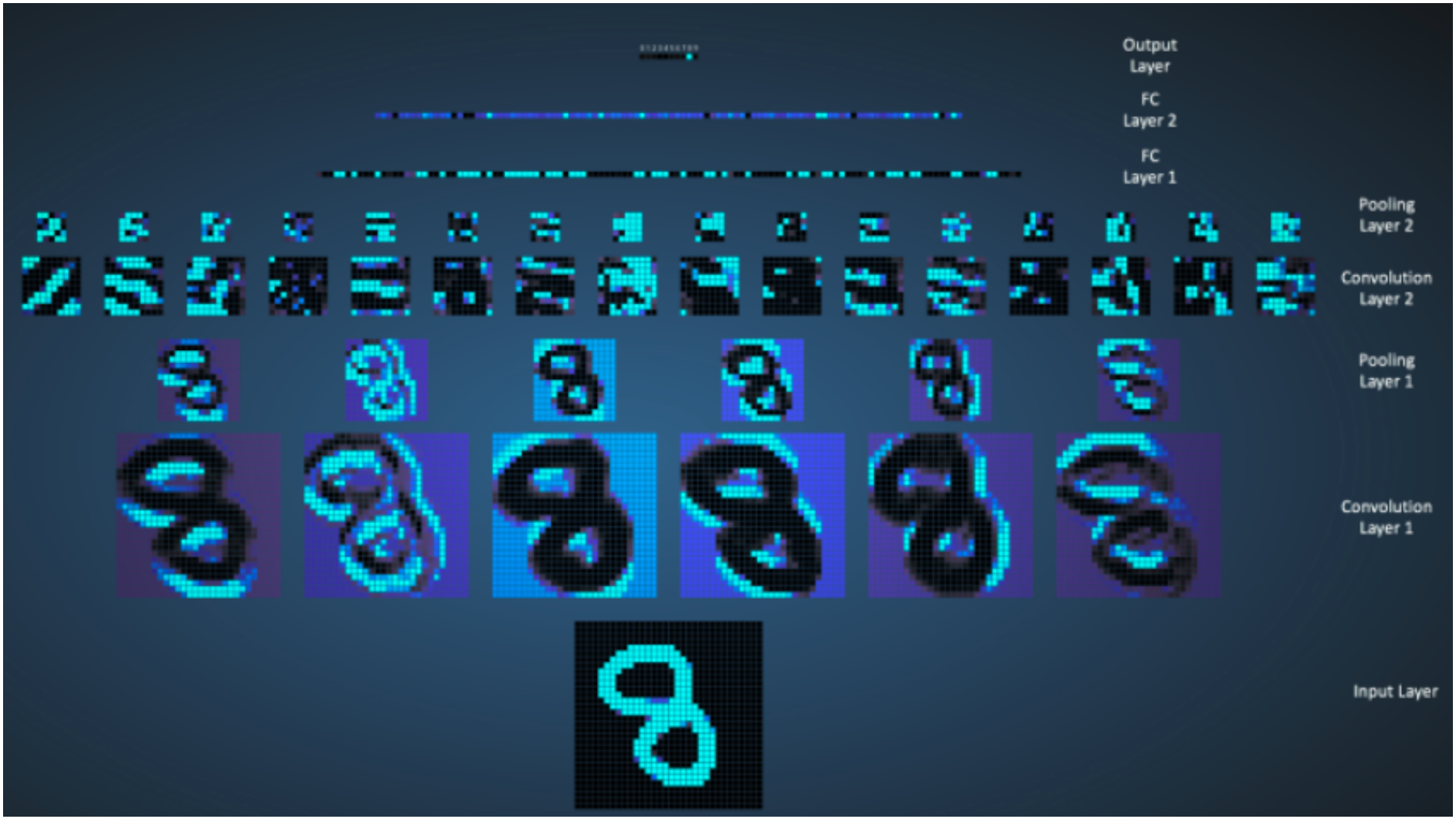
Convolutional Neural Networks

Classificação (Fully Connected Layer)

- Essa camanda é uma MLP com função de ativação softmax na camada de saída (que garante estimação de probabilidades a posteriori), que garante que a soma das saídas seja = 1.
- A saída das camadas de convolução+ReLU+Pooling representam características de alto nível que servem como entrada da MLP.
- Essa representação pode ser utilizada com qualquer outro classificador.



Convolutional Neural Networks



Convolutional Neural Networks

O processo de treinamento da CNN pode ser resumido da seguinte forma:

- Passo 1: Iniciar aleatoriamente os pesos da redes
- Passo 2: A rede recebe uma imagem e realiza a propagação na rede (forward) e encontra os valores na camada de sáida.
- Passo 3: Calcula o erro na camada de saída
- Passo 4: Aplica Backpropagation para atualizar os pesos da rede e assim minimizar o erro na saída
- Passo 5: Repete os passos 2-4 para todas as imagens da base de treinamento.

Data Augmentation

- Como a CNN possui um número bastante grande de pesos que devem ser aprendidos durante a fase de treinamento, em geral, uma grande quantidade de dados é necessária.
- Entretanto, em alguns casos a base de treinamento é limitada.
- Nesses casos, alguma técnica para gerar mais exemplos para a aprendizagem é necessária.
  - Adição de ruído,
  - Mudança de escala
  - Rotação e translação

CNN como extrator de características

- A CNN pode ser vista como um extrator universal de características.
- Uma rede usada frequentemente é a ImageNet (camada fc7)