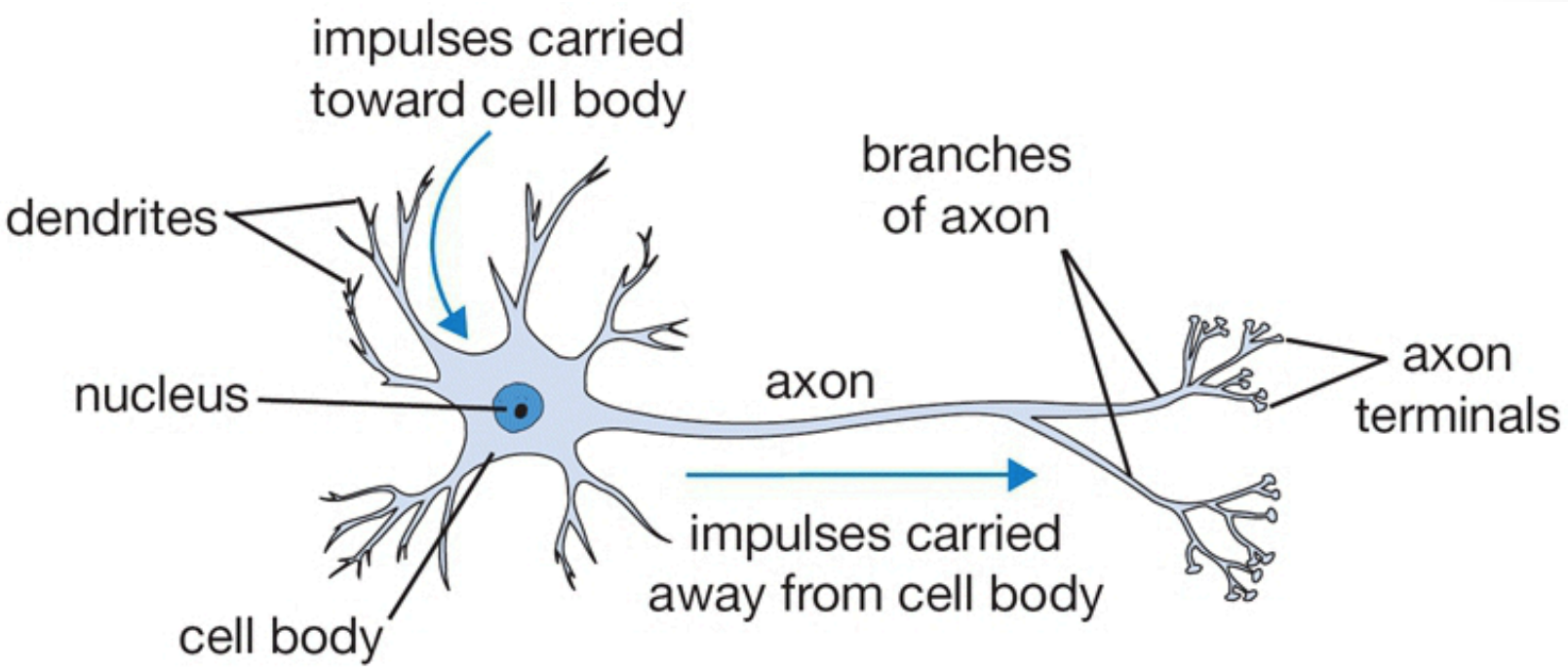
Redes Neurais Artificiais - Introdução

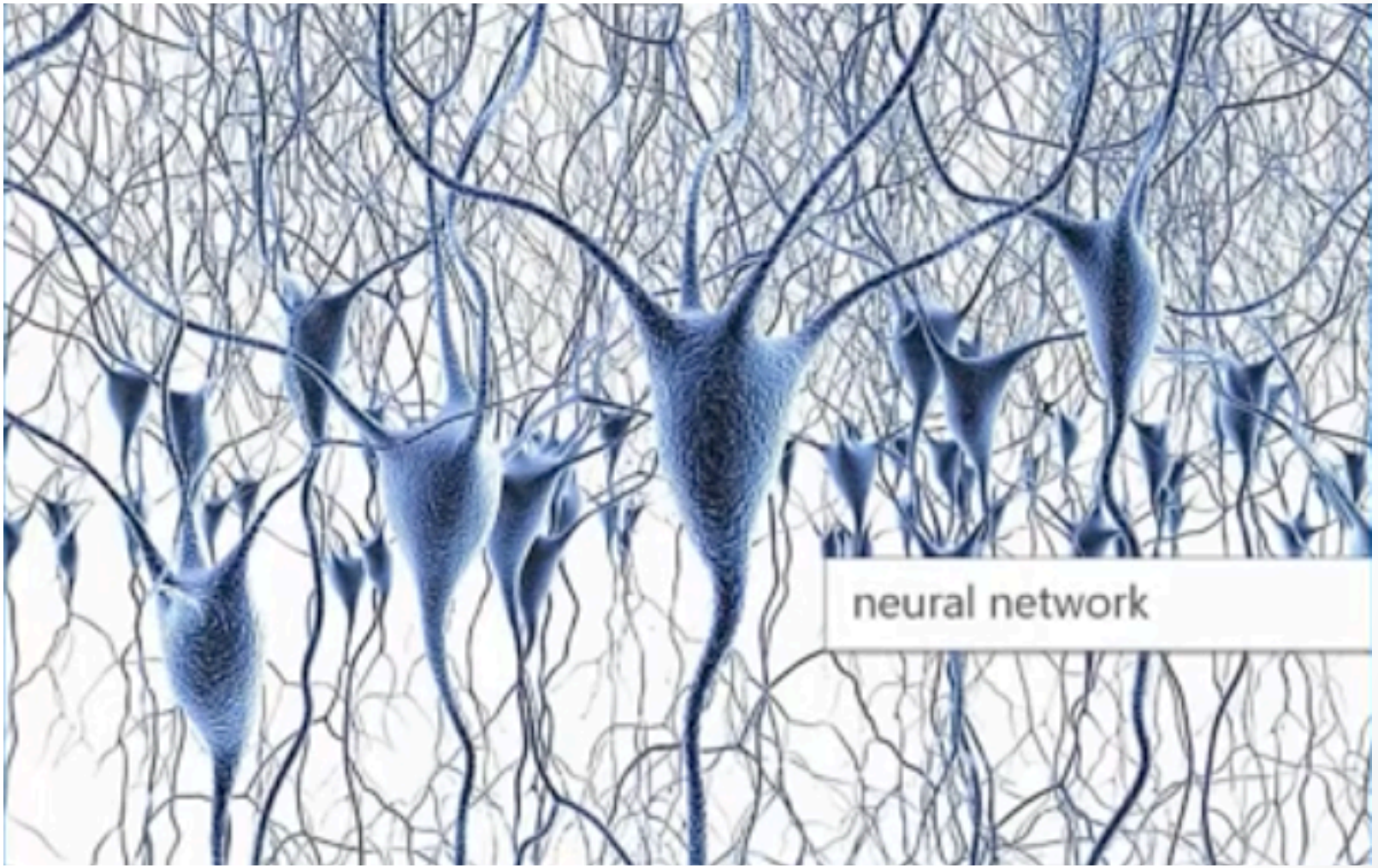
Visão Computacional

Prof. Geraldo Braz Junior

Inspiração

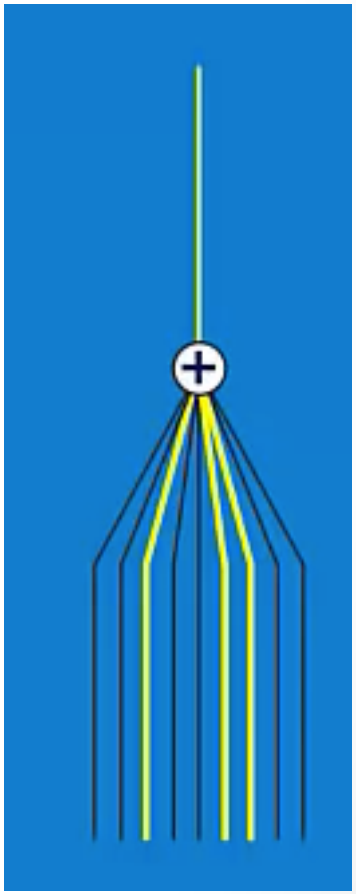

impulses carried
toward cell body
dendrites
branches
of axon
nucleus
axon
axon
terminals
impulses carried
away from cell body
cell body

Inspiração



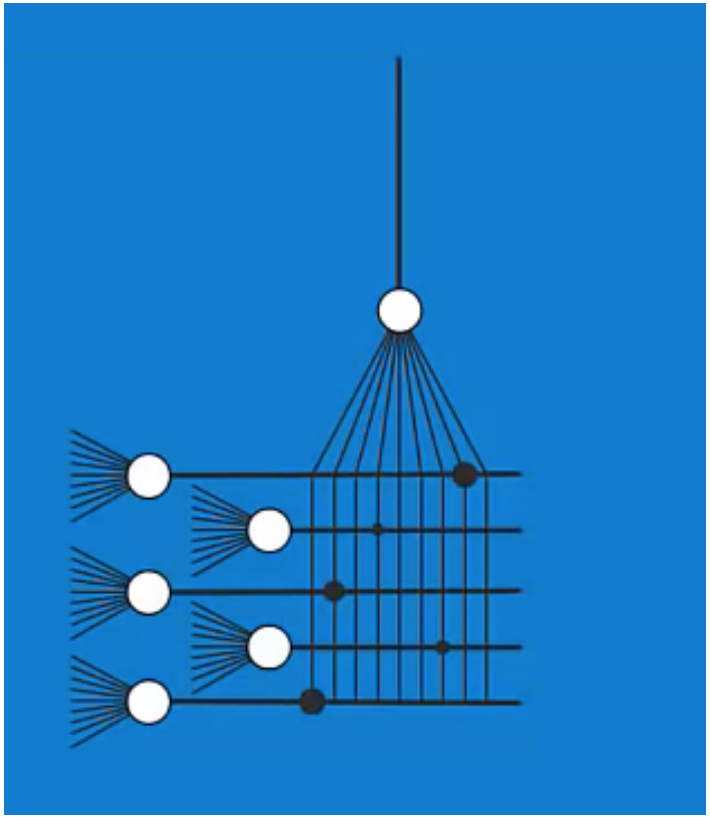
Inspiração

- Atividade seletiva nas conexões
- “Soma” os impulsos e passa a diante

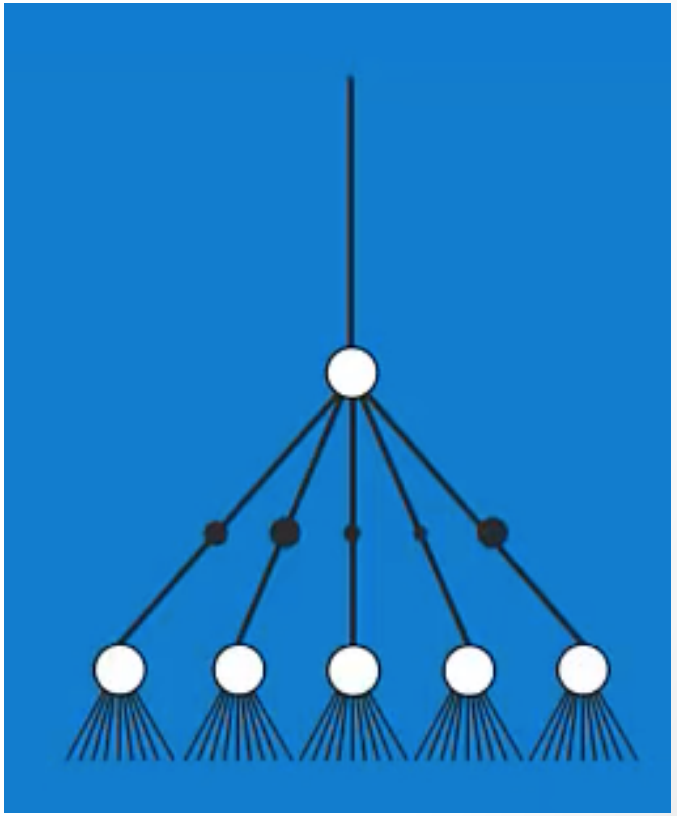


Inspiração

- As conexões entre os dendritos informam como a informação deve trafegar (força de conexão)

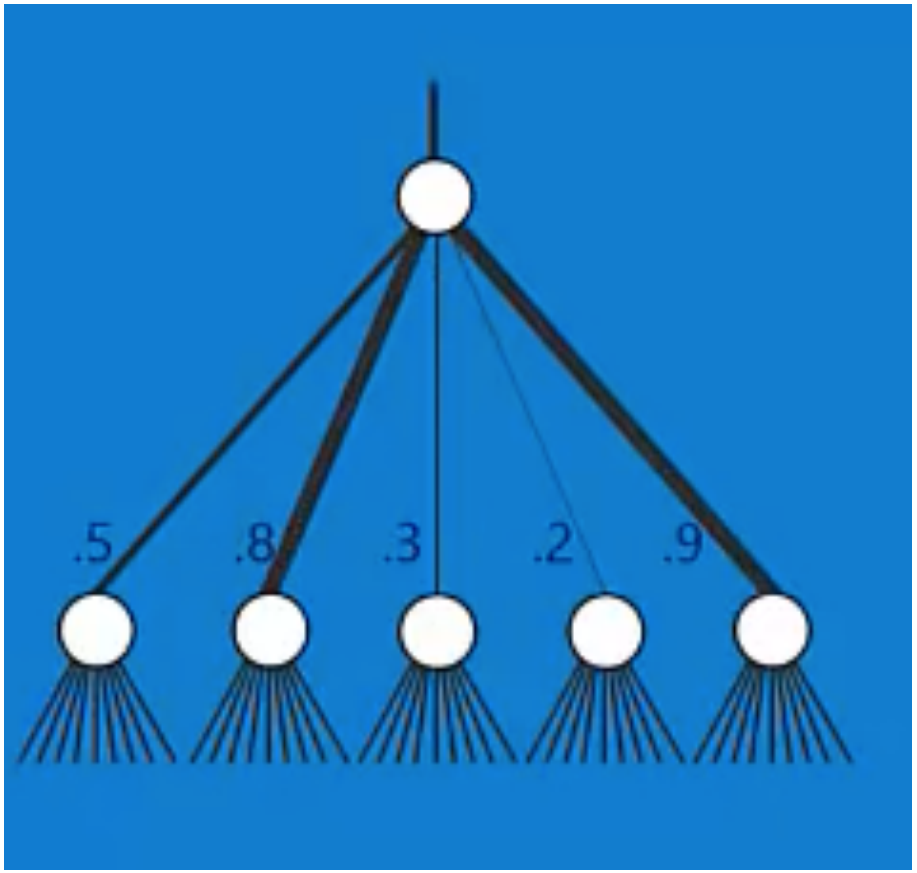


ou



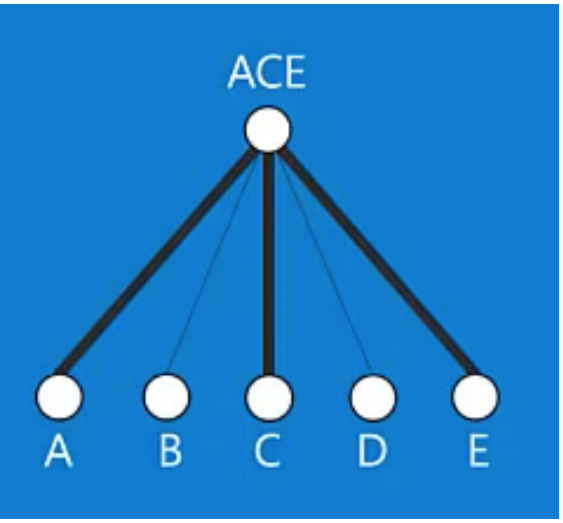
Inspiração

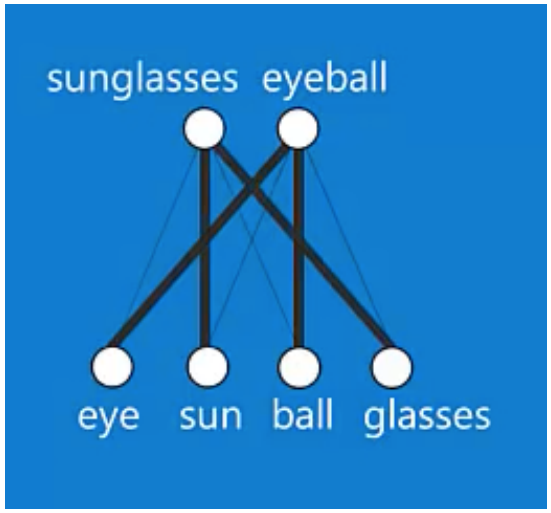
- A força pode ser representada por pesos

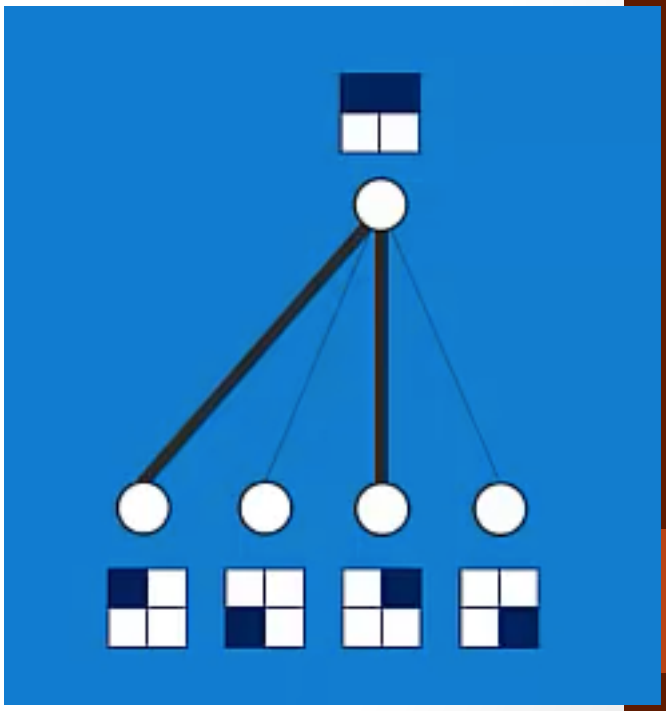


Inspiração

- As conexões determinam como reconhecer um objeto específico.
- Ou quais ligações ativar para atingir o objetivo





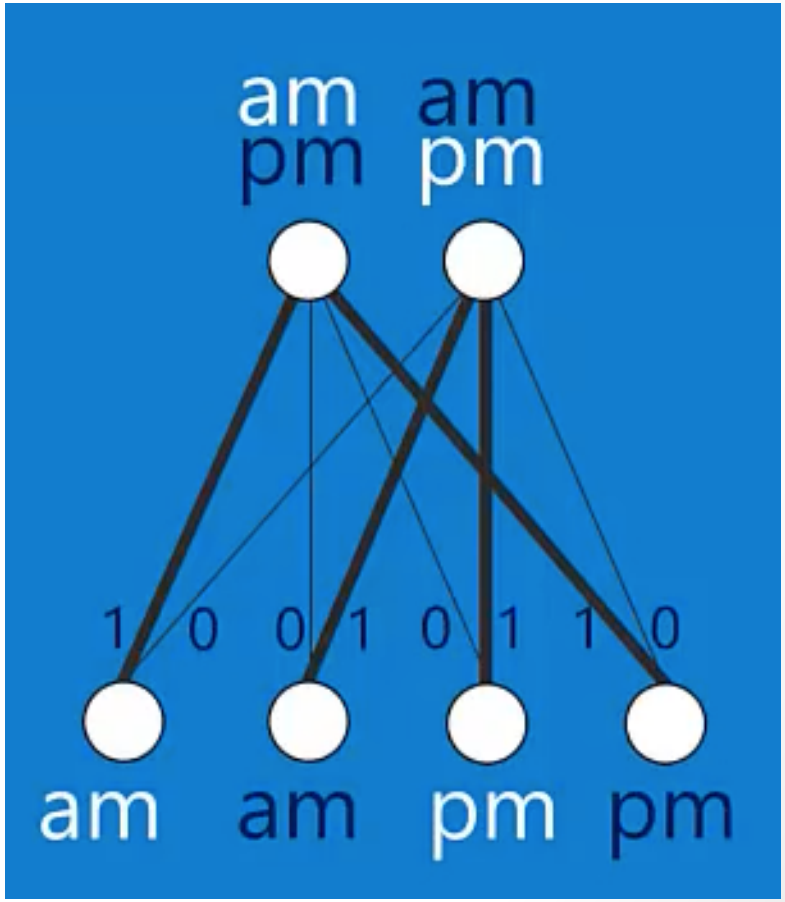


Inspiração

- Mas como aprender os pesos das conexões de maneira a representar a informação que desejamos?

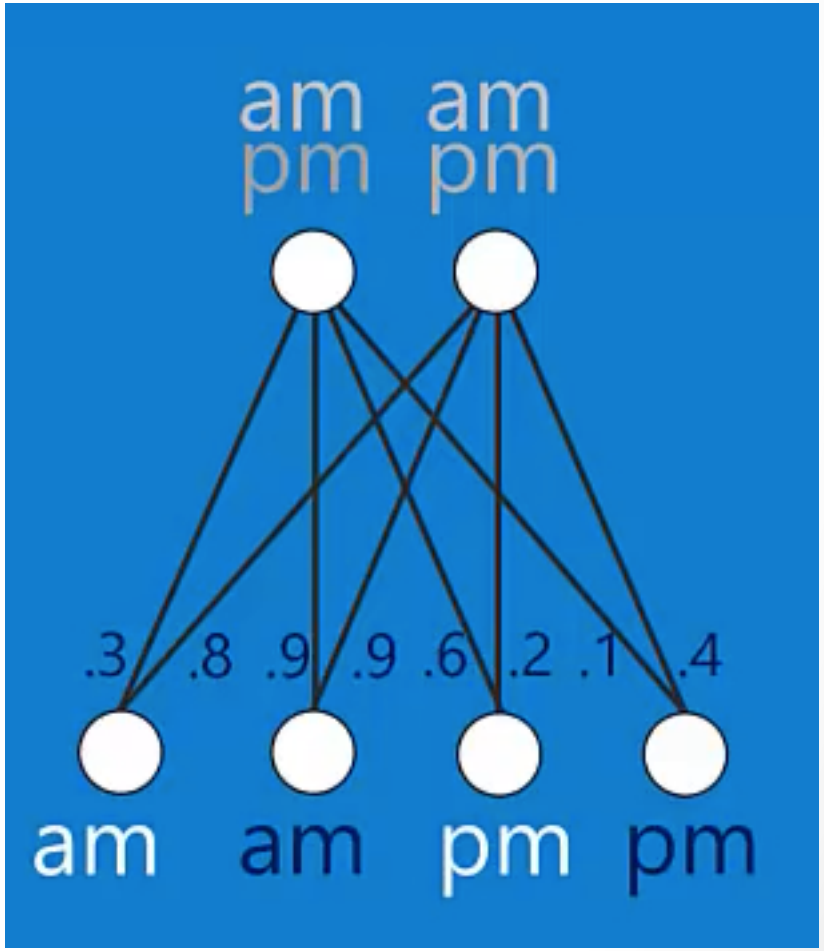
- Funcionário que trabalha pela manha folga a noite
- Funcionário que trabalha a noite folga pela manhã

Nós queremos →



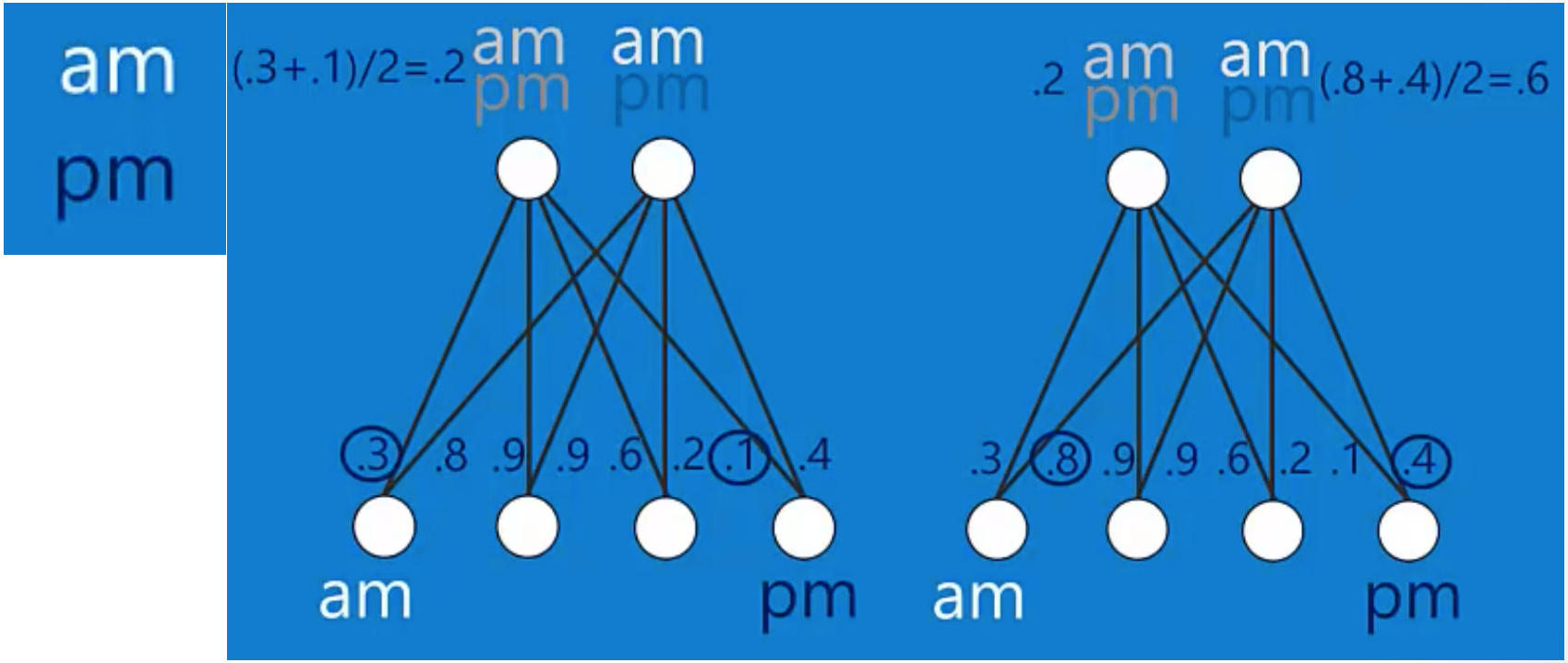
Inspiração

- Mas como chegar na resposta?
- Parta do aleatório



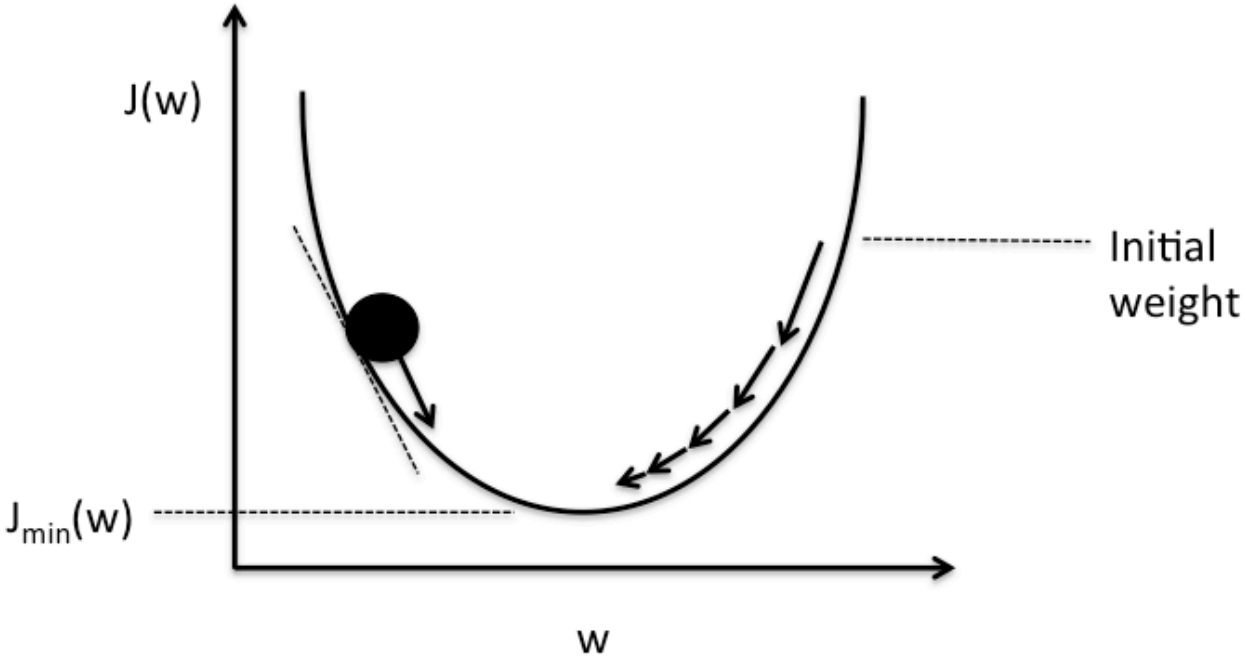
Inspiração

- A partir de um treinamento → **Ative os neurônios** (Feed Forward)



Inspiração

- Aprenda com os erros
- Gradient Descent (obtem a direção da diferença que minimiza o erro J)

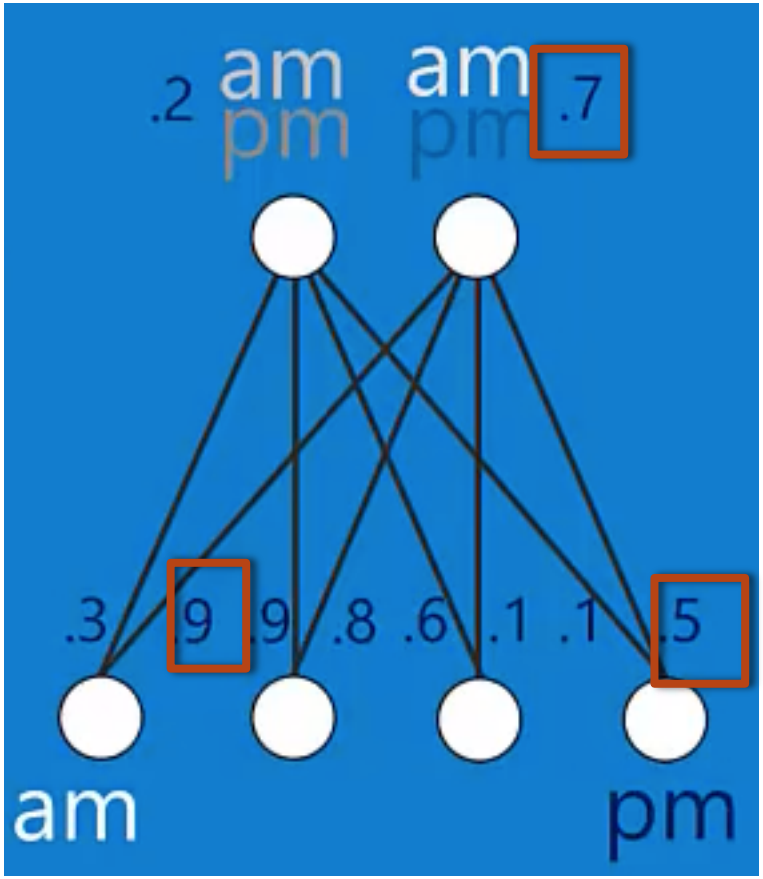


Schematic of gradient descent.

Inspiração

- Ajuste os pesos.
- Direção do ajuste: camada superior → camada abaixo
  - Backpropagation
  - ajuste = erro * gradiente * delta

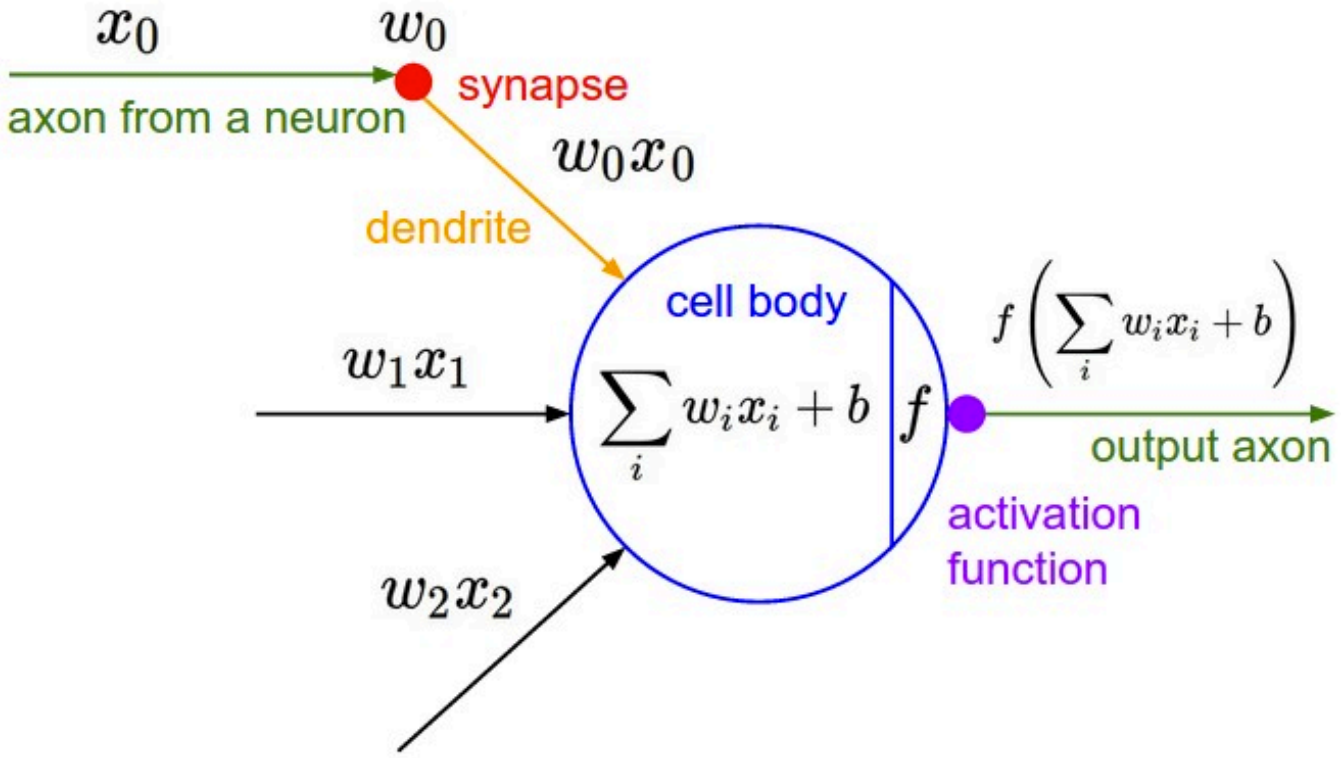
Inspiração



- Repita o processo até que os pesos parem de mudar

Neurônio Artificial

- Ele tem um meio de computar seu nível de ativação dados sinais de entrada **xi** e pesos numéricos **wi**.



Representações

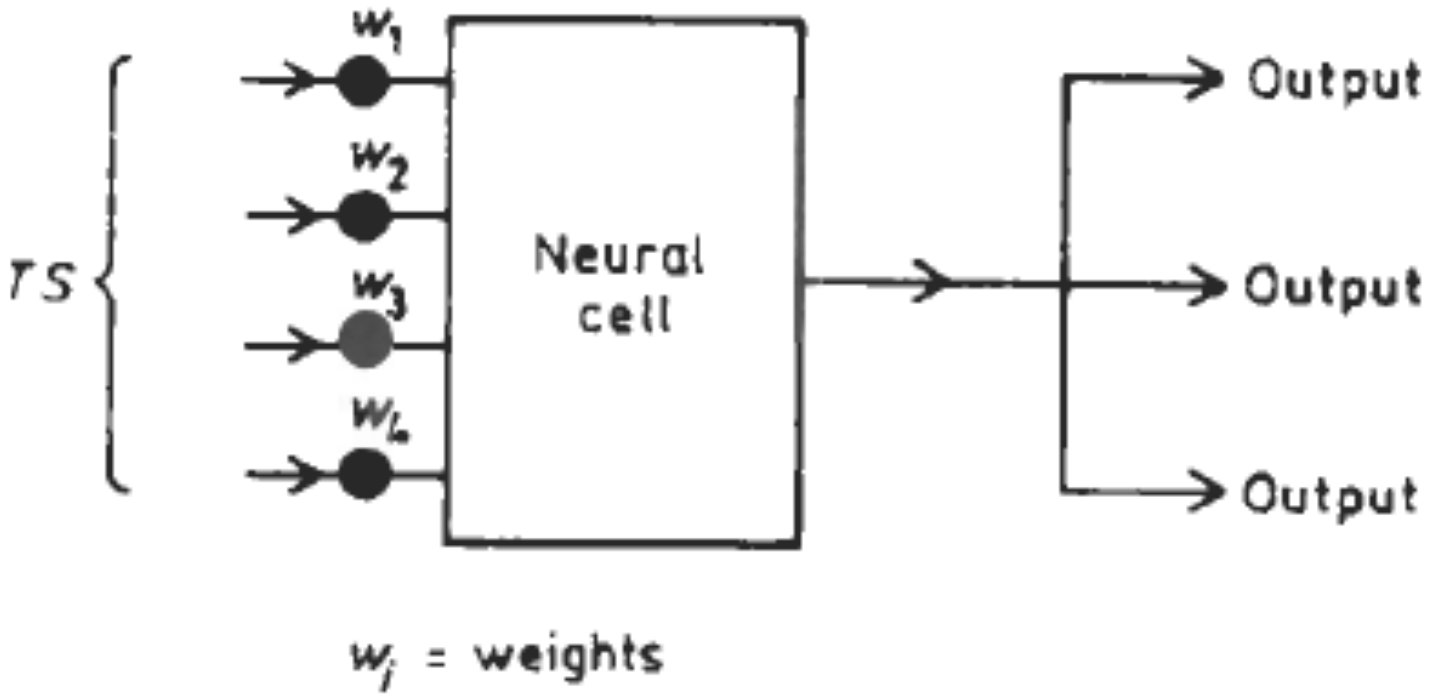


Fig. 2.4. Schematic analog of a biological neural cell.

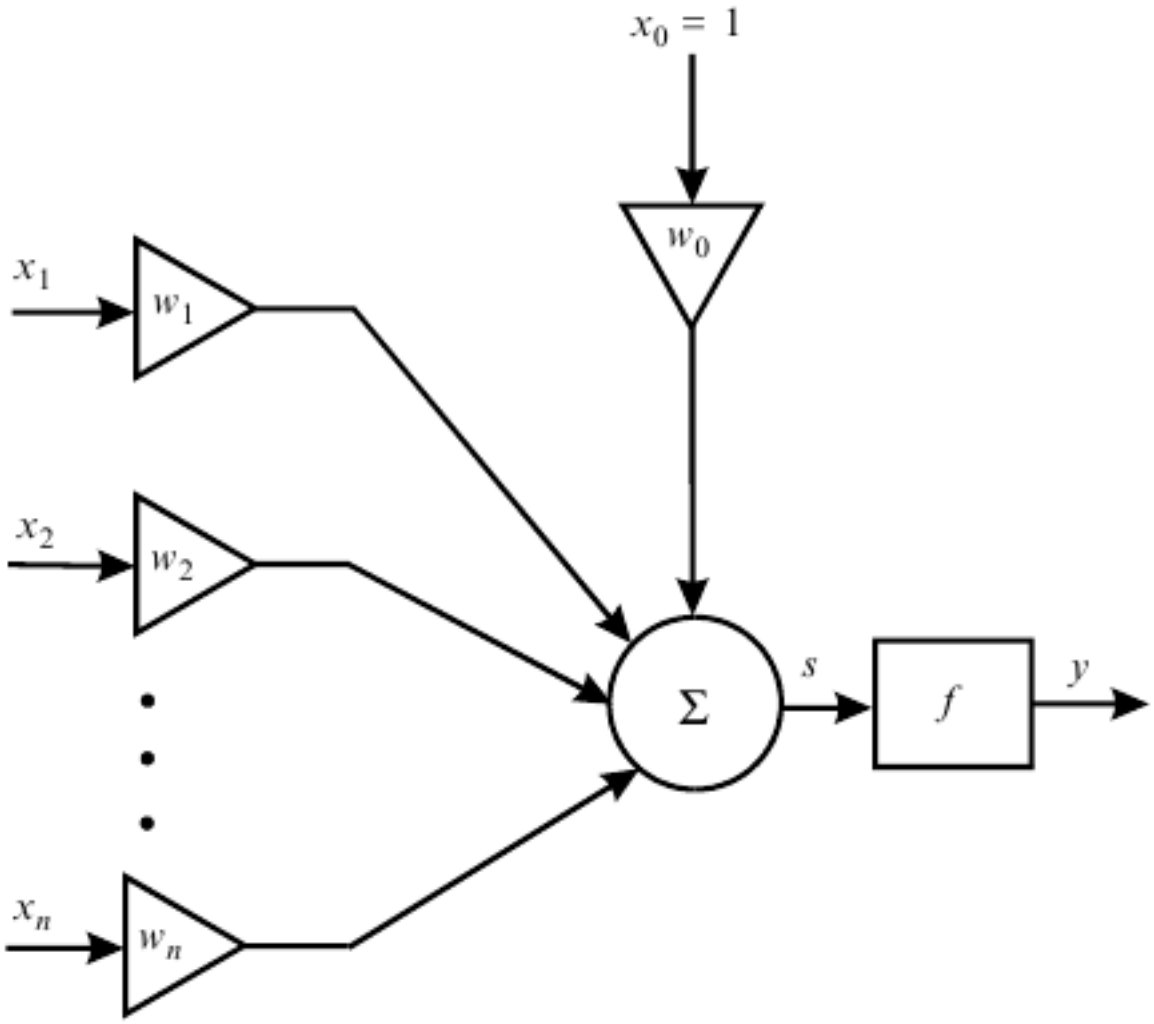


FIGURE 6.2. Model of a neuron

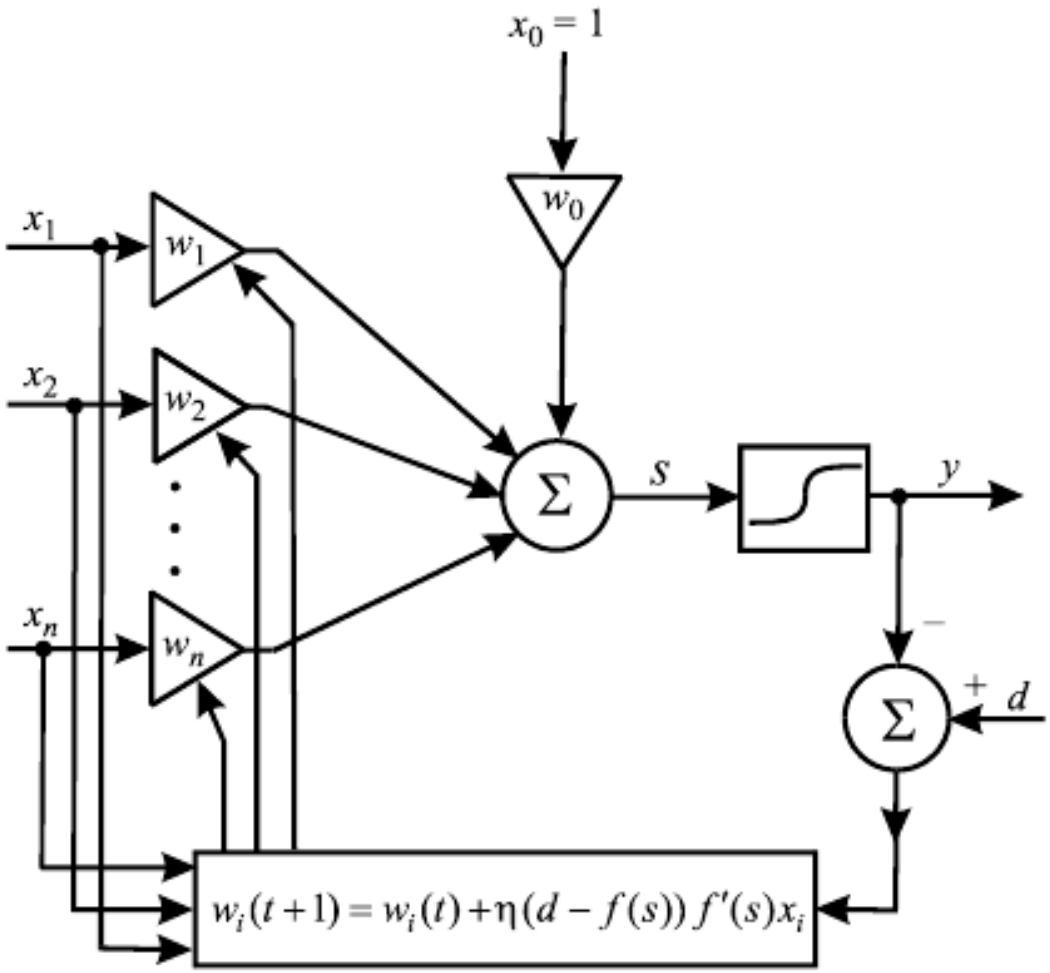


FIGURE 6.11. The scheme of a sigmoidal neuron

Idéia Inicial

$$\left.\begin{array}{l} \text{Out} = 1, \text{Net} > 0 \\ \\ = 0, \text{Net} \leq 0. \end{array}\right\} \qquad (8.1)$$

Net in the McCulloch-Pitts model is formally written as

$$\text{Net} = \sum_{i=1}^{n} w_i x_i - \theta \qquad (8.2)$$

where

x_i is an input for i=1 to n,
w_i is the weight of input x_i, and
θ is a bias term.

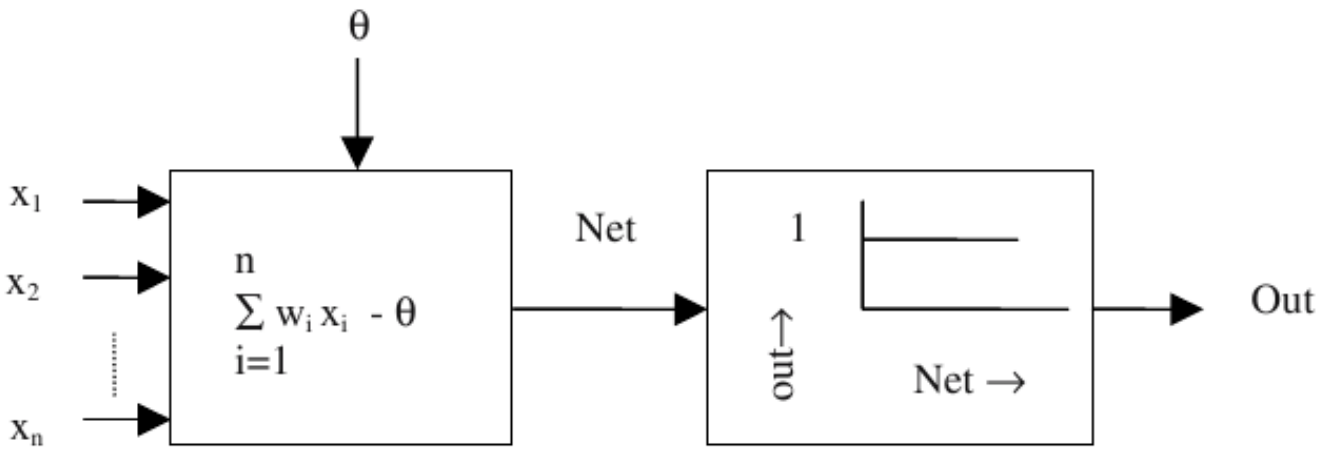


Fig. 8.1: The McCulloch-Pitts neuron containing a weighted summer and a step type non-linearity.

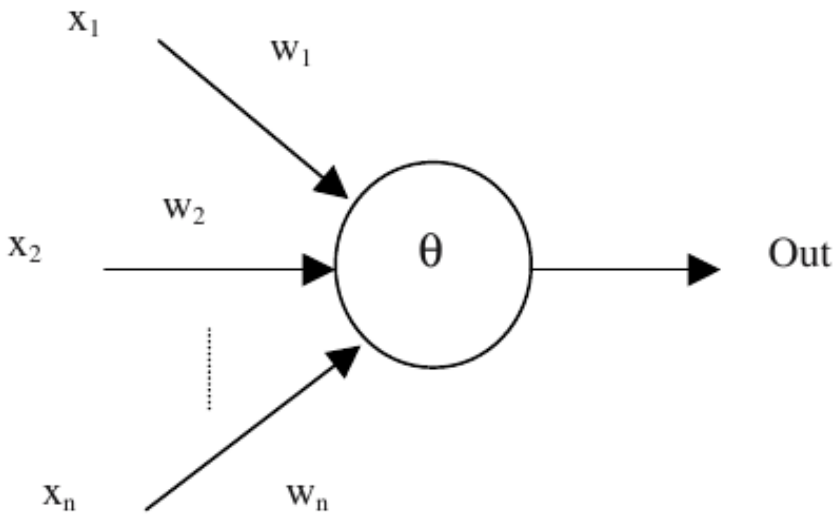
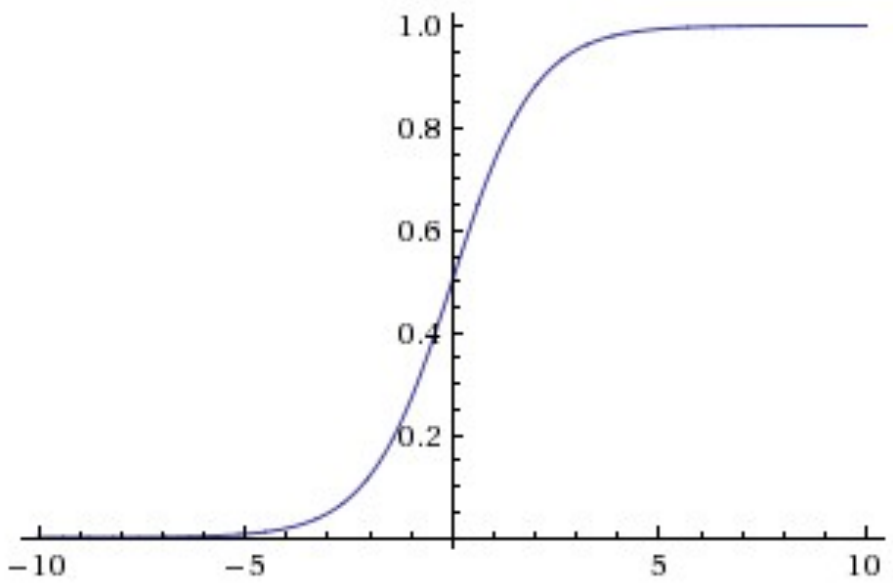


Fig. 8.2: A symbolic representation of Fig. 8.1.

Funções de Ativação

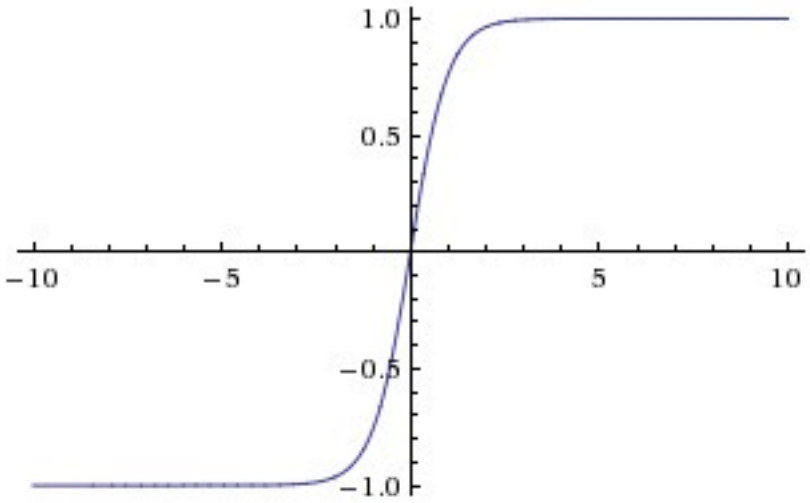
- Sigmoid: $\sigma(x) = 1/(1 + e^{-x})$



- Contras: não é alinhada em zero | gradientes próximos de regiões próximas a 0 e 1 são quase zero | pouco usada

Funções de Ativação

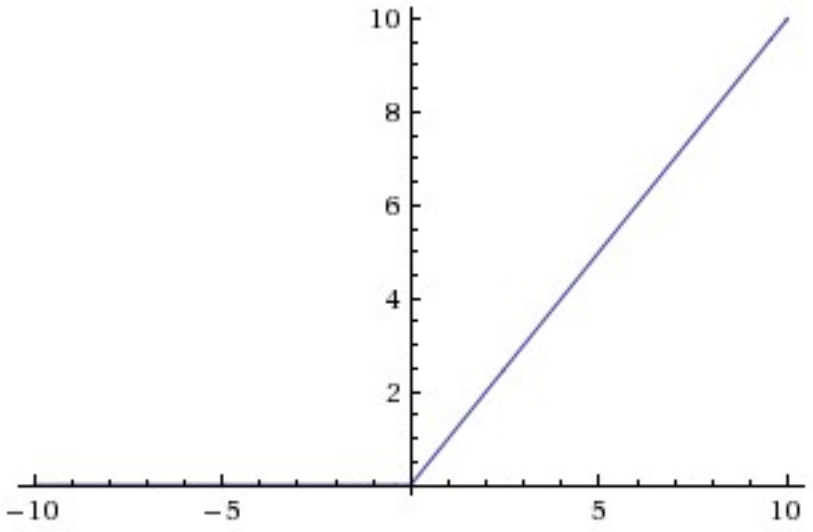
- Tanh: $\tanh(x) = 2\sigma(2x) - 1$



- Diferencia de modo contínuo | centrada em zero | é um deslocamento da Sigmoid

Funções de Ativação

- ReLU. The Rectified Linear Unit $f(x) = \max(0, x)$.



- Rápida | mas neurônios podem morrer com o valor “0” | sol: adiciona “folga” (0,01)

Tipos de Redes Neurais

- Uma camada (também chamada de Perceptron)
- Consistem em um simples neurônio, com ponderações ajustáveis de sinapses e um limitador fixo

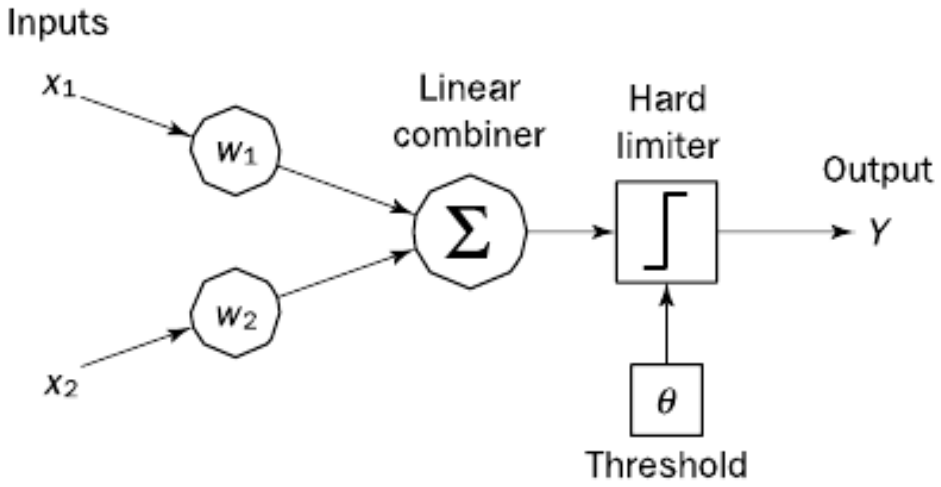
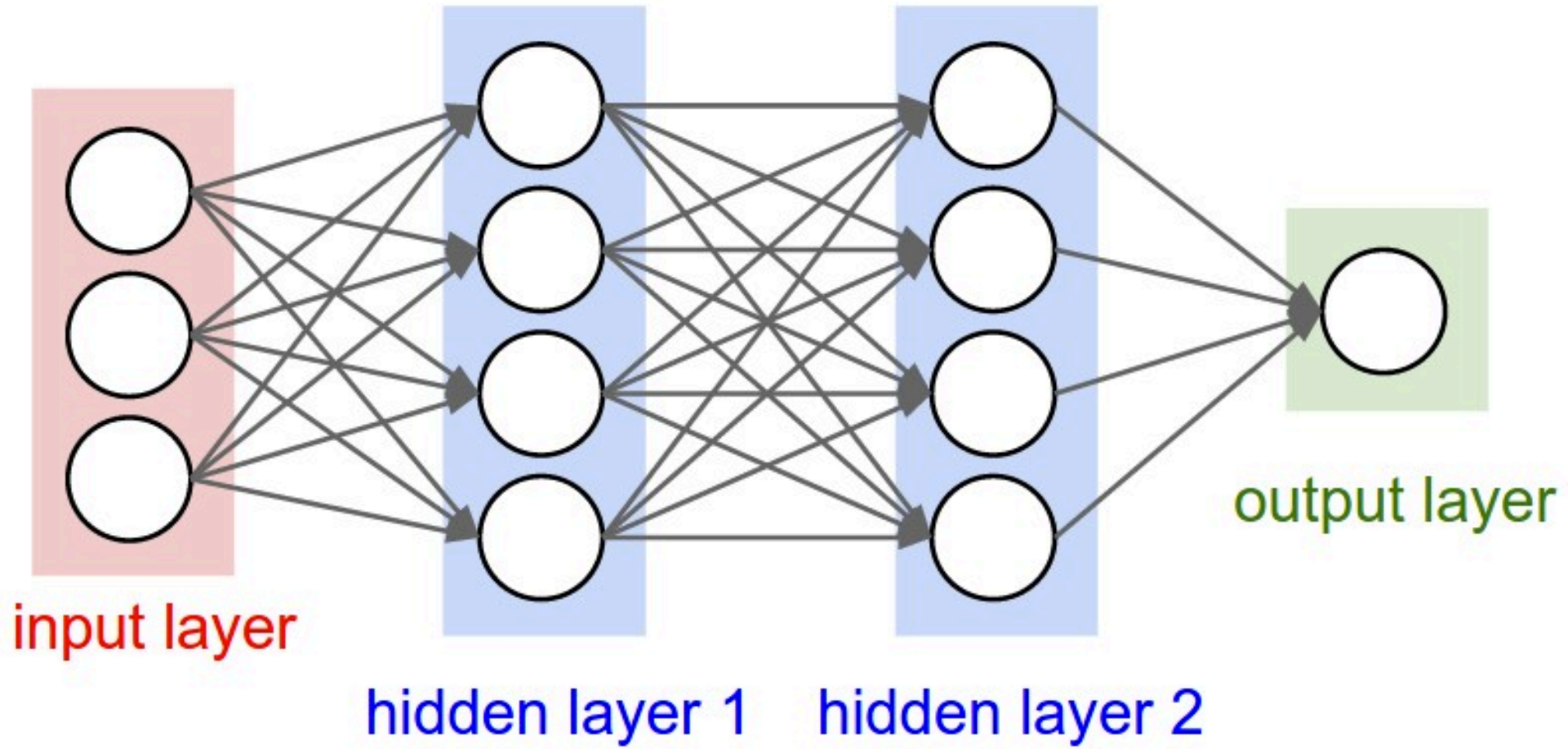


Figure 6.5 Single-layer two-input perceptron

Várias Camadas

- Uma rede neural Multilayer perceptron:
  - Contém uma ou mais camadas escondidas
- Consiste basicamente:
  - 1 camada de entrada
  - 1 ou mais escondidas
  - 1 camada de saída
- O sinal é propagado de uma camada para outra (forward)

Exemplo 2 camadas escondidas

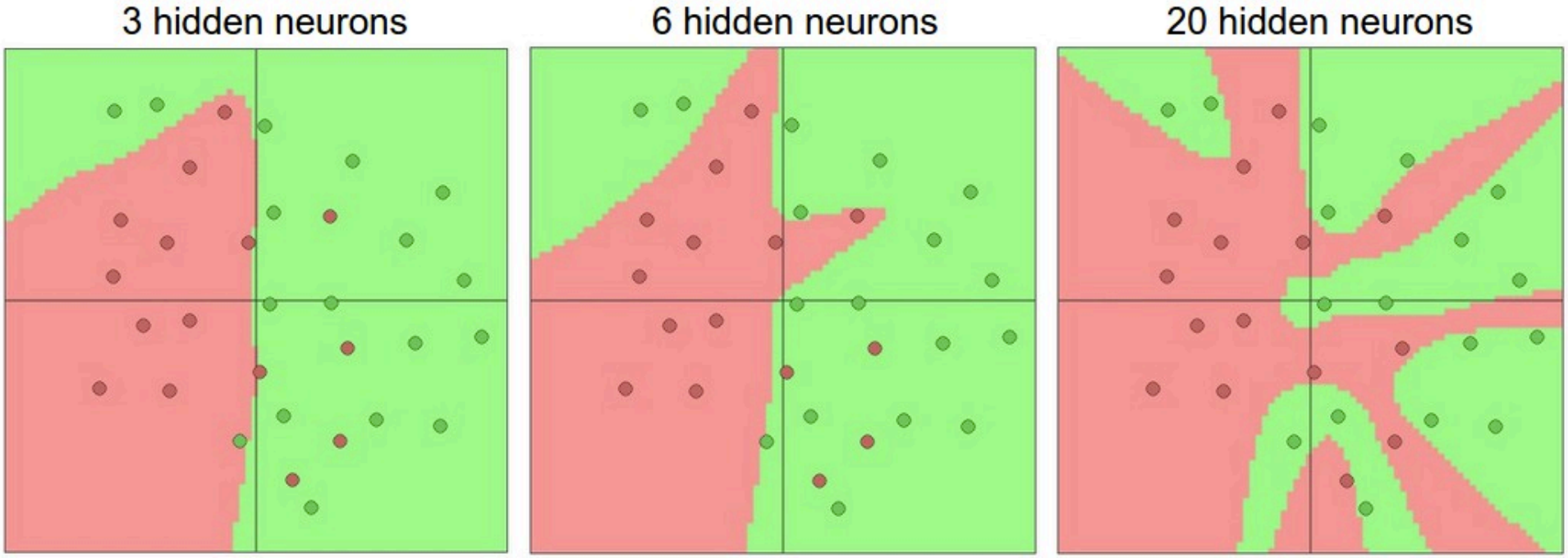


Camadas Escondidas

- Cada camada tem sua funcionalidade específica
  - Os neurônios dessas camadas detectam características presentes nos padrões que não eram detectadas por fora

- Com uma camada escodida é possível representar qualquer função contínua
- Com duas é possível representar funções contínuas e descontínuas
- A camada funciona como uma caixa-preta (**hidden**)

Poder de Representação



Como fica o OverFitting?

Aprendizado em Redes Neurais

Arquitetura (Hyperparameters)

- Definida a teste:
  - uma única camada
  - várias camadas
  - quantos neurônios em cada camada
  - interligação entre os neurônios nas várias camadas, etc

- Depois disso, se escolhe o algoritmo de aprendizagem (vai efetivamente diferenciar as redes neurais)

Aprendizado Supervisionado

- O algoritmo ajusta o erro da rede, e tenta em uma nova iteração diminuir este erro:
  - O algoritmo de aprendizagem Perceptron (regra Delta)
  - O método do menor error quadrado (Least Mean Square)
  - O algoritmo Backpropagation

O Algoritmo Perceptron

- A operação do perceptron de Rosenblatt (Rosenblatt, 1960) é baseada no modelo de neurônio de Mc Culloch and Pitts.

- Consiste em um combinador linear seguido por um hard limiter.
  - A soma ponderada das entradas é aplicada ao hard limiter, que produz uma saída igual a +1 se sua entrada for positiva e -1 se for negativa.

Esquema

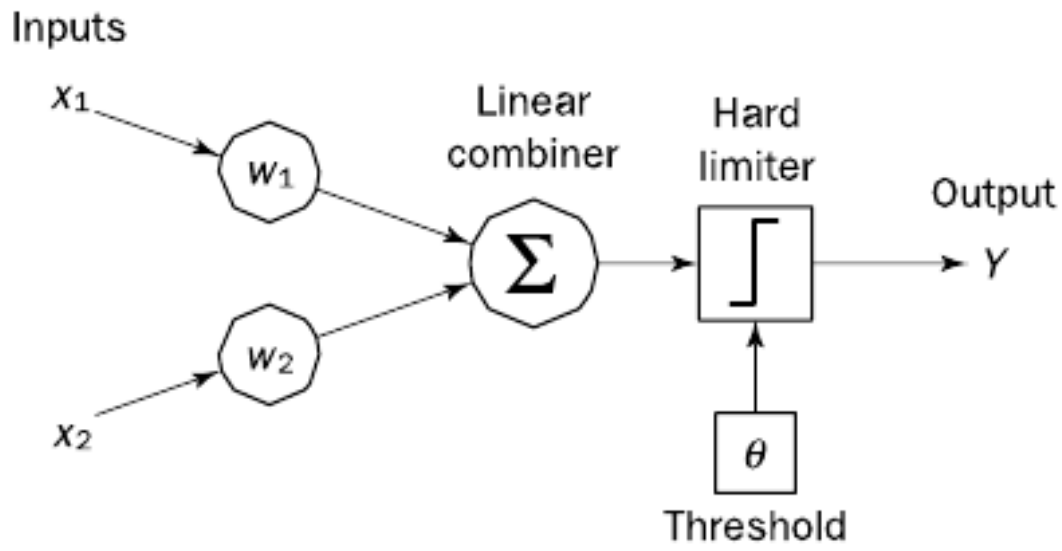


Figure 6.5 Single-layer two-input perceptron

O Algoritmo Perceptron

- Em outras palavras, o neurônio de Rosenblatt usa a seguinte **função de transferência** ou **função de ativação**:

$$X = \sum_{i=1}^{n} x_i w_i$$

$$Y = \begin{cases} +1 & \text{if } X \geqslant \theta \\ -1 & \text{if } X < \theta \end{cases}$$

O Algoritmo Perceptron

- A função de ativação utilizada é chamada de **sign** (função sinal).
- Assim a saída do neurônio com a função de ativação **sign** pode ser representada como

$$Y = sign\left[\sum_{i=1}^{n} x_i w_i - \theta\right]$$

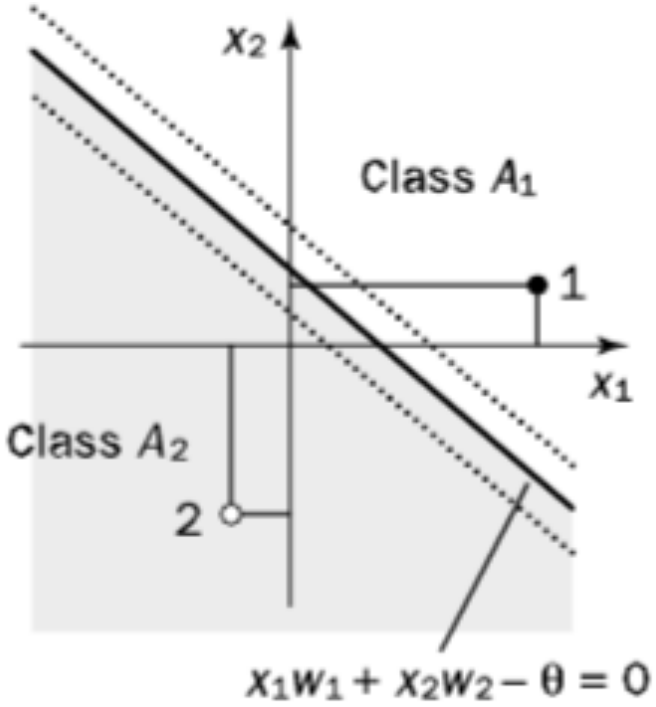
Perceptrons como classificadores

- O objetivo do perceptron é classificar entradas, ou em outras palavras classificar estímulos externos x_1, x_2, . . . x_n em uma de duas classes A1 e A2.
- O espaço n - dimensional é dividido por um **hyperplano** em duas regiões de decisão. O hyperplano é definido pela função linearmente separável

$$\sum_{i=1}^{n} x_i w_i - \theta = 0$$

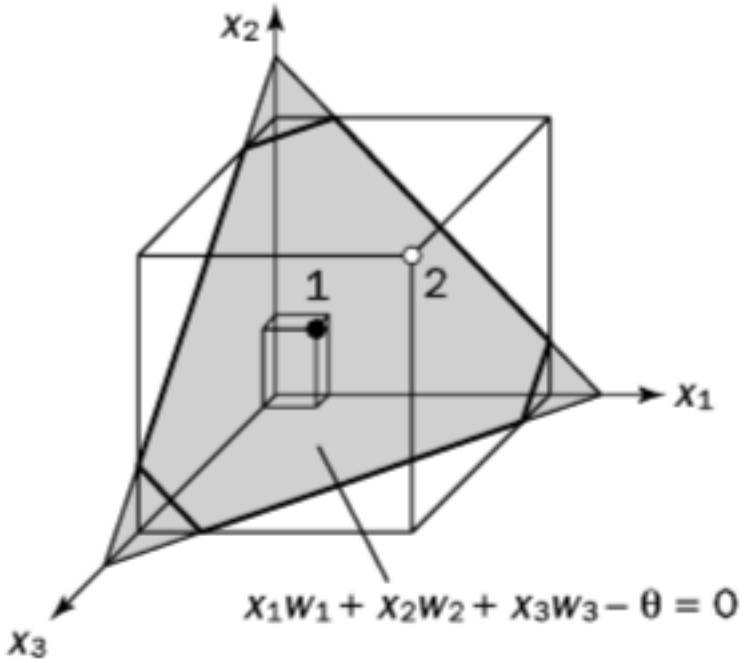
Superfície de Decisão

- Para o caso de duas entradas x_1 and x_2, a fronteira de decisão toma a forma de uma linha reta mostrada em negrito:



Superfície de Decisão

- Com três entradas o hyperplano pode ainda ser visualizado graficamente
- O plano de separação entre as classes A1 e A2 é definido pela equação



Mas como o perceptron aprende?

- Realizando ajustes na ponderação, partindo de uma solução incial aleatória
- Supondo que a função de classificação seja:

$$x_1 . w_1 + x_2 . W_2 - \theta = 0$$

sendo w_i [-0.5, 0.5]

O Erro na iteração p é calculada como erro em relação a classificação

Erro

Se

- Y(p) for a saída calculada
- $Y_d(p)$ a saída correta (desired)

então:

- $e(p) = Y_d(p) – Y(p)$

onde p é a iteração (geração)

- Se erro é positivo então precisamos incrementar a saída do perceptron Y(p)
- Se negativo, decrementar

Regra Delta

- Assim, a seguinte **regra de aprendizagem do perceptron** ou **regra Delta** pode ser estabelecida:

$$w_i(p+1) = w_i(p) + \alpha \, . \, x_i(p) \, . \, e(p)$$

- onde α é a **taxa de aprendizagem**, uma constante positiva menor do que 1.

Algoritmo

Step 1: ***Initialisation***

Set initial weights $w_1, w_2, \ldots, w_n$ and threshold θ to random numbers in the range $[-0.5, 0.5]$.

Step 2: ***Activation***

Activate the perceptron by applying inputs $x_1(p), x_2(p), \ldots, x_n(p)$ and desired output $Y_d(p)$. Calculate the actual output at iteration $p = 1$

$$Y(p) = step\left[\sum_{i=1}^{n} x_i(p)w_i(p) - \theta\right], \tag{6.6}$$

where n is the number of the perceptron inputs, and *step* is a step activation function.

Algoritmo

Step 3: ***Weight training***

Update the weights of the perceptron

$$w_i(p+1) = w_i(p) + \Delta w_i(p), \tag{6.7}$$

where $\Delta w_i(p)$ is the weight correction at iteration p.

The weight correction is computed by the **delta rule**:

$$\Delta w_i(p) = \alpha \times x_i(p) \times e(p) \tag{6.8}$$

Step 4: ***Iteration***

Increase iteration p by one, go back to Step 2 and repeat the process until convergence.

Backpropagation e MLP

Introdução

- Redes de Perceptrons
  - Resolvem apenas problemas lineares
  - Não existem muitos deles

- Solução
  - Redes de múltiplas camadas
  - Backpropagation

Feedforward MLP

- Rede Neural FeedFoward Multicamadas
  - Uma ou mais camadas escondidas
  - Sendo uma de entrada
    - Processar a entrada e distribuir o sinal
  - Uma de saída
    - Gerar um saída
  - E pelo menos uma intermediária

Representação

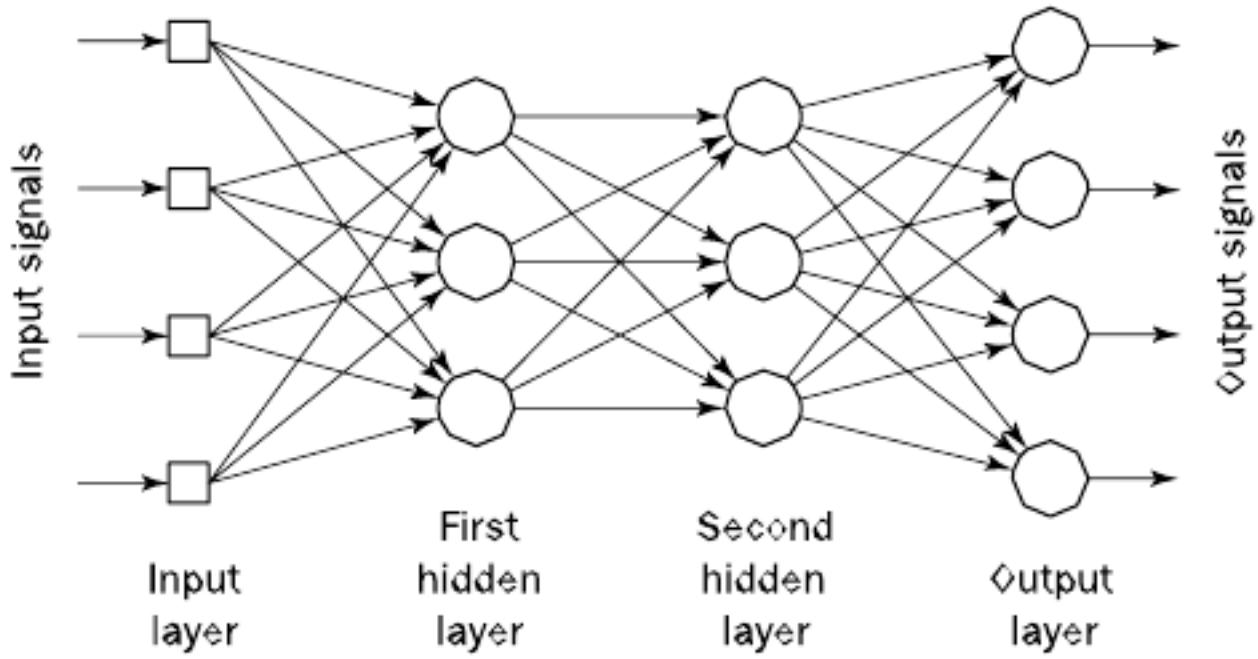


Figure 6.8 Multilayer perceptron with two hidden layers

Porque mais de uma camada?

- Neurônios na(s) camada(s) escondida(s) detecta(m) as características;
  - os pesos dos neurônios representam as características escondidas nos padrões de entrada.
  - Estas características são então usadas pela camada de saída na determinação do padrão de saída.

Como aprende?

- O algoritmo mais popular é o **backpropagation**

- O processo é o mesmo:
  - Um conjunto de entrada é apresentado
  - É calculado a saída
  - Se houver erro é realizado o ajuste dos pesos para diminuir o erro

Diferença

- Com apenas um perceptron
  - Existe apenas 1 conjunto de pesos
  - E apenas 1 saída

- **Em multicamadas**
  - **Existem vários pesos aplicados a mesma entrada**
  - **E várias saídas**

Avaliação de Erro

- Em uma rede neural backpropagation, o algoritmo de aprendizagem tem duas fases.
  1. um padrão de entrada de treinamento é apresentado a camada de entrada da rede.

A rede então propaga (**feedfowards**) o padrão de entrada de camada a camada até o padrão de saída ser gerado pela camada de saída.

Avaliação de Erro

2. Se este padrão é diferente da saída desejada, um erro é calculado e então propagado de volta (**backpropagated**) através da rede partindo da camada de saída para a camada de entrada.

Os pesos são modificados a medida que o erro é propagado.

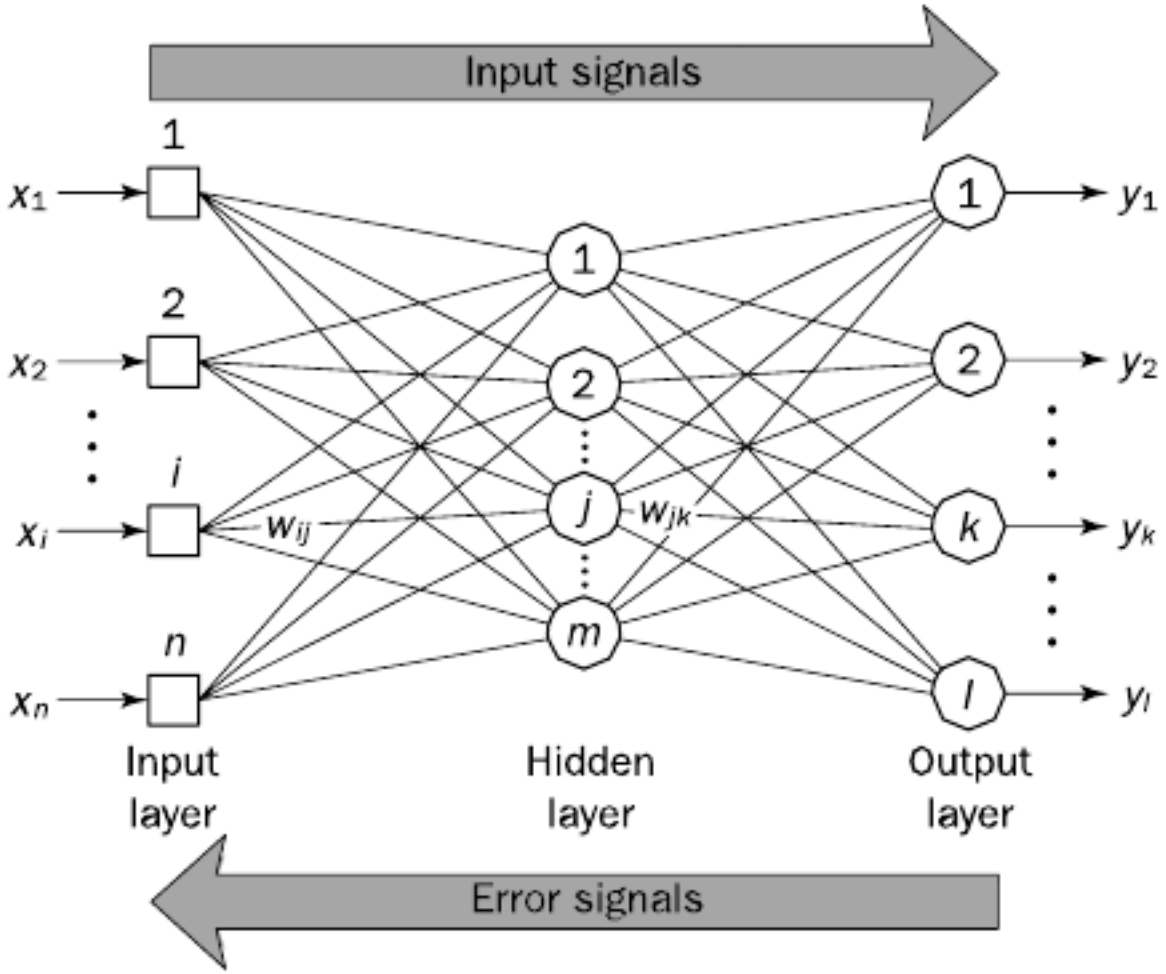


Figure 6.9 Three-layer back-propagation neural network

Rede Backpropagation

- Como qualquer outra rede neural, uma rede backpropagation é determinada por:
  - as **conexões entre neurônios** (a arquitetura da rede),
  - a **função de ativação** utilizada pelos neurônios, e
  - o **algoritmo de aprendizagem** (ou a lei de aprendizagem) que especifica o procedimento para o ajuste de pesos.

Arquitetura da Rede

- Tipicamente, uma rede backpropagation é uma rede multicamadas que tem três ou quatro camadas.

- As camadas são completamente conectadas (**Full Connected (FC)**):
  - isto é, cada neurônio em cada camada está conectado a cada outro neurônio na camada adjacente.

Função de Ativação

- Primeiro, ele computa soma ponderada das entradas.
- Em seguida, a soma é passada através de uma função de ativação.
- Função de ativação para fins didáticos **sigmoid**:

$$Y^{sigmoid} = \frac{1}{1 + e^{-X}}$$

Lei de Aprendizagem

- Considerarando uma rede de três camadas

sendo i, j e k referem-se aos neurônios nas camadas de **entrada**, **escondida** e de **saída**, respectivamente.

Sinais de **entrada**, $x_1, x_2, \ldots, x_n$, são propagados através da rede da **esquerda para a direita**, e sinais de **erros**, $e_1, e_2, \ldots, e_l$, da **direita para a esquerda**.

Propagação

- Para propagar sinais de erro:
  - começamos na camada de saída e trabalhamos no sentido de volta para a camada escondida.
  - O sinal de erro na saída do neurônio k na iteração p é definido por:

$$e_k(p) = y_{d,k}(p) - y_k(p),$$

onde $\mathbf{y_{d,k}(\ p\ )}$ é a saída desejada do neurônio k na iteração p.

- Para camada de saída!

Correção dos pesos

- Regra semelhante

$$w_{jk}(p+1) = w_{jk}(p) + \Delta w_{jk}(p),$$

- A entrada não é mesma na camada y
- Assim, nós usamos a saída do neurônio j na camada escondida, y_j, ao invés da entrada x_i.

Atualização

$$\Delta w_{jk}(p) = \alpha \times y_j(p) \times \delta_k(p),$$

- onde δ_k (p) é o gradiente de erro no neurônio k da camada de saída na iteração p.

Gradiente do erro

- O gradiente de erro é determinado como a derivada da função de ativação multiplicado pelo erro no neurônio de saída.

$$\delta_k(p) = \frac{\partial y_k(p)}{\partial X_k(p)} \times e_k(p),$$

- onde y_k (p) é a saída do neurônio k na iteração p, e X_k (p) é a soma ponderada das entradas do neurônio k na mesma iteração.

Para a função sigmoid

- O gradiente do erro aplicada a função de ativação sigmoid:

$$\delta_k(p) = y_k(p) \times [1 - y_k(p)] \times e_k(p),$$

Peso nas camadas escondidas

$$\Delta w_{ij}(p) = \alpha \times x_i(p) \times \delta_j(p),$$

onde δ_j (p) representa o gradiente de erro do neurônio j na camada escondida:

$$\delta_j(p) = y_j(p) \times [1 - y_j(p)] \times \sum_{k=1}^{l} \delta_k(p) w_{jk}(p),$$

onde i é o número de neurônios na camada de saída;

Peso nas camadas escondidas

$$\delta_j(p) = y_j(p) \times [1 - y_j(p)] \times \sum_{k=1}^{l} \delta_k(p) w_{jk}(p),$$

$$y_j(p) = \frac{1}{1 + e^{-X_j(p)}};$$

$$X_j(p) = \sum_{i=1}^{n} x_i(p) \times w_{ij}(p) - \theta_j;$$

Algoritmo – Passo 1

Step 1: ***Initialisation***

Set all the weights and threshold levels of the network to random numbers uniformly distributed inside a small range (Haykin, 1999):

$$\left(-\frac{2.4}{F_i}, +\frac{2.4}{F_i}\right),$$

where F_i is the total number of inputs of neuron i in the network. The weight initialisation is done on a neuron-by-neuron basis.

Algoritmo – Passo 2(a)

Step 2: ***Activation***

Activate the back-propagation neural network by applying inputs $x_1(p), x_2(p), \ldots, x_n(p)$ and desired outputs $y_{d,1}(p), y_{d,2}(p), \ldots, y_{d,n}(p)$.

(a) Calculate the actual outputs of the neurons in the hidden layer:

$$y_j(p) = sigmoid\left[\sum_{i=1}^{n} x_i(p) \times w_{ij}(p) - \theta_j\right],$$

where n is the number of inputs of neuron j in the hidden layer, and *sigmoid* is the sigmoid activation function.

Algoritmo - Passo 2 (b)

(b) Calculate the actual outputs of the neurons in the output layer:

$$y_k(p) = sigmoid\left[\sum_{j=1}^{m} x_{jk}(p) \times w_{jk}(p) - \theta_k\right],$$

where m is the number of inputs of neuron k in the output layer.

Algoritmo – Passo 3 (a)

Step 3: ***Weight training***

Update the weights in the back-propagation network propagating backward the errors associated with output neurons.

(a) Calculate the error gradient for the neurons in the output layer:

$$\delta_k(p) = y_k(p) \times [1 - y_k(p)] \times e_k(p)$$

where

$$e_k(p) = y_{d,k}(p) - y_k(p)$$

Calculate the weight corrections:

$$\Delta w_{jk}(p) = \alpha \times y_j(p) \times \delta_k(p)$$

Update the weights at the output neurons:

$$w_{jk}(p+1) = w_{jk}(p) + \Delta w_{jk}(p)$$

Algoritmo- Passo 3 (b)

(b) Calculate the error gradient for the neurons in the hidden layer:

$$\delta_j(p) = y_j(p) \times [1 - y_j(p)] \times \sum_{k=1}^{l} \delta_k(p) \times w_{jk}(p)$$

Calculate the weight corrections:

$$\Delta w_{ij}(p) = \alpha \times x_i(p) \times \delta_j(p)$$

Update the weights at the hidden neurons:

$$w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)$$

Algoritmo – Passo 4

Step 4: ***Iteration***

Increase iteration p by one, go back to Step 2 and repeat the process until the selected error criterion is satisfied.

Exemplo - XOR

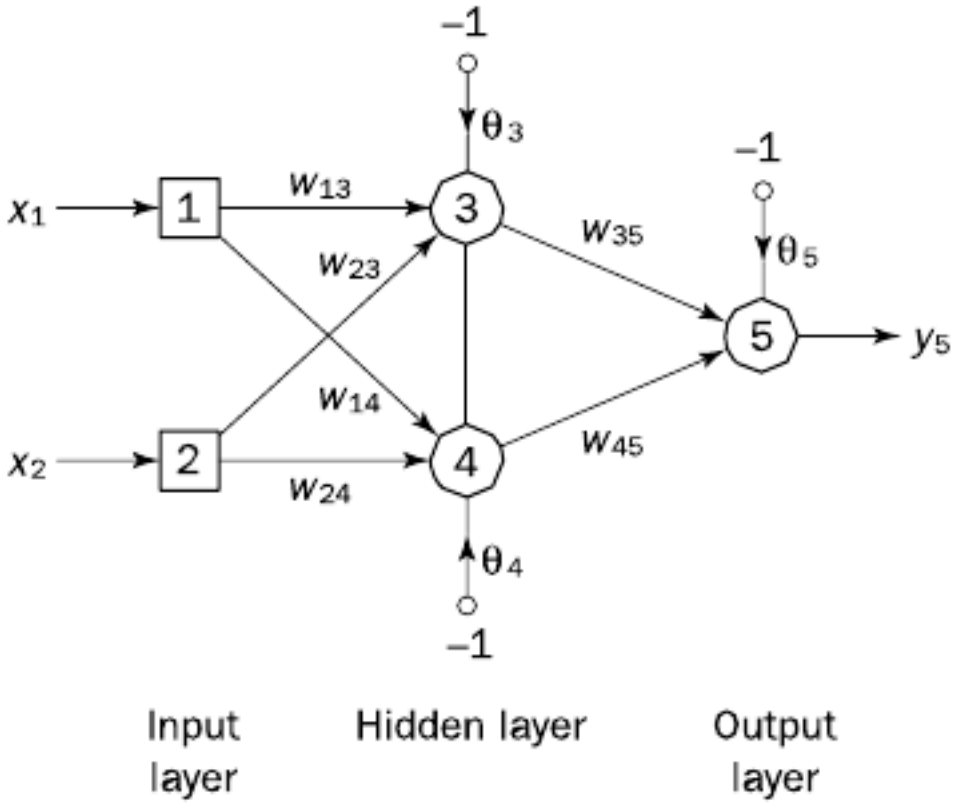


Figure 6.10 Three-layer network for solving the Exclusive-OR operation

Passo 1: Inicialização

- $w_{13} = 0.5$
- $w_{14} = 0.9$
- $w_{23} = 0.4$
- $w_{24} = 1.0$
- $w_{35} = -1.2$
- $w_{45} = 1.1$
- $\theta_3 = 0.8$
- $\theta_4 = -0.1$
- $\theta_5 = 0.3$

Passo 2: Ativação

- Cosidere as entradas $x_1 = x_2 = 1$; neste caso a saída desejada é $y_{d,5} = 0$.
- As saídas atuais dos neurônios 3 e 4 na camada escondida são calculados como:

$$y_3 = sigmoid\,(x_1 w_{13} + x_2 w_{23} - \theta_3) = 1/[1 + e^{-(1\times 0.5 + 1\times 0.4 - 1\times 0.8)}] = 0.5250$$

$$y_4 = sigmoid\,(x_1 w_{14} + x_2 w_{24} - \theta_4) = 1/[1 + e^{-(1\times 0.9 + 1\times 1.0 + 1\times 0.1)}] = 0.8808$$

Passo 2: Ativação

- Assim a saída atual do neurônio 5 na camada de saída pode ser determinado por

$$y_5 = sigmoid\,(y_3 w_{35} + y_4 w_{45} - \theta_5)$$

$$= 1/[1 + e^{-(-0.5250 \times 1.2 + 0.8808 \times 1.1 - 1 \times 0.3)}] = 0.5097$$

Erro saída

- onde, o seguinte erro é obtido:

$e = y_{d,5} - y_5 = 0 - 0.5097 = -0.5097$

Passo 3: Atualização de Pesos

- Para atualizar os pesos da rede, o erro é propagado de volta da camada de saída para a camada de entrada.

- Primeiro, calcula-se o gradiente de erro para o neurônio 5 na camada de saída:

$$\delta_5 = y_5(1 - y_5)e = 0.5097 \times (1 - 0.5097) \times (-0.5097) = -0.1274$$

Passo 3

- Em seguida, determinam-se as correções de pesos (no exemplo, assume-se que a taxa de aprendizagem α = 0.1):

$$\Delta w_{35} = \alpha \times y_3 \times \delta_5 = 0.1 \times 0.5250 \times (-0.1274) = -0.0067$$

$$\Delta w_{45} = \alpha \times y_4 \times \delta_5 = 0.1 \times 0.8808 \times (-0.1274) = -0.0112$$

$$\Delta \theta_5 = \alpha \times (-1) \times \delta_5 = 0.1 \times (-1) \times (-0.1274) = 0.0127$$

Passo 3

- Depois, calculam-se os gradientes de erro para os neurônios 3 e 4 na camada escondida:

$$\delta_3 = y_3(1 - y_3) \times \delta_5 \times w_{35} =$$

$$0.5250 \times (1 - 0.5250) \times (-0.1274) \times (-1.2) = 0.0381$$

$$\delta_4 = y_4(1 - y_4) \times \delta_5 \times w_{45} =$$

$$0.8808 \times (1 - 0.8808) \times (-0.1274) \times 1.1 = -0.0147$$

Passo 3

- Depois, determinam-se as correções de pesos:

$$\Delta w_{13} = \alpha \times x_1 \times \delta_3 = 0.1 \times 1 \times 0.0381 = 0.0038$$

$$\Delta w_{23} = \alpha \times x_2 \times \delta_3 = 0.1 \times 1 \times 0.0381 = 0.0038$$

$$\Delta \theta_3 = \alpha \times (-1) \times \delta_3 = 0.1 \times (-1) \times 0.0381 = -0.0038$$

$$\Delta w_{14} = \alpha \times x_1 \times \delta_4 = 0.1 \times 1 \times (-0.0147) = -0.0015$$

$$\Delta w_{24} = \alpha \times x_2 \times \delta_4 = 0.1 \times 1 \times (-0.0147) = -0.0015$$

$$\Delta \theta_4 = \alpha \times (-1) \times \delta_4 = 0.1 \times (-1) \times (-0.0147) = 0.0015$$

Passo 3

- E por fim, atualizam-se todos os pesos da rede:

$$w_{13} = w_{13} + \Delta w_{13} = 0.5 + 0.0038 = 0.5038$$

$$w_{14} = w_{14} + \Delta w_{14} = 0.9 - 0.0015 = 0.8985$$

$$w_{23} = w_{23} + \Delta w_{23} = 0.4 + 0.0038 = 0.4038$$

$$w_{24} = w_{24} + \Delta w_{24} = 1.0 - 0.0015 = 0.9985$$

$$w_{35} = w_{35} + \Delta w_{35} = -1.2 - 0.0067 = -1.2067$$

$$w_{45} = w_{45} + \Delta w_{45} = 1.1 - 0.0112 = 1.0888$$

$$\theta_3 = \theta_3 + \Delta \theta_3 = 0.8 - 0.0038 = 0.7962$$

$$\theta_4 = \theta_4 + \Delta \theta_4 = -0.1 + 0.0015 = -0.0985$$

$$\theta_5 = \theta_5 + \Delta \theta_5 = 0.3 + 0.0127 = 0.3127$$

Critério de Parada

- O processo de treinamento é repetido até que **a soma dos erros quadráticos** seja menor que uma margem de erro pequena pré-estabelecida (e.g., 0.001).

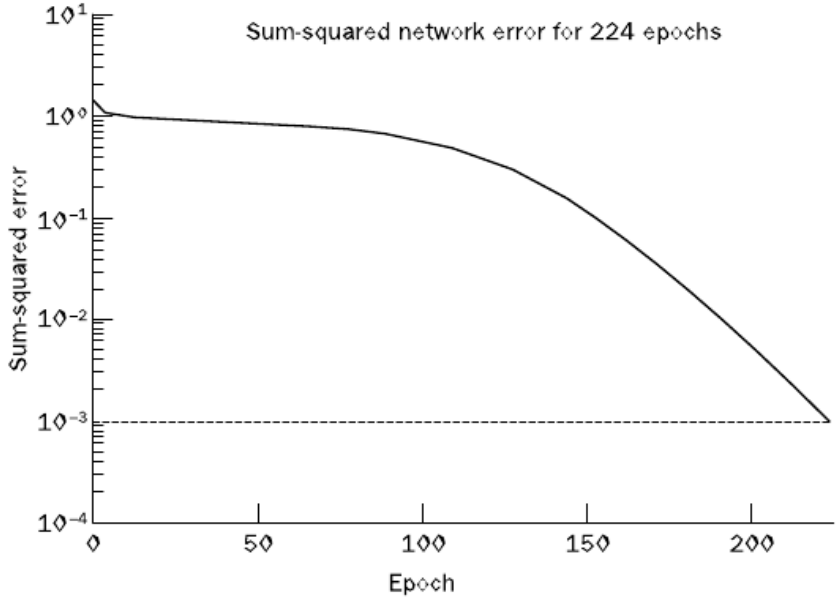


Figure 6.11 Learning curve for operation Exclusive-OR

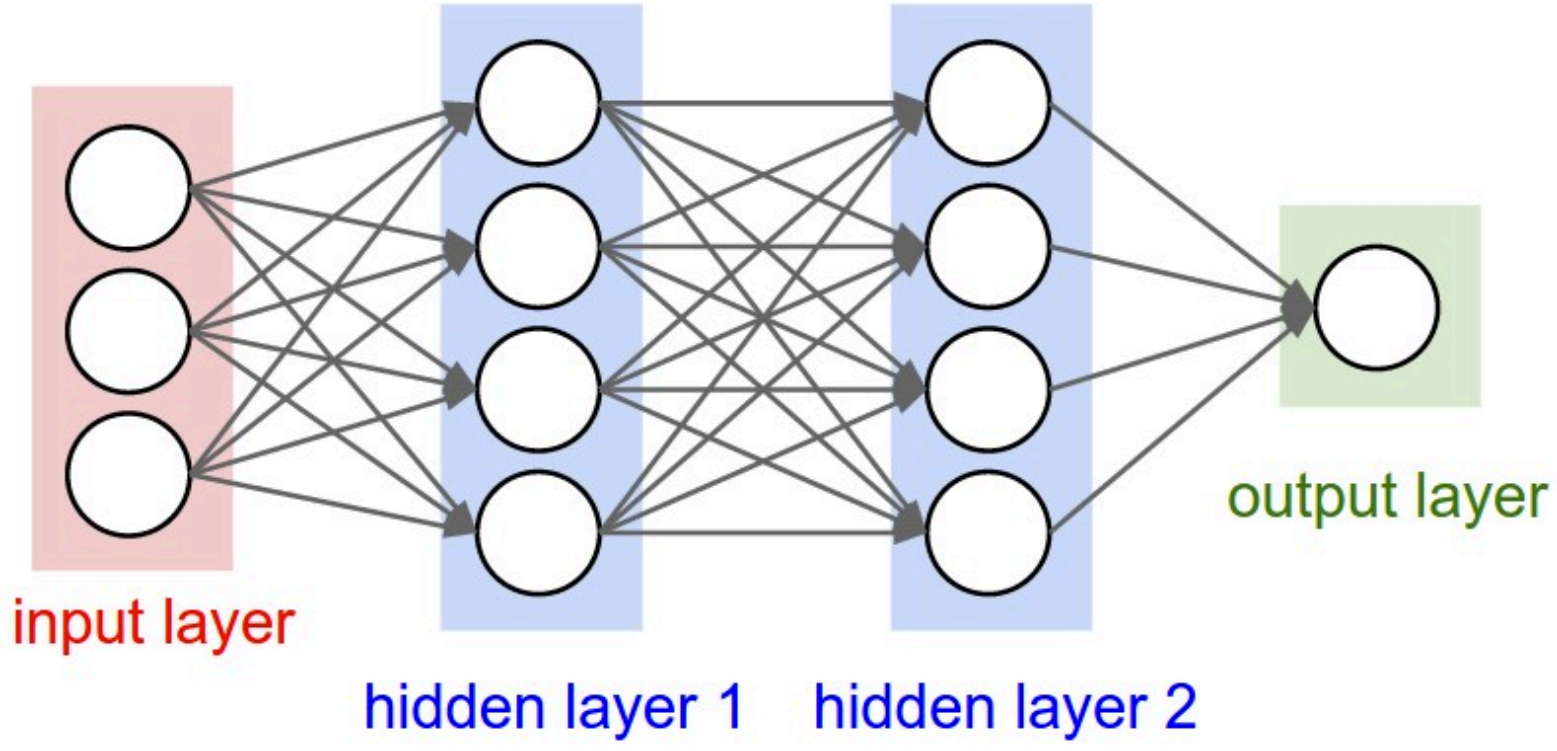
Convolutional Neural Networks - CNN

Visão Computacional

Prof. Geraldo Braz Junior

Motivação

- Imagine uma MLP
  - Múltiplas camadas



Motivação

- Conecte como entrada uma **imagem** de 32x32x3 (RGB)
- Quantos pesos necessitam ser ajustados nessa rede?
  - 3072 (só na camada de entrada)
  - A coisa piora quando a imagem cresce

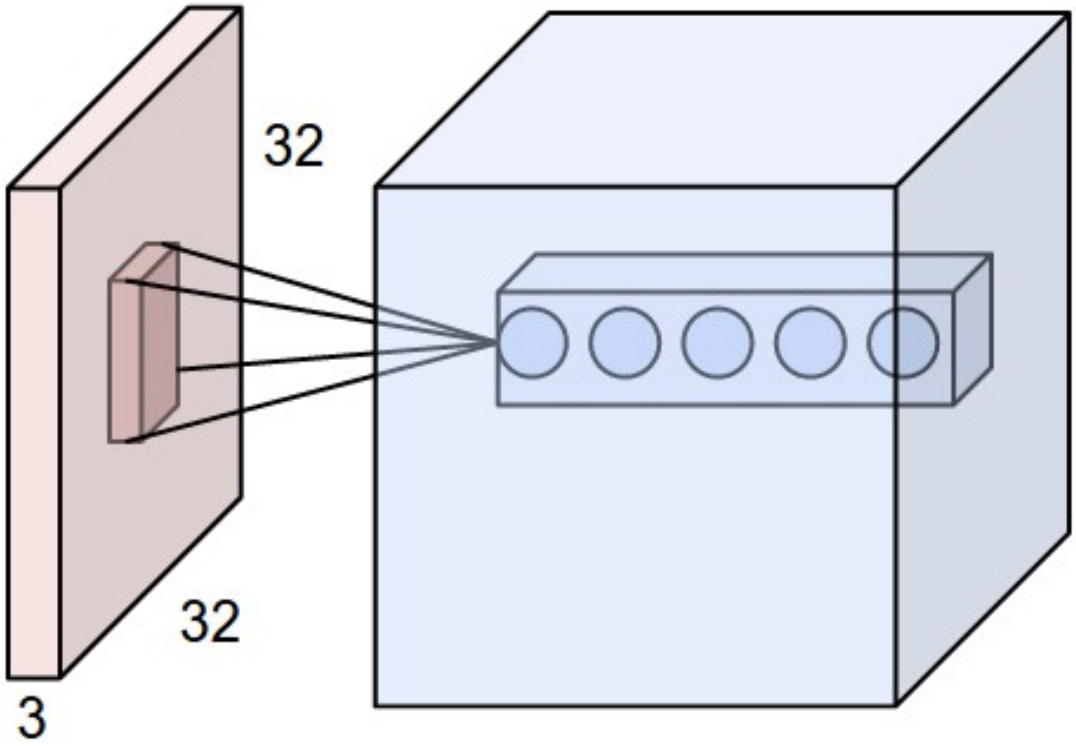
- **Pense sobre isso….**
  - **quais as implicações? (tempo, generalização …)**

Ideia

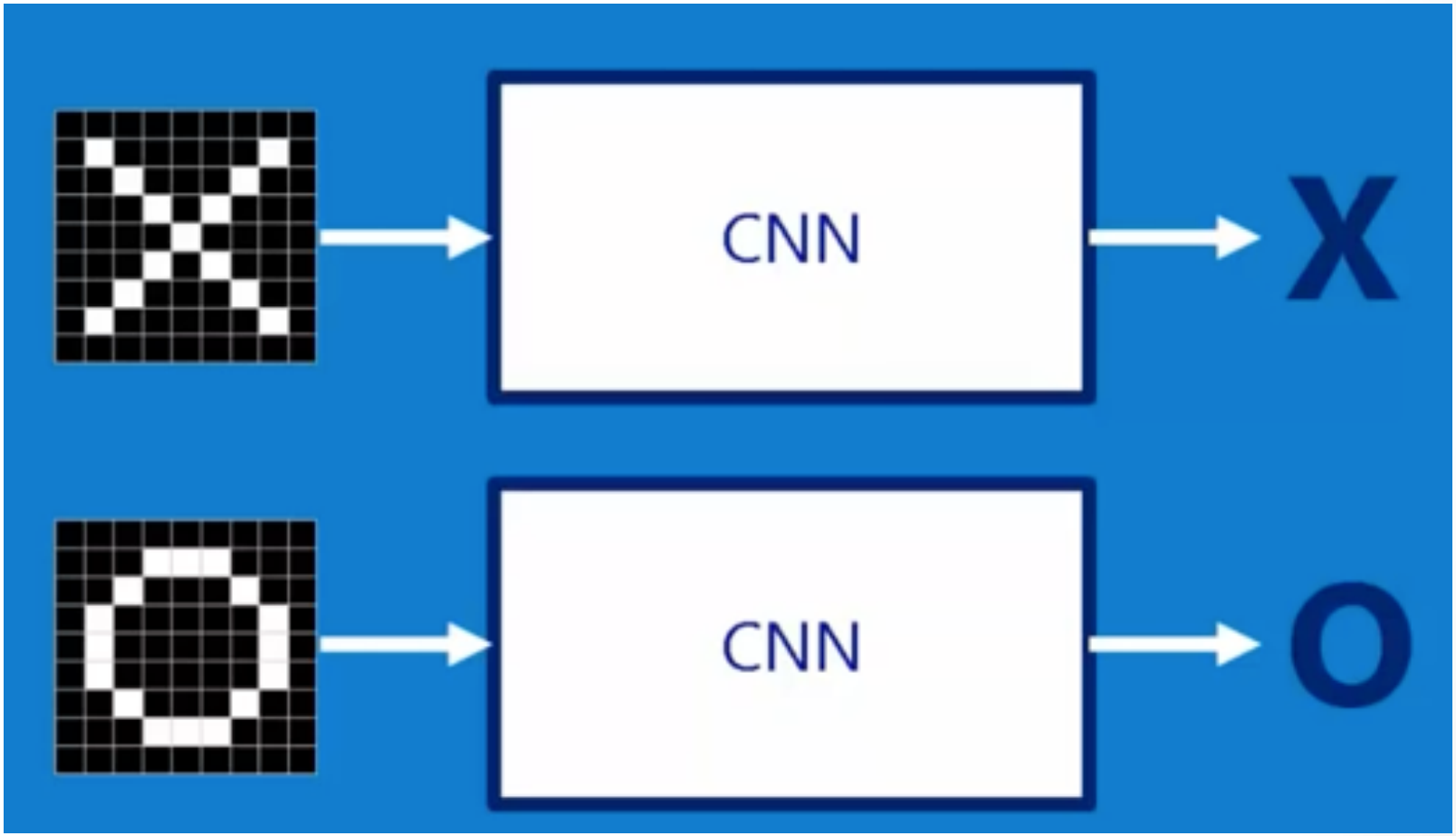
- Sabendo que se trata de uma imagem, podemos assumir que:
  - Existe um **relacionamento espacial** entre pixels próximos
  - Operações realizadas em um canal da imagem, podem ser realizadas em paralelo para todos os canais (o peso poderia ser único)

Ideia

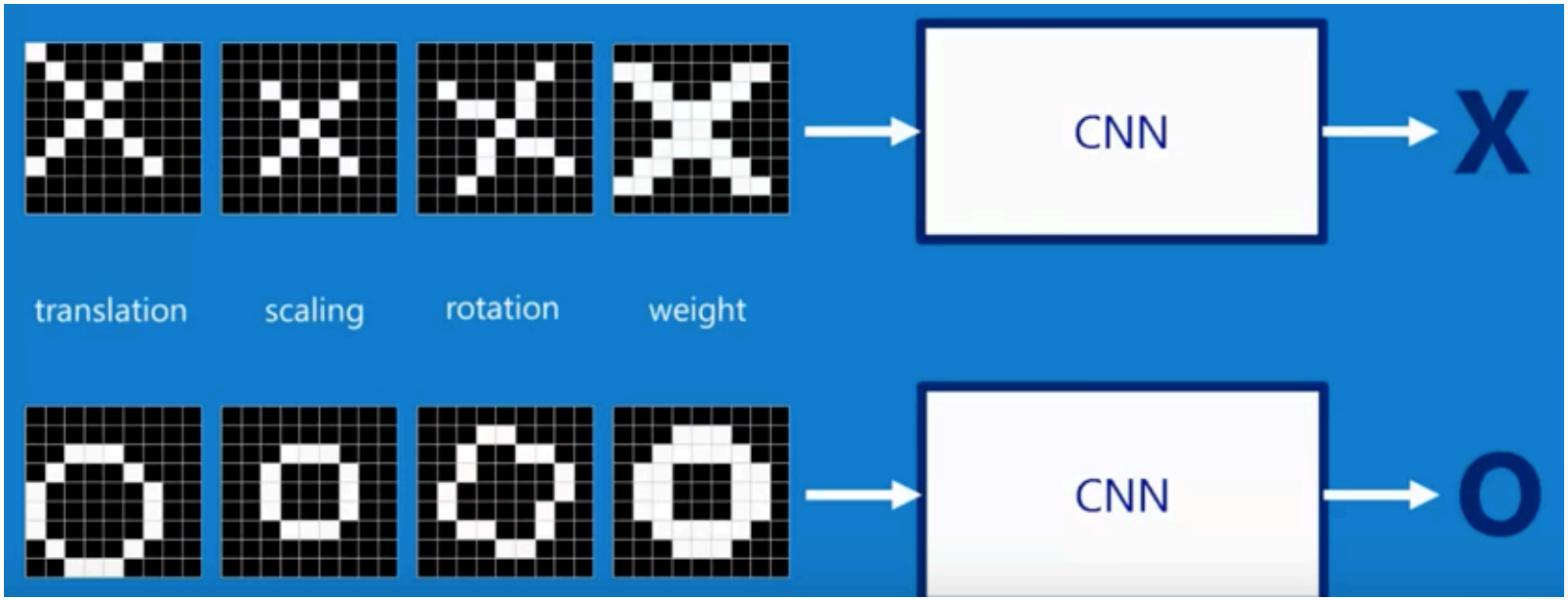
- Assim as **ativações** na imagem podem ser **localizadas** a uma região, em busca de **features** que definam a região



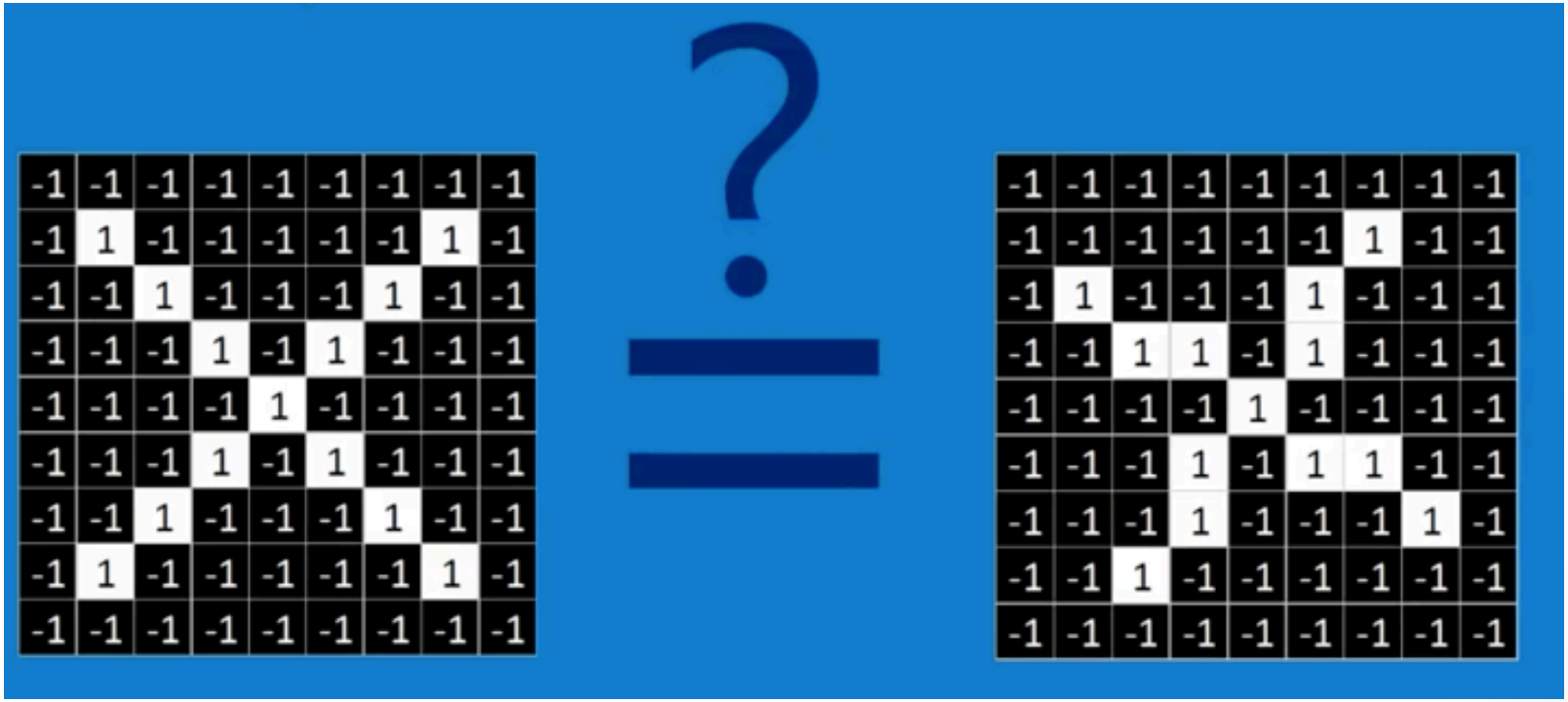
Ideia



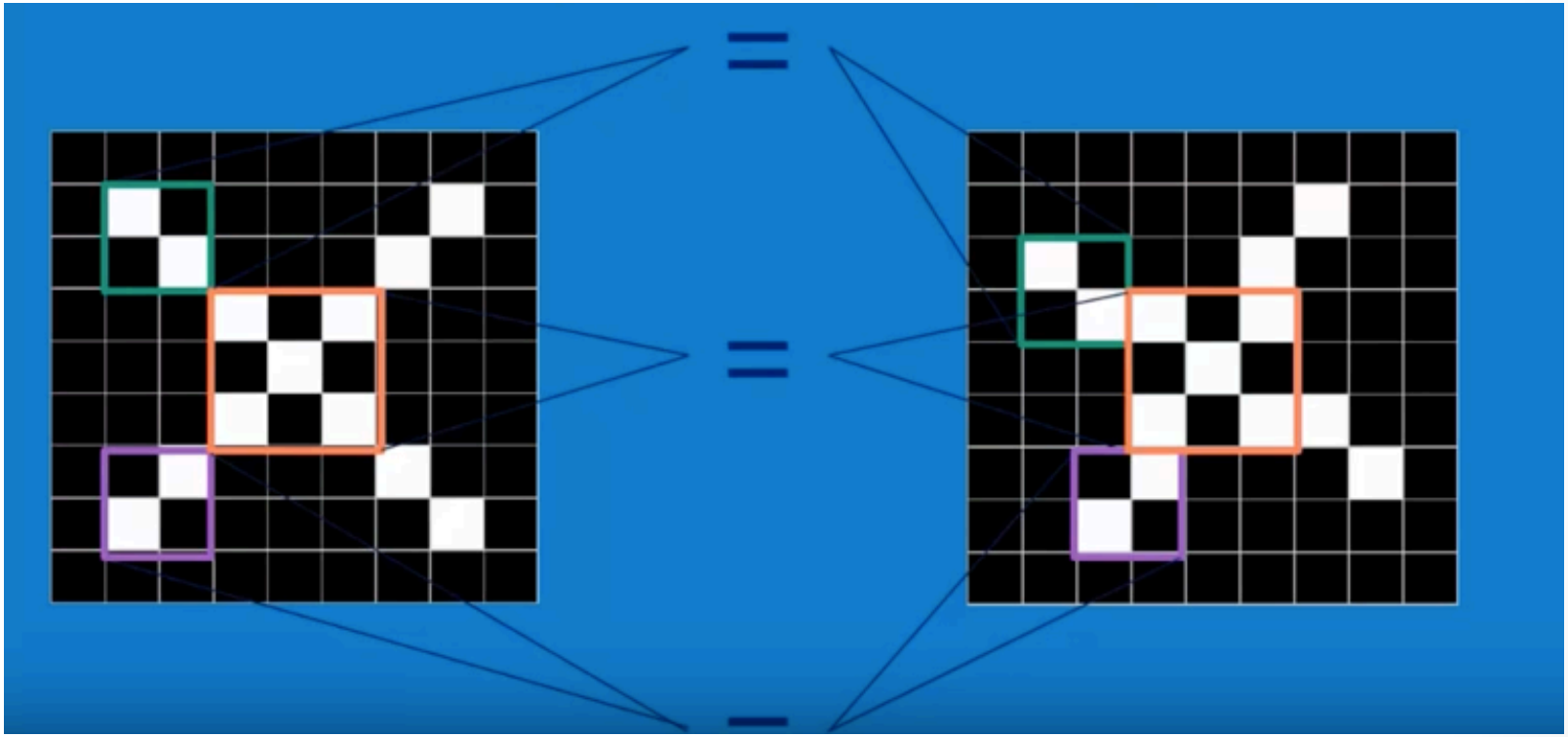
Ideia



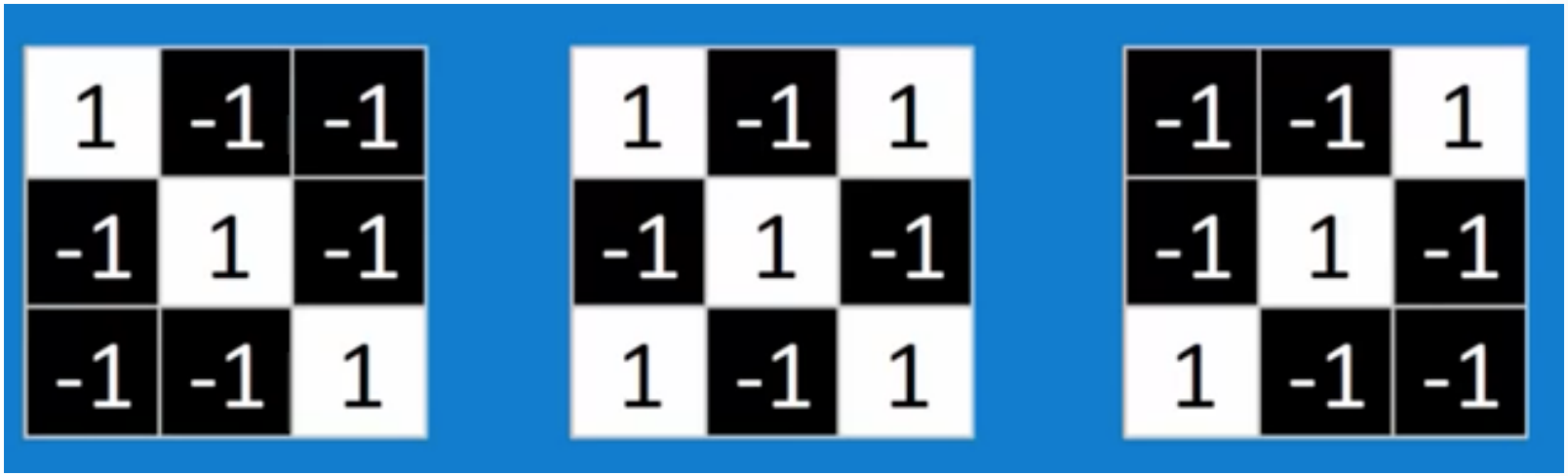
Ideia


-1 -1 -1 -1 -1 -1 -1 -1 -1
-1 1 -1 -1 -1 -1 -1 1 -1
-1 -1 1 -1 -1 -1 1 -1 -1
-1 -1 -1 1 -1 1 -1 -1 -1
-1 -1 -1 -1 1 -1 -1 -1 -1
-1 -1 -1 1 -1 1 -1 -1 -1
-1 -1 1 -1 -1 -1 1 -1 -1
-1 1 -1 -1 -1 -1 -1 1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1
?
-1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 1 -1 -1
-1 1 -1 -1 -1 1 -1 -1 -1
-1 -1 1 1 -1 1 -1 -1 -1
-1 -1 -1 -1 1 -1 -1 -1 -1
-1 -1 -1 1 -1 1 1 -1 -1
-1 -1 -1 1 -1 -1 -1 1 -1
-1 -1 1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1

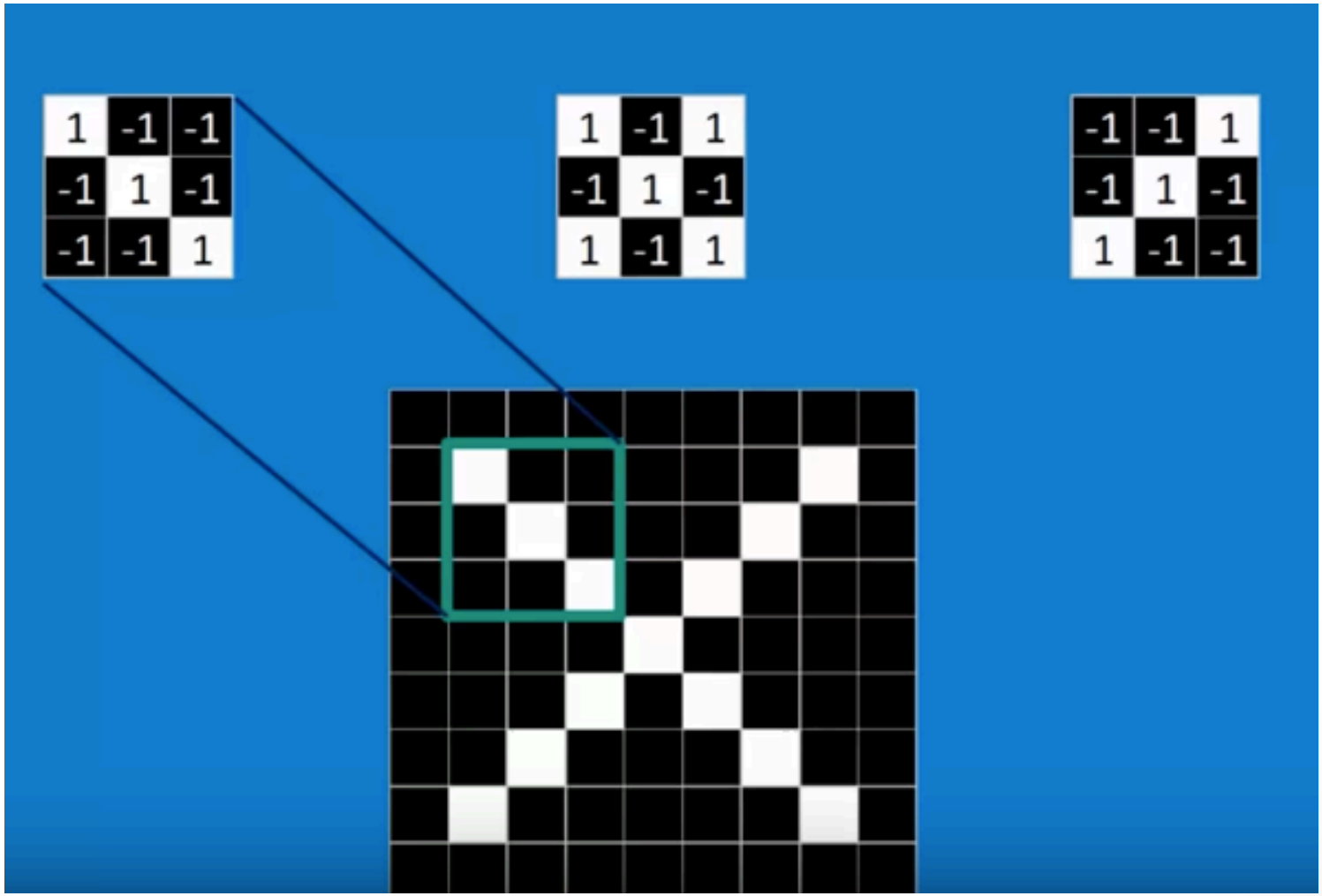
Ideia



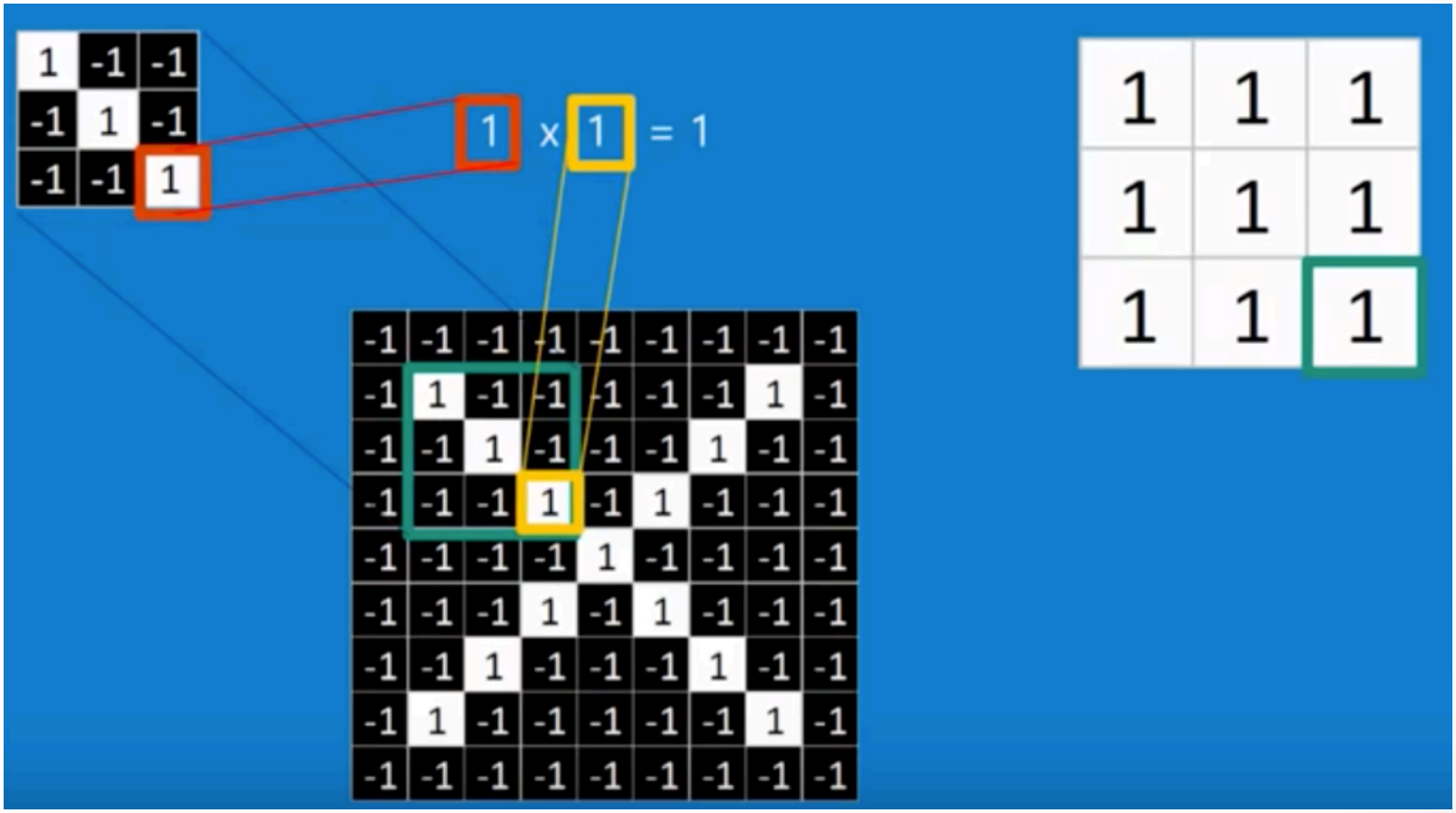
Ideia



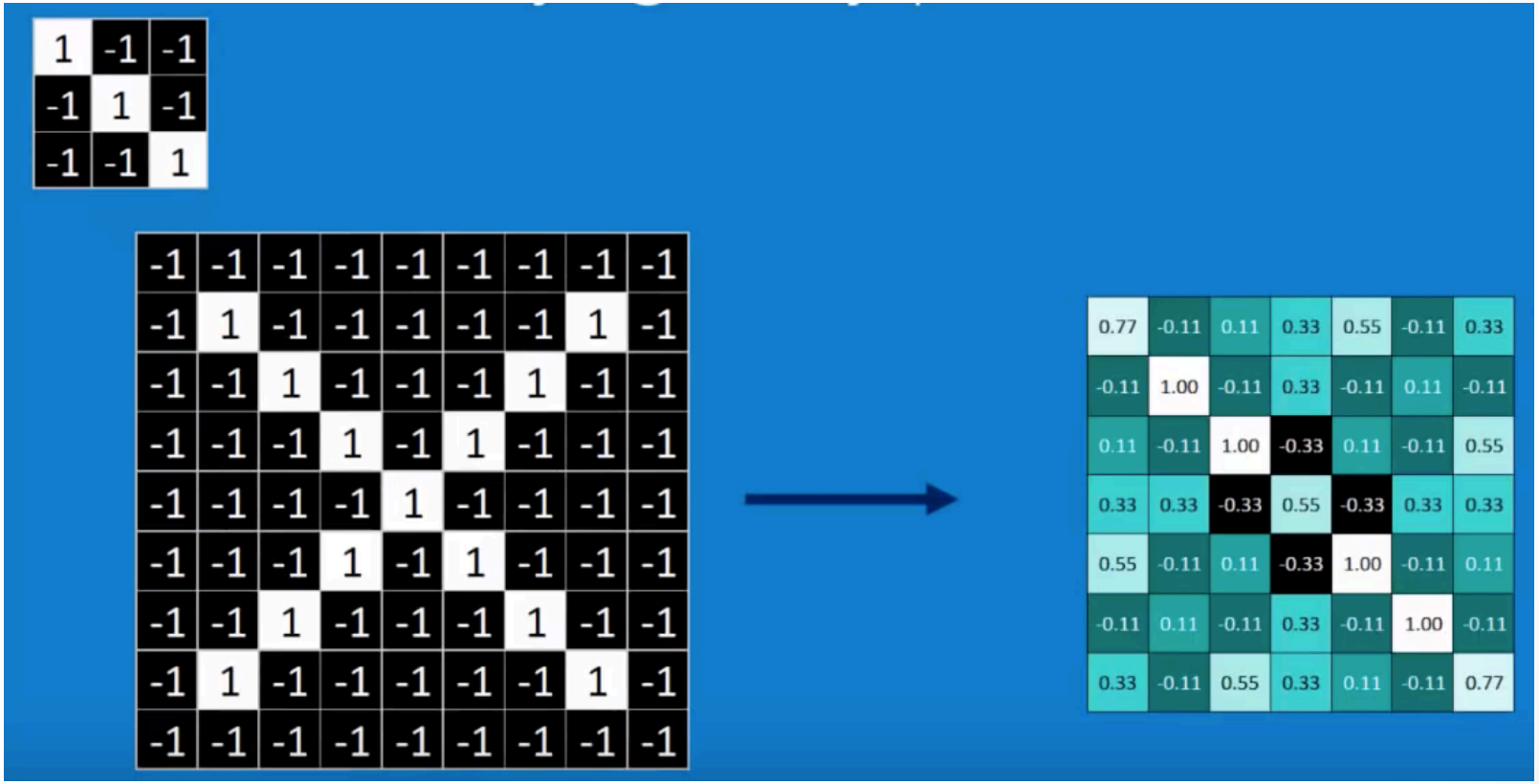
Ideia



Ideia


1 -1 -1
-1 1 -1
-1 -1 1
1 x 1 = 1
-1 -1 -1 -1 -1 -1 -1 -1 -1
-1 1 -1 -1 -1 -1 -1 1 -1
-1 -1 1 -1 -1 -1 1 -1 -1
-1 -1 -1 1 -1 1 -1 -1 -1
-1 -1 -1 -1 1 -1 -1 -1 -1
-1 -1 -1 1 -1 1 -1 -1 -1
-1 -1 1 -1 -1 -1 1 -1 -1
-1 1 -1 -1 -1 -1 -1 1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1
1 1 1
1 1 1
1 1 1

Ideia


1 -1 -1
-1 1 -1
-1 -1 1
-1 -1 -1 -1 -1 -1 -1 -1 -1
-1 1 -1 -1 -1 -1 -1 1 -1
-1 -1 1 -1 -1 -1 1 -1 -1
-1 -1 -1 1 -1 1 -1 -1 -1
-1 -1 -1 -1 1 -1 -1 -1 -1
-1 -1 -1 1 -1 1 -1 -1 -1
-1 -1 1 -1 -1 -1 1 -1 -1
-1 1 -1 -1 -1 -1 -1 1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1
0.77 -0.11 0.11 0.33 0.55 -0.11 0.33
-0.11 1.00 -0.11 0.33 -0.11 0.11 -0.11
0.11 -0.11 1.00 -0.33 0.11 -0.11 0.55
0.33 0.33 -0.33 0.55 -0.33 0.33 0.33
0.55 -0.11 0.11 -0.33 1.00 -0.11 0.11
-0.11 0.11 -0.11 0.33 -0.11 1.00 -0.11
0.33 -0.11 0.55 0.33 0.11 -0.11 0.77

Ideia

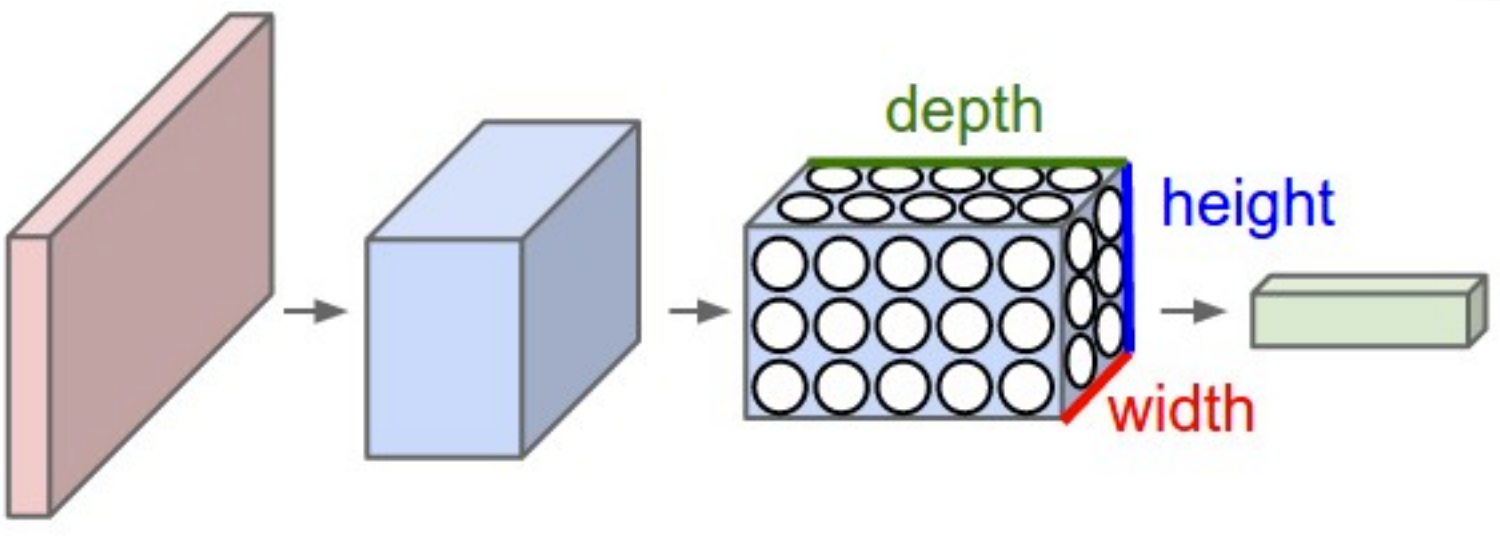
Ideia

- Pode-se aprender várias features.
  - Gerando um volume que representam as ativações dos neurônios (pesos)

- Uma feature também é uma matriz
  - A ativação pode então ser obtida por uma operação de **convolução**

  - Que preserva ou não o tamanho original

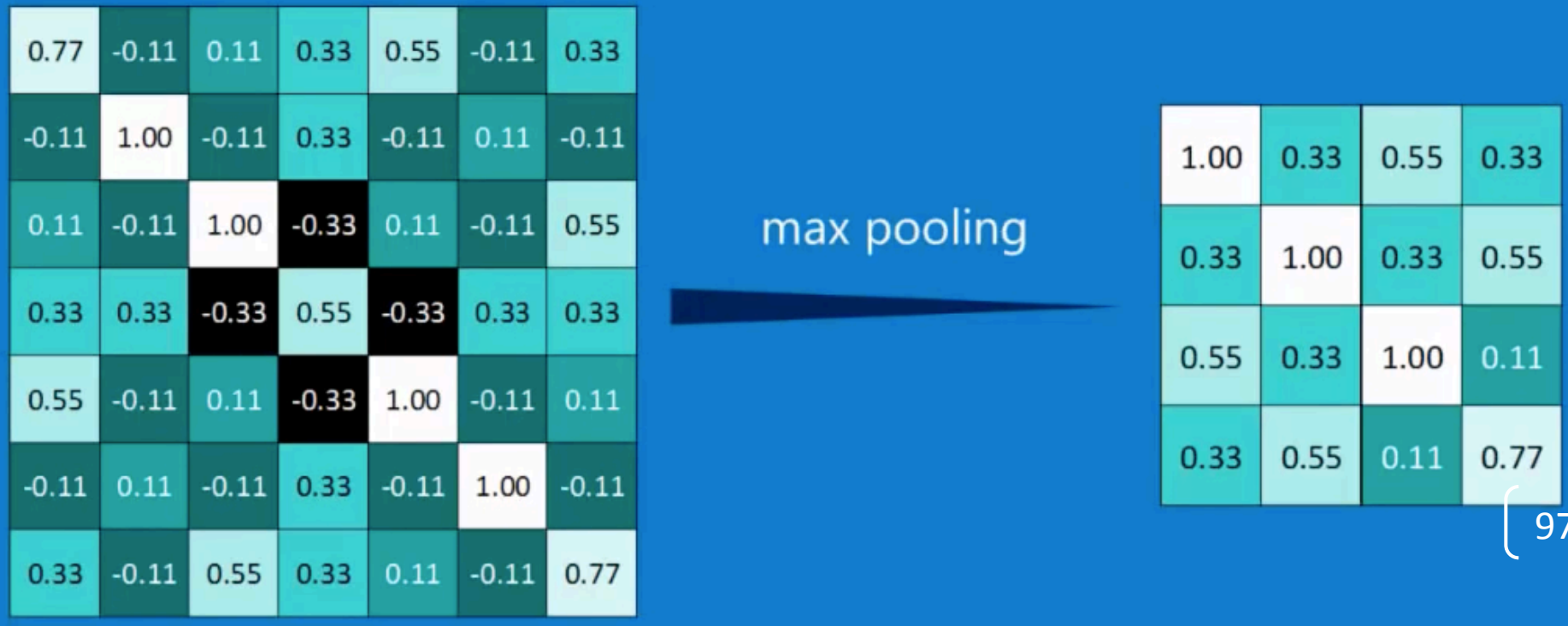
Ideia

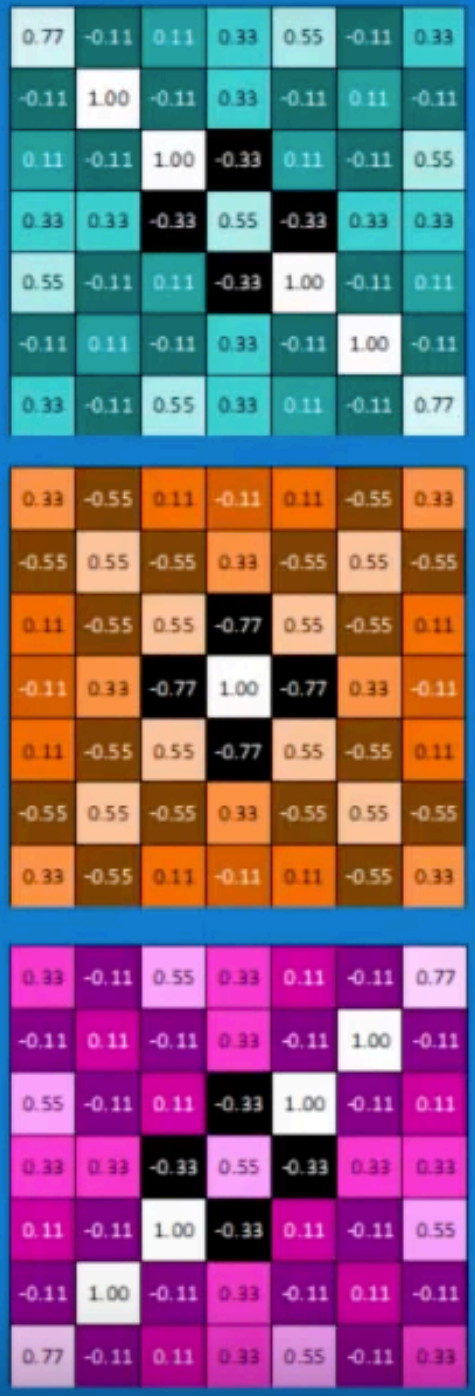


O processo consiste em codificar a imagem numa quantidade menor de neurônios que preserve os relacionamentos espaciais anteriores

Ideia

- Mas não vai gerar a mesma quantidade? Reduza o tamanho com Pooling





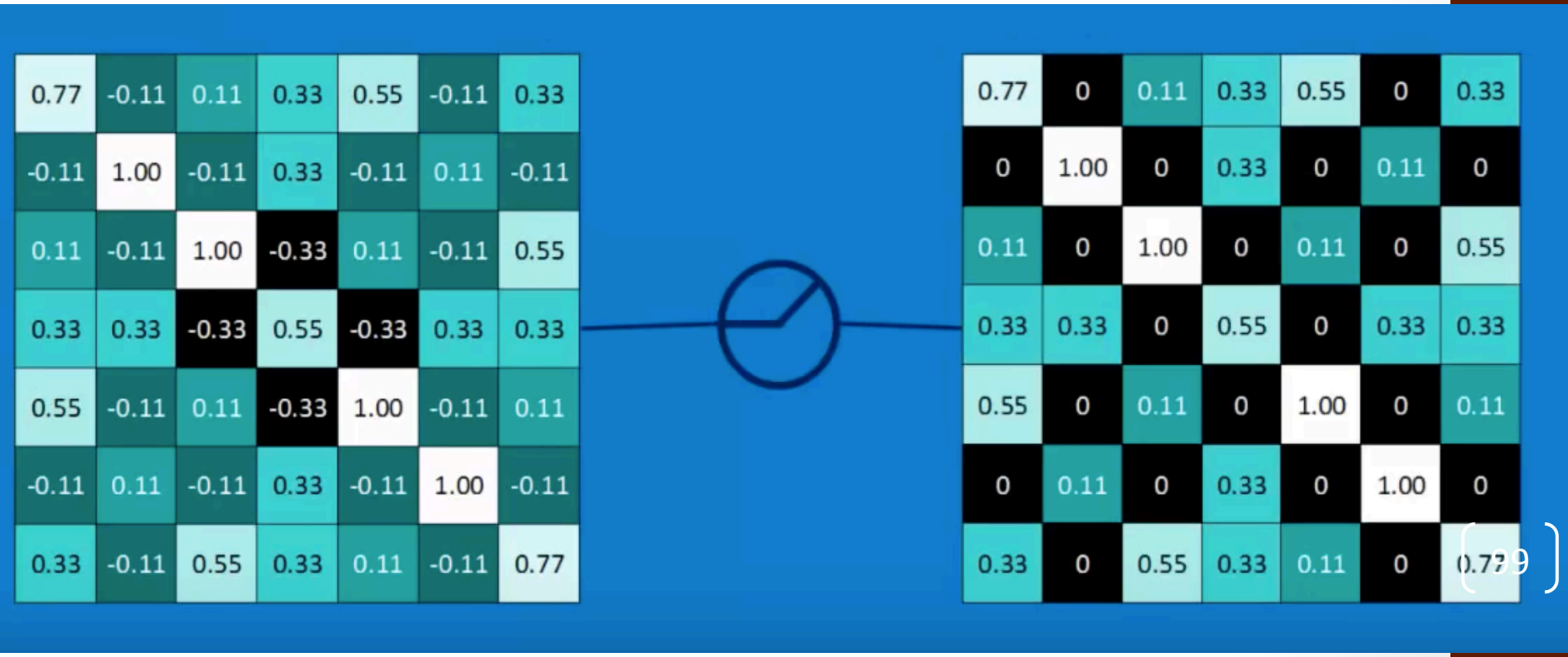
1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77

0.55	0.33	0.55	0.33
0.33	1.00	0.55	0.11
0.55	0.55	0.55	0.11
0.33	0.11	0.11	0.33

0.33	0.55	1.00	0.77
0.55	0.55	1.00	0.33
1.00	1.00	0.11	0.55
0.77	0.33	0.55	0.33

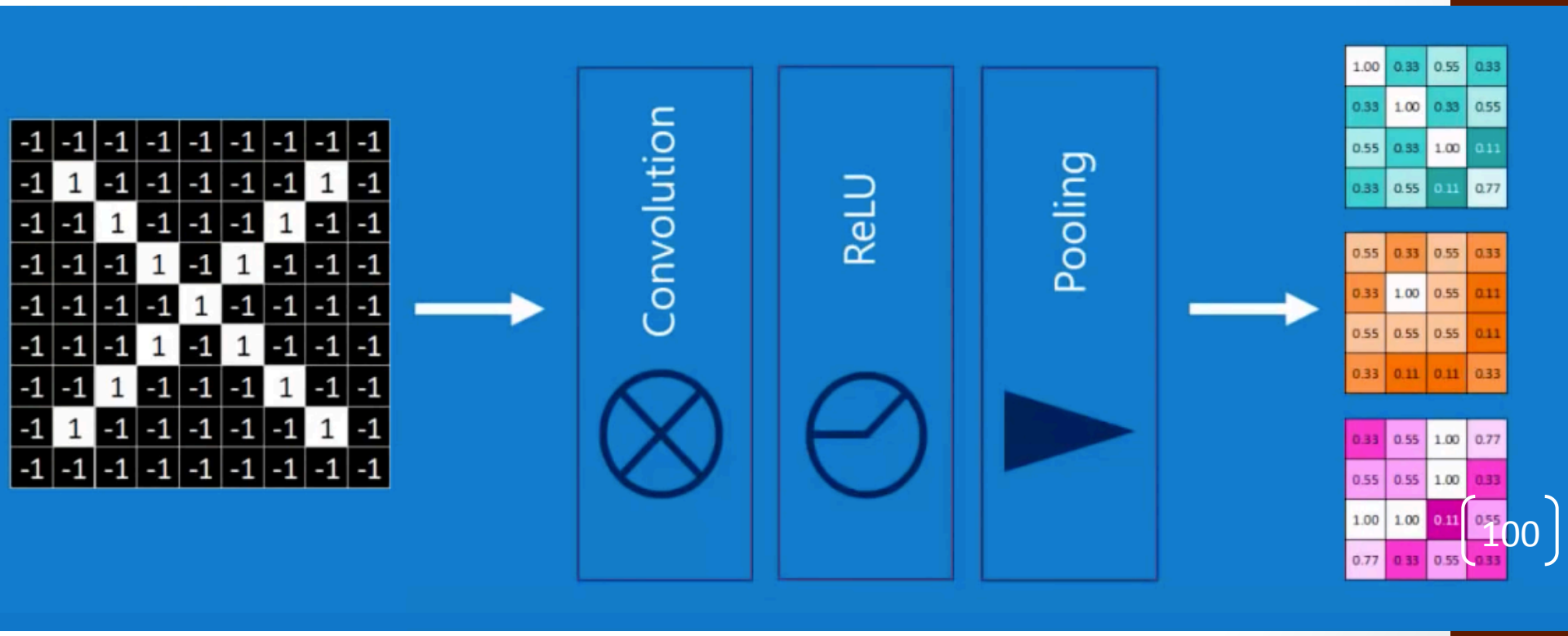
Ideia

- E esses valores negativos? Imagem não tem! RELU



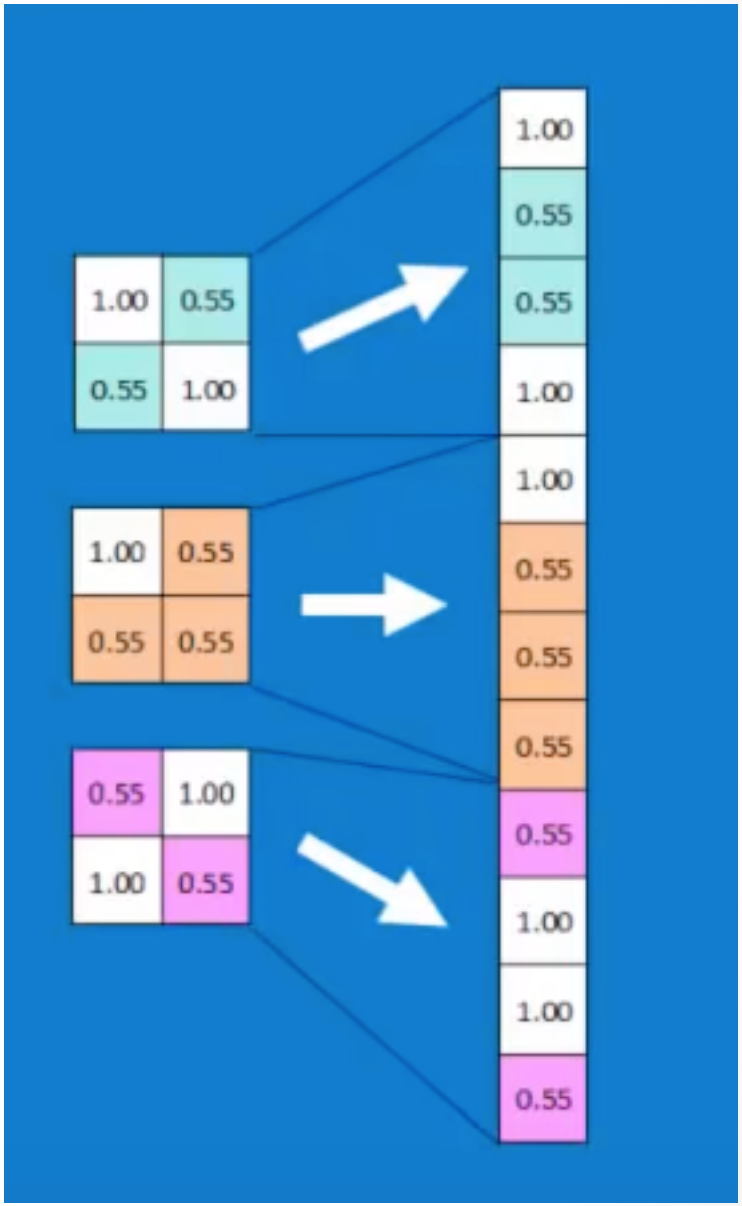
Ideia

- E quem reconhece?

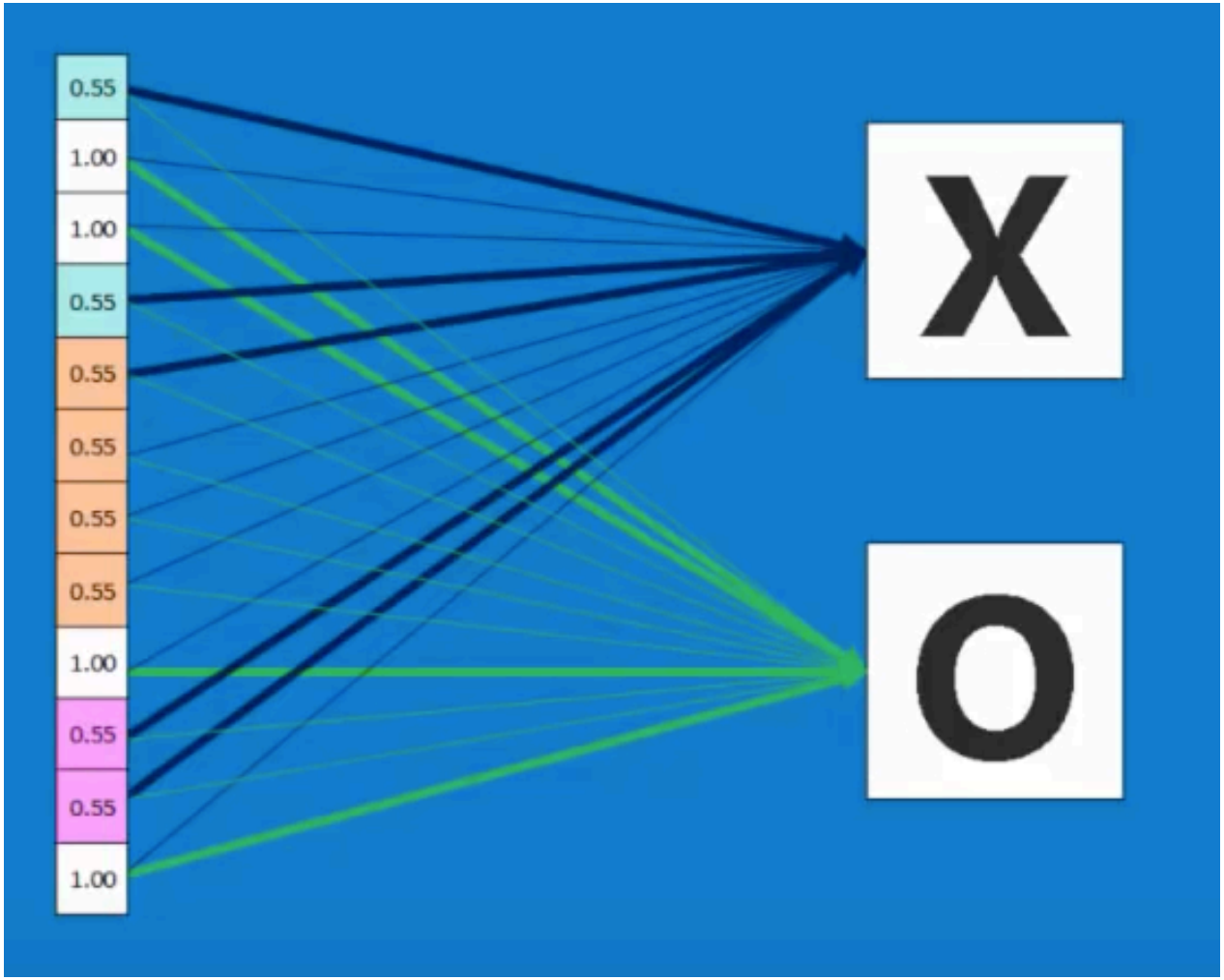


Ideia

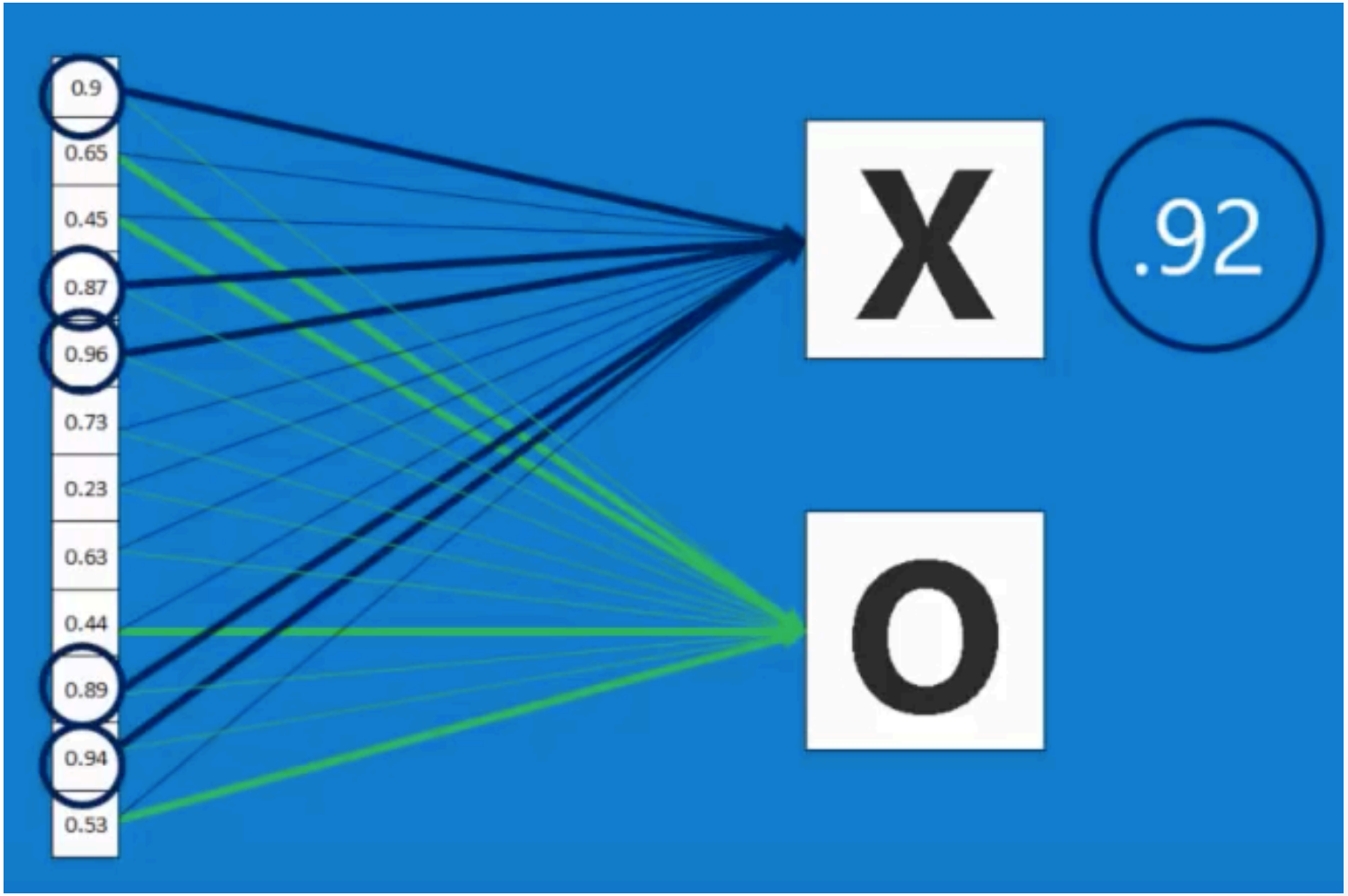
- Quem reconhece?
- MLP
  - Camada Totalmente Conectada



Ideia

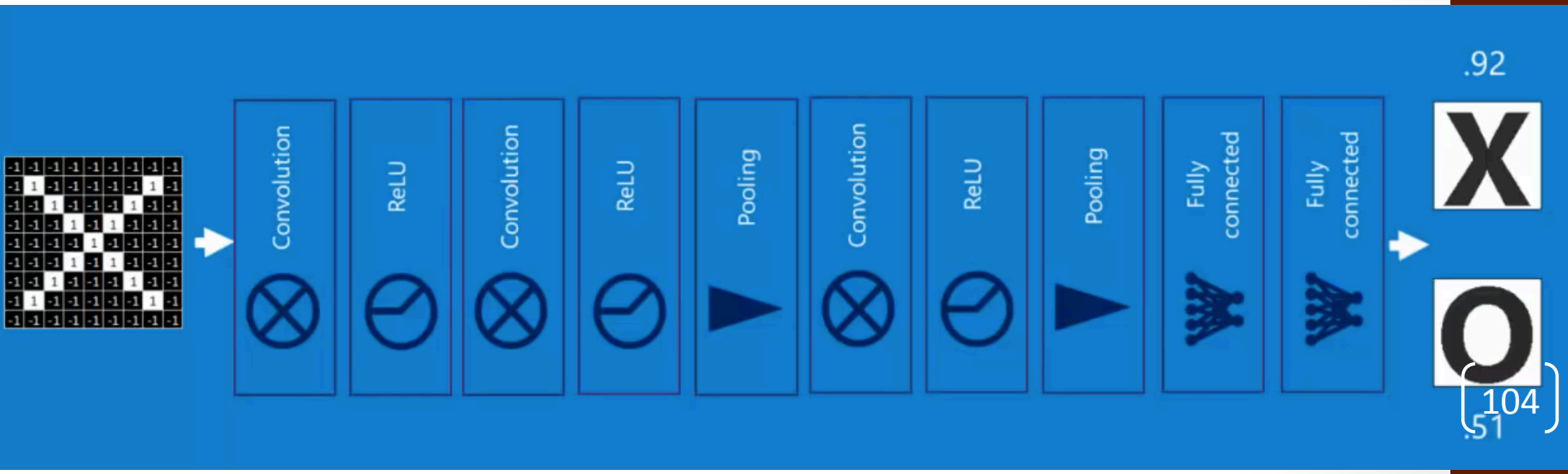


Ideia



Ideia

- O que eu faço com o erro?
- **Backpropagation**

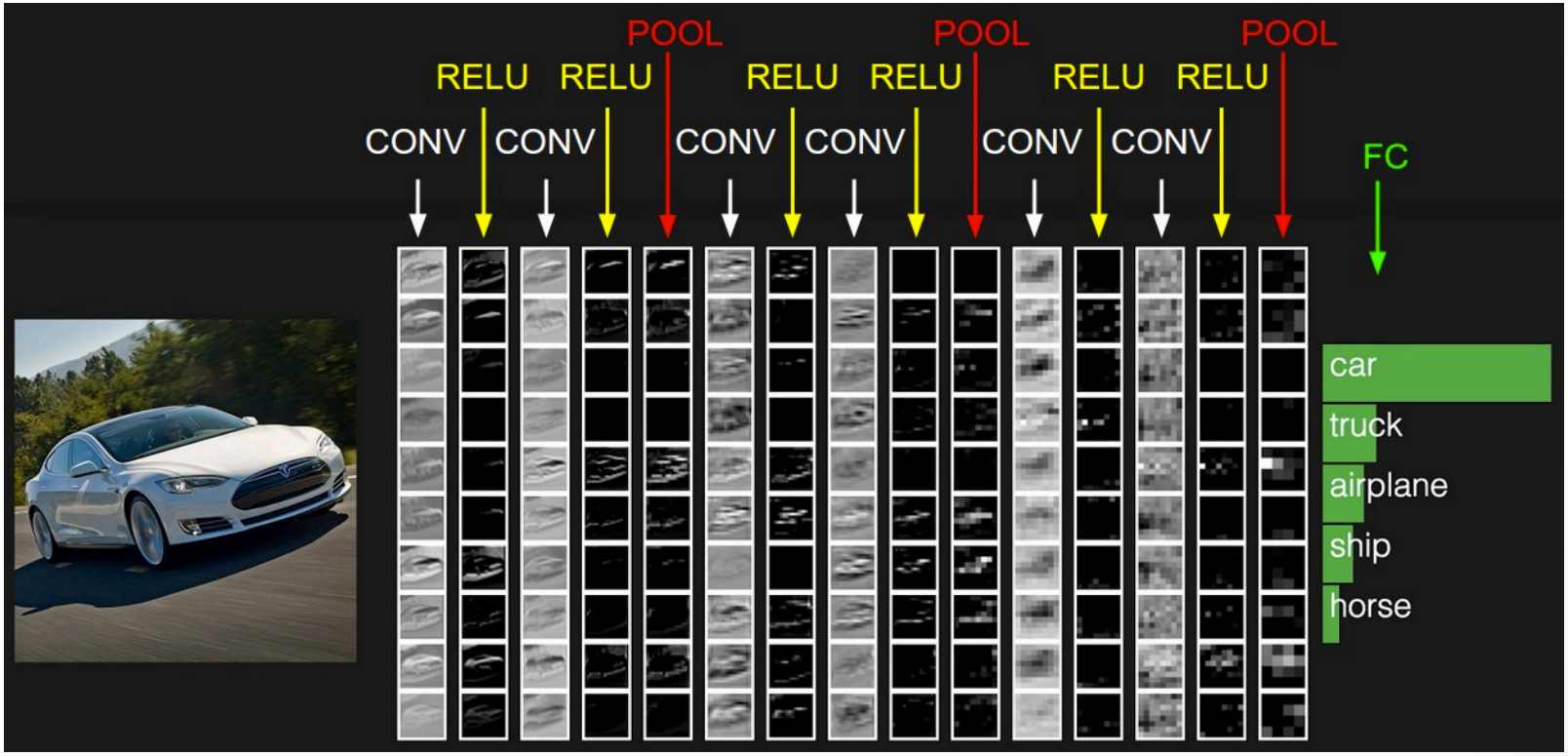


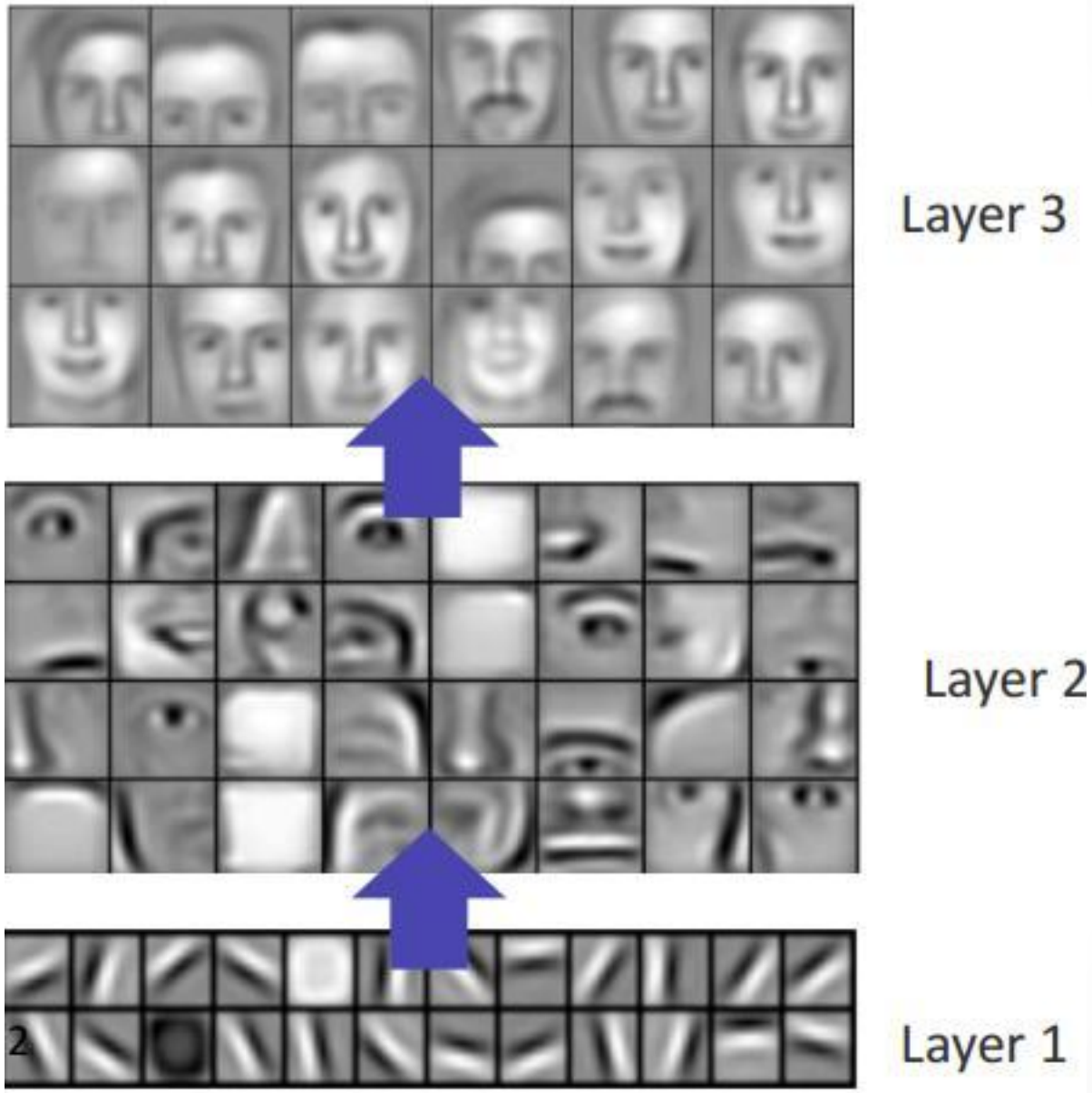
Porque Deep?

- A ideia é conseguir trabalhar com o espaço grande de características presente
- E capturar as informações que compõe o padrão
- **Do nível mais básico para o mais elevado**

- **De características de baixo nível → para características de alto nível**
  - Hierarquicamente

Visualizando




Layer 3
Layer 2
Layer 1

Camadas Normalmente Usadas numa CNN

- **Convolucional** – gerar neurônios conectados a regiões das imagens
- **RELU** – Rectified Linear Unit
- **POOL** – Redução de dimensionalidade
- **FC** – Realizar a classificação

Por Camada: Convolucional

- Predicado Local: não funciona conectar cada pixel a um neurônio
- Ideia: conectar um neurônio a uma região espacial da imagem

- Regulada por um hyperparameter: **receptive field** (se considerar como um filtro, então o **tamanho do filtro**)
  - Mesma profundidadade da imagem. → **sempre**

Convolucional – Tamanho da Saída

- O tamanho do volume de resultado é controlado em termos de 3 variáveis (**de uso mutuamente exclusivas**):
  - **Depth** (hyperparameter): quantidade de filtros
  - **Stride** (hyperparameter): Quantidade de pixels que o filtro pula durante a convolução. Padrão 1 ou 2
    - Quando 1, o tamanho permanece igual em termos de alturaxlargura
    - Quando maior, diminui

Convolucional - Tamanho da Saída

- **Zero-Padding**: preenchimento com zeros a borda para prevenir efeito de corte da borda pela convolução
- Relacionamento **(a conta sempre tem que ser inteira)**:

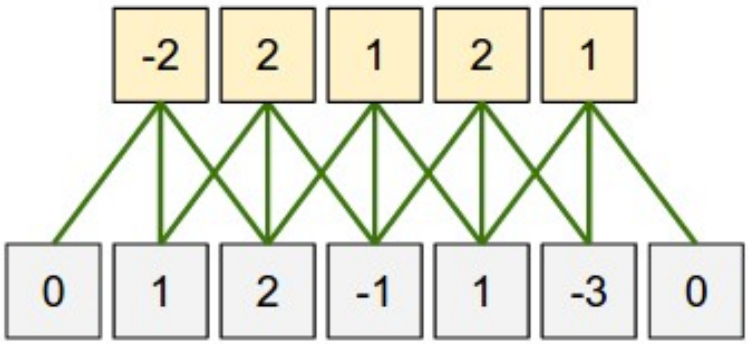
$$\frac{(W-F+2P)}{S} + 1$$

onde W = tamanho da entrada

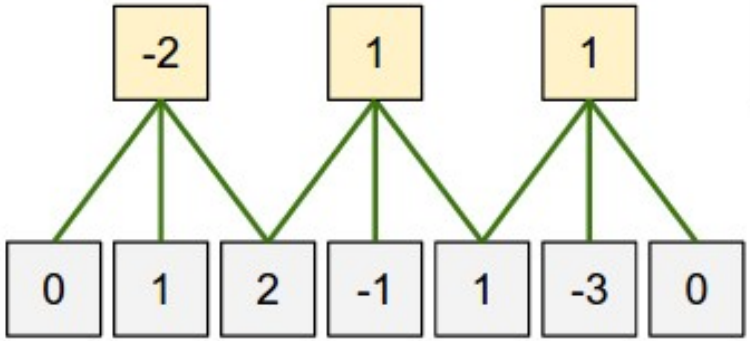
F = tamanho do filtro

P = tamanho do zero-padding

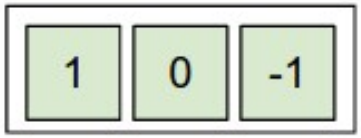
S = tamanho do stride



F = 3
W =5
S = 1
P =1



F = 3
W =5
S = 2
P = 1



Um exemplo: AlexNet

- Aceita imagens [227x227x3]
- Primeira Camada de Convolução
  - F= 11
  - S = 4
  - P = 0
  - K = 96 (quantidade de filtros)

  - (227-11)/4 + 1 = 55

  - Final: (55x55x96) conectado cada um a uma região (11x11x3) → este último é o tamanho do filtro

Funcionamento

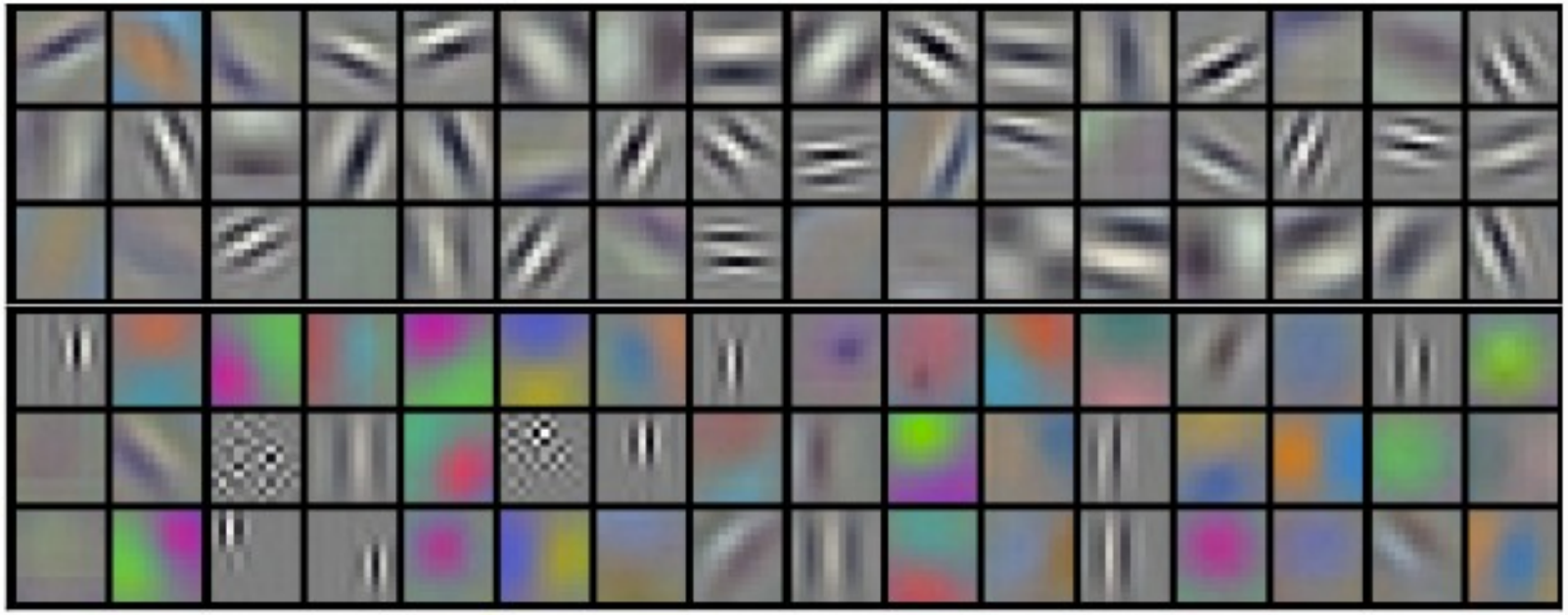
Summary. To summarize, the Conv Layer:

- 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.

Os filtros

- São aleatoriamente obtidos
  - Ou se usa uma base pré-treinada (**Transfer Learning**)
  - Mesmo com Transfer ainda é necessário se adaptar a nova base
- Representam os fatores de peso da rede (**w**) (neurônios na convolução)
  - Com o seu próprio bias (b)
- São apreendidos via **backpropagation**
- Cada filtro representa uma característica local na imagem
  - **Quantos são necessários? Tamanho? São parâmetros**

Exemplos de Filtros Aprendidos



Uma dica para saber se colocou a quantidade suficiente de filtros:
- visualizar os filtros nas várias camadas e
- checar se estão se aproximando a algo relacionado com a base e se não são “aleatórios”

RELU

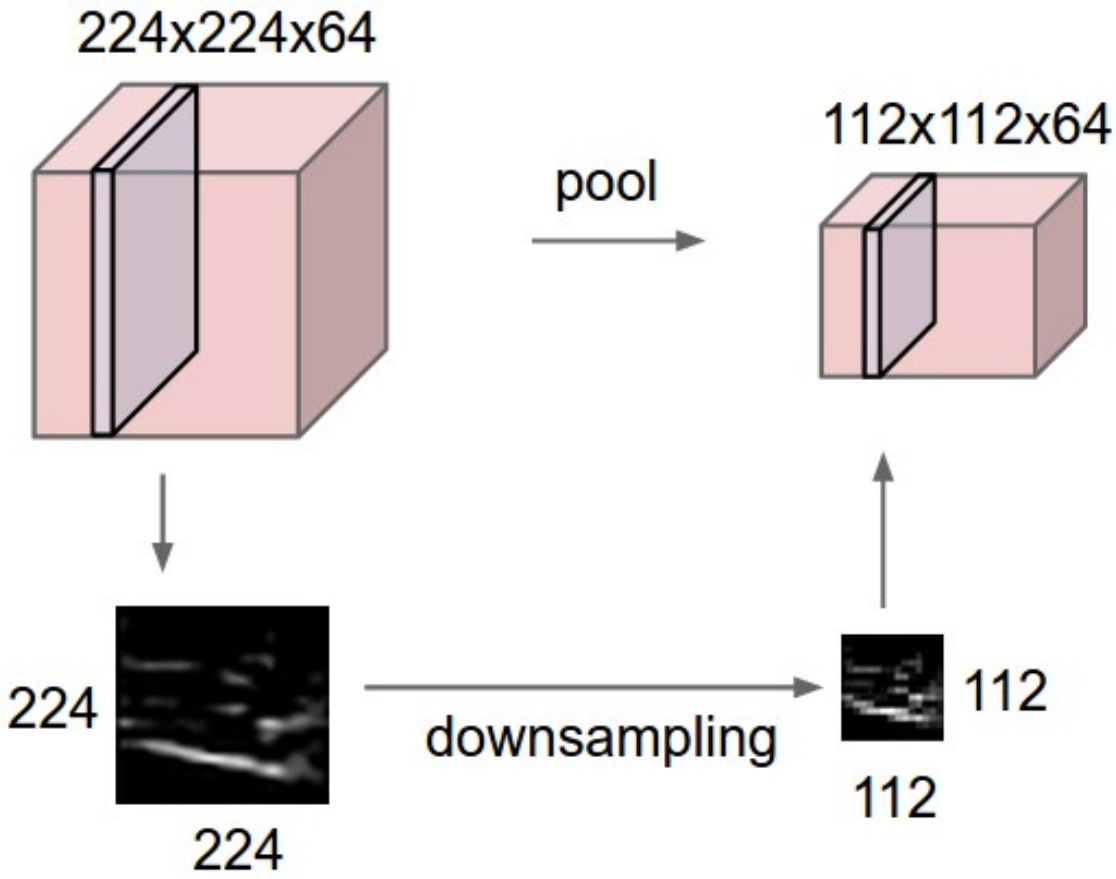
- Ao final da convolução, pode-se aplicar RELU para remover valores negativos

Pooling

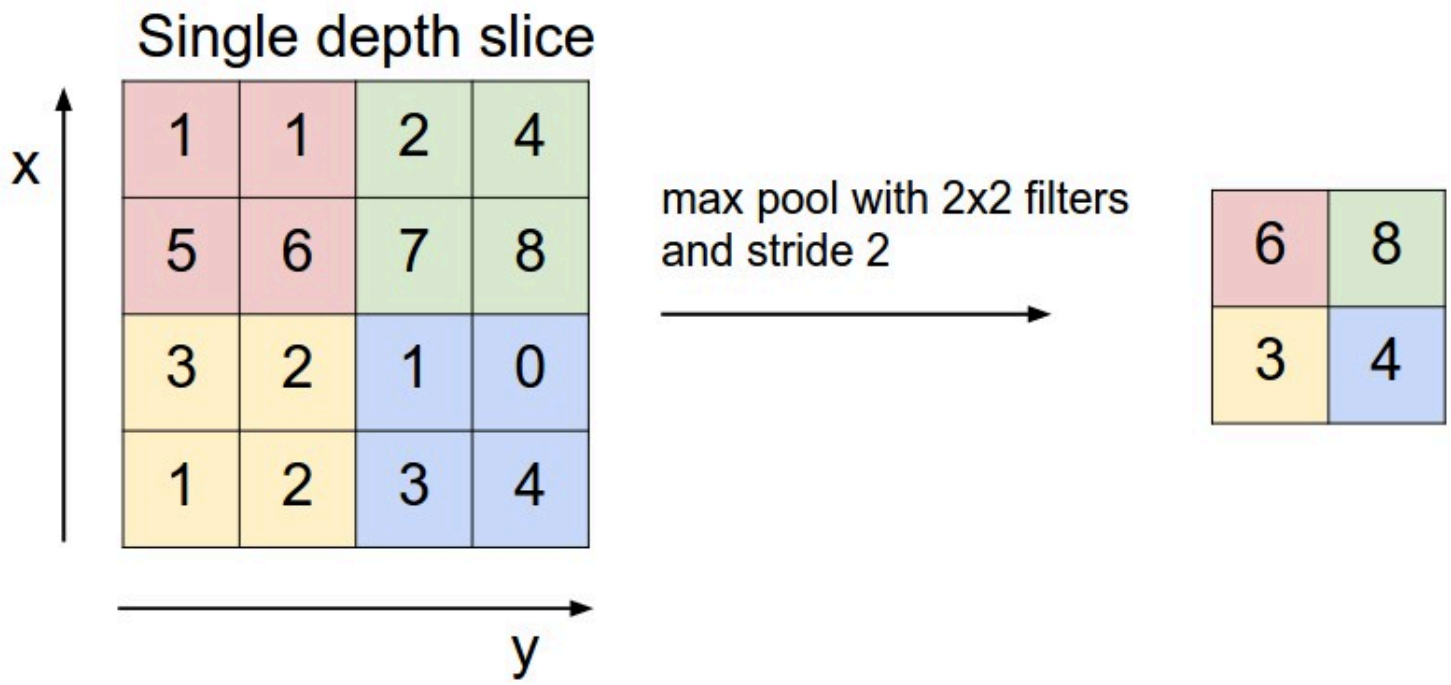
- Basicamente reduz dimensionalidade
  - Controla overfitting
  - Supõe que um valor represente bem uma área pequena
- Max pooling: mais comum, obtém o maior valor
- Mean pooling: obtém a média

- **Normal: F=2, S=2**

Pooling



Pooling


Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

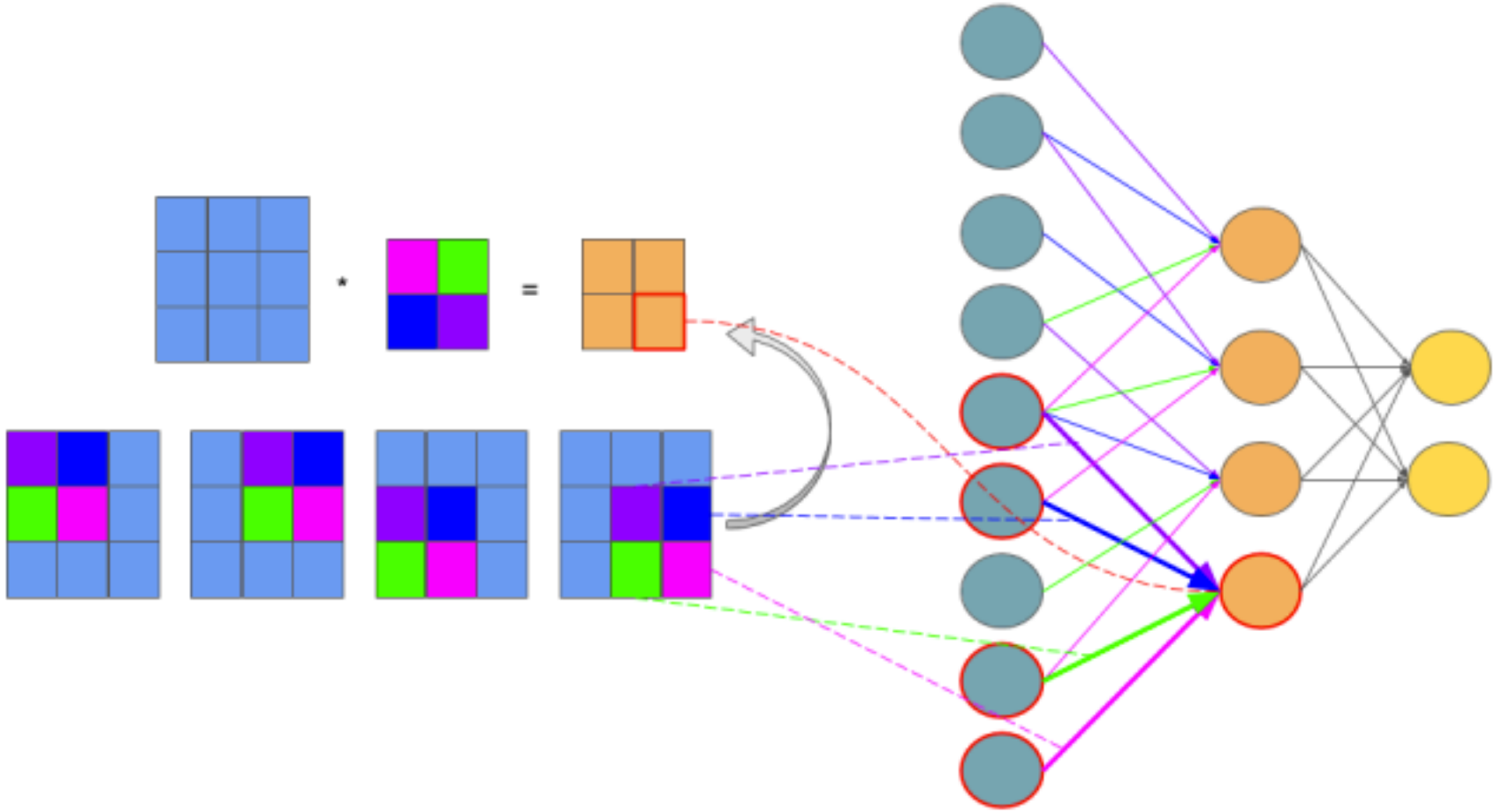
Fully-Connected

- Basicamente uma **MLP** (qualquer outra técnica que se adapte ao modelo de aprendizado)
- O resultado das camadas anteriores e convertido em apenas um canal e vetorizado (sob sua convenção):
  - Na última camada = 7x7x512
  - Na entrada da MLP
    - troca entrada por uma convolução por F=7 e K=4096
    - Fica [1x1x4096]

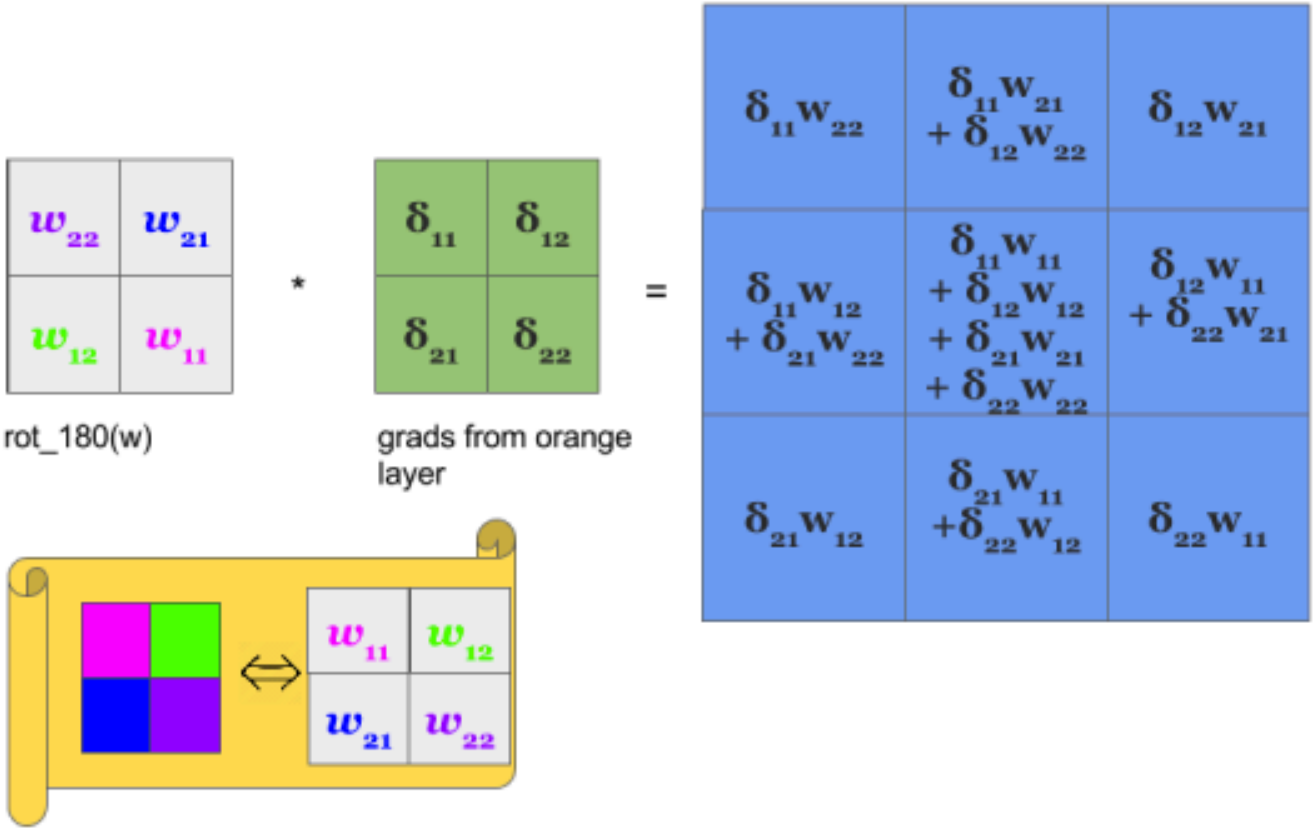
Fully-Connected

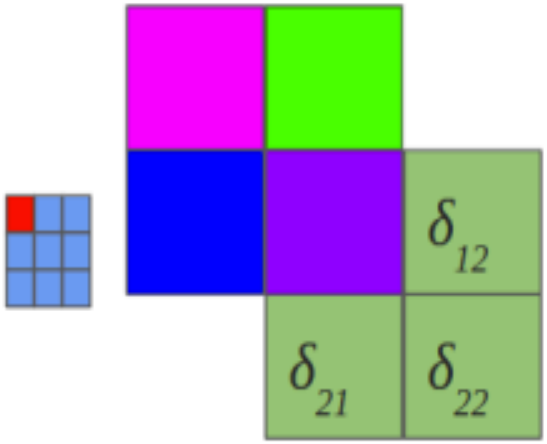
- Ao final se tem o erro de classificação
- Esse erro é devolvido para ajustar todos os pesos (backpropagation):
  - Dos neurônios na MLP
  - Dos neurônios no filtro de Pooling
  - Dos neurônios do filtro de Convolução

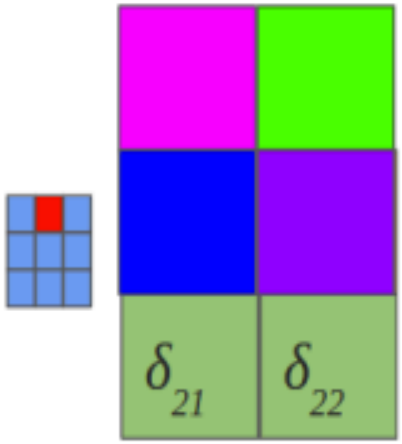
Backpropagation

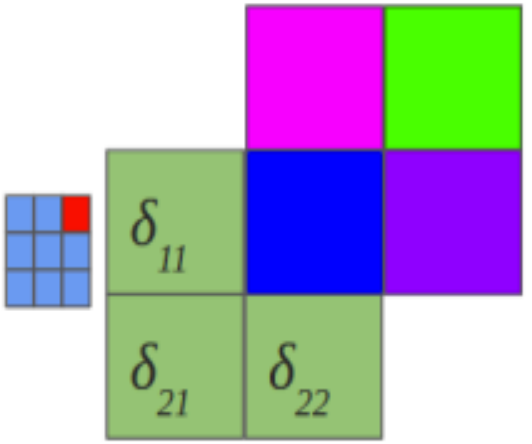


Backpropagation

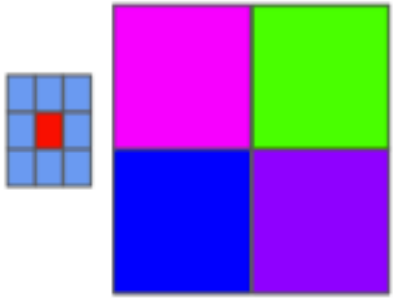



δ_{12}
δ_{21}
δ_{22}

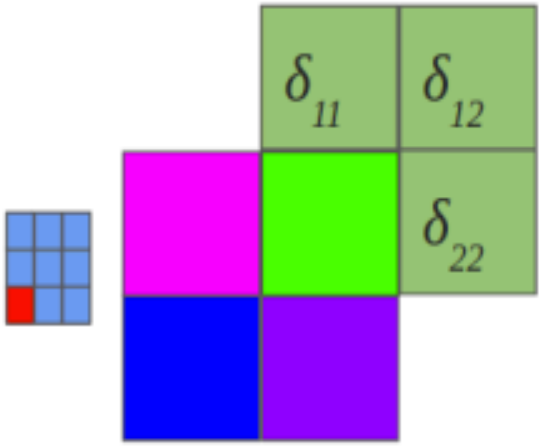

δ_{21}
δ_{22}

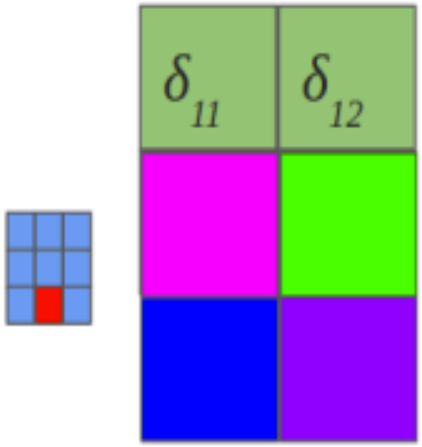

δ_{11}
δ_{21}
δ_{22}

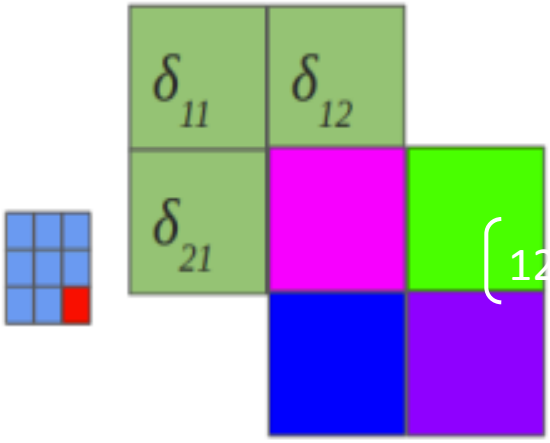

δ_{12}
δ_{22}




δ_{11}
δ_{21}


δ_{11}
δ_{12}
δ_{22}


δ_{11}
δ_{12}


δ_{11}
δ_{12}
δ_{21}

Backpropagation

- Também é uma convolução
  - O erro volta também como um filtro (convolução)

Gradiente do erro:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l}$$

onde (usando convolução):

$$z_{x,y}^{l+1} = w_{x,y}^{l+1} * \sigma(z_{x,y}^l) + b_{x,y}^{l+1} = \sum_a \sum_b w_{a,b}^{l+1} \sigma(z_{x-a,y-b}^l) +$$

$$b_{x,y}^{l+1}$$

sendo (ativação do neurônio)

$$a_j^l = \sigma(z_j^l)$$

Backpropagation

Contribuição do erro de classifcação na camada seguinte x a derivada do erro na camada atual levando em consideração a camada seguinte

$$\sum_{x'}\sum_{y'}\delta^{l+1}_{x',y'}\frac{\partial(\sum_a\sum_b w^{l+1}_{a,b}\sigma(z^l_{x'-a,y'-b})+b^{l+1}_{x',y'})}{\partial z^l_{x,y}} = \sum_{x'}\sum_{y'}\delta^{l+1}_{x',y'}w^{l+1}_{a,b}\sigma'(z^l_{x,y})$$

Só conta neurônios ativados

x =x'-a y = y'-b

O resto zera

Backpropagation

Troca a e b pelos valores de entrada

$$\sum_{x'}\sum_{y'}\delta^{l+1}_{x',y'}w^{l+1}_{a,b}\sigma'(z^l_{x,y}) = \sum_{x'}\sum_{y'}\delta^{l+1}_{x',y'}w^{l+1}_{x'-x,y'-y}\sigma'(z^l_{x,y})$$

Substitui pela convolução da função de erro pela função de peso multiplicada pela ativação

$$\sum_{x'}\sum_{y'}\delta^{l+1}_{x',y'}w^{l+1}_{x'-x,y'-y}\sigma'(z^l_{x,y}) = \delta^{l+1}_{x,y} * w^{l+1}_{-x,-y}\sigma'(z^l_{x,y})$$

Backpropagation

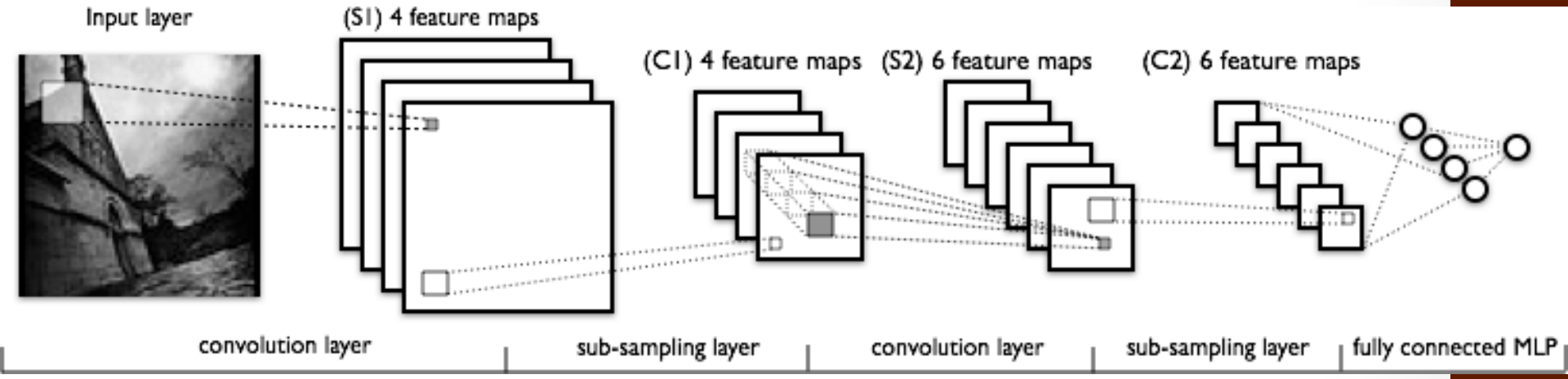
Sendo: $ROT180(w^{l+1}_{x,y}) = w^{l+1}_{-x,-y}$

$$\frac{\partial C}{\partial w^l_{a,b}} = \sum_x \sum_y \frac{\partial C}{\partial z^l_{x,y}} \frac{\partial z^l_{x,y}}{\partial w^l_{a,b}} = \sum_x \sum_y \delta^l_{x,y} \frac{\partial (\sum_{a'} \sum_{b'} w^l_{a',b'} \sigma(z^l_{x-a',y-b'}) + b^l_{x,y})}{\partial w^l_{a,b}} =$$

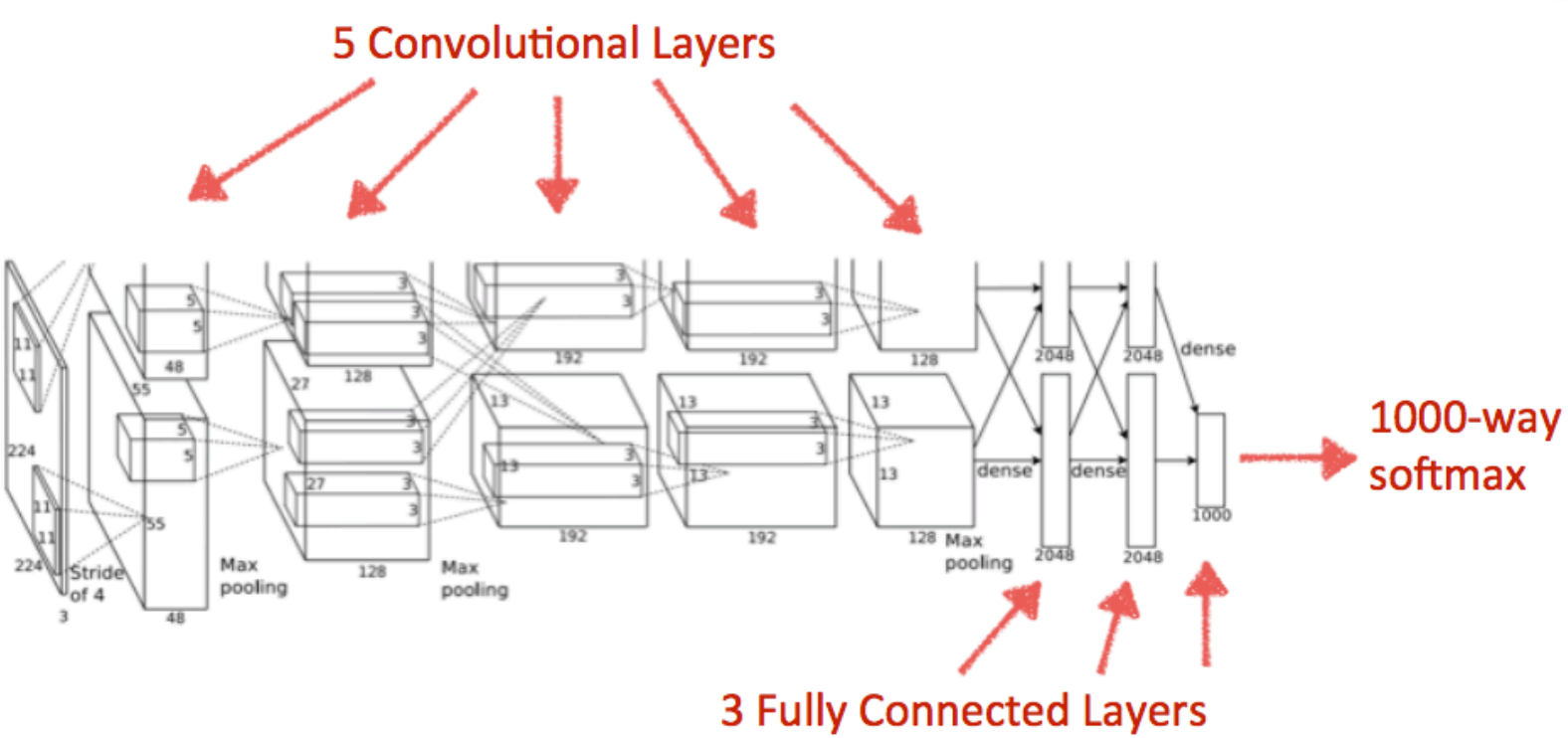
$$\sum_x \sum_y \delta^l_{x,y} \sigma(z^{l-1}_{x-a,y-b}) = \delta^l_{a,b} * \sigma(z^{l-1}_{-a,-b}) = \delta^l_{a,b} * \sigma(ROT180(z^{l-1}_{a,b}))$$

Algumas arquiteturas famosas

- LeNet (1990)

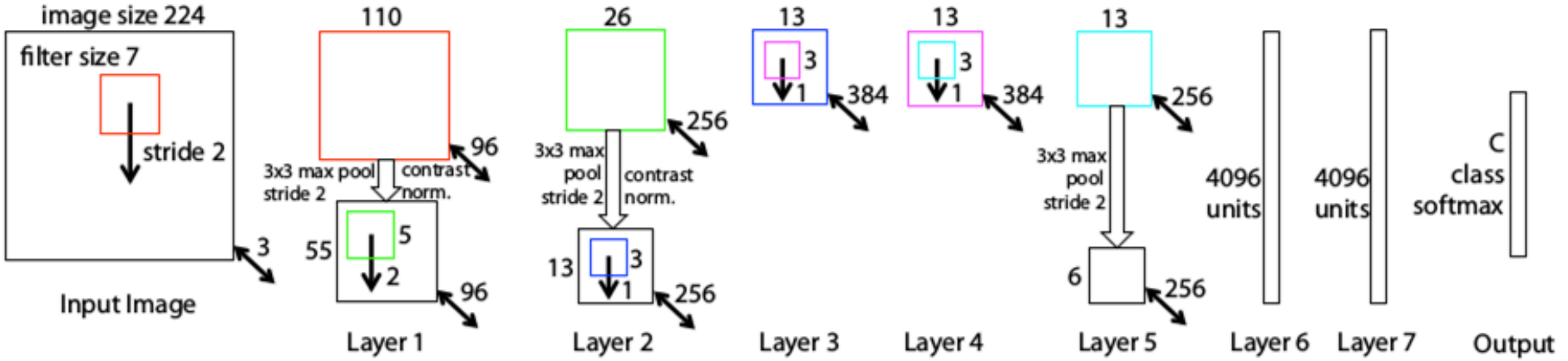


AlexNet (2012)



ZFNet (2013)

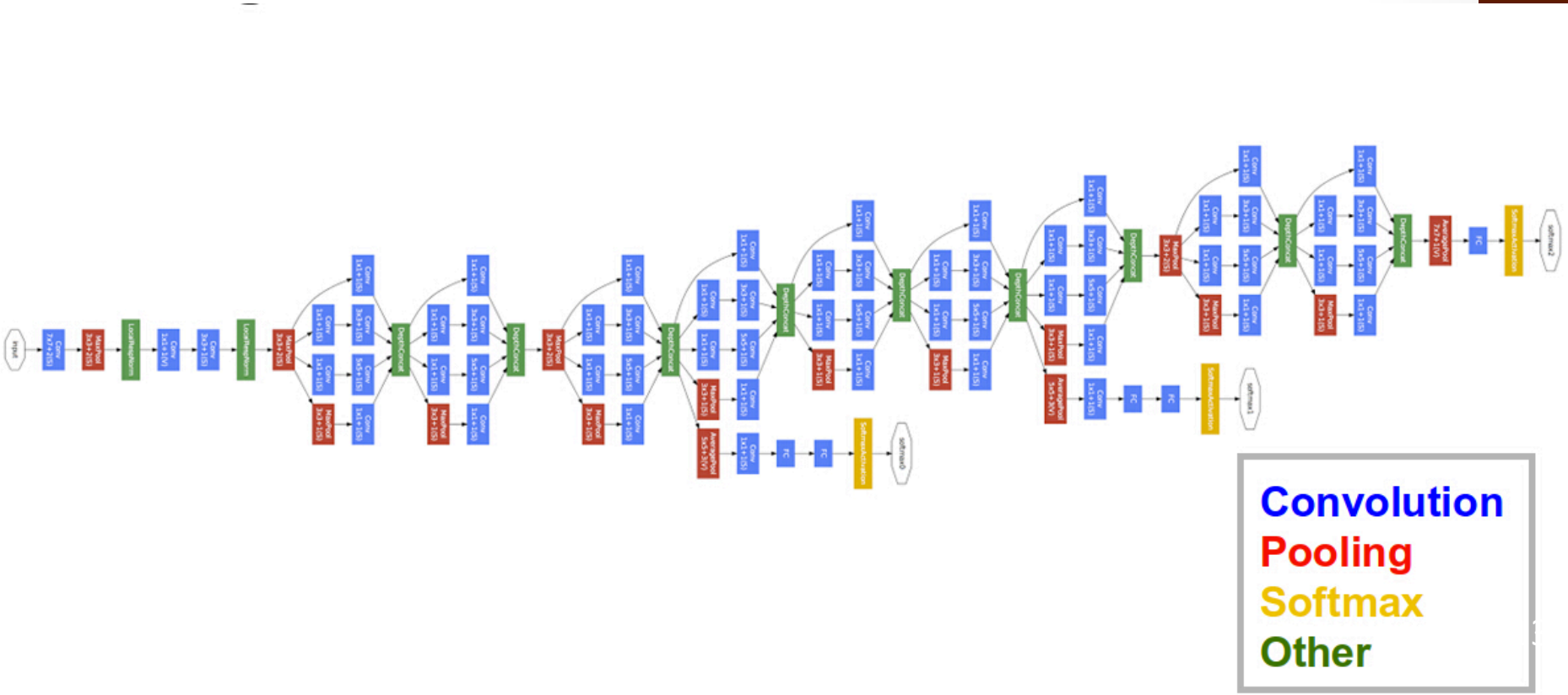
- Aumenta o tamanho da AlexNet



ZF Net Architecture

GoogLeNET (2014)

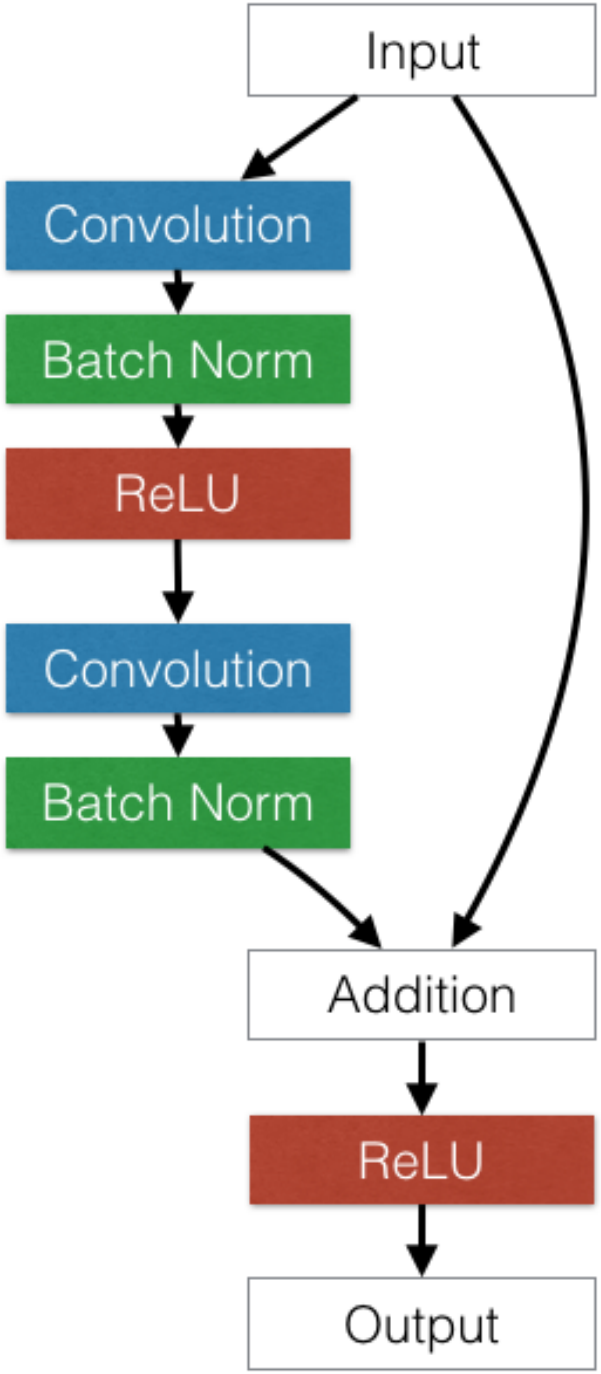
- Inception Mode e Average Pooling ao invés de FC



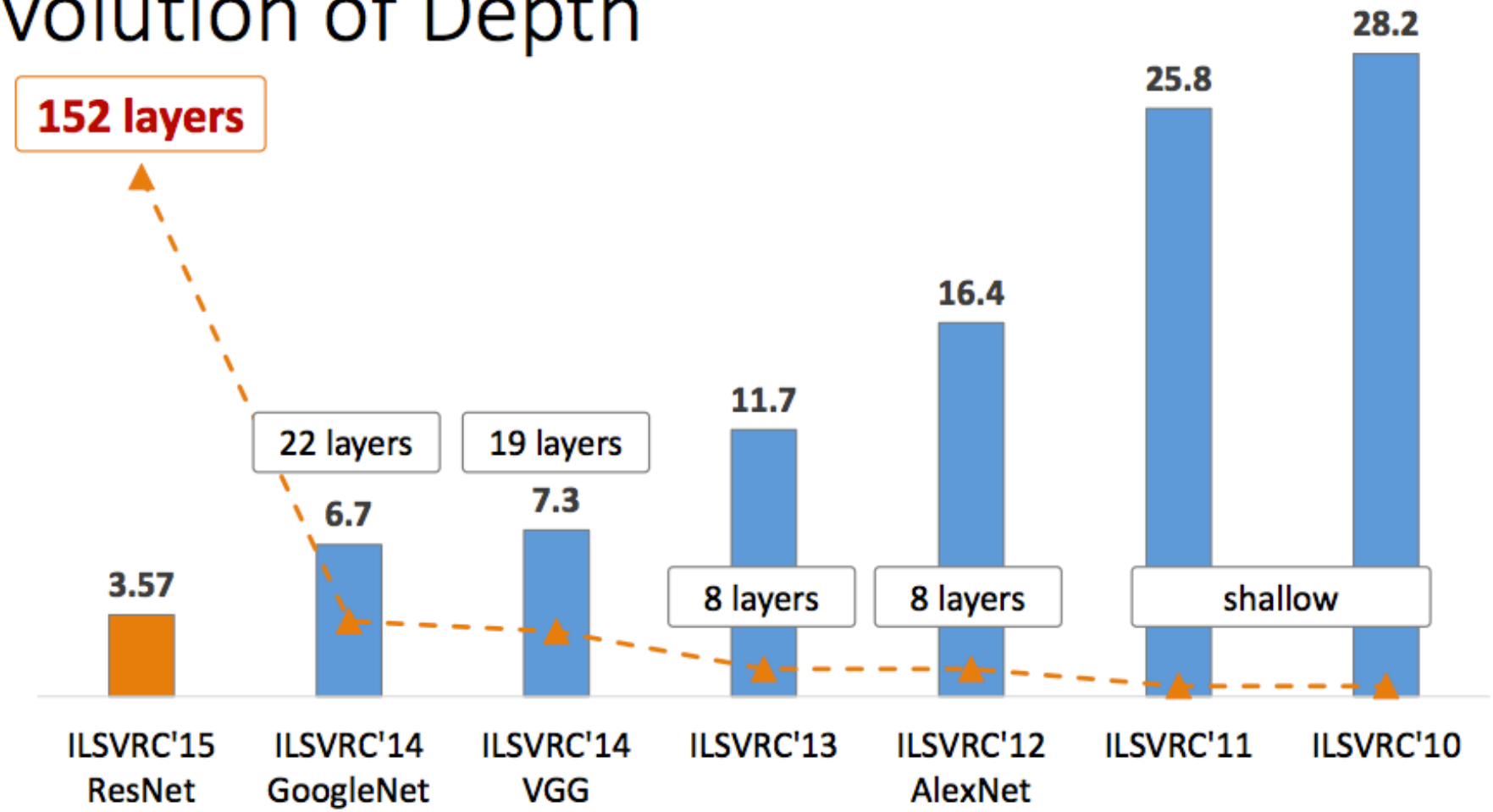
ResNet (2015)

- Aprende a pular camadas quando necessário

- Originalmente 152 camadas
  - Mas a medida que treina, identifica "saltos"



Revolution of Depth



Considerações

- Principal consideração
  - Como definir a arquitetura e os hyperparametros?

  - Por enquanto:
    - Partir de um que existe, e
    - Otimizar
    - Visualize os filtros
      - O que está saindo faz sentido?

Considerações

- Existem vários tipos de aprendizado profundo ou métodos de aprendizado (além da CNN)
  - Stacked Auto-Encoders→ aprendizado via reconstrução
  - Deep Belief Networks (DBN) → recurrent
  - Hierarchical Temporal Memory → reconhecimento espacial e inferência temporal
  - Deep Spatial Temporal Inference Network (DESTIN) → hierarquia de aprendizados não supervisionados com relacionamento temporal e local
- Leia mais ;)