PPGTI3007 - Visão Computacional

Redes Neurais Convolucionais

Redes Neurais Artificiais

- Cérebro humano é responsável pelo processamento e controle de diversas informações
- Realizamos ações que requerem atenção a diversos eventos ao mesmo tempo e processamentos variados
  - Ex. pegar objeto, caminhar, envolvem ação de diversos componentes, como memória, coordenação, aprendizado
- ⇒ Motivação na construção de máquinas inteligentes

Redes Neurais Artificiais

- Sistemas distribuídos inspirados na estrutura e funcionamento do sistema nervoso
- Objetivo: simular capacidade de aprendizado do cérebro na aquisição de conhecimento

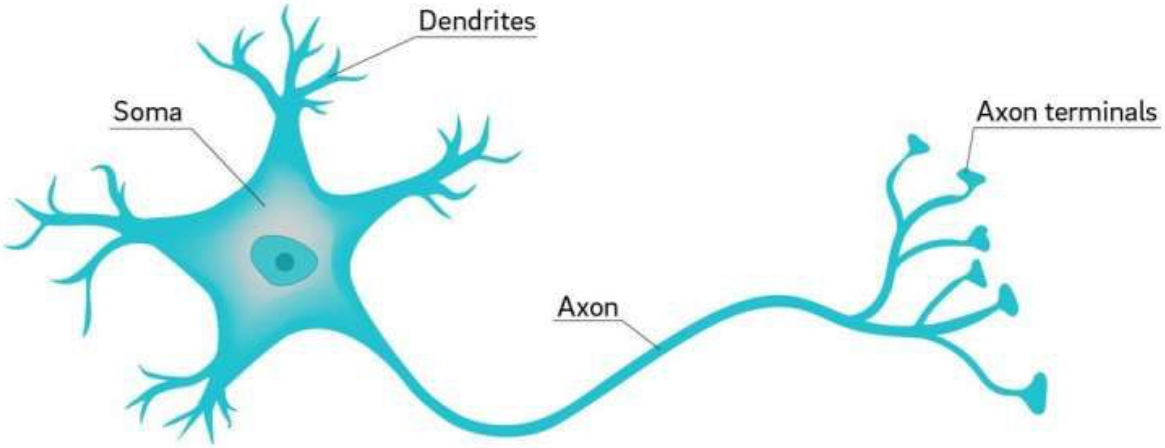
Compostas por várias unidades de processamento (“neurônios”)

Interligadas por um grande número de conexões (“sinapses”)

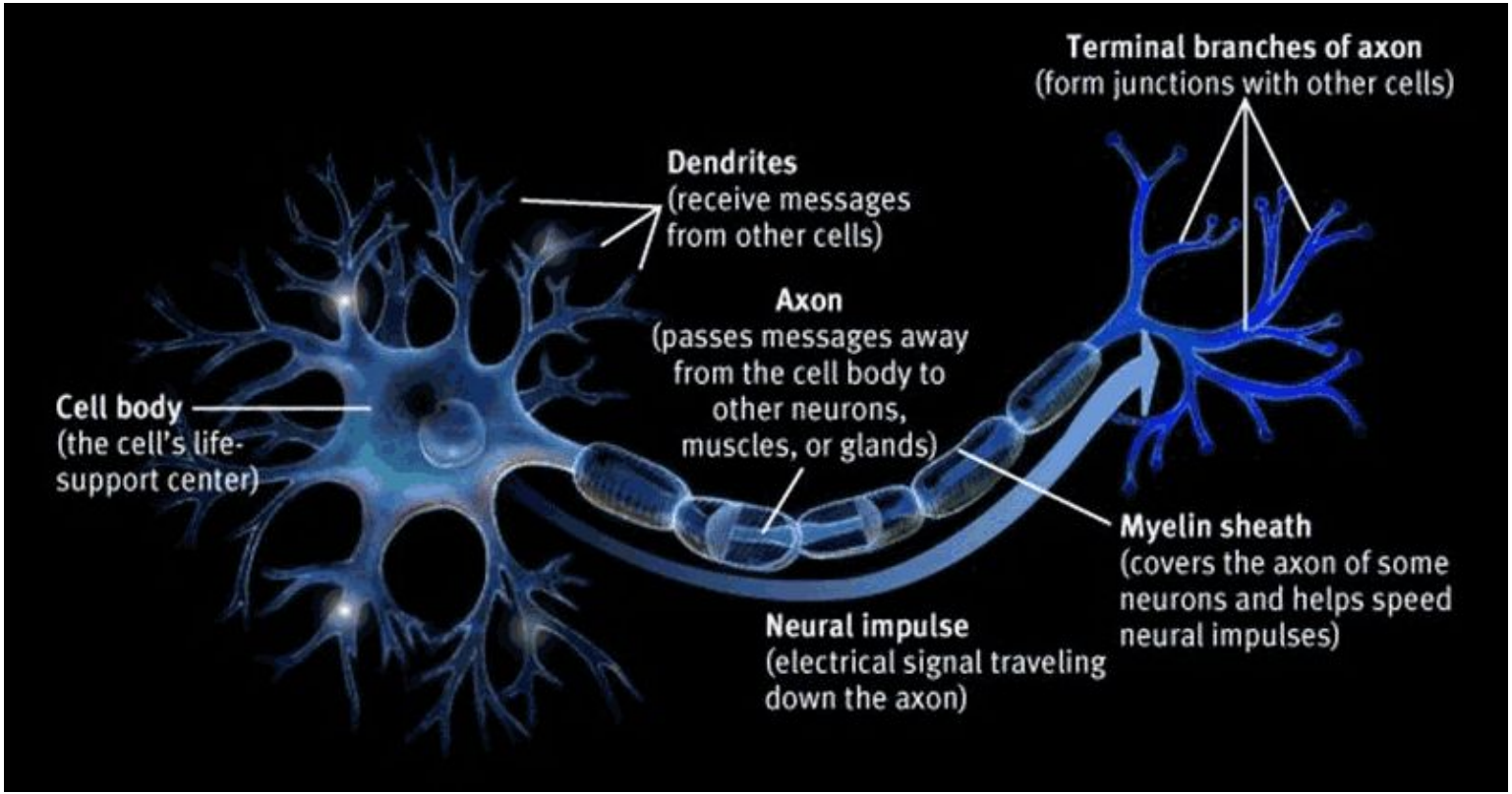
Redes Biológicas

- Cérebro humano: 10^{11} neurônios
- Cada neurônio processa e se comunica com milhares de outros continuamente e em paralelo
- Cérebro: responsável por funções cognitivas e execução de funções sensoriomotoras e autônomas
- Tem capacidade de reconhecer padrões e relacioná-los, usar e armazenar conhecimento por experiência e interpretar observações

Neurônio Natural



Neurônio Natural



Redes Biológicas

- Neurônios são bem mais lentos que os circuitos elétricos, mas o cérebro é capaz de realizar muitas tarefas mais rápido que qualquer computador
  - Redes neurais biológicas trabalham de forma massivamente paralela
  - Neurônios estão organizados em cerca de 1000 nódulos principais, cada um com 500 redes neurais
  - E cada neurônio pode estar ligado a centenas ou até milhares de outros neurônios

Redes Biológicas

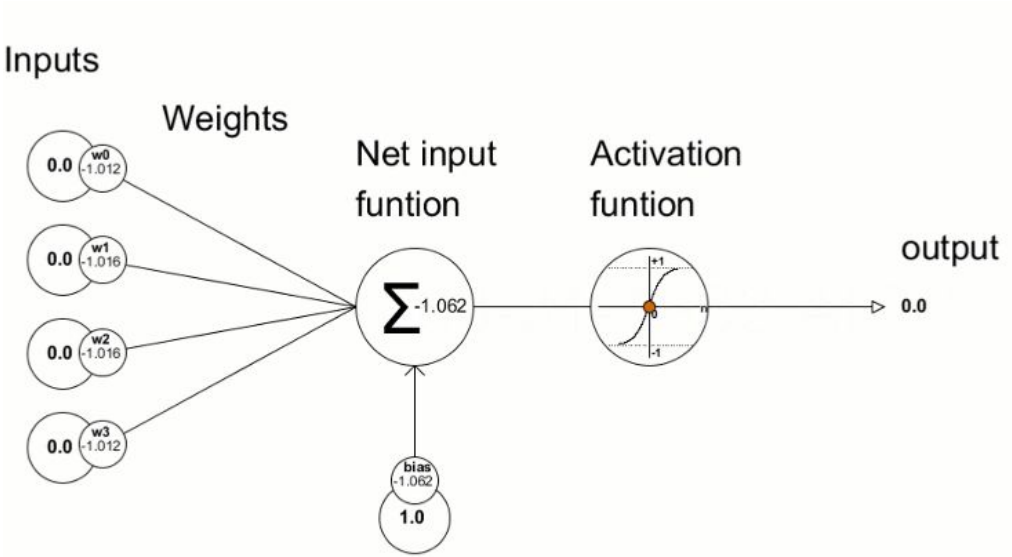
Rede Neural Artificial

Uma Rede Neural Artificial (RNA) é um sistema computacional que apresenta um modelo inspirado na estrutura neural do cérebro humano

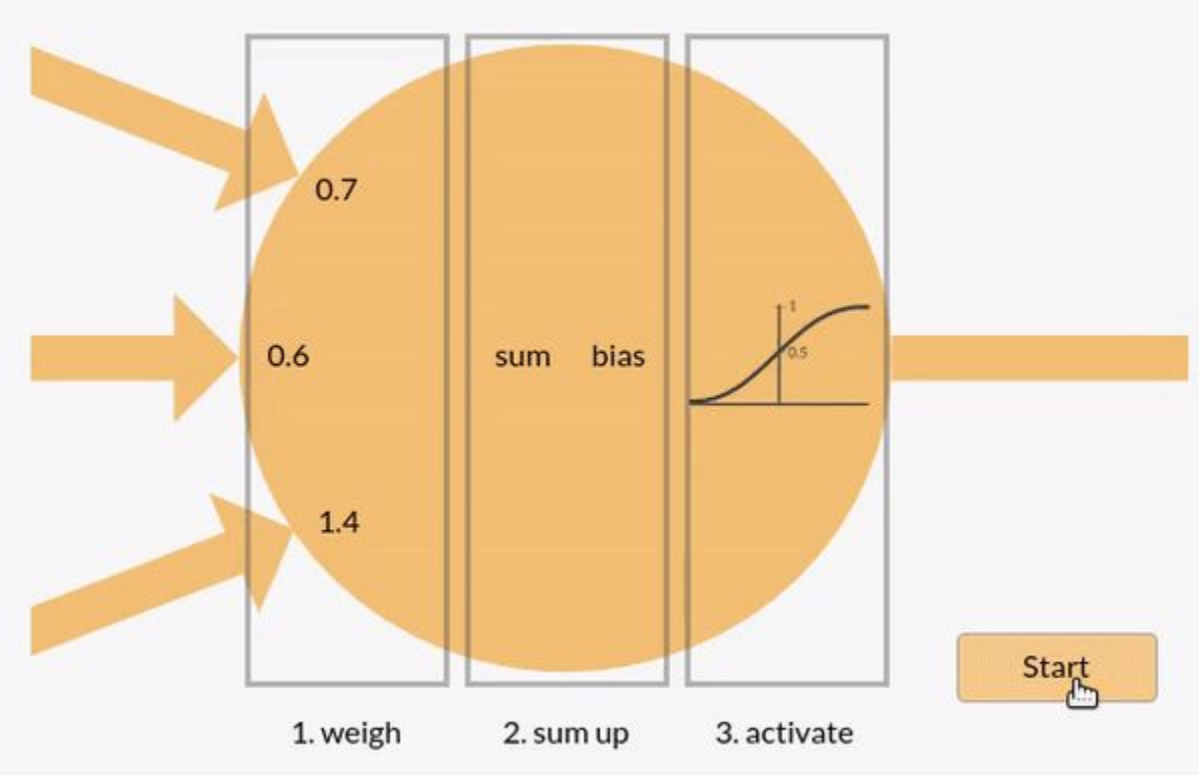
- Componentes básicos:
  - Neurônio: unidade computacional básica da rede
  - Arquitetura: estrutura topológica de como os neurônios são conectados
  - Aprendizagem: processo que adapta a rede de modo a computar uma função desejada, ou realizar uma tarefa

Neurônio Artificial

Objeto **x** com *d* atributos fornece entrada
Pesos para as entradas são dados pelo vetor **w**
É realizada uma soma ponderada da entrada, à qual é aplicada uma função de ativação, que fornece a saída final (previsão)



Funções de Ativação

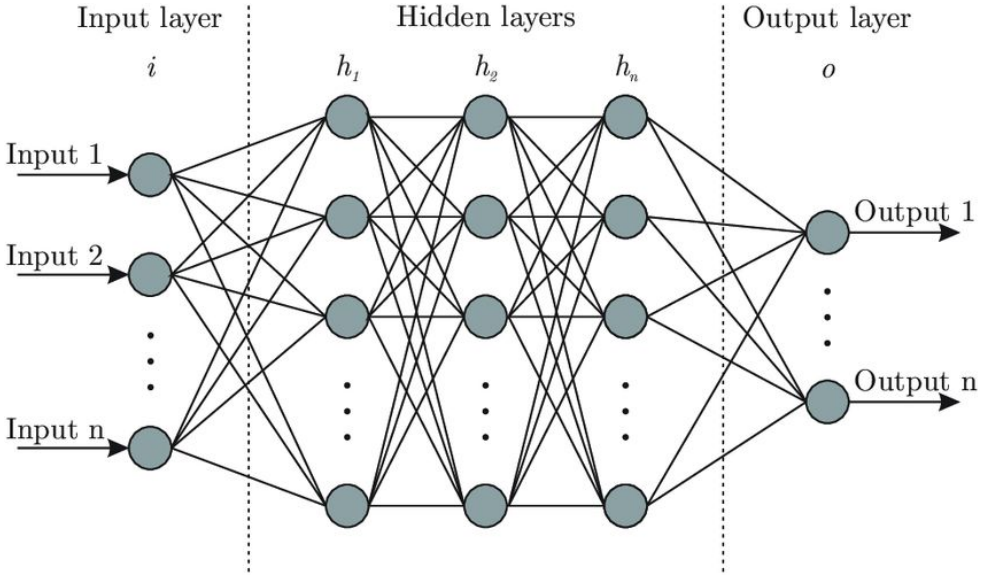


Topologia

- Definida por:
  - Número de camadas da RNA
  - Número de neurônios em cada camada
  - Grau de conectividade dos neurônios
  - Presença ou não de conexões de retropropagação

Topologia

- Neurônios podem estar dispostos em camadas
  - Neurônio pode receber como entrada a saída de neurônios da camada anterior
  - E enviar sua saída para entrada de neurônios em camada seguinte



Função custo

Toda rede neural gera uma saída.

Essa saída deve ser analisada e comparada com o dado que se tem sobre o objeto que foi classificado. Para realizar essa tarefa, é utilizada uma função custo da rede para um conjunto de dados é:

$$L = \frac{1}{N} \sum_{i} L_i(f(x_i, W), y_i)$$

Essa função mede o quão diferente a saída da rede está do rótulo real dos objetos.

Aprendizado

Quatro grupos de algoritmos:

Correção de erros: procuram ajustar pesos para reduzir erros cometidos pela rede (supervisionado)

Hebbiano: baseados na regra de Hebb, se dois neurônios estão simultaneamente ativos, a conexão entre eles deve ser reforçada

(não supervisionado)

Competitivo: promovem competição entre neurônios para definir quais terão pesos ajustados, geralmente os que respondem mais fortemente à entrada

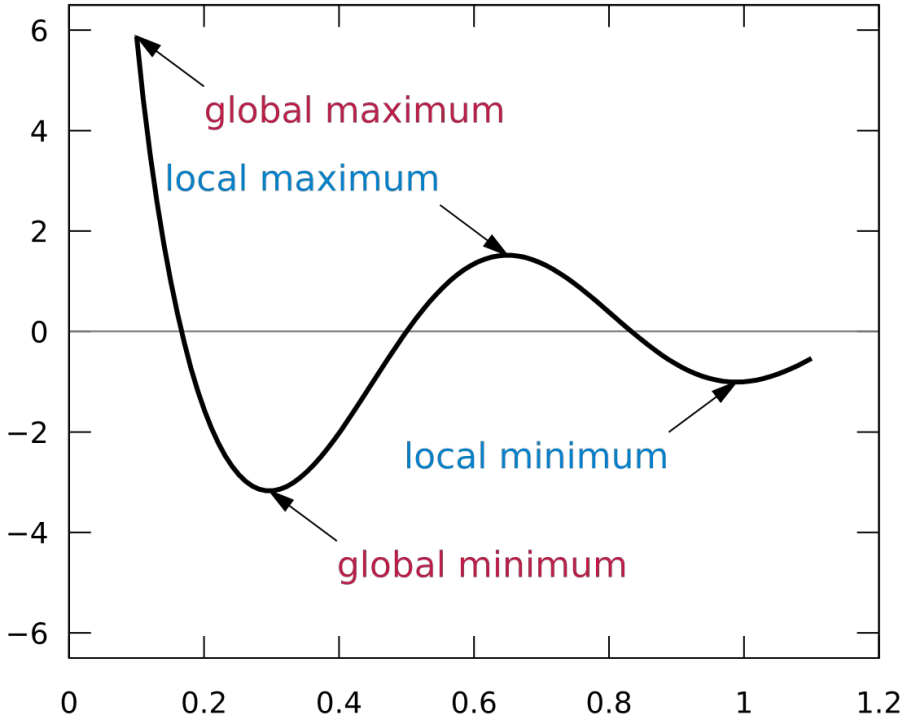
(não supervisionado)

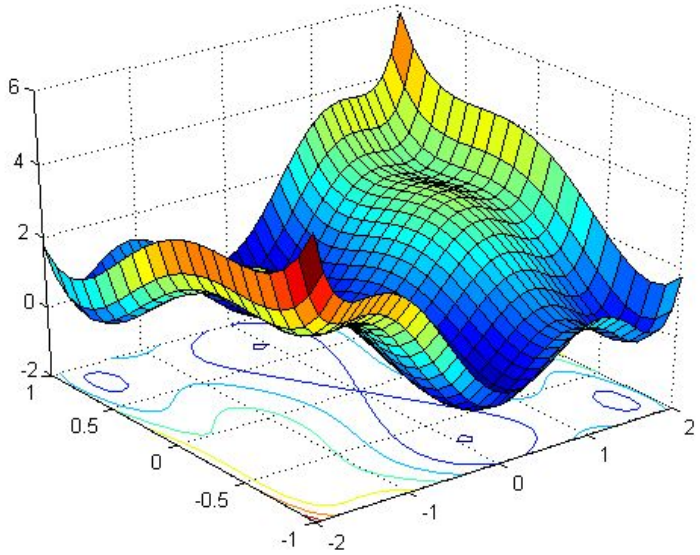
Termodinâmico (Boltzmann): algoritmos estocásticos baseados em princípios observados na metalurgia

Aprendizado por Correção de Erro

- Superfície de erro
  - Superfície multi-dimensional representando gráfico da função de custos *X* peso
  - Objetivo do aprendizado:
    - *A partir de um ponto qualquer da superfície, mover em direção a um mínimo global*

Superfície de erro





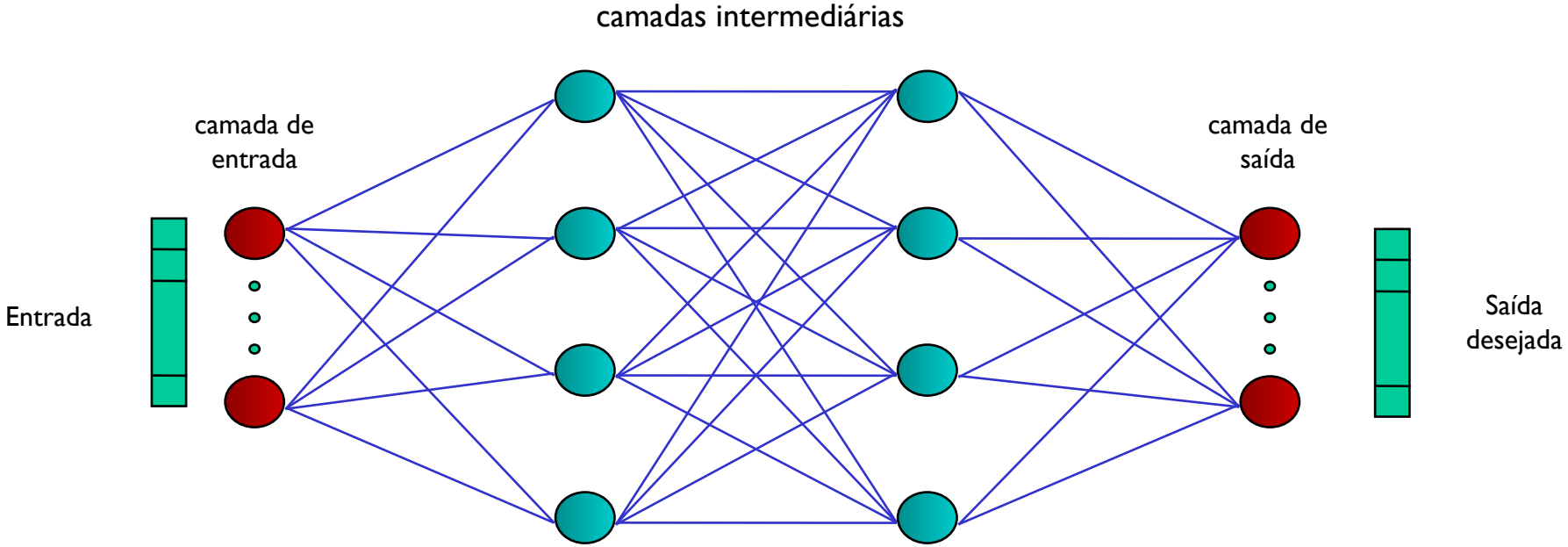
Backpropagation

- Treinamento por correção de erros
  - Camada de saída: comparação entre vetor de saída dos neurônios e vetor de valores desejados
    - Classificação: rede classifica objeto corretamente quando a saída mais elevada é a do neurônio correspondente à classe correta do exemplo
      - Se valores são baixos ou mais de um neurônio dá valor de saída alto, a rede não tem condições de prever
  - Camadas intermediárias: Qual a saída desejada de uma camada intermediária?
    - Algoritmo *Backpropagation* (retropropagação de erros)

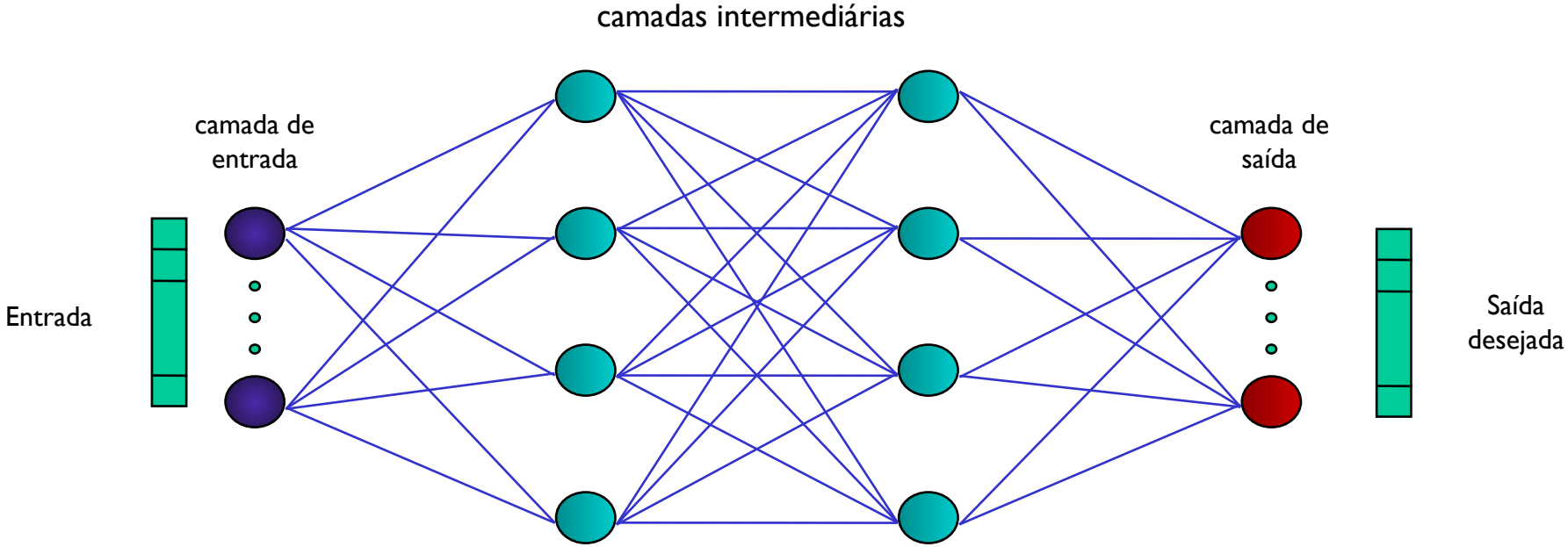
Backpropagation

- Treinamento: iteração de duas fases
  - Cada fase percorre a rede em dois sentidos
    - Sinal (forward)
    - Erro (backward)

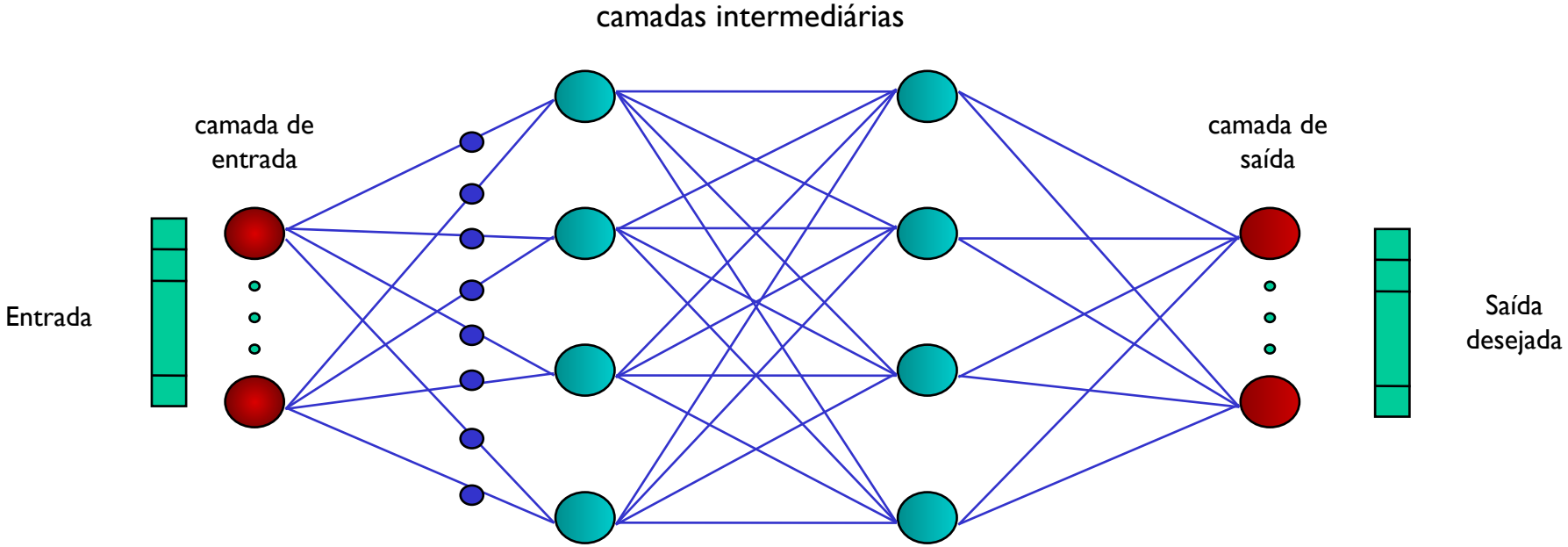
Aprendizado



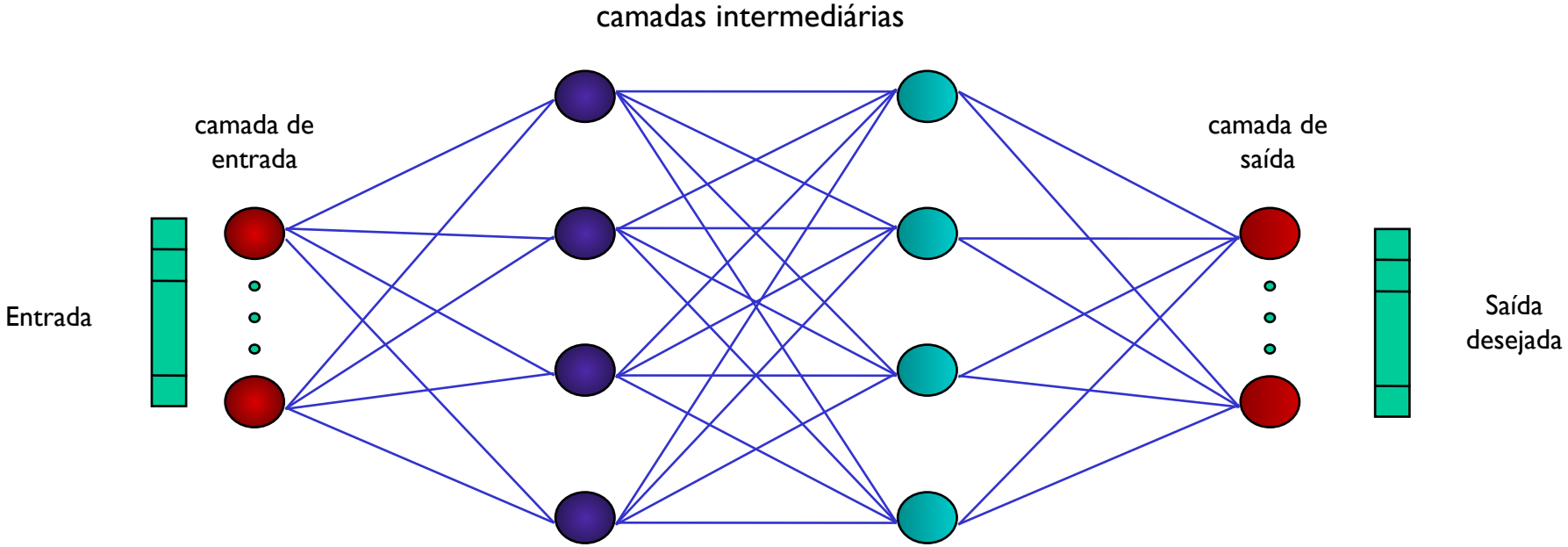
Aprendizado



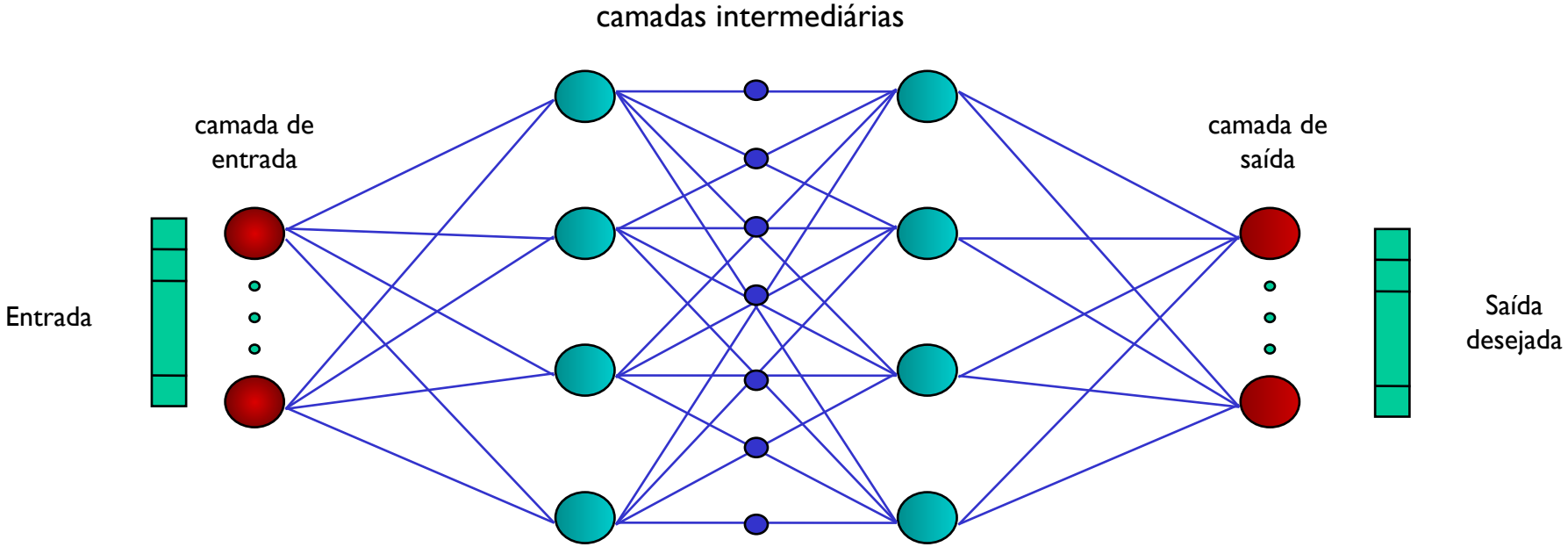
Aprendizado



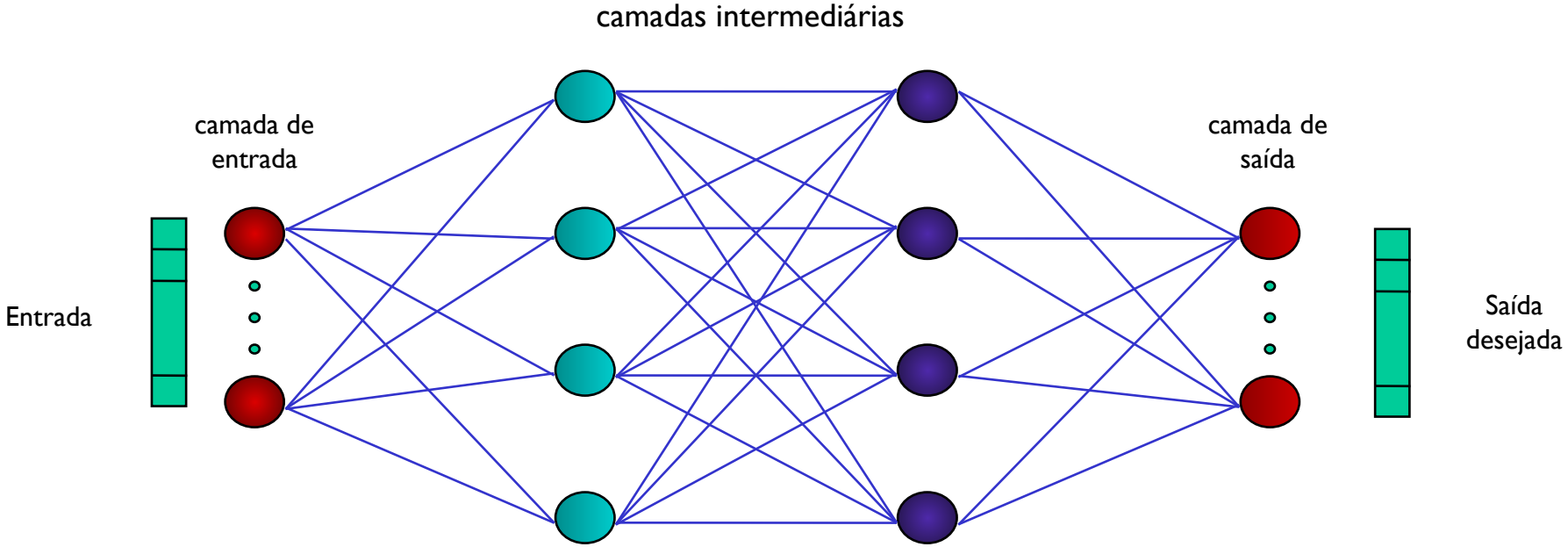
Aprendizado



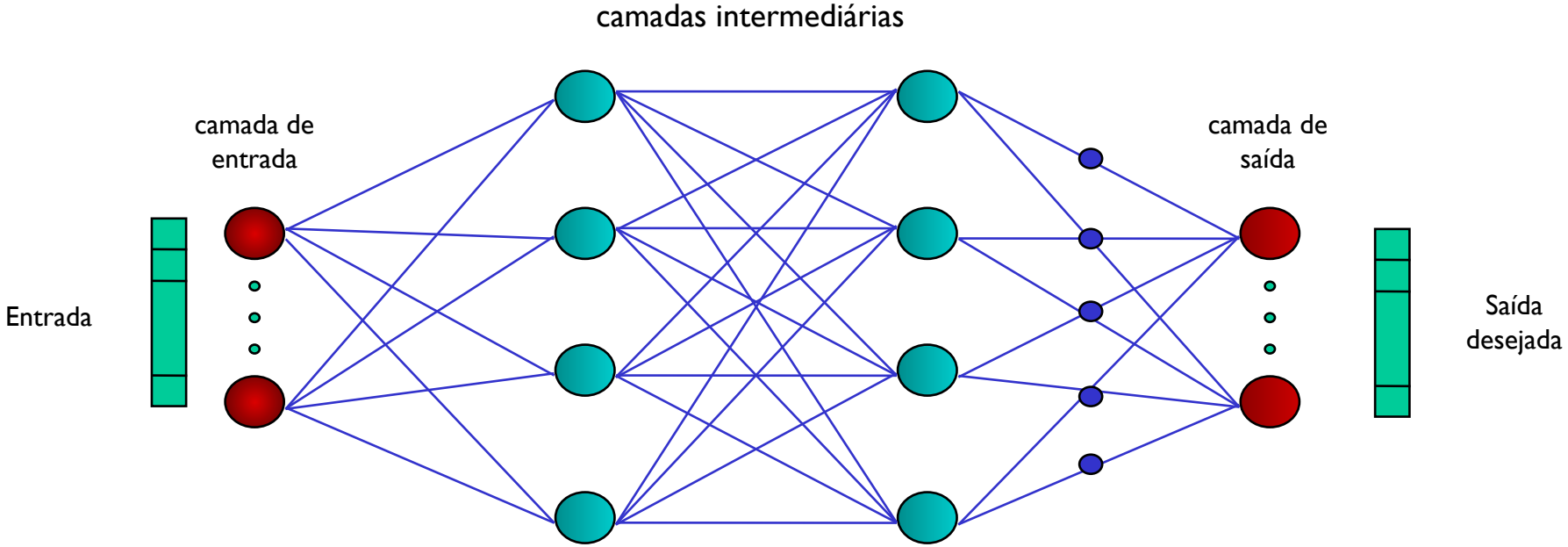
Aprendizado



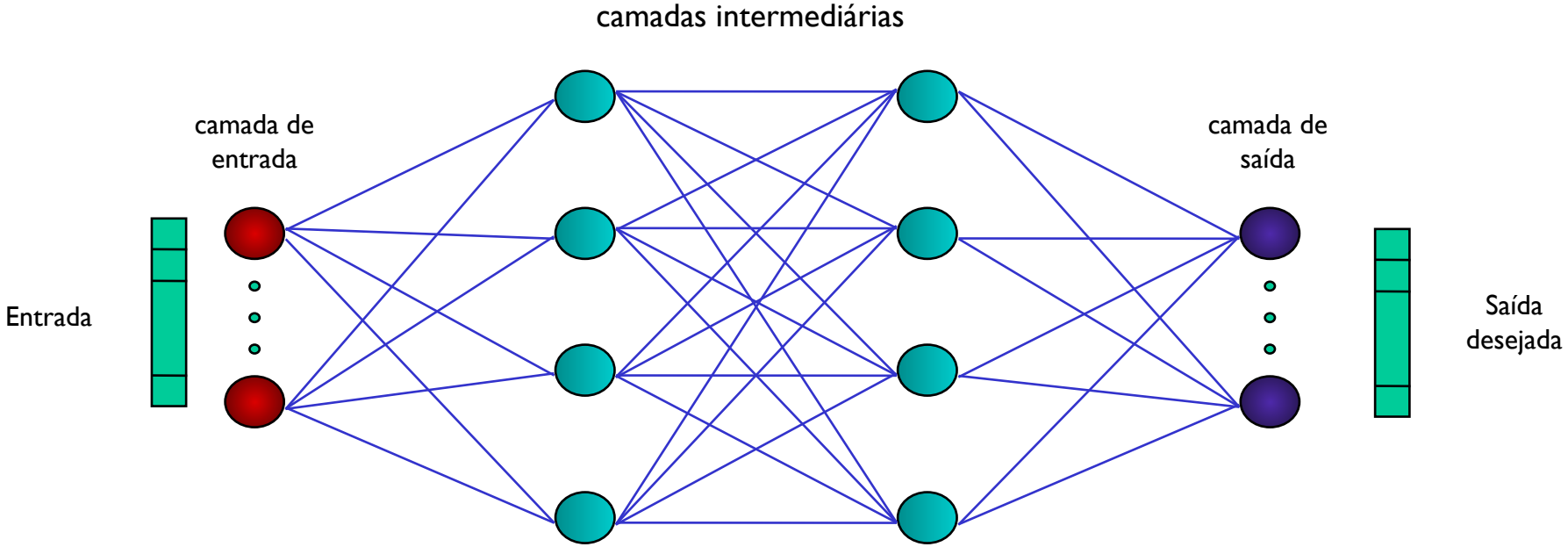
Aprendizado



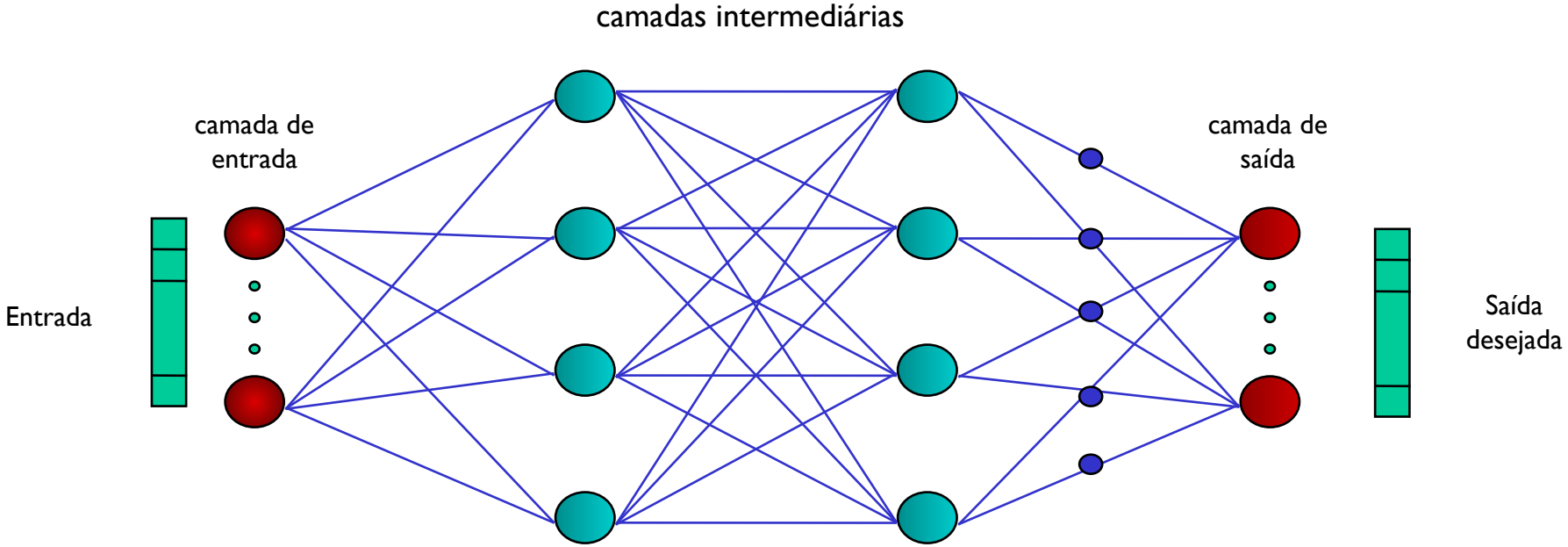
Aprendizado



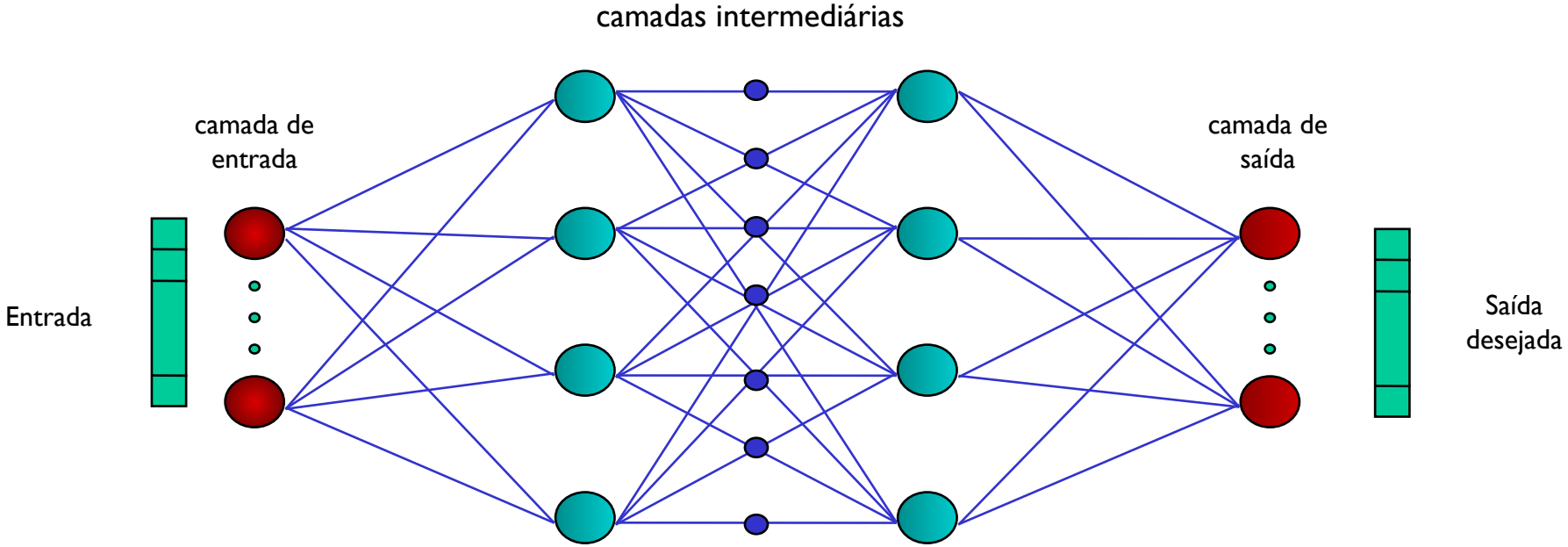
Aprendizado



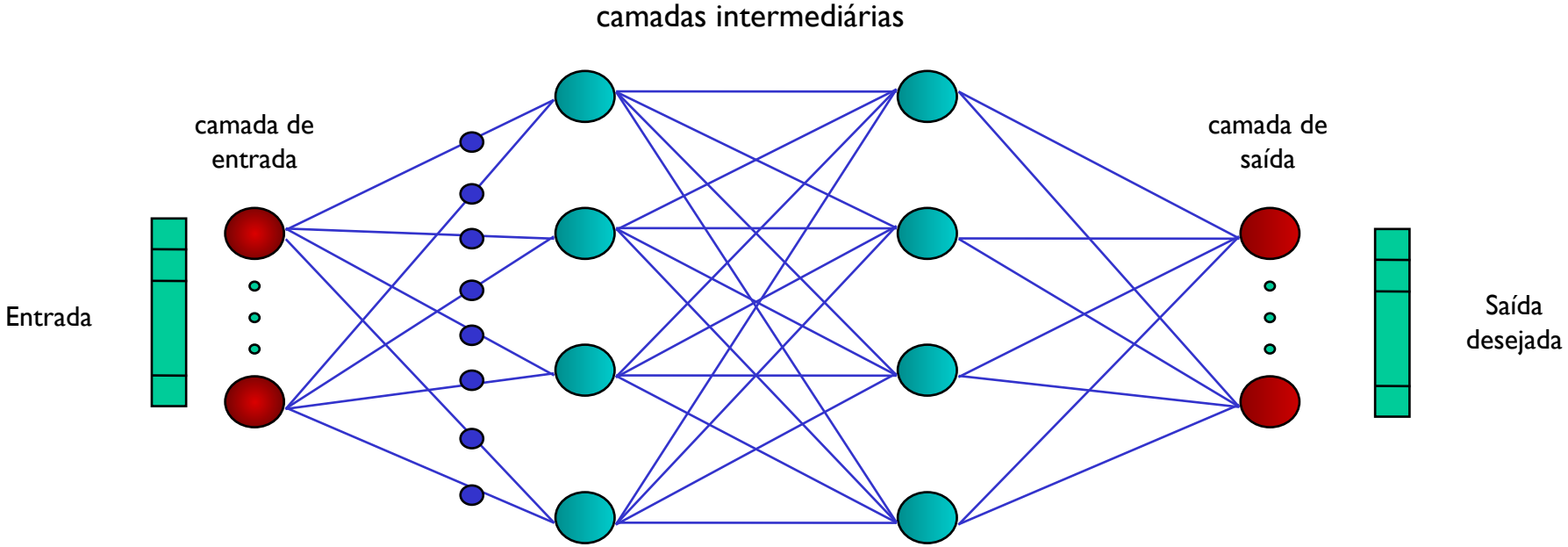
Aprendizado



Aprendizado



Aprendizado



Aprendizado

- Versão padrão: ajuste de pesos para cada objeto individualmente

- Variação *batch*: pesos são ajustados uma única vez para cada ciclo

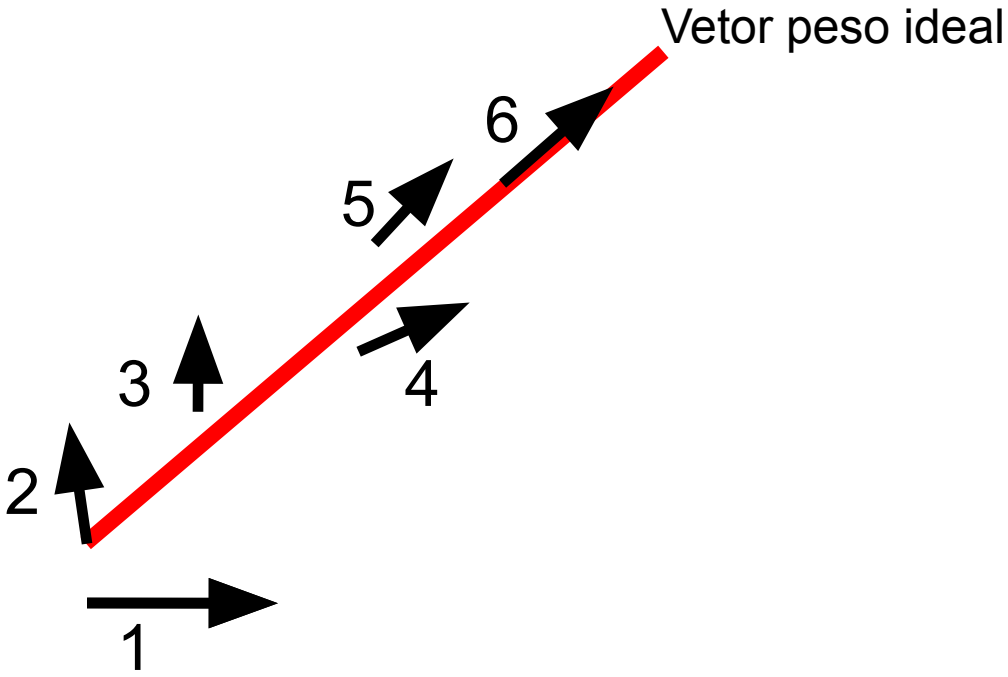
Aprendizado

- Ajuste dos pesos das conexões

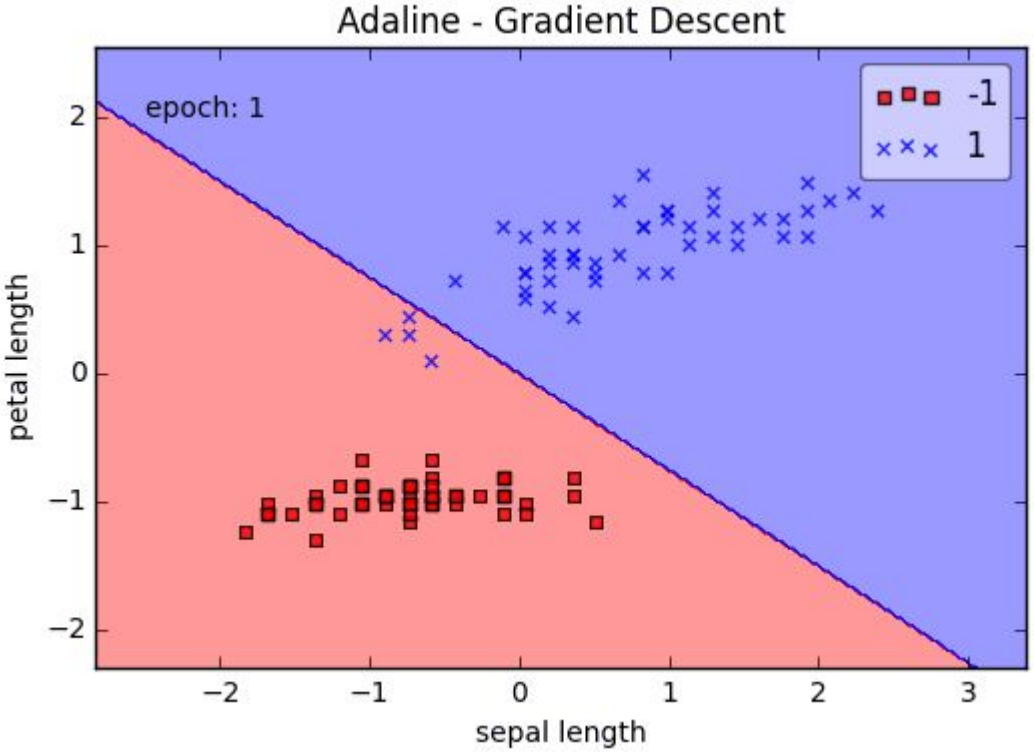
$$w(t+1) = w(t) + \nabla w(t)$$

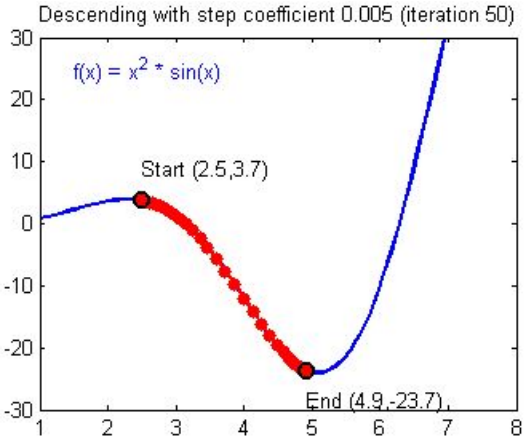
- Algoritmos de aprendizado
  - Conjunto de regras bem definidas para ensinar a rede a resolver um dado problema
  - Divergem na maneira como os pesos são ajustados
    - Em como ∇w é calculado

Treinamento

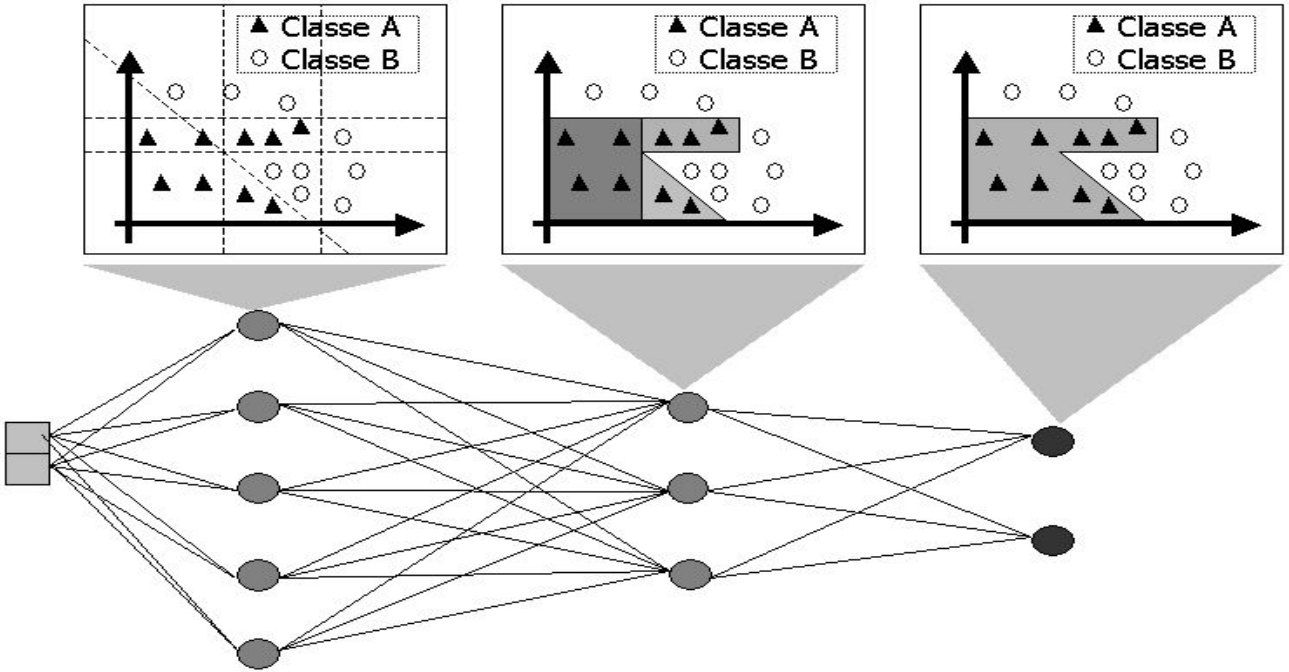


Treinamento



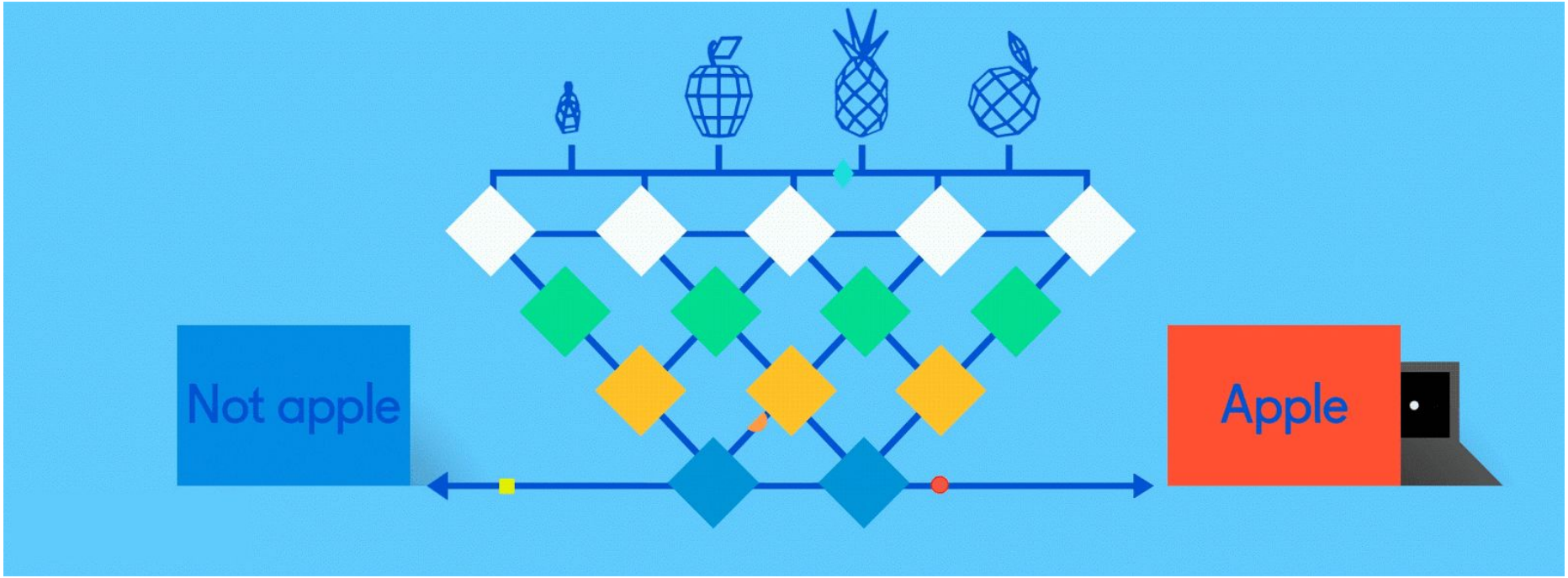


Intuição



Combinação das funções desempenhadas por cada neurônio define a função associada à RNA

MLP

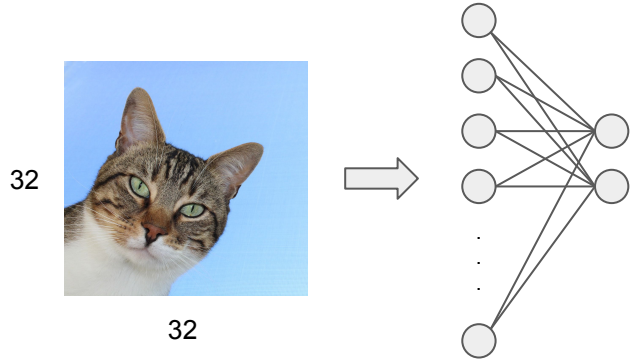


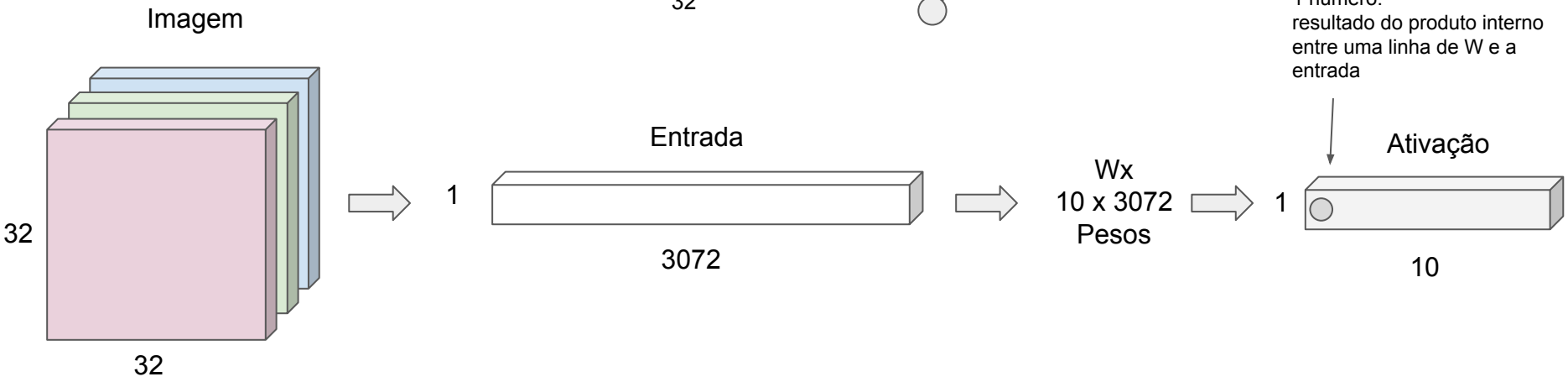
Considerações

- Mínimos locais: solução estável que não é a melhor solução
  - Incidência pode ser reduzida
    - Empregando taxa de aprendizado decrescente
    - Adicionando nós intermediários
    - Utilizando termo de momentum
- Backpropagation é muito lento em superfícies complexas
  - Utilizar métodos de segunda ordem
    - Outros algoritmos
    - Ex.: RPROP, Newton, etc.

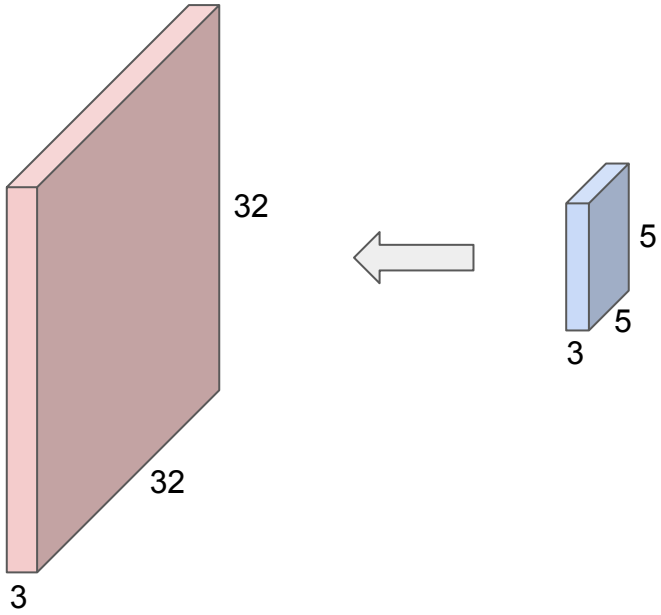
Redes Neurais Convolucionais

Redes "convencionais"





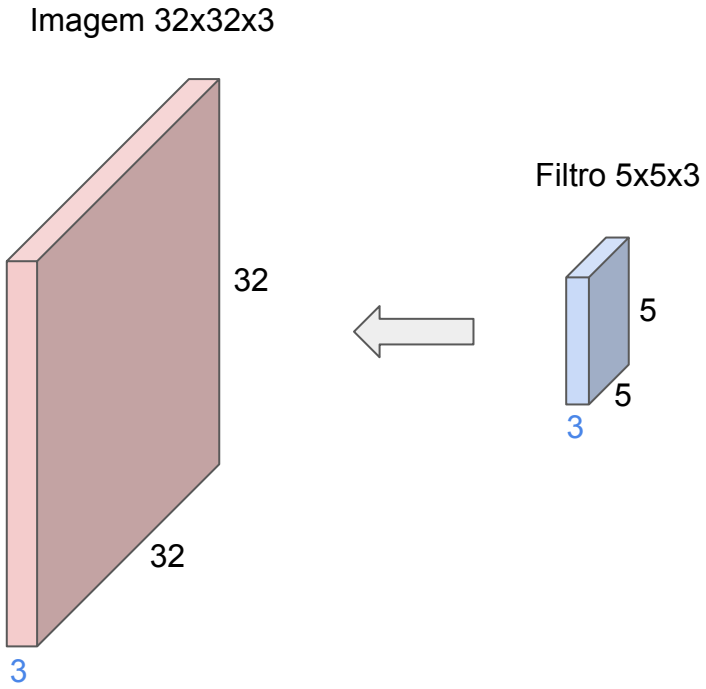
Camada convolucional



Convolução do filtro com a imagem = deslizar o filtro na sobre a imagem computando produtos internos.

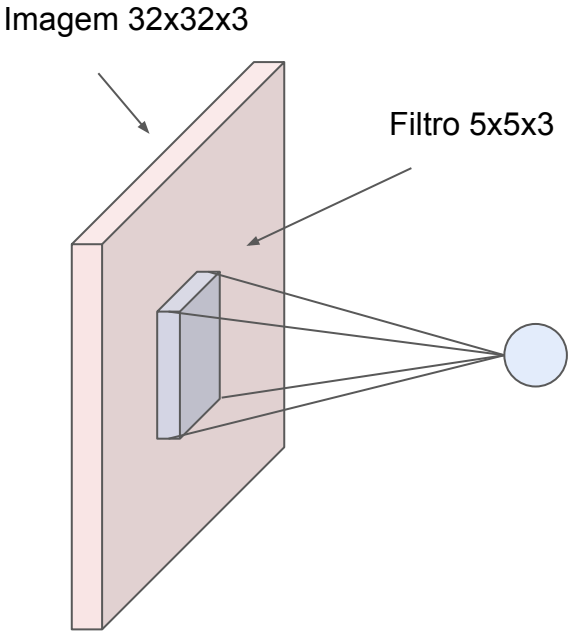
A ideia leva em consideração a estrutura espacial da imagem.

Camada convolucional



Filtros sempre possuem a mesma profundidade que a entrada.

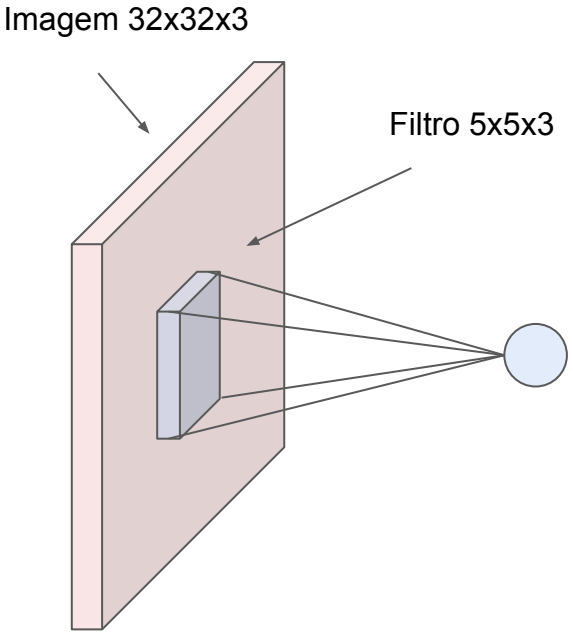
Camada convolucional



1 número:
resultado do produto interno entre um filtro 5x5x3 com uma porção 5x5x3 da imagem.

$w^T x$

Camada convolucional

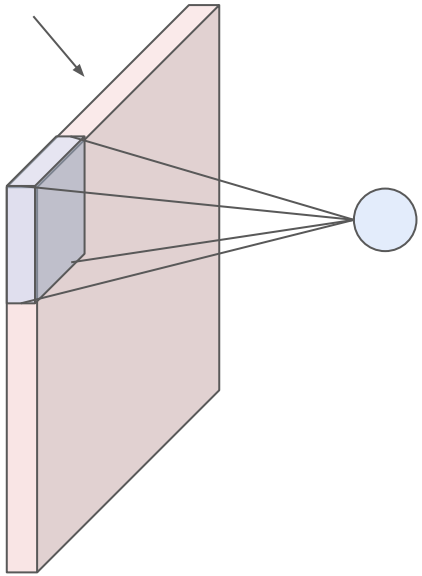


1 número:
resultado do produto interno entre um filtro 5x5x3 com uma porção 5x5x3 da imagem.
Se considerar o bias, temos um produto interno de dois vetores com dimensão 75 (5x5x3) + 1

$w^T x + b$

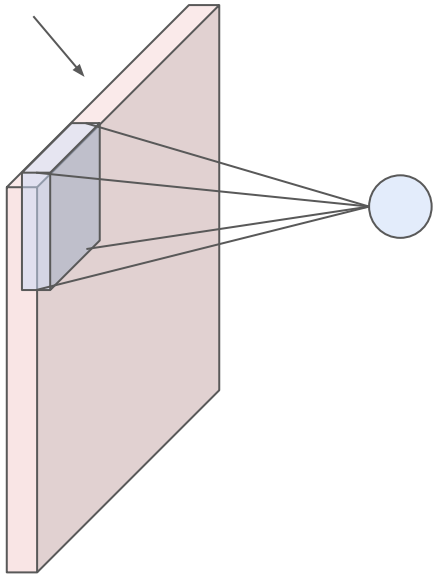
Camada convolucional

Imagem 32x32x3



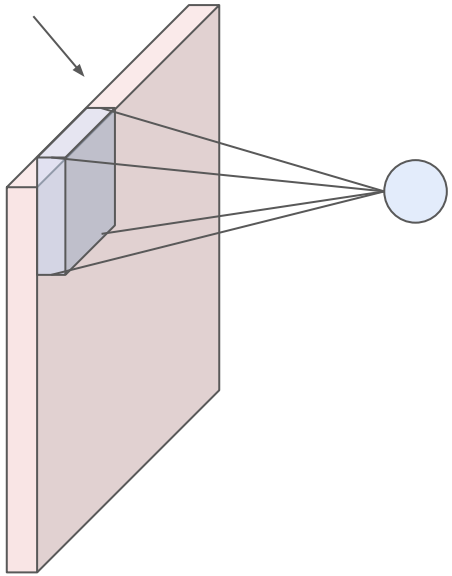
Camada convolucional

Imagem 32x32x3



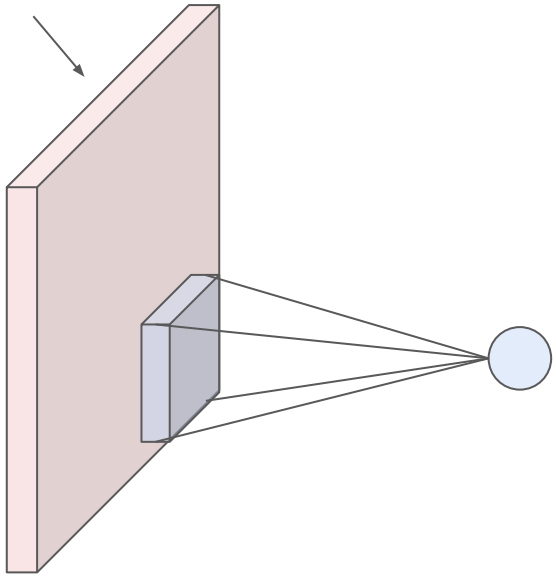
Camada convolucional

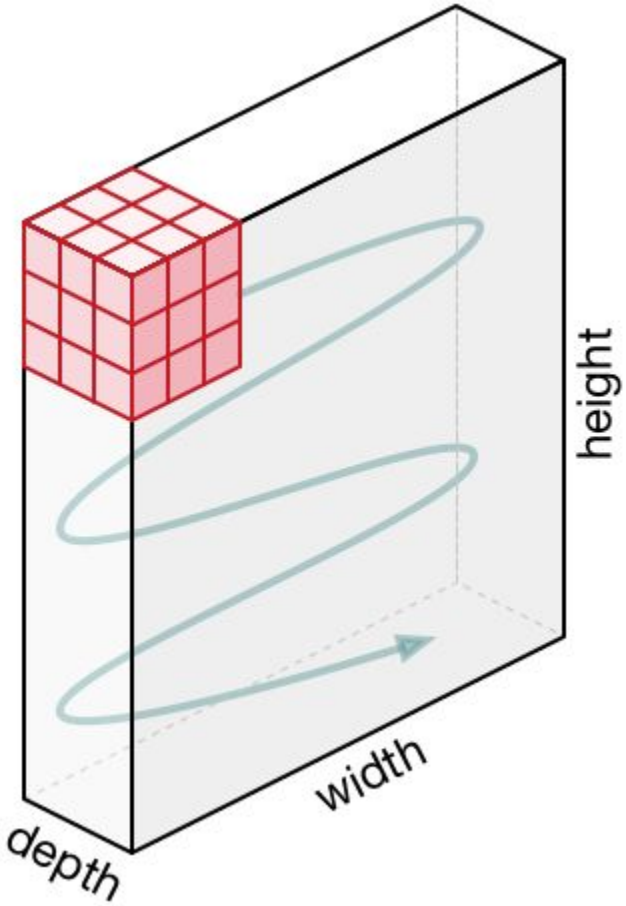
Imagem 32x32x3

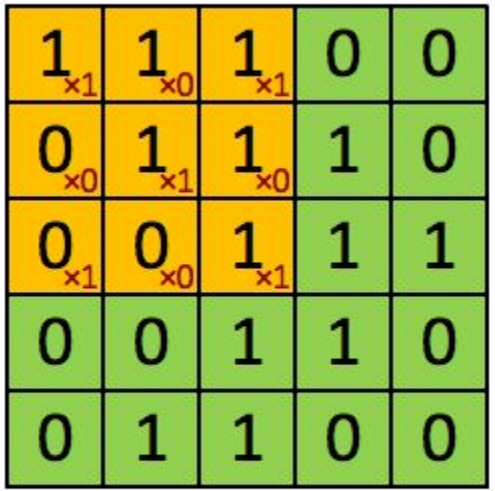


Camada convolucional

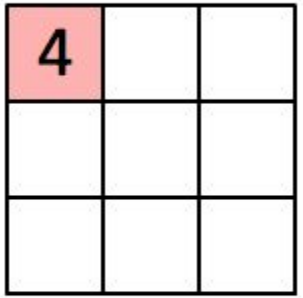
Imagem 32x32x3




height
width
depth

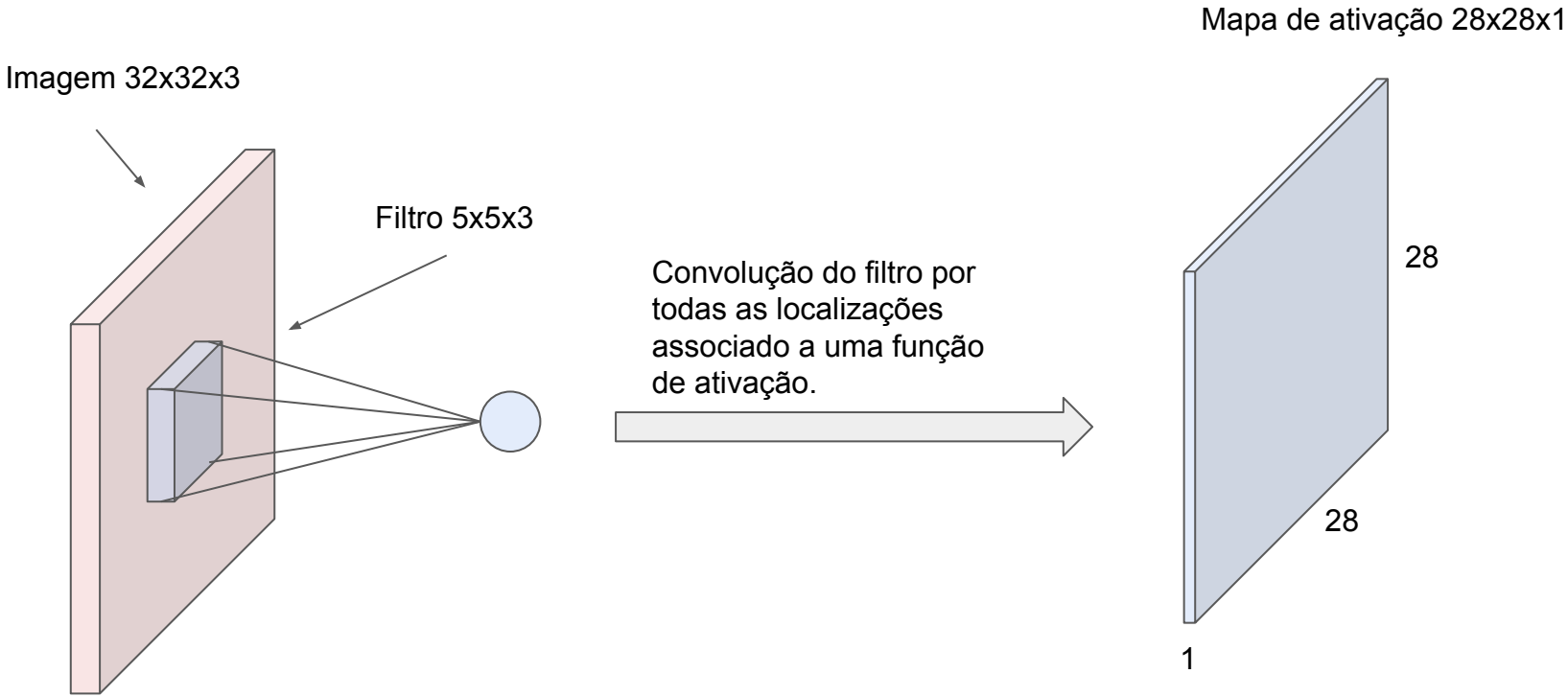


Image

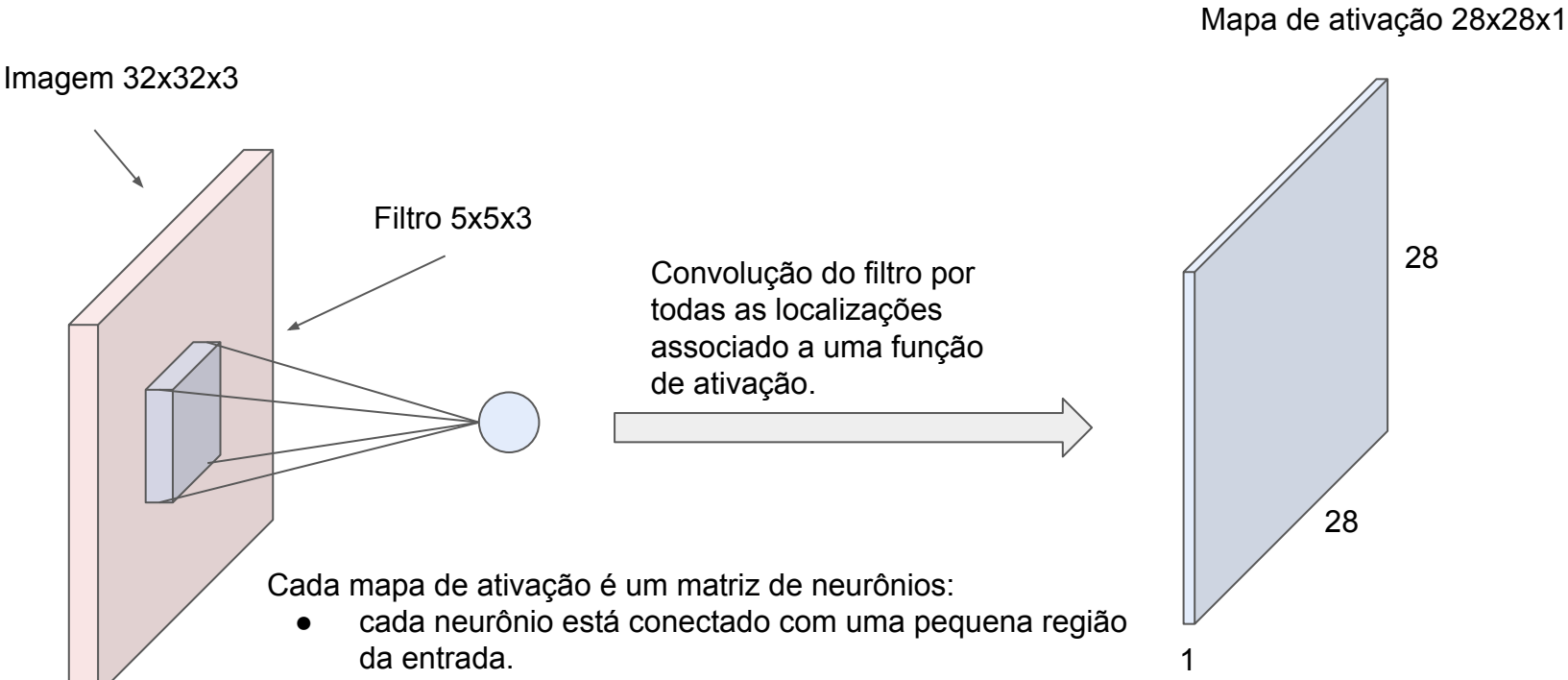


Convolved Feature

Camada convolucional



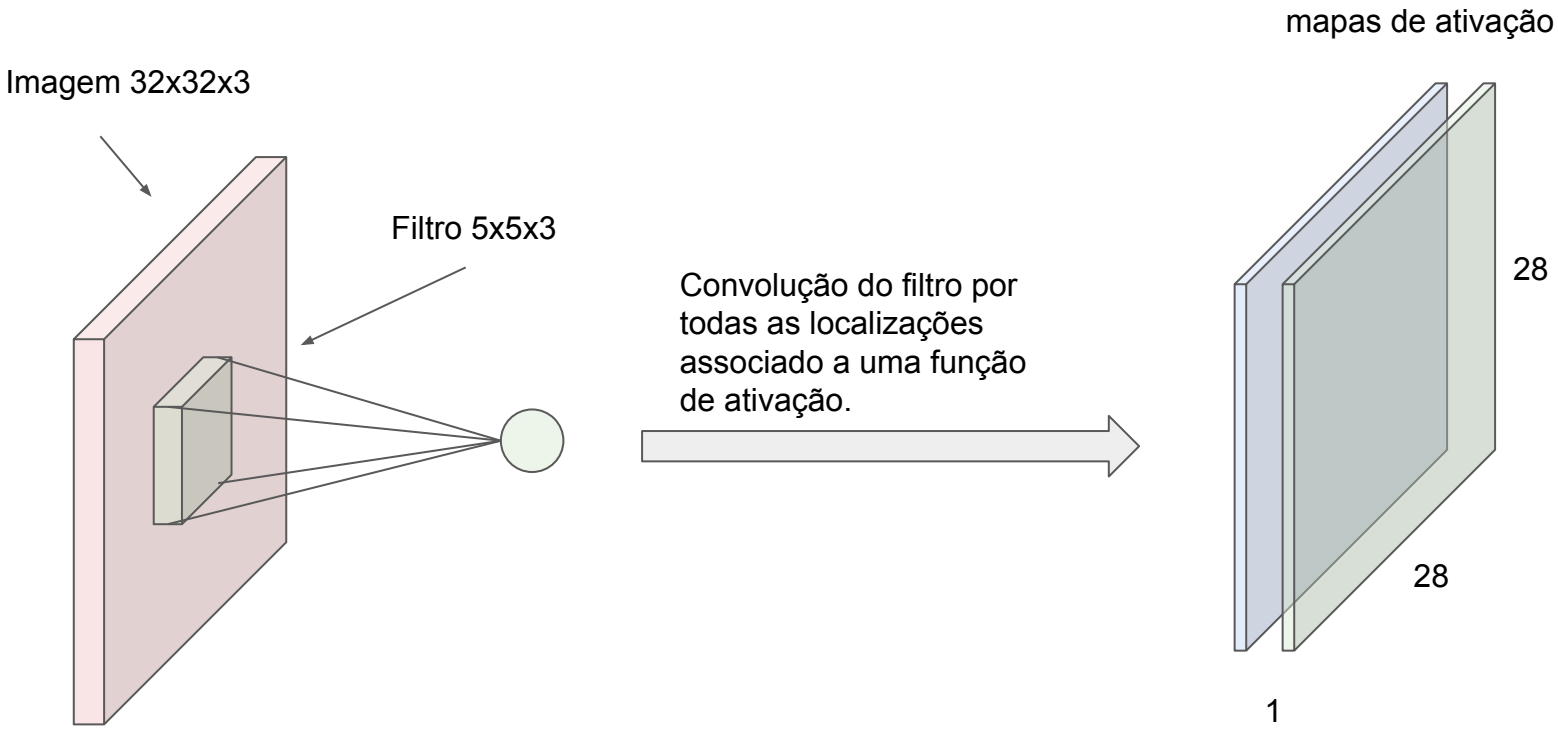
Camada convolucional



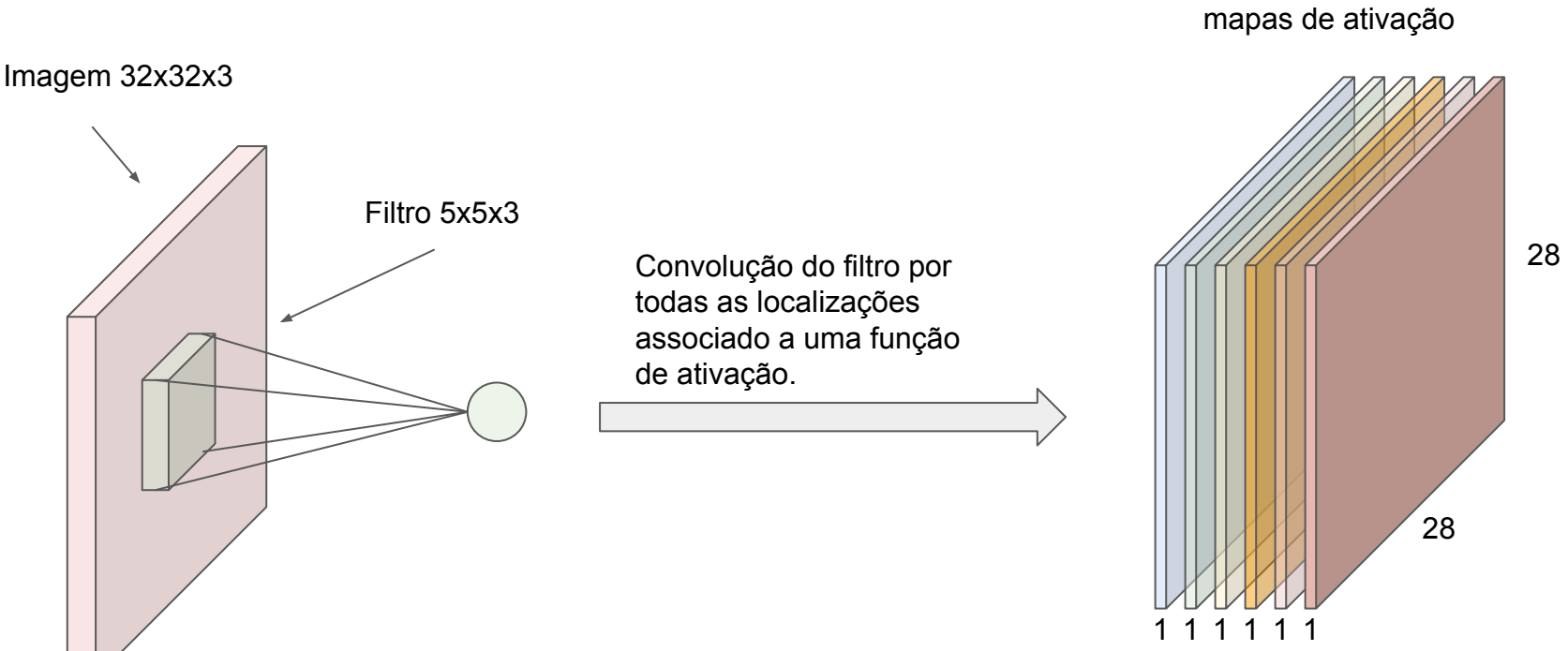
Cada mapa de ativação é um matriz de neurônios:

- cada neurônio está conectado com uma pequena região da entrada.
- todos compartilham os mesmos parâmetros.
- um filtro representa um campo receptivo para cada neurônio.

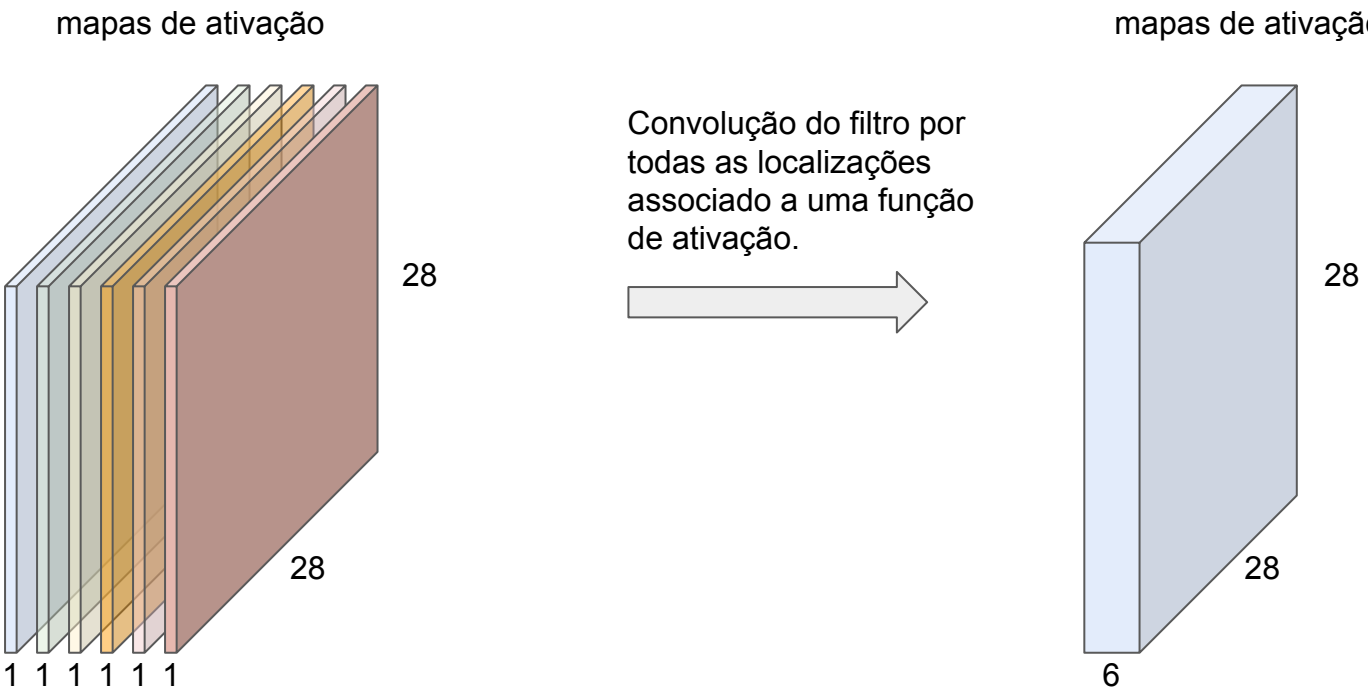
Camada convolucional



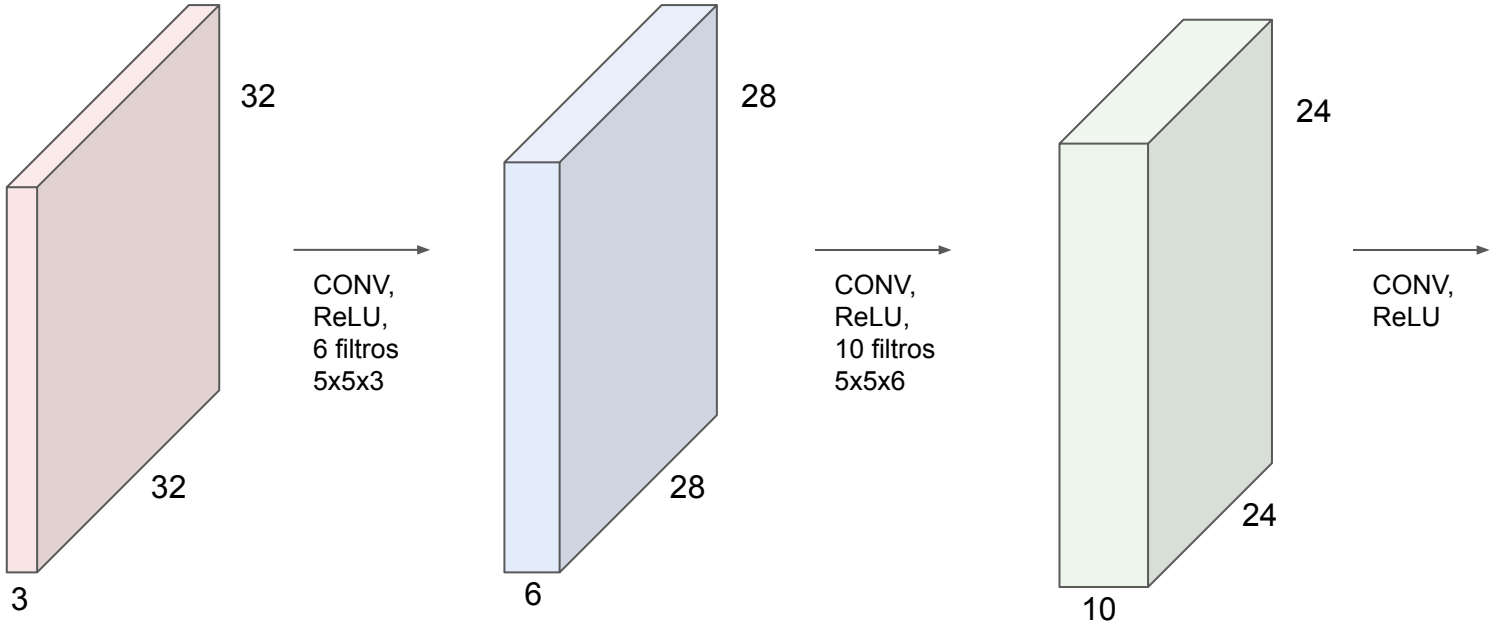
Camada convolucional

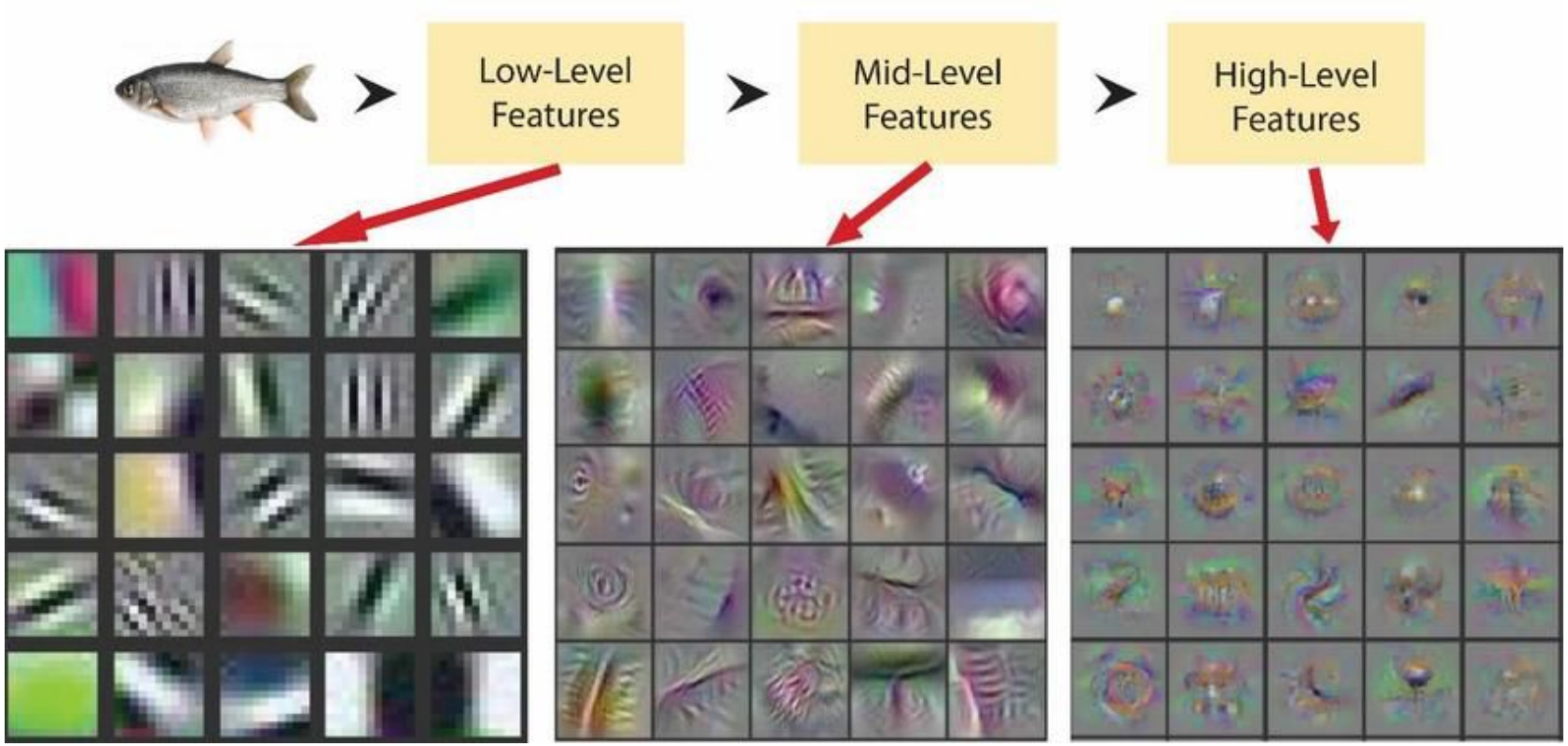


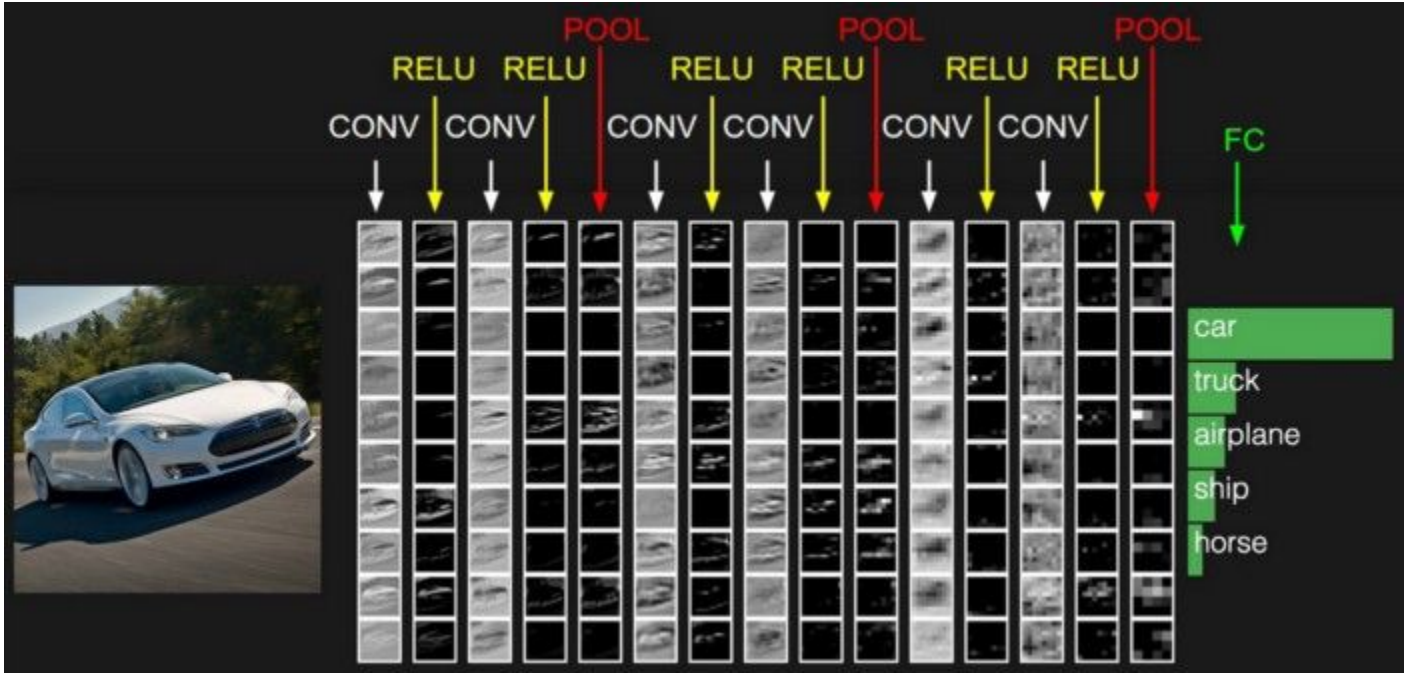
Camada convolucional



Uma rede convolucional consiste em uma sequência de camadas convolucionais combinadas com funções de ativação.

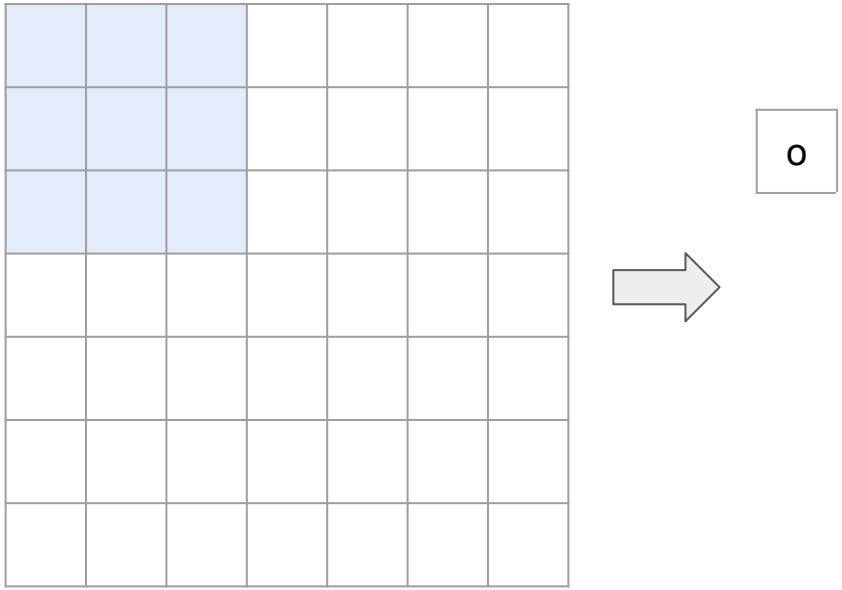



Low-Level Features
Mid-Level Features
High-Level Features


POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

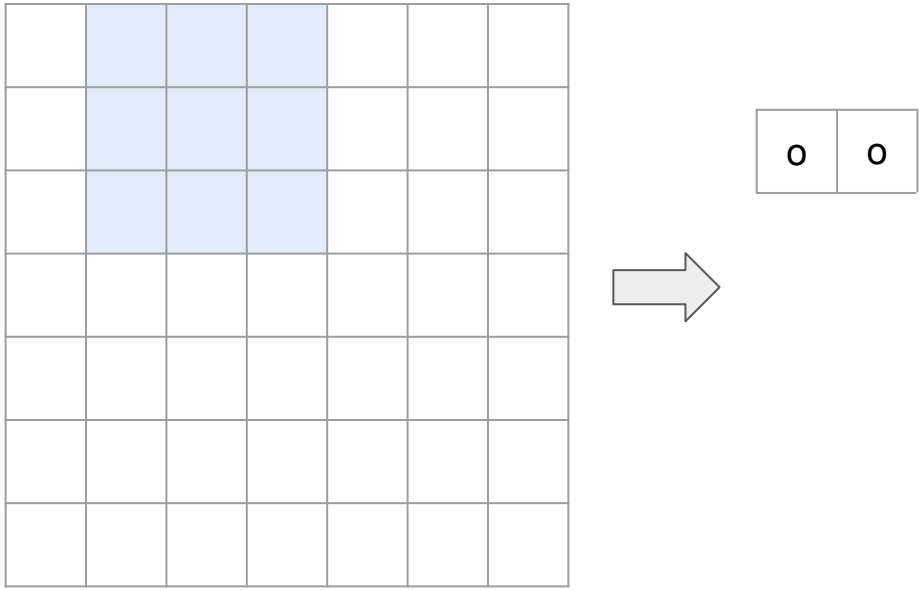
Dimensões

Imagem 7 x 7 com filtro de 3x3



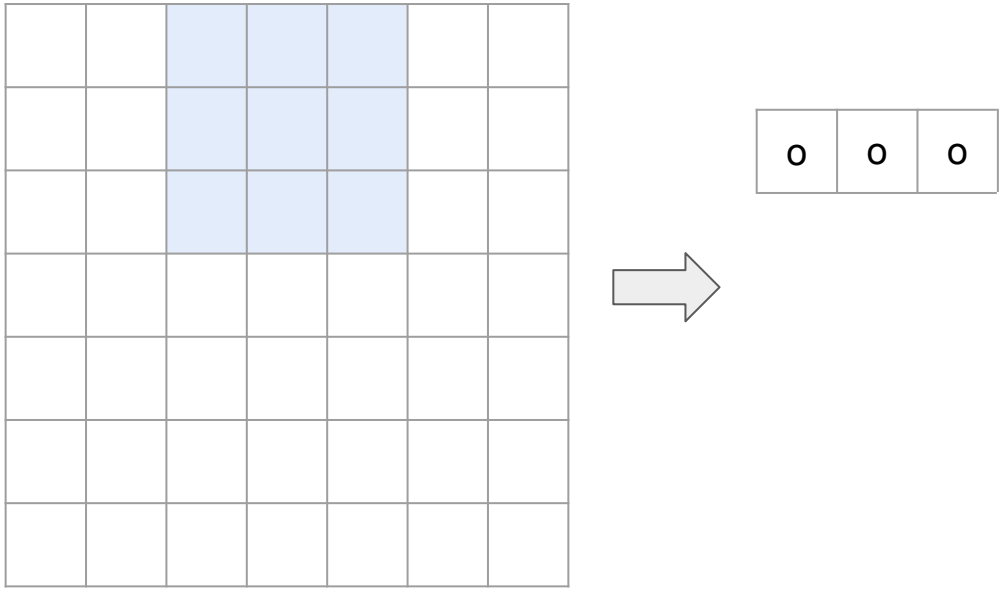
Dimensões

Imagem 7 x 7 com filtro de 3x3



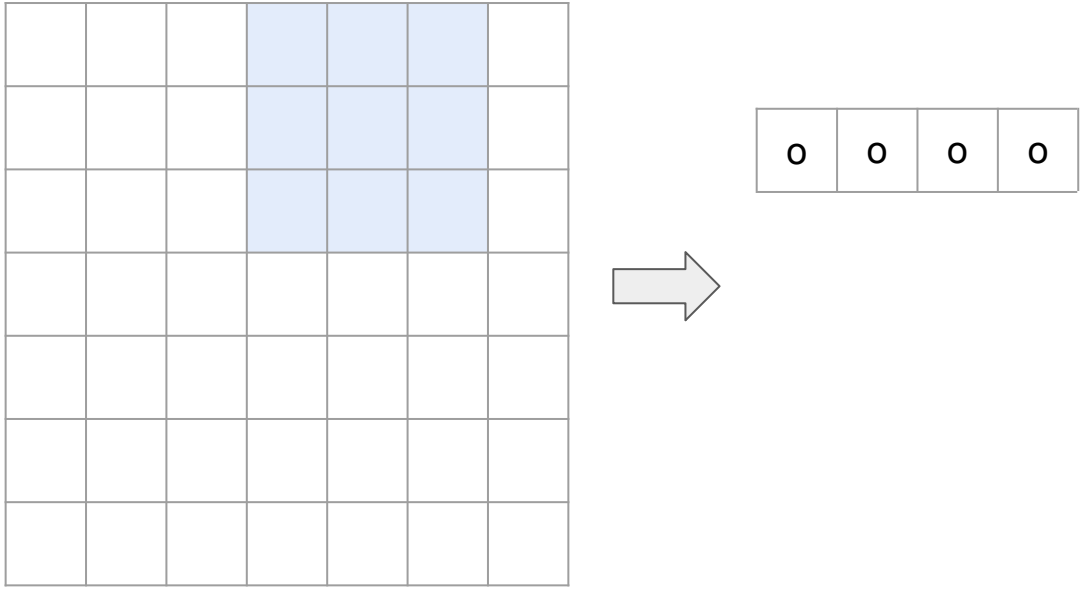
Dimensões

Imagem 7 x 7 com filtro de 3x3



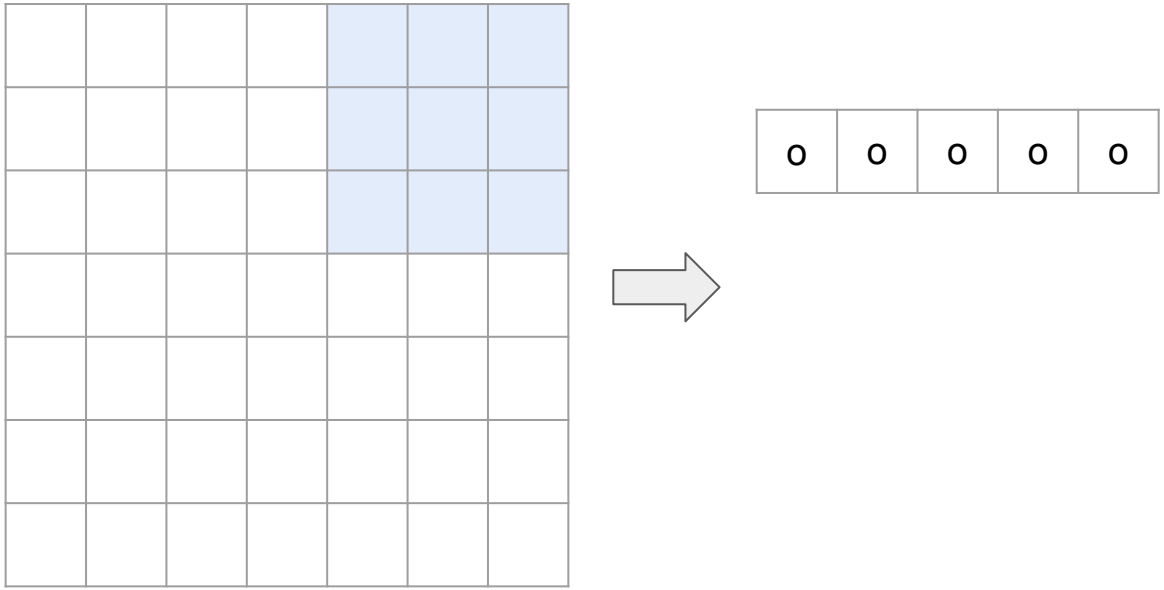
Dimensões

Imagem 7 x 7 com filtro de 3x3



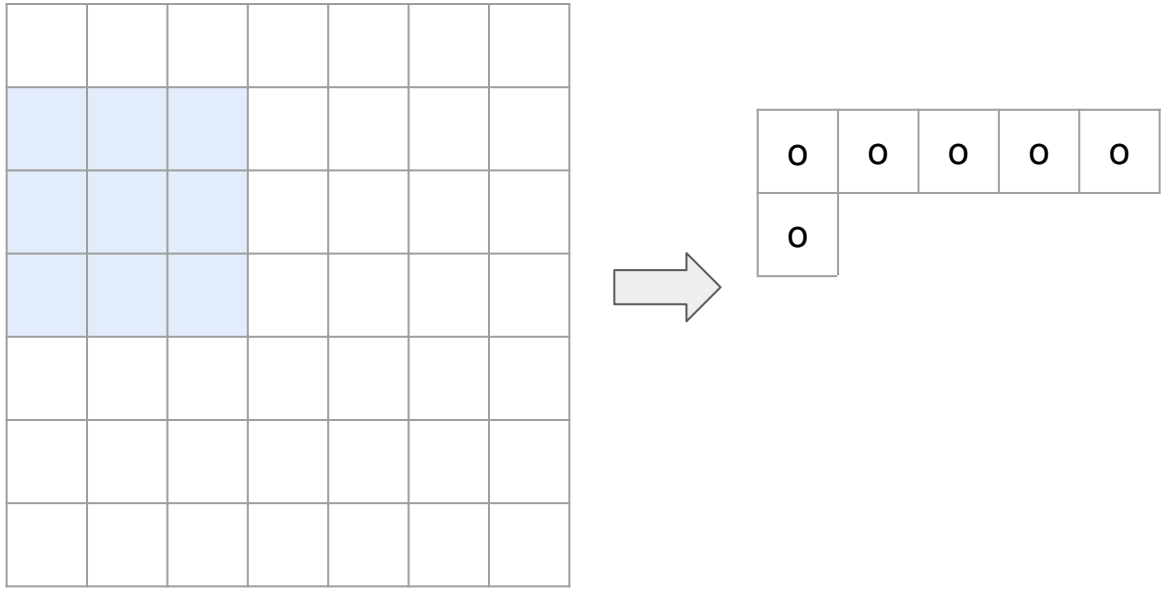
Dimensões

Imagem 7 x 7 com filtro de 3x3



Dimensões

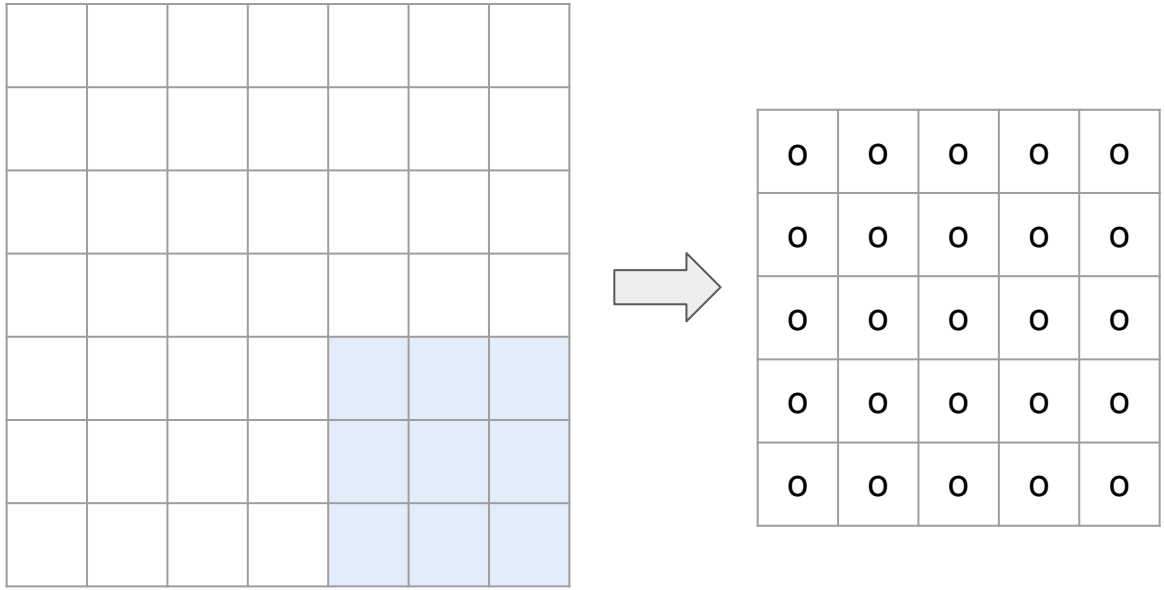
Imagem 7 x 7 com filtro de 3x3



Dimensões

Imagem 7 x 7 com filtro de 3x3.

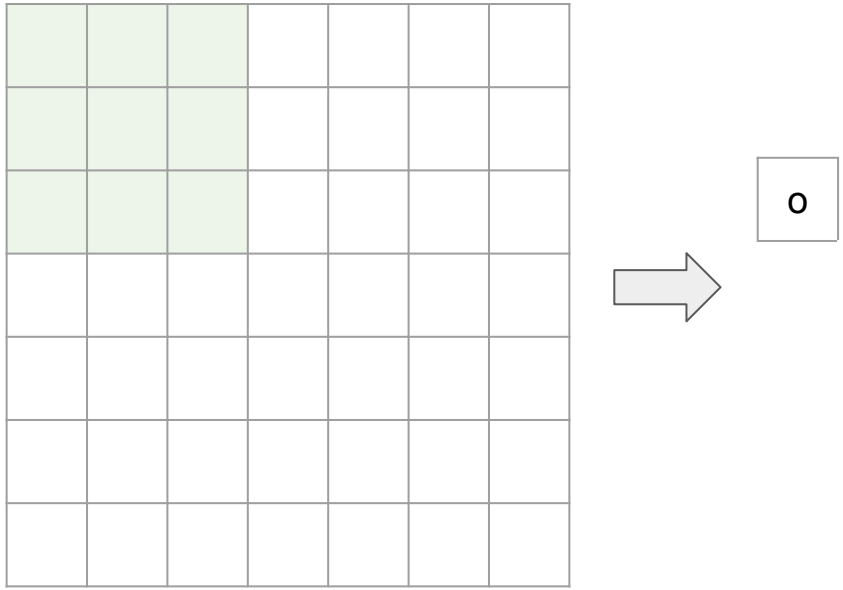
Mapa de ativação de saída com dimensões 5x5.



Dimensões

Podemos utilizar filtros diferentes com "passadas" (strides) distintas.

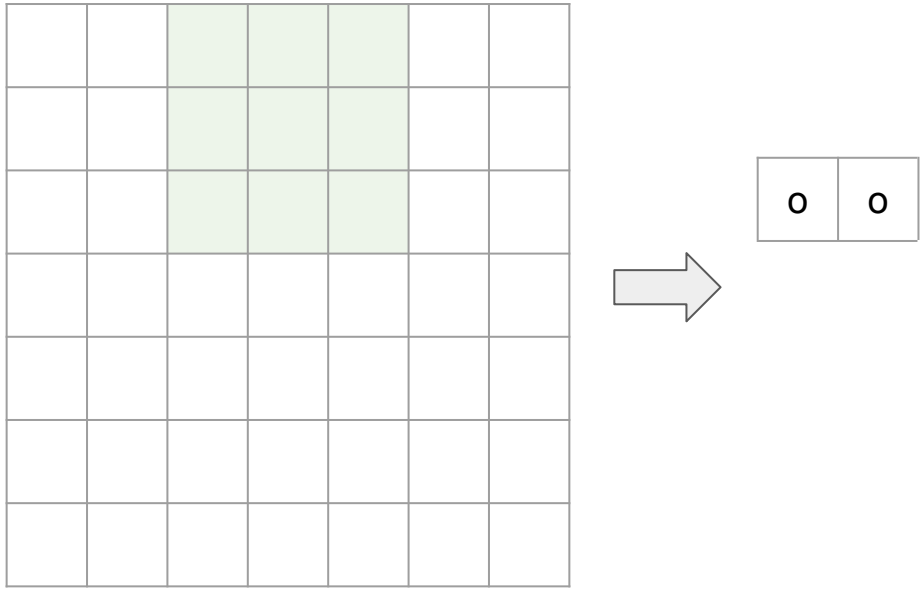
A mesma imagem 7x7 com um filtro 3x3, mas com stride 2.



Dimensões

Podemos utilizar filtros diferentes com "passadas" (strides) distintas.

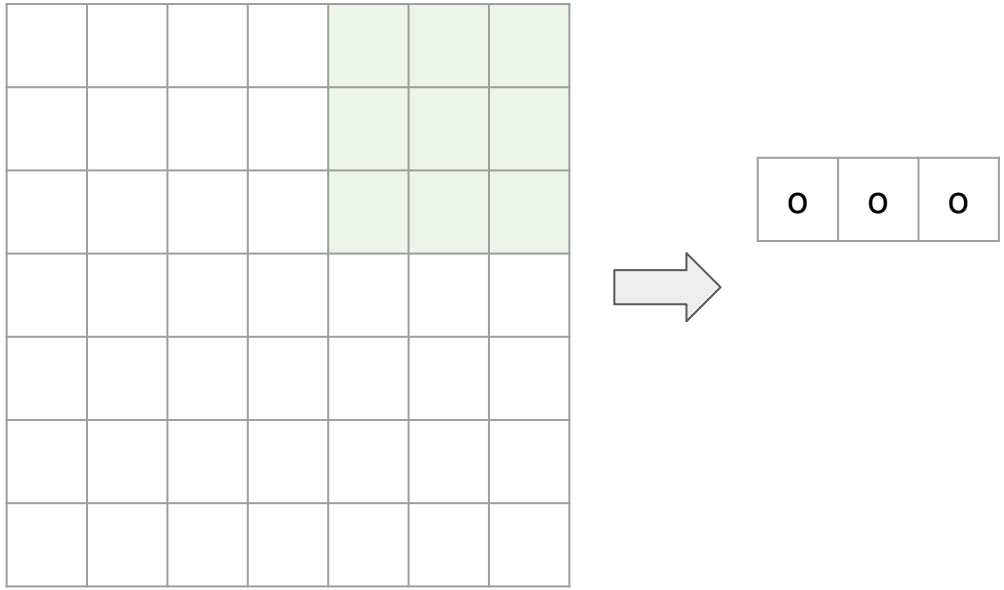
A mesma imagem 7x7 com um filtro 3x3, mas com stride 2.



Dimensões

Podemos utilizar filtros diferentes com "passadas" (strides) distintas.

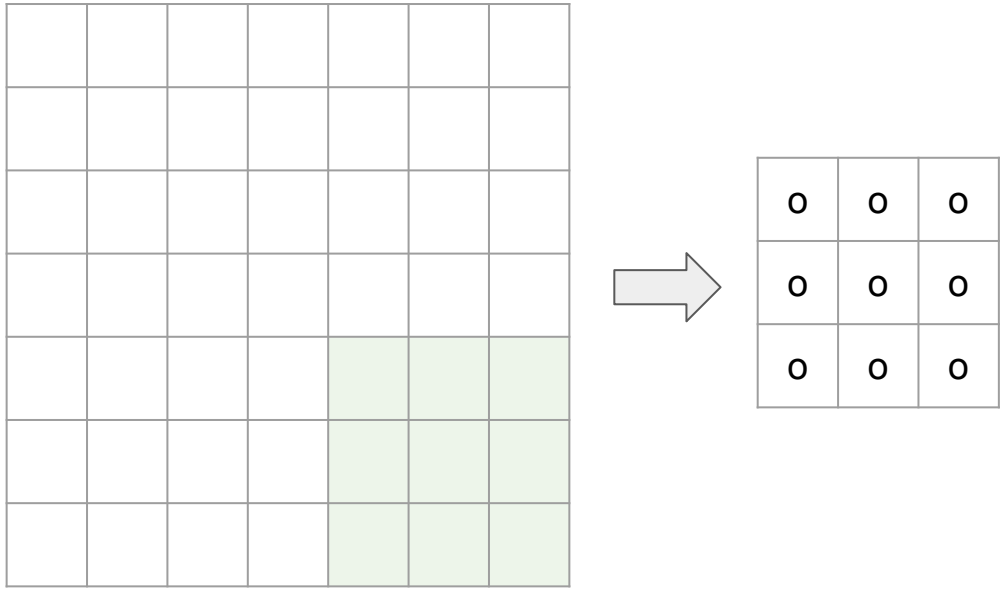
A mesma imagem 7x7 com um filtro 3x3, mas com stride 2.



Dimensões

Podemos utilizar filtros diferentes com "passadas" (strides) distintas.

A mesma imagem 7x7 com um filtro 3x3, mas com stride 2 gera uma saída com tamanho 3x3.



Dimensões

Se o valor de stride for aumentado muito ou estiver em desacordo com as dimensões da imagem original, pode existir um problema.

De uma maneira simples, o tamanho da saída pode ser obtida pela seguinte equação:

(N-F) / stride + 1

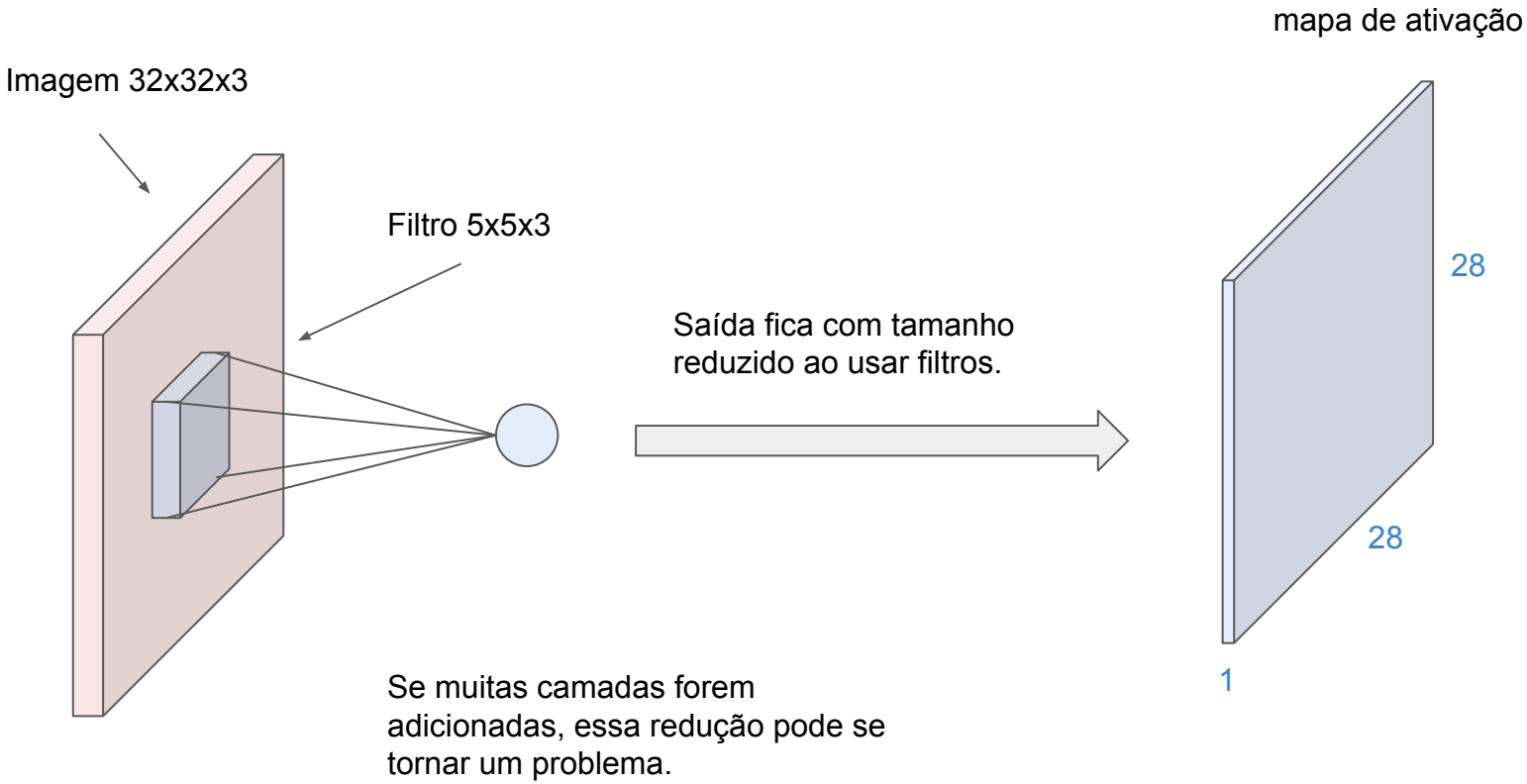
N = dimensão da imagem

F = dimensão do filtro

exemplo: N=7, F=3

- stride 1 => (7-3)/1+1 = 5
- stride 2 => (7-3)/2+1 = 3
- stride 3 => (7-3)/3+1 = 2,33

Redução de dimensionalidade



Se muitas camadas forem adicionadas, essa redução pode se tornar um problema.

Zero padding

Para manter a dimensão original das imagens, é comum realizar um preenchimento "sem significado" (*zero padding*) na imagem.

Para conseguir manter o tamanho da imagem, basta usar a equação abaixo para definir o tamanho do padding.

P = (F-1)/2

exemplo:

- F = 3 => padding de 1
- F = 5 => padding de 2
- F = 7 => padding de 3

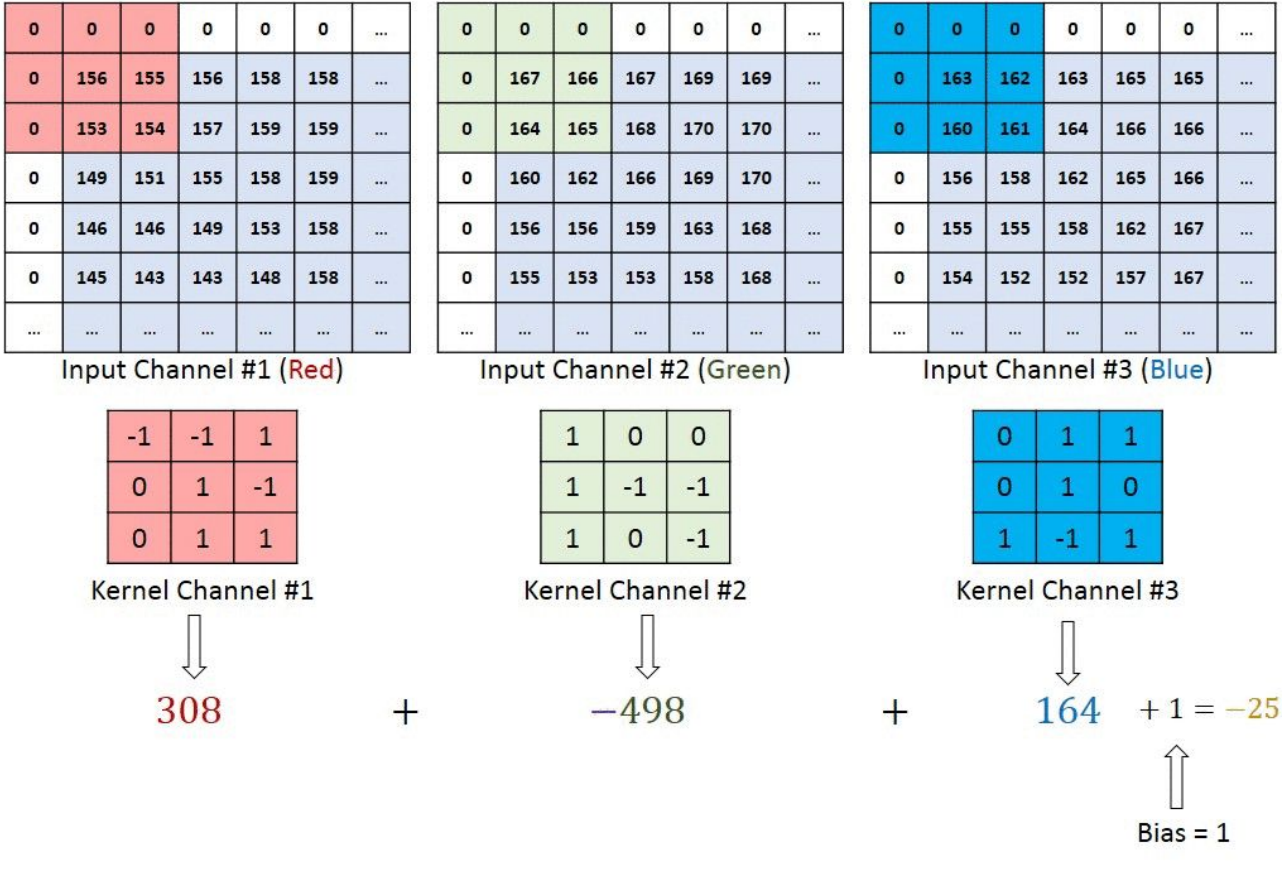
Se considerarmos uma imagem 7x7, com filtro 3x3 e zero padding de 1, a imagem de saída, vai ter 7x7.

Zero padding

Na prática, o zero padding significa acrescentar uma linha e coluna nas margens verticais e horizontais da imagem.

Ao fazer o deslizamento dos filtros sobre a imagem, a saída resultante do produto interno entre o filtros e a imagem possui as mesmas dimensões da entrada.

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

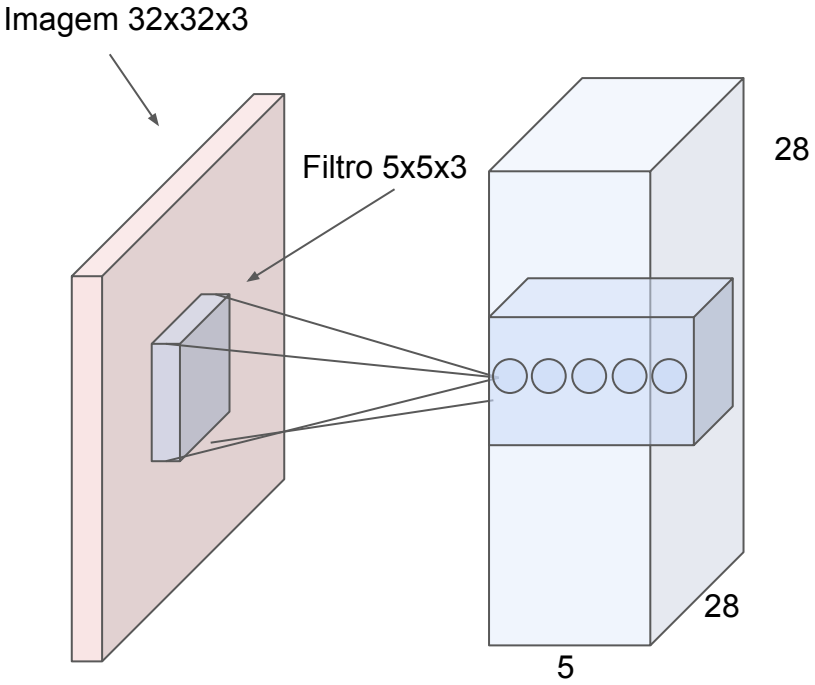

0 0 0 0 0 0 ...
0 156 155 156 158 158 ...
0 153 154 157 159 159 ...
0 149 151 155 158 159 ...
0 146 146 149 153 158 ...
0 145 143 143 148 158 ...
... ...
Input Channel #1 (Red)
0 0 0 0 0 0 ...
0 167 166 167 169 169 ...
0 164 165 168 170 170 ...
0 160 162 166 169 170 ...
0 156 156 159 163 168 ...
0 155 153 153 158 168 ...
... ...
Input Channel #2 (Green)
0 0 0 0 0 0 ...
0 163 162 163 165 165 ...
0 160 161 164 166 166 ...
0 156 158 162 165 166 ...
0 155 155 158 162 167 ...
0 154 152 152 157 167 ...
... ...
Input Channel #3 (Blue)
-1 -1 1
0 1 -1
0 1 1
Kernel Channel #1
1 0 0
1 -1 -1
1 0 -1
Kernel Channel #2
0 1 1
0 1 0
1 -1 1
Kernel Channel #3
308
+
−498
+
164
+ 1 = −25
Bias = 1
Output
-25 ... ...
... ...
... ...
... ...
... ...

Pipeline da camada convolucional

Cada camada convolucional:

- Aceita um volume de entrada com tamanho $W_1 \times H_1 \times D_1$
- Requer quatro hiperparâmetros:
  - número de filtros K,
  - dimensão dos filtros, F,
  - stride S,
  - quantidade de zero padding P
- Produz um volume de saída $W_2 \times H_2 \times D_2$, em que:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$
  - $D_2 = K$
- Introduz $F.F.D_1$ pesos (parâmetros) por filtro e um total de $(F.F.D1).K$ pesos e K biases para a camada.
- No volume de saída, a fatia da profundidade d_i (com tamanho $W_2 \times H_2$) é o resultado de uma convolução do filtro d_i stride S sobre o volume de entrada.

Camada convolucional



Cada mapa de ativação é um matriz de neurônios:

- cada neurônio está conectado com uma pequena região da entrada.
- todos compartilham os mesmos parâmetros.
- um filtro representa um campo receptivo para cada neurônio.

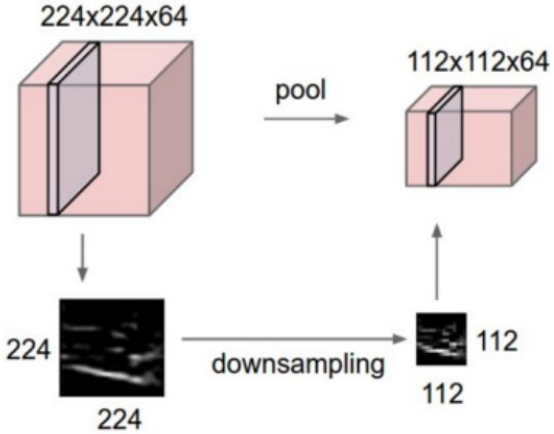
Com um filtro 5x5x3, uma camada convolucional consiste em neurônios organizados em um grid 5x5x5

Isso significa que vão existir 5 neurônios "olhando" para a mesma região do volume de entrada.

Camadas de Pooling

Camadas de pooling

- As camadas de pooling são utilizadas para representar as camadas convolucionais de maneira mais simplificada com o objetivo de melhorar o gerenciamento dos parâmetros.
- Opera sobre cada mapa de ativação individualmente.



Max Pooling

Fatia da camada convolucional

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

max pooling com filtro 2x2 e stride 2

Saída

6	8
3	4

Pipeline da camada de pooling

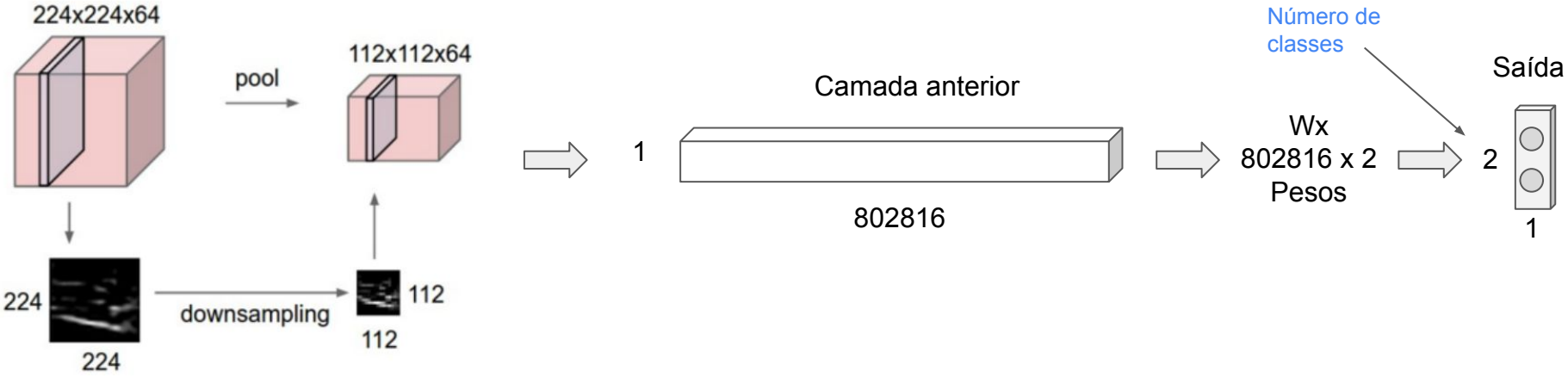
Cada camada de pooling:

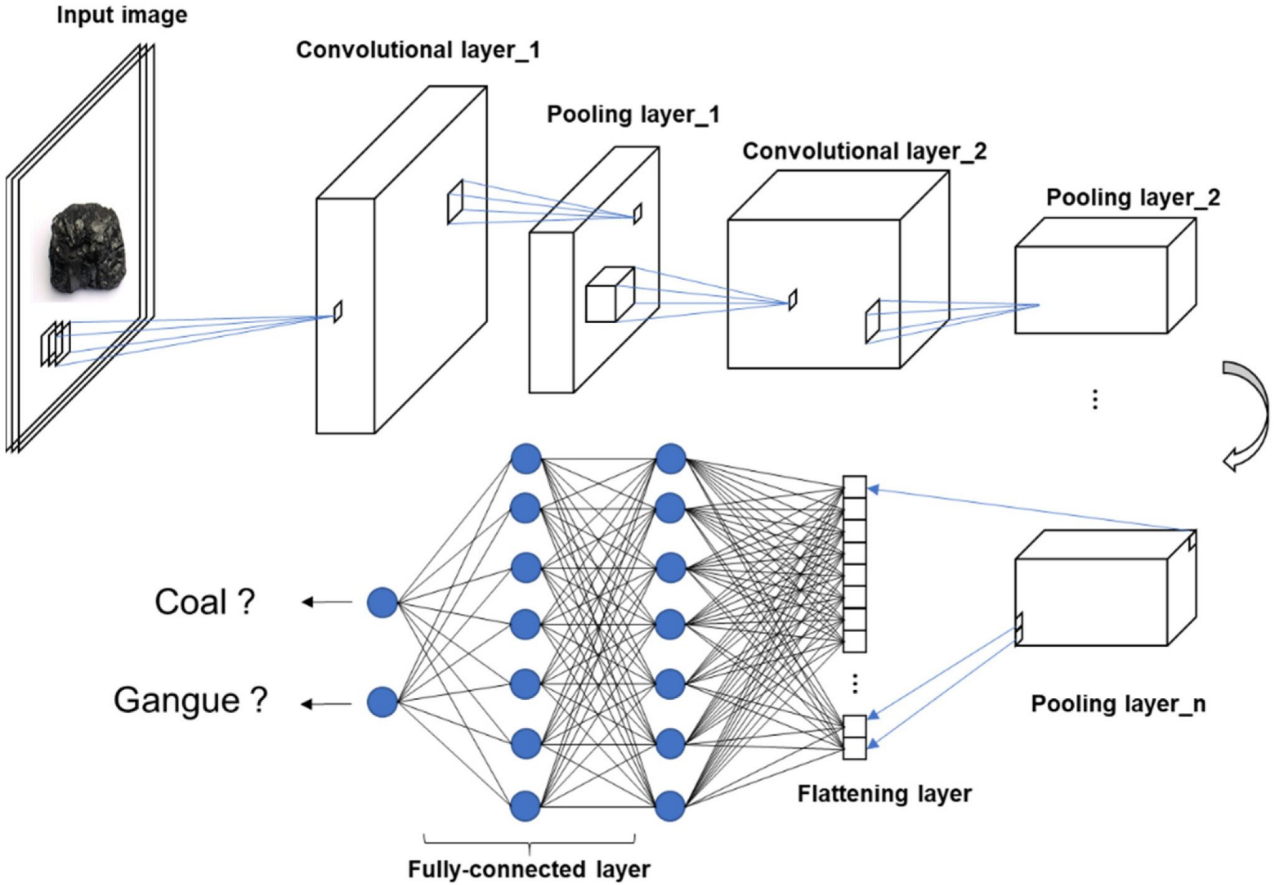
- Aceita um volume de entrada com tamanho $W_1 \times H_1 \times D_1$
- Requer dois hiperparâmetros:
  - dimensão dos filtros F,
  - stride S
- Produz um volume de saída $W_2 \times H_2 \times D_2$, em que:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Não introduz parâmetros já que computa uma função fixa da entrada.
- Não é comum utilizar padding para esse tipo de camada.

Camada Totalmente Conectada

Camada totalmente conectada

- Normalmente, a última camada de uma rede convolucional é uma camada totalmente conectada.
- O objetivo dela é realizar a classificação final de acordo com as classes do conjunto de dados.



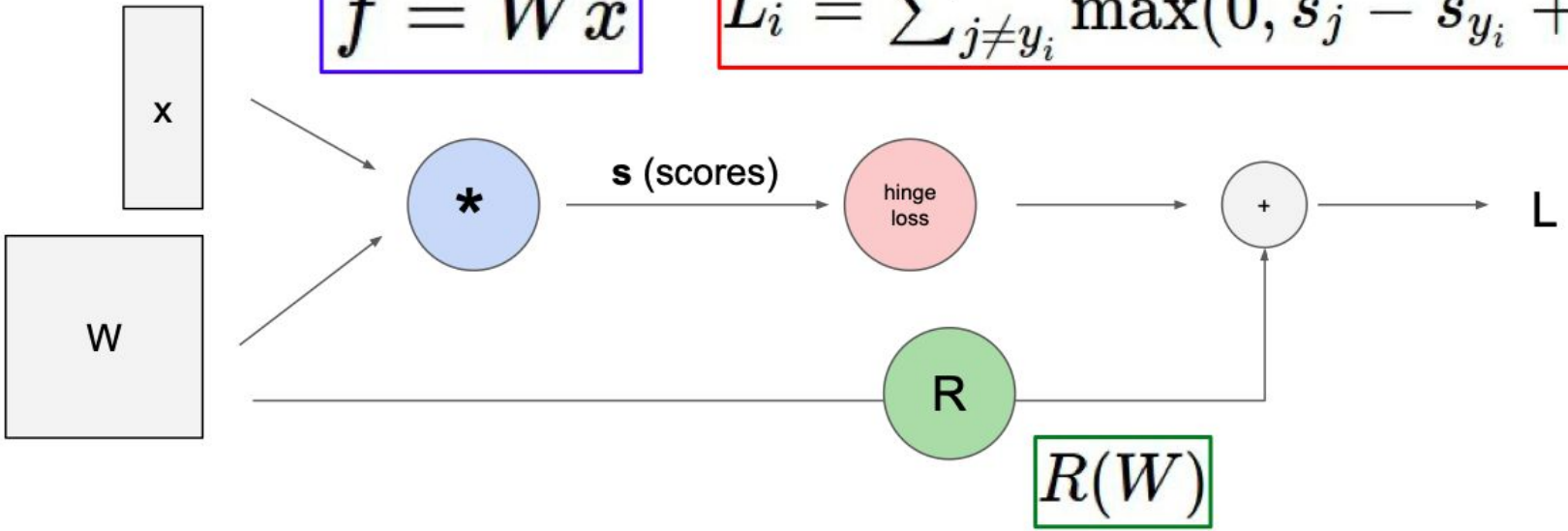

Input image
Convolutional layer_1
Pooling layer_1
Convolutional layer_2
Pooling layer_2
Pooling layer_n
Flattening layer
Fully-connected layer
Coal ?
Gangue ?

Treinamento

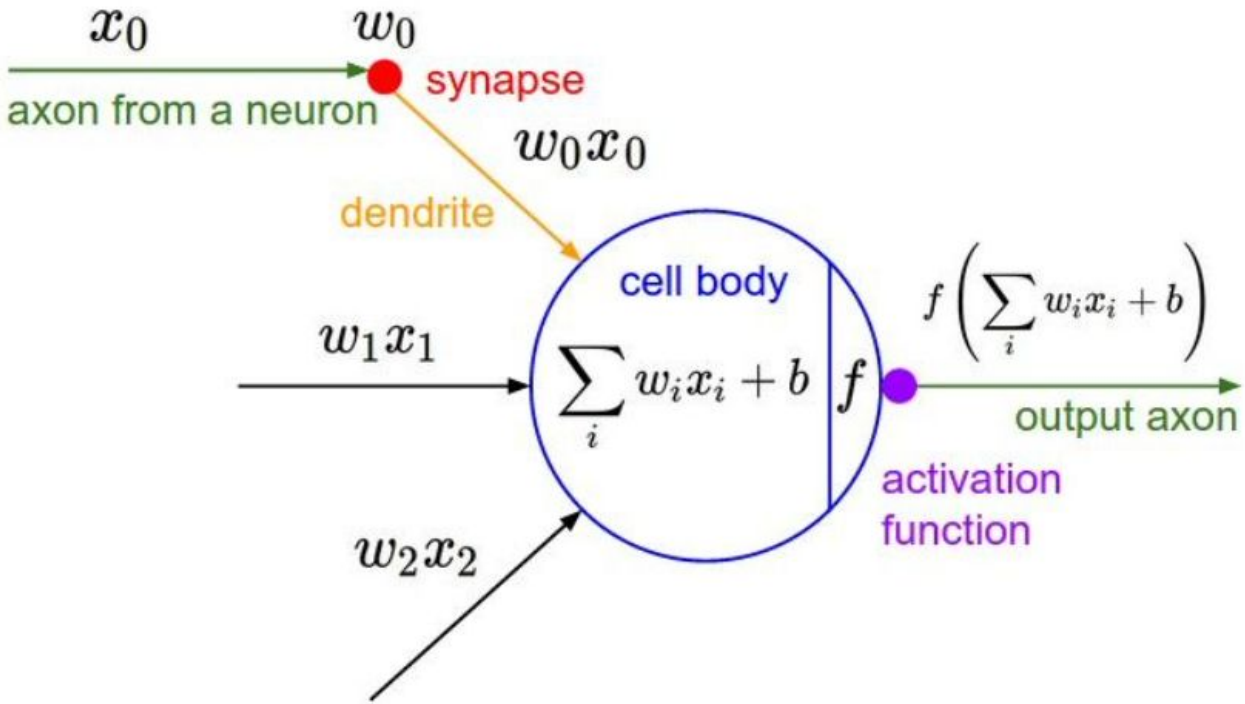
Funções de ativação

$$f = Wx$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$



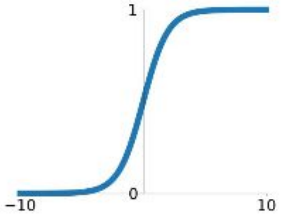
Funções de ativação



Funções de ativação

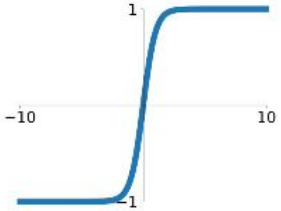
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



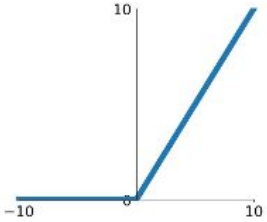
tanh

$$\tanh(x)$$



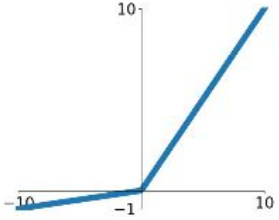
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

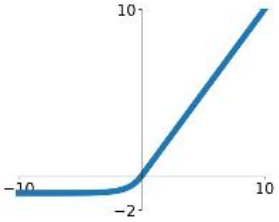


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

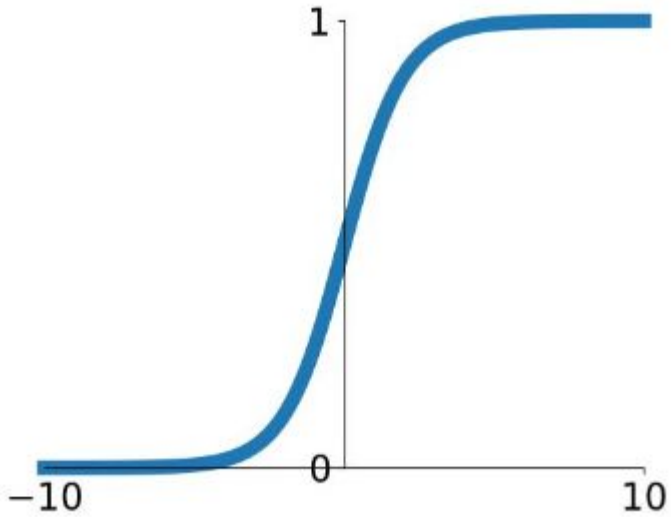
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Sigmóide (sigmoid)

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



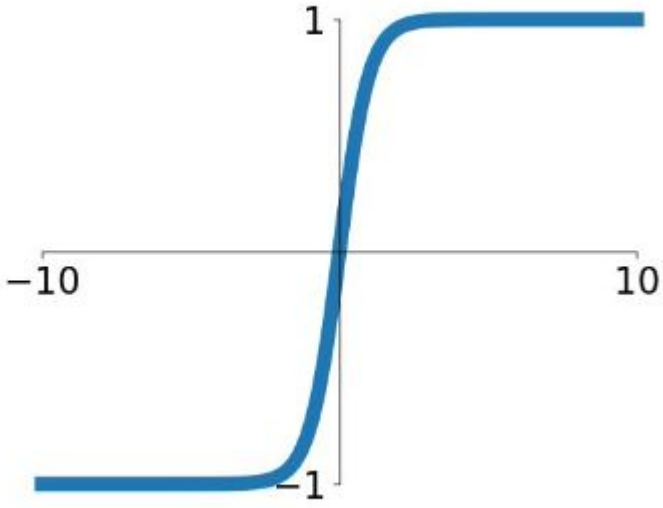
- Valores ficam no intervalo [0,1]
- Popular nos modelos MLP
- Satura valores altos (positivos ou negativos)

Problemas:

- Neurônios saturados "matam" o gradiente
- Saída não está centralizada em zero
- exp() é computacionalmente custoso

Tangente Hiperbólica (tanh)

$$\tanh(x)$$



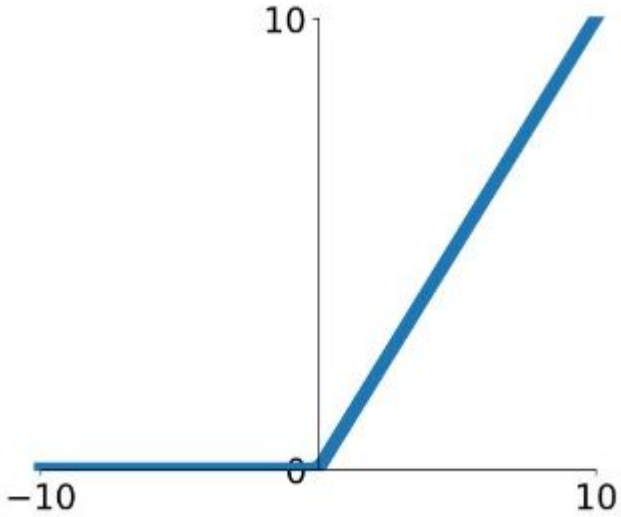
- Valores ficam no intervalo [-1,1]
- Satura valores altos (positivos ou negativos)

Problemas:

- Neurônios saturados "matam" o gradiente
- Saída está centralizada em zero

Rectified Linear Unit (ReLU)

$$f(x) = \max(0, x)$$



- Valores ficam no intervalo [0,+inf]
- Não satura na região positiva
- Converge mais rápido que sigmóide e tangente hiperbólica

Problemas:

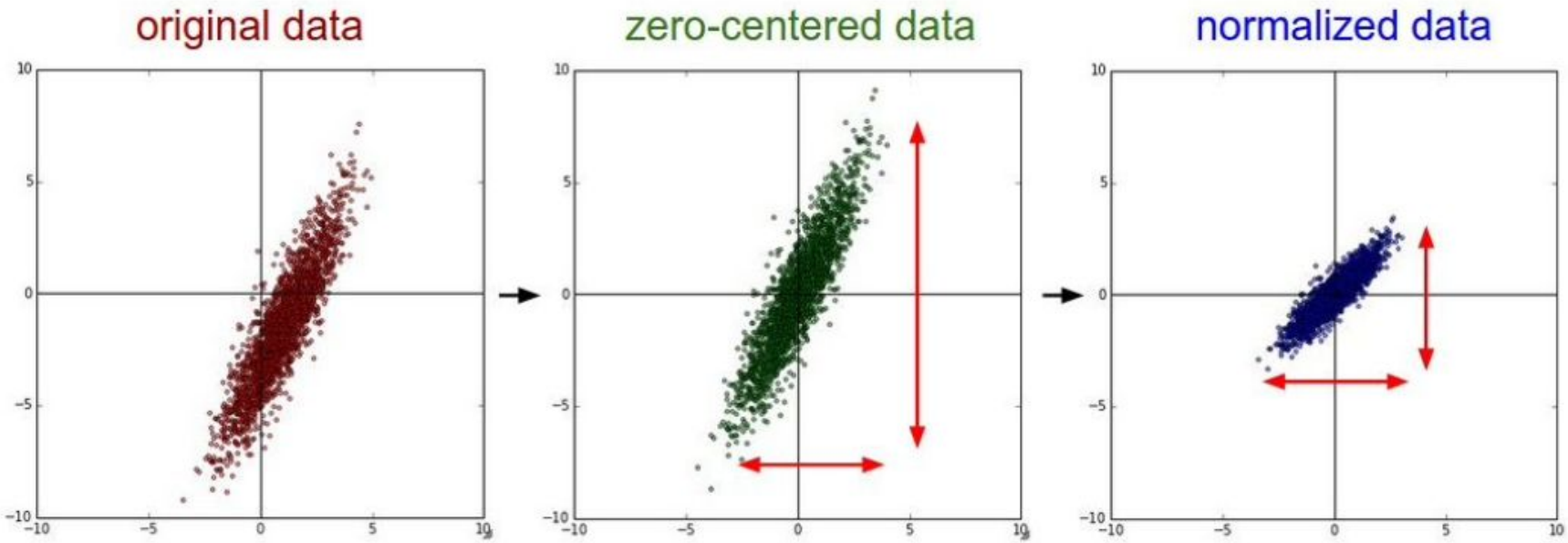
- Saída não está centralizada em zero
- Sensível a inicialização dos pesos (ainda satura no lado negativo)

Outras funções de ativação

- Existem ainda variações da ReLU que buscam corrigir limitações:
  - Leaky ReLU
  - Parametric Rectifier (PReLU)
  - Exponential Linear Units (ELU)
  - Maxout "Neuron"
- Na prática, a ReLU é a função de ativação mais utilizada.

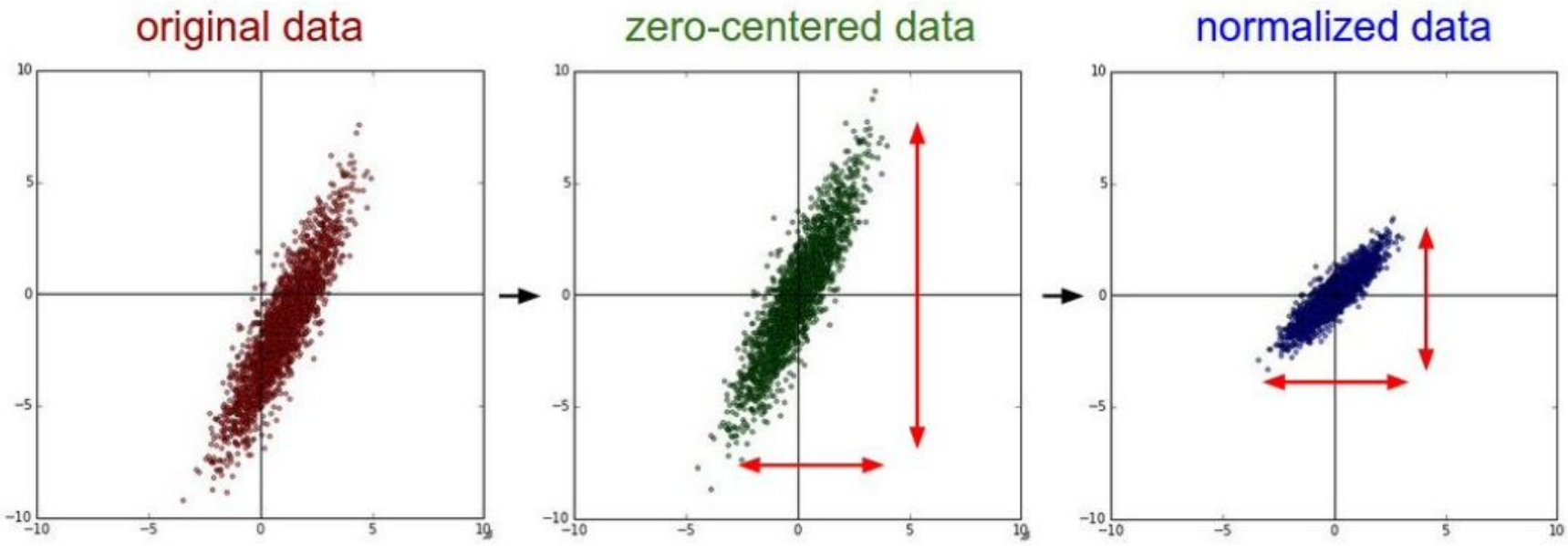
Pré-processamento

Pré-processamento



Pré-processamento

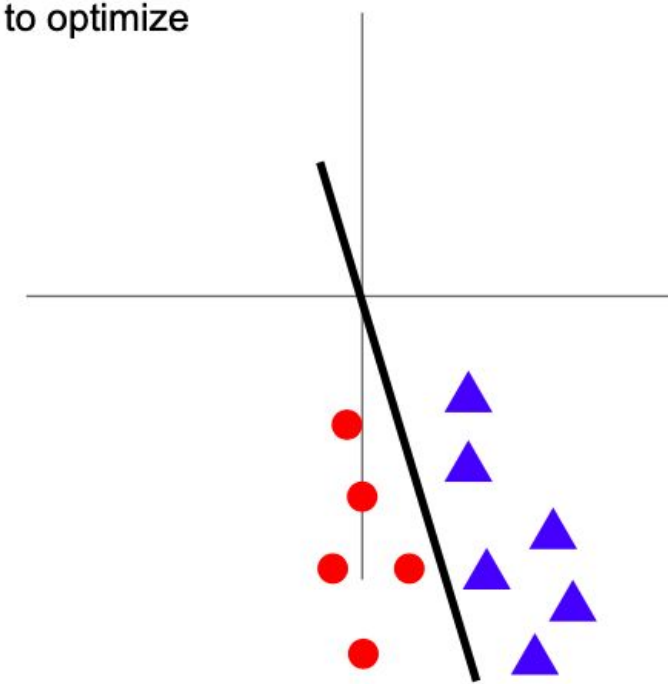
Normalmente feito no contexto de imagens.



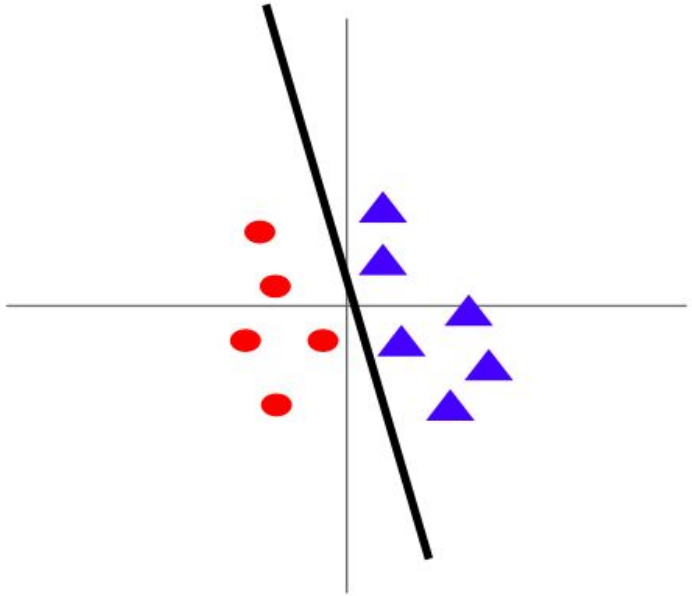
Normalmente é feito a cada camada no contexto de imagens.

Importância da centralização

Before normalization: classification loss very sensitive to changes in weight matrix; hard to optimize



After normalization: less sensitive to small changes in weights; easier to optimize



Inicialização dos pesos

Inicialização dos pesos

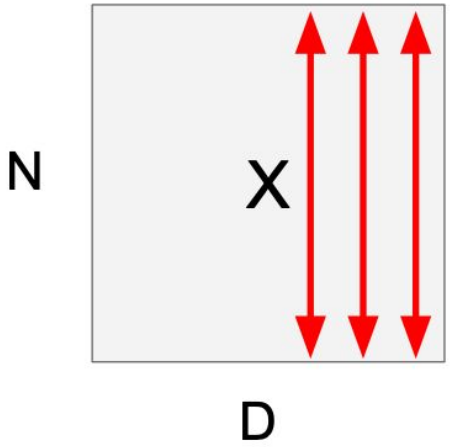
- Não iniciar todos com zero.
  - Faz com que todos os neurônios tenham o mesmo comportamento.
- Inicializar com valores aleatórios pequenos.
  - Pesos tendem a ter média e desvio padrão igual a zero.
  - Neurônios não ativam e não mudam com o tempo e a rede não converge.
- Inicializar com valores aleatórios grandes.
  - Pesos tendem a saturar
  - Gradientes ficam próximos a zero (rede não converge)
- Inicialização com heurística
  - Xavier
    - Variância da saída deve ser próxima a da entrada.
    - Alguns estudos apontam que usar a metade da variância de entrada na saída é melhor quando utilizamos a ReLU como função de ativação.

Como regra geral, utilizar a inicialização Xavier.

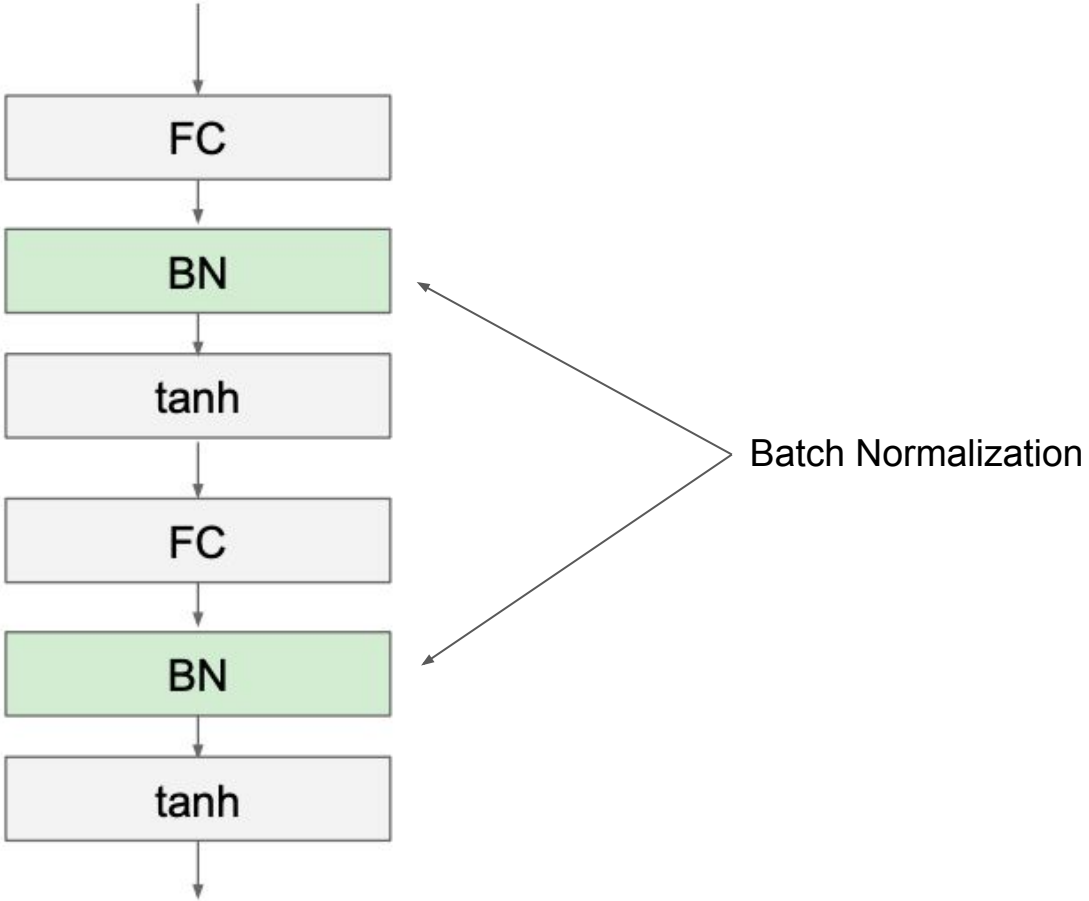
Batch Normalization

Batch Normalization

- Utilizada para manter a saída de cada camada dentro de uma distribuição gaussiana.
- Feito a partir da média e variância amostral do *batch* de dados atual.
- Feito a cada camada ao invés de realizar a normalização somente na inicialização dos pesos.
    - Nas camadas convolucionais, a média e variância são calculados por mapa de ativação e os valores são usados para normalizar os valores de todas as entradas do *batch*.
    - Feito para manter a informação espacial da imagem.



$$\hat{x}(k) = \frac{x^{(k)} - E\left[x^{(k)}\right]}{\sqrt{Var(x^k)}}$$



A etapa de Batch Normalization:

- melhora o fluxo do gradiente na rede
- Permite valores maiores para taxa de aprendizado
- Reduz o impacto da inicialização dos pesos
- Atua como uma forma de regularizacão
- Pode reduzir a necessidade de dropout.

Monitorar treinamento

Sanity Check

- Configurar arquitetura básica para o modelo.
  - Entrada (tamanho da imagem), camada(s) escondida(s), camada de saída (número de classes).
- Executar o treinamento sem regularização e observar o *loss*.
- Ao adicionar alguma regularização o *loss* deve aumentar inicialmente.
- Conseguir superajustar o modelo para porção de dados pequenas (loss = ~0, acurácia = ~1).

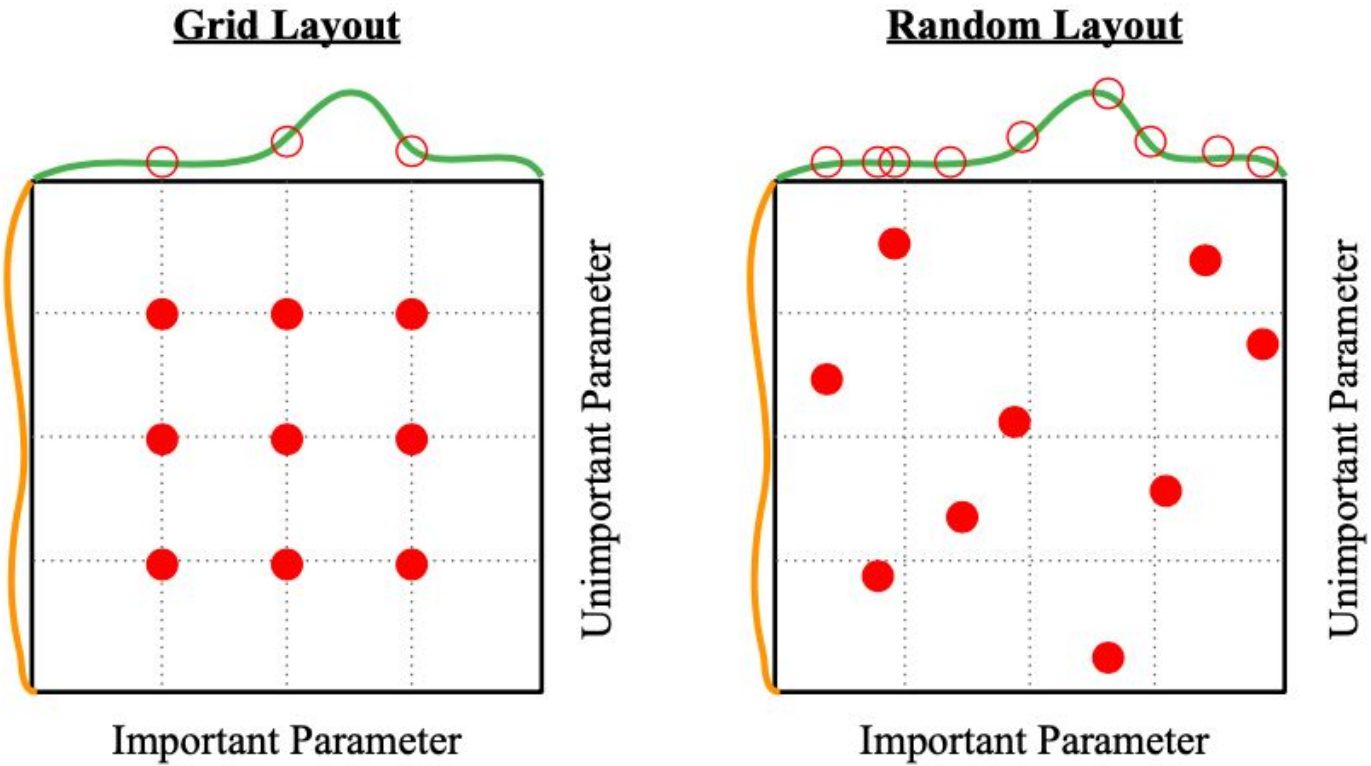
Treinamento inicial

- Configurar arquitetura básica para o modelo.
  - Entrada (tamanho da imagem), camada(s) escondida(s), camada de saída (número de classes).
- Executar o treinamento com valores de regularização baixos.
- Configurar taxa de aprendizado para valores não muito pequenos (entre 1e-3 e 1e-4).
- Utilizar poucas épocas para verificar comportamento.

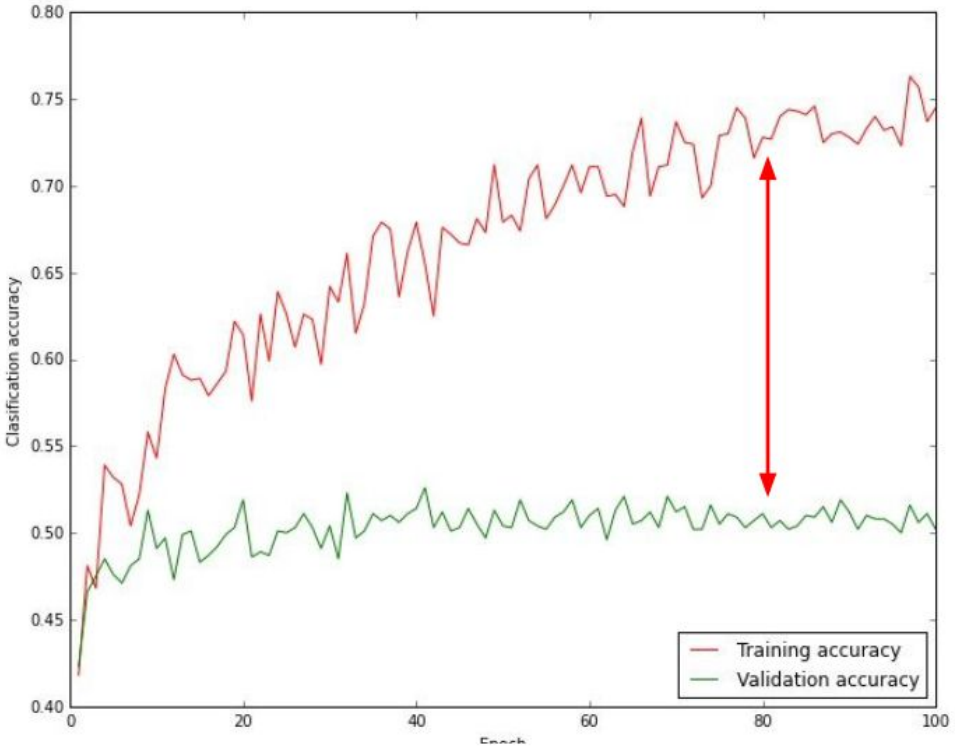
Otimização dos hiperparâmetros

- Modificar os hiperparâmetros e verificar o comportamento da rede.
  - Definir intervalos amplos para os valores dos hiperparâmetros
  - Identificar valores onde os hiperparâmetros funcionam melhor e redefinir os intervalos para as regiões próximas a esses valores.
  - Modificar um ou dois parâmetros por vez.
- Problema: muitos hiperparâmetros para otimizar.
  - Cada hiperparâmetro possui seu próprio intervalo de valores ótimo.
- É possível adotar uma estratégia para buscar os melhores valores para cada hiperparâmetro.
  - Mais comuns são a busca aleatória e a busca em grid.

Otimização dos hiperparâmetros



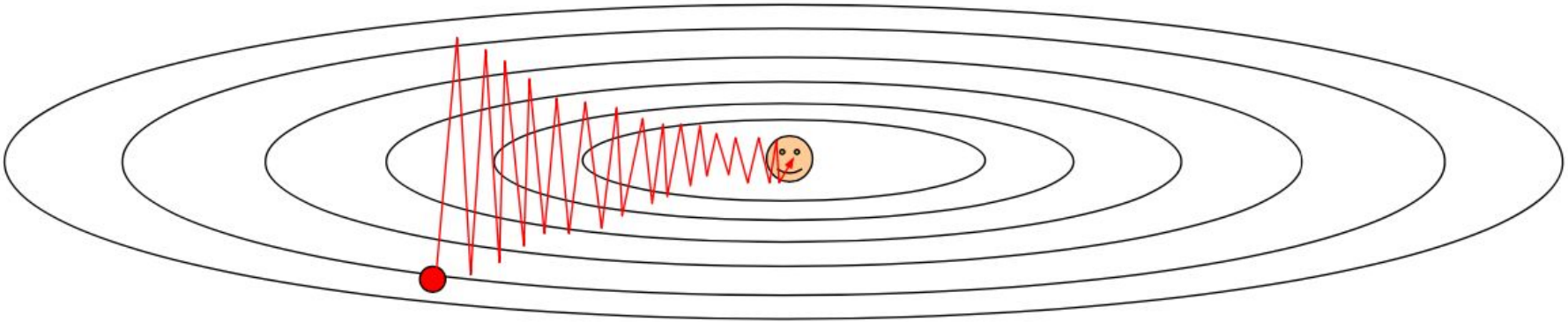
Monitorar o superajustamento



Otimização

SGD

Se o loss mudar rapidamente em uma direção mas não na outra, a rede demora muito para convergir.

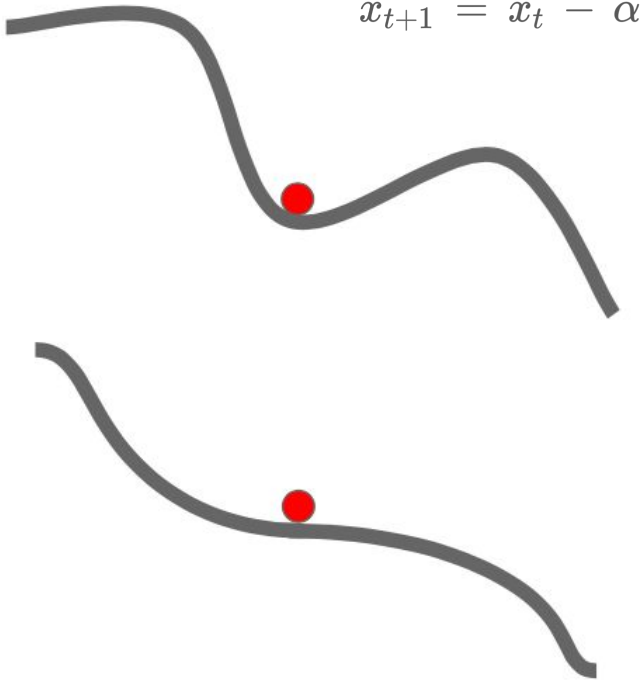


Stochastic Gradient Descent (SGD)

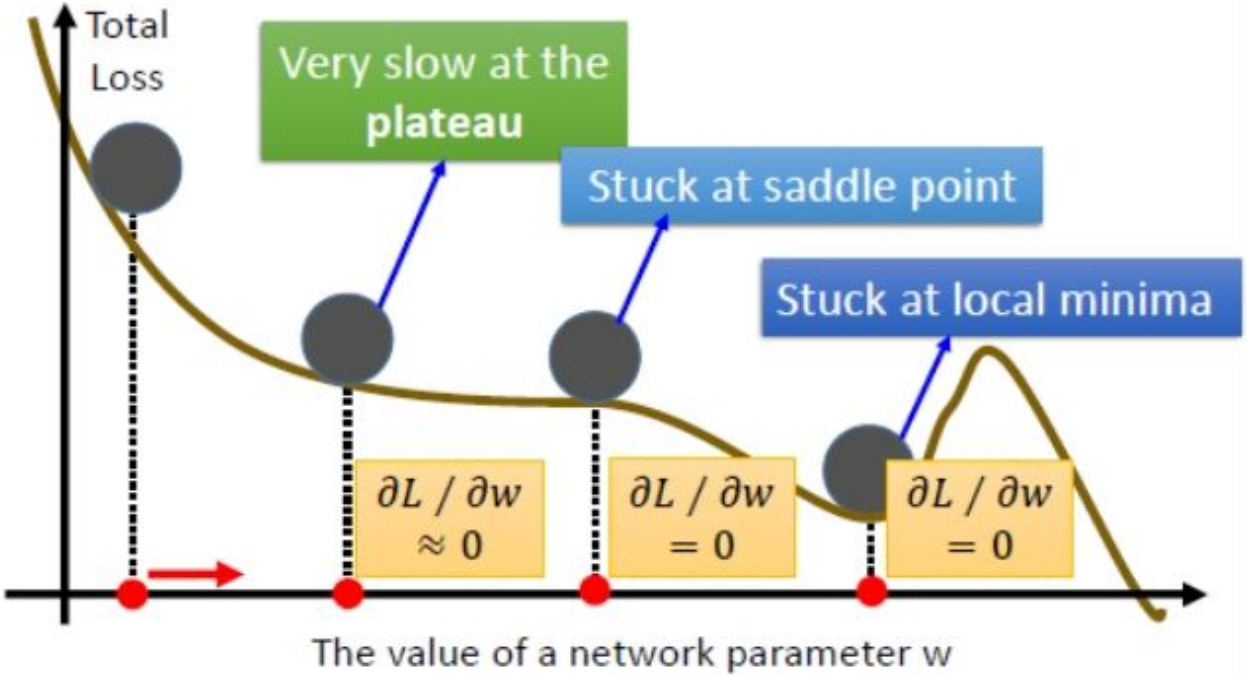
Existem ainda problemas em mínimos locais ou pontos sem inclinação (saddle point).

- o problema de mínimo local não é tão importante em situações de muitas dimensões.
  - seria muito raro ter todas as dimensões um aumento na função custo.
- em locais onde existe um vale, o gradiente fica próximo a zero e o progresso é muito lento.
  - situação bem mais comum em altas dimensões.

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$



Hard to find optimal network parameters



SGD + Momentum

Um incremento que ajuda a reduzir o efeito dos pontos de gradiente baixo é a adição de um elemento de velocidade à equação.

- o momentum representa esse elementos de velocidade.
- ele é calculado pela média móvel dos gradientes anteriores.
- como existe um termo agregado ao gradiente, em casos de locais de pouca inclinação, a *velocidade acumulada* ajuda a superar esses locais com mais eficiência.

SGD

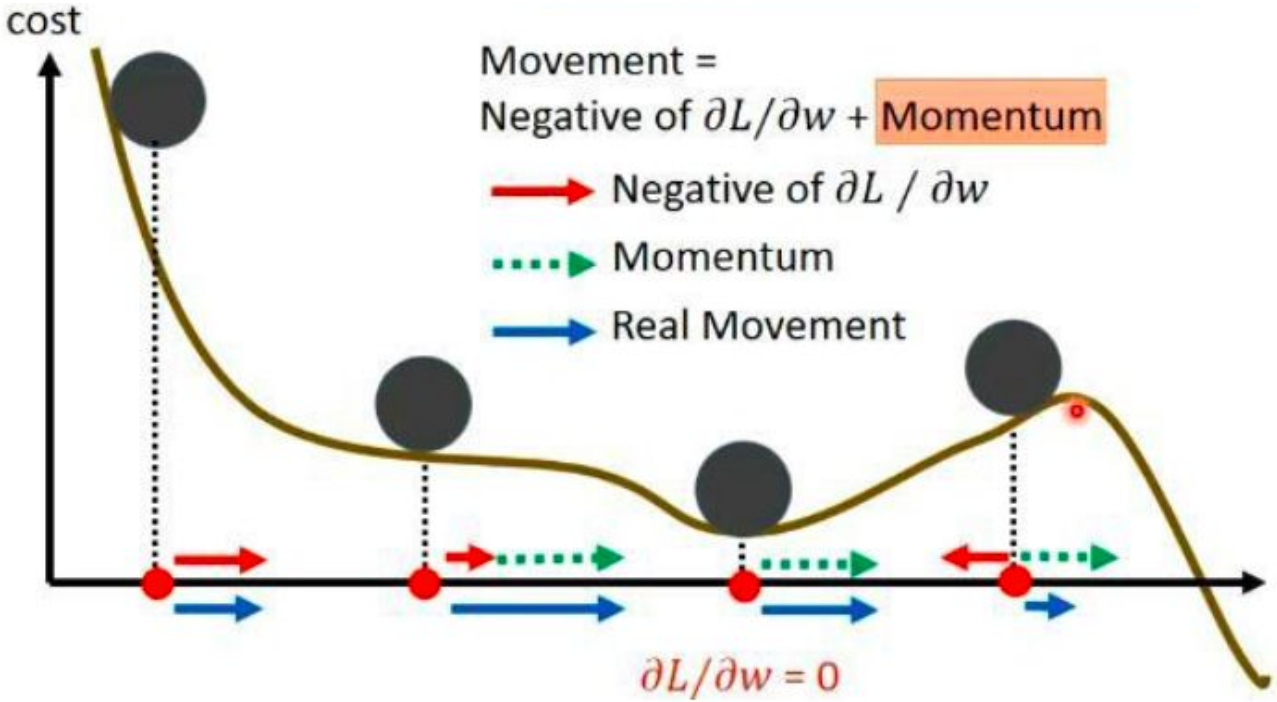
$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

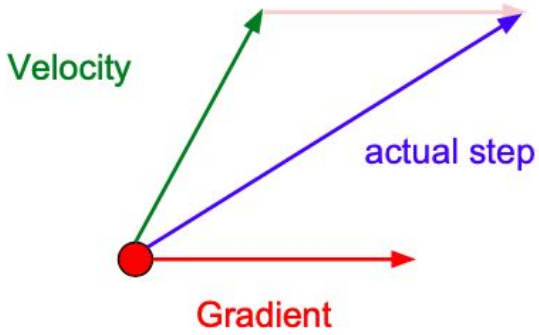
SGD + Momentum

$$x_{t+1} = x_t - \alpha v_{t+1}$$

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

SGD + Momentum





AdaGrad e RMSProp

AdaGrad atualiza o SGD para considerar um escalonamento do gradiente quadrático acumulado levando em consideração cada dimensão (parâmetro) para adaptar a taxa de aprendizado.

- Acelera para dimensões onde o gradiente varia pouco e desacelera onde o gradiente varia muito.
- pode ficar muito lento quando os passos reduzem muito.

RMSProp adiciona um elemento de decaimento do fator de escalonamento da taxa de aprendizado.

AdaGrad

$$x_{t+1} = x_t - \alpha \frac{\nabla f(x_t)}{\sqrt{g^2}}$$

g^2 é o gradiente quadrático acumulado.

RMSProp

$$x_{t+1} = x_t - \alpha \frac{\nabla f(x_t)}{\sqrt{\gamma g^2 + (1-\gamma)(\nabla f(x_t))^2}}$$

y é o fator de decaimento.

Adaptive Momentum (Adam)

Utiliza a ideia do RMSProp com a adição de um momentum. São utilizados dois termos de momentum.

- o primeiro termo funciona como elemento de velocidade similar ao momentum adicionado ao SGD.
- o segundo termo tem função similar ao proposto pelo AdaGrad de decaimento da taxa de aprendizado.

Atualmente, o Adam é o otimizador mais utilizado para realizar a atualização dos parâmetros da rede.

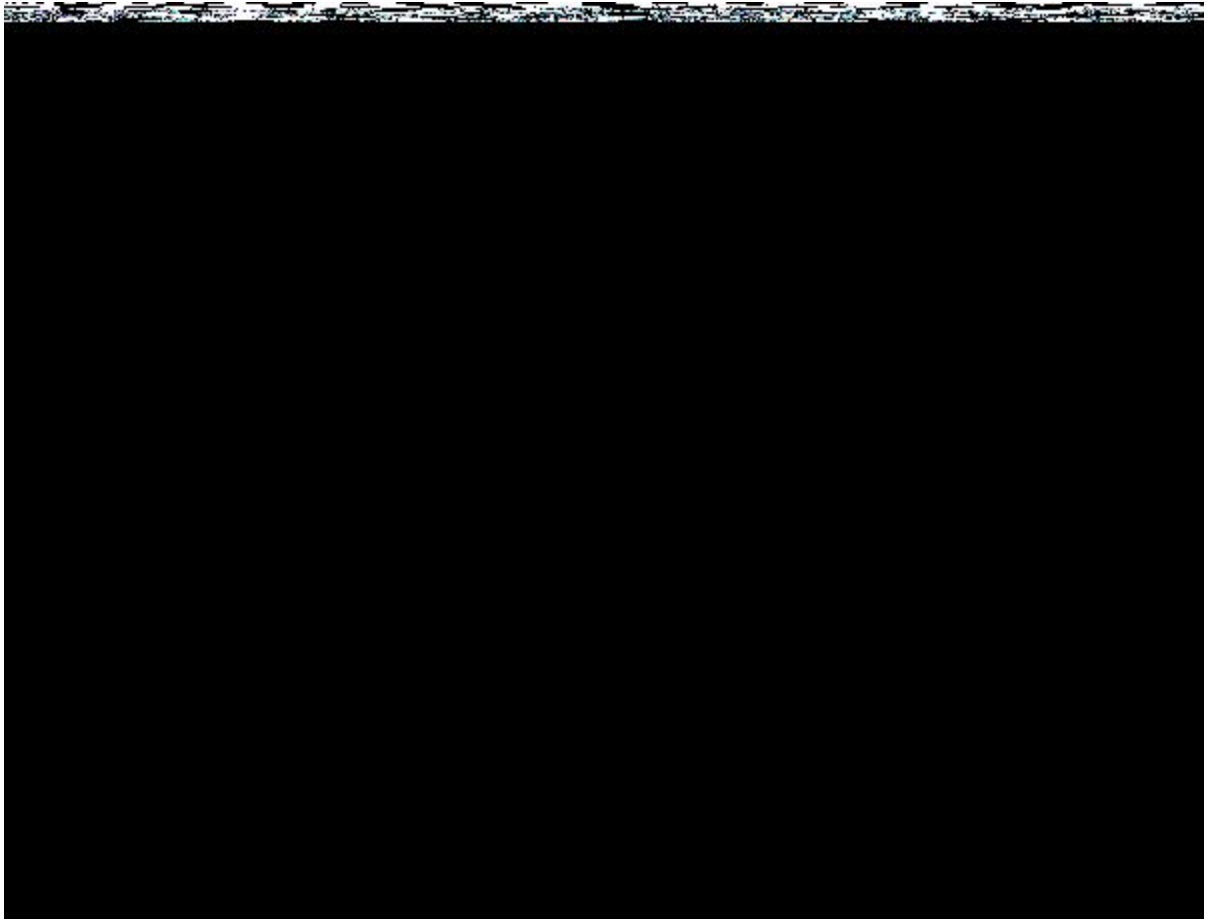
Adam

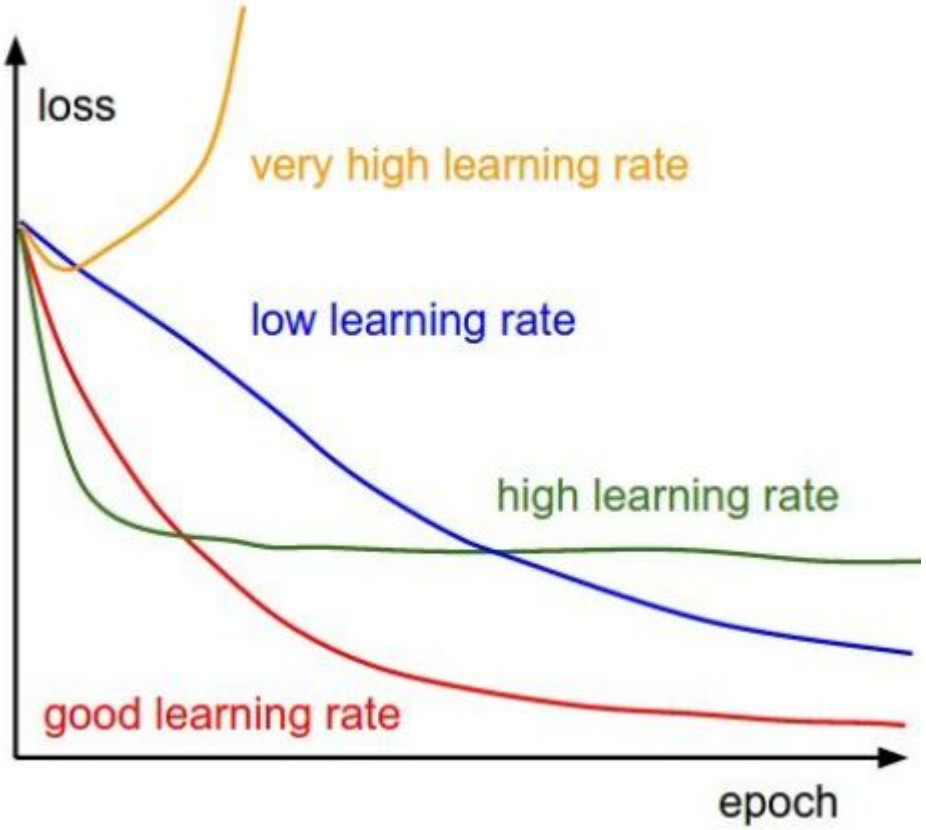
$$m_{t+1} = \beta_1 m_t + (1-\beta_1)\nabla_w L_t$$

$$v_{t+1} = \beta_2 + (1-\beta_2)(\nabla_w L_t)^2$$

$$x_{t+1} = x_t - \alpha \frac{m_{t+1}}{\sqrt{v_{t+1}} + \epsilon}$$

m é o primeiro momentum, *v* é o segundo momentum, $betta_1$ e $betta_2$ são parâmetros de ajuste do método.




loss
very high learning rate
low learning rate
high learning rate
good learning rate
epoch

Regularização

Regularização

É comum utilizar algum método para evitar overfitting nos dados de treino.

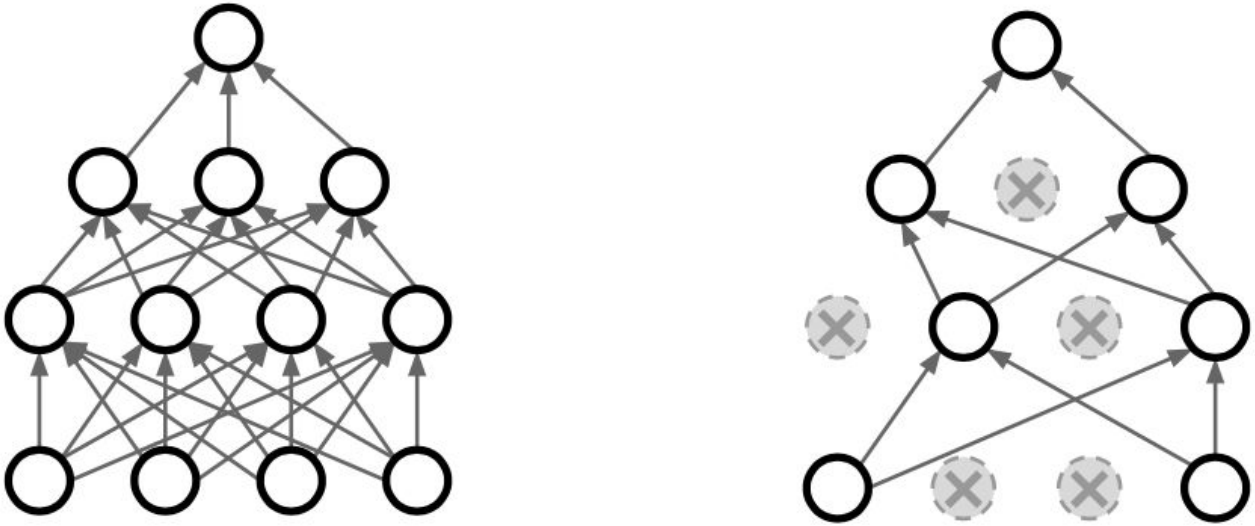
A regularização é uma das maneiras de atingir esse objetivo. Existem algumas formas de realizar isso, adicionando um termo ao custo da rede:

- L2
- L1
- Elastic net (L1+L2)

No entanto, existe uma forma mais prática de conseguir resultados semelhantes.

Dropout

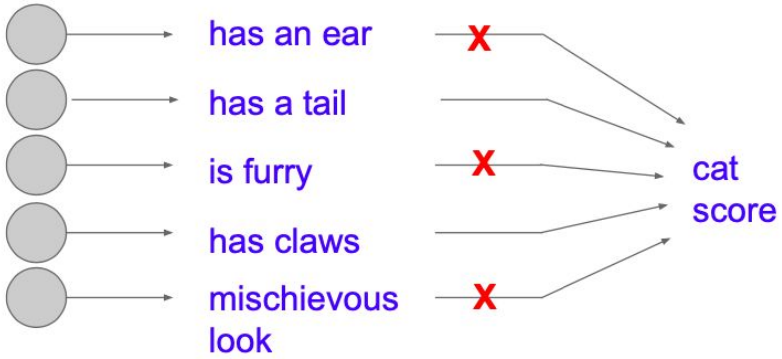
A cada passada para frente da rede (forward pass), alguns neurônios são configurados para ter saída igual a zero.



Dropout

O dropout é um hiperparâmetro da rede e é comumente utilizado um valor de 0.5: 50% dos neurônios de cada camada são aleatoriamente escolhidos para ter sua saída configurada para zero.

Esse mecanismo força a rede a criar representações redundantes para cada classe.



Dropout

Uma outra interpretação para o Dropout é que sua utilização promove a criação implícita de comitês de classificadores dentro de uma mesma rede:

- cada máscara de dropout representa um modelo diferente.
- cada modelo tem um desempenho diferente.

Dropout

No momento de teste da rede, como o dropout é considerado?

- no treinamento, a cada passagem, um conjunto de neurônios era escolhido aleatoriamente para serem desativados.
- no teste, qual máscara deve ser usada?

Na prática, o que se faz é utilizar a probabilidade definida a prioridade para escalonar a saída de cada camada.

- se o dropout foi definido para 0.5, então, esse valor é usado na saída de cada camada.

Estratégias de regularização

A regularização na maioria dos casos consiste em inserir algum valor aleatório ou não na estrutura da rede para evitar o overfitting. Além do dropout, outras estratégias também atuam como regularizadores:

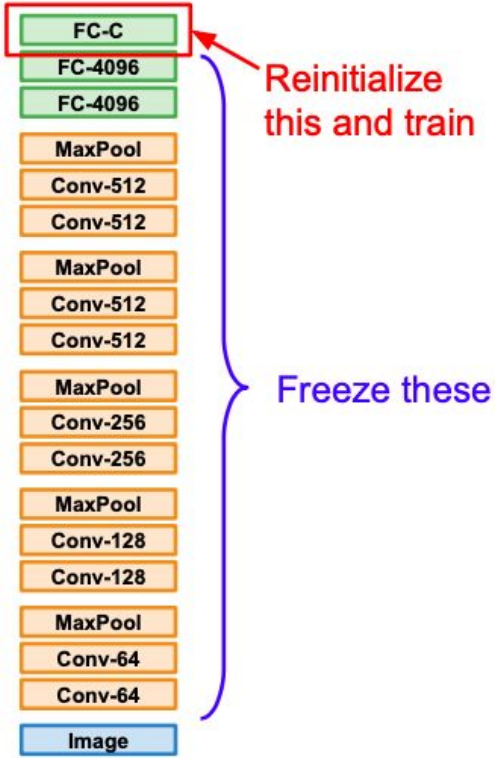
- batch normalization: mini batches possuem distribuições de dados diferentes.
- data augmentation: flips, rotações, translações, recortes aleatórios da imagem, etc.

Outras estratégias incluem anular os pesos de neurônios ou anular camadas inteiras da rede.

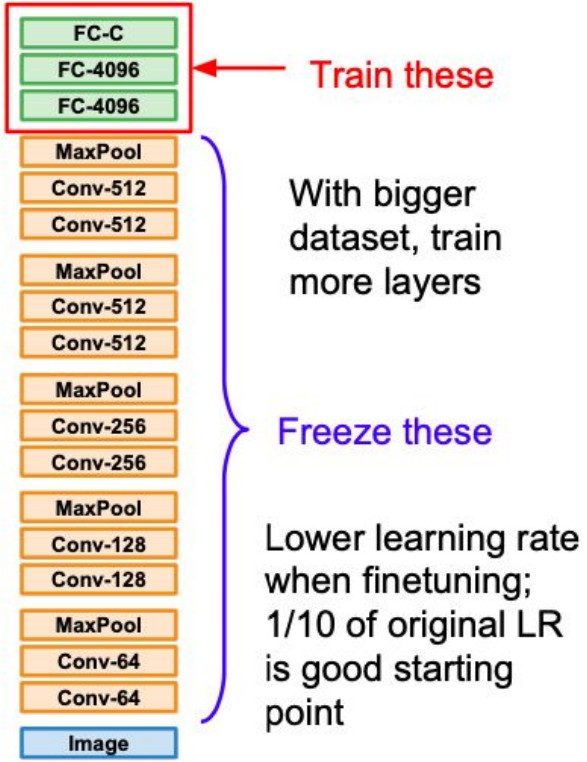
Transfer Learning

1. Train on Imagenet

2. Small Dataset (C classes)



3. Bigger dataset



Regras gerais

	dados similares	dados diferentes
poucos dados	usar classificador linear na última camada	problema… usar classificador linear em alguns estágios
muitos dados	ajustar (finetuning) algumas camadas	ajustar (finetuning) muitas camadas

Na prática, se não tiver muitos dados (~1M), é melhor buscar um modelo treinado em um conjunto de dados que possua imagens semelhantes e realizar o transfer learning.

Tarefas de Visão Computacional

Tarefas de Visão Computacional

Claramente, as redes neurais convolucionais podem ser utilizadas para realizar classificação de imagens. Elas conseguem extrair as características e construir um mapa de ativação para conseguir distinguir os objetos de uma maneira eficiente.

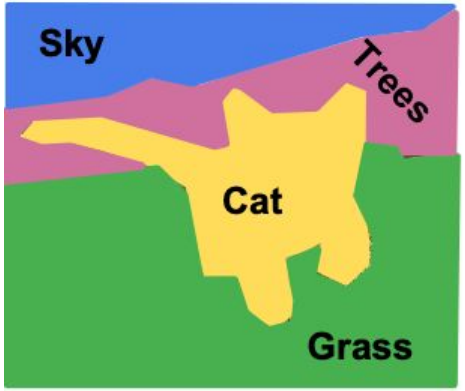
Existem outras tarefas que podem ser realizadas:

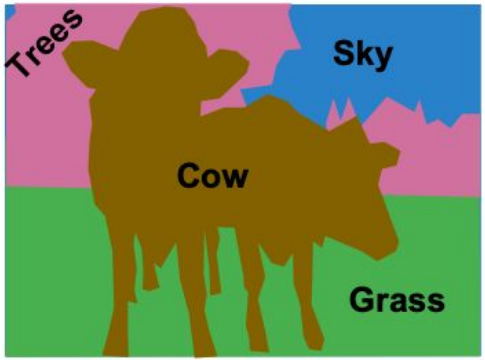
- segmentação semântica
- localização e classificação
- detecção de objetos

Segmentação semântica

Rotular cada pixel da imagem com uma categoria

Não diferencia instâncias, pois está dedicada ao contexto de pixels





Localização e Classificação

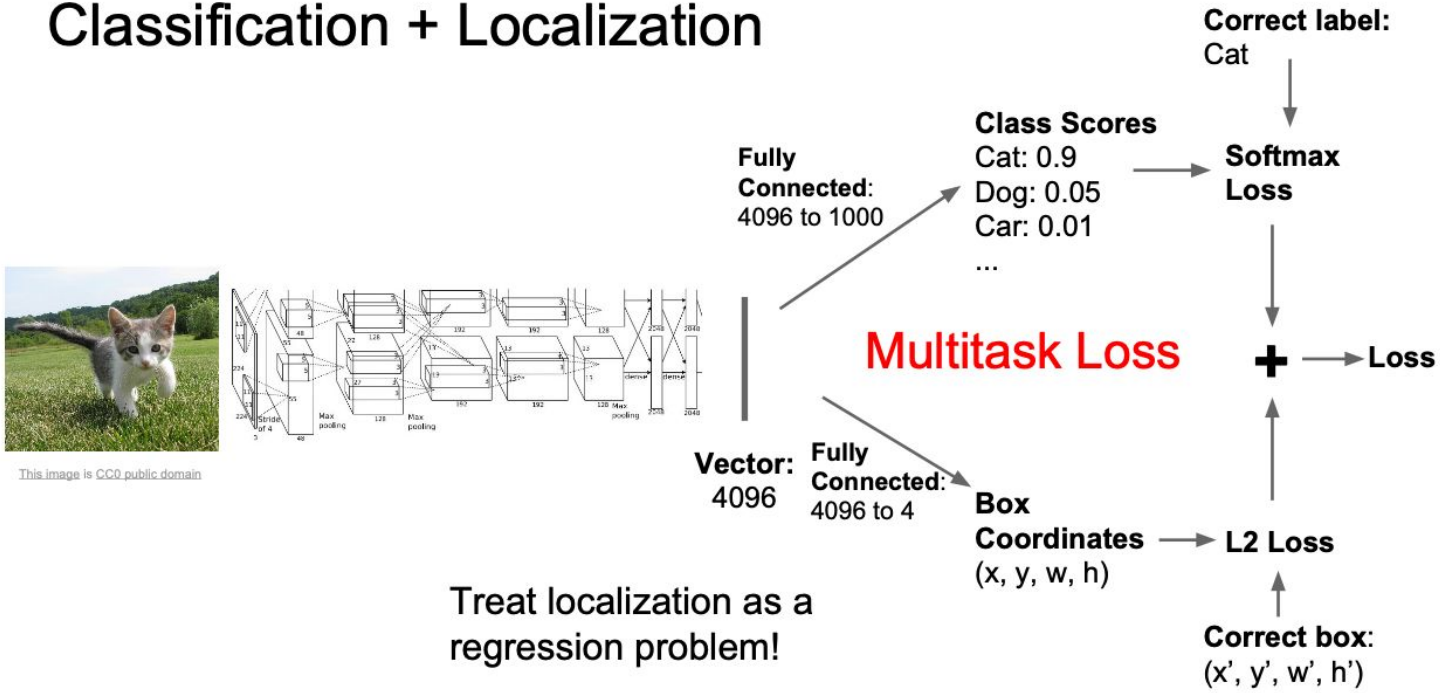
Em algumas situações, além de classificar, é interessante saber a localização do objeto de interesse dentro da imagem.

Quando sabemos que só existe um objeto de interesse dentro da imagem, essa tarefa é denominada de localização e classificação.

Por outro lado, se não sabemos quantos elementos de interesse existem na imagem, essa tarefa se torna de detecção de objetos.

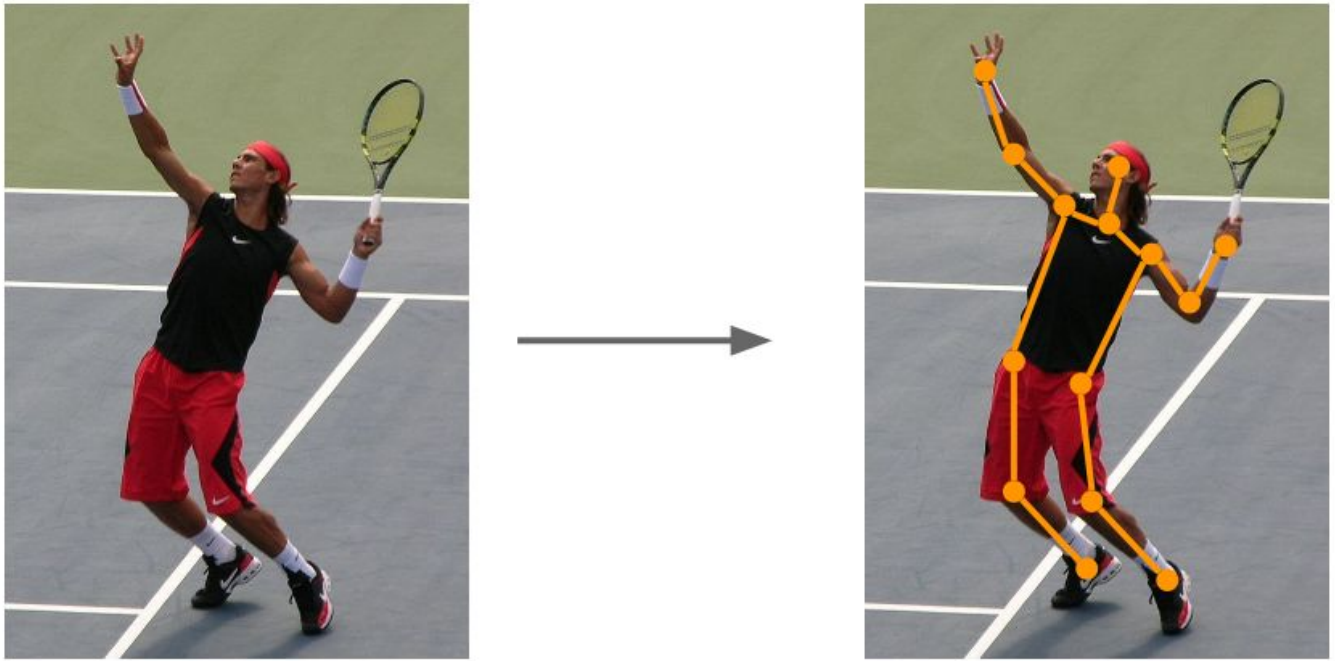
As duas abordagens são tratadas de forma semelhante.

Classification + Localization



A localização é encarada como um problema de regressão: prever as coordenadas e tamanho do bound box.

Exemplo: detectar pose humana



Represent pose as a set of 14 joint positions:

Left / right foot
Left / right knee
Left / right hip
Left / right shoulder
Left / right elbow
Left / right hand
Neck
Head top

Detecção de objetos

Verificar em toda a imagem se um conjunto de objetos predeterminados estão presentes nela.

- podem existir mais de um objeto na imagem
- os objetos podem estar em qualquer lugar da imagem

Como o número de outputs da rede pode variar muito, tratar o problema de definir o bounding box como regressão não é mais interessante.

Janelas deslizantes

A estratégia mais simples para tratar o problema é realizar a classificação utilizando a ideia de janelas deslizantes:

Janelas deslizantes

A estratégia mais simples para tratar o problema é realizar a classificação utilizando a ideia de janelas deslizantes:

Gato: não

Cachorro: não

Backgound: sim

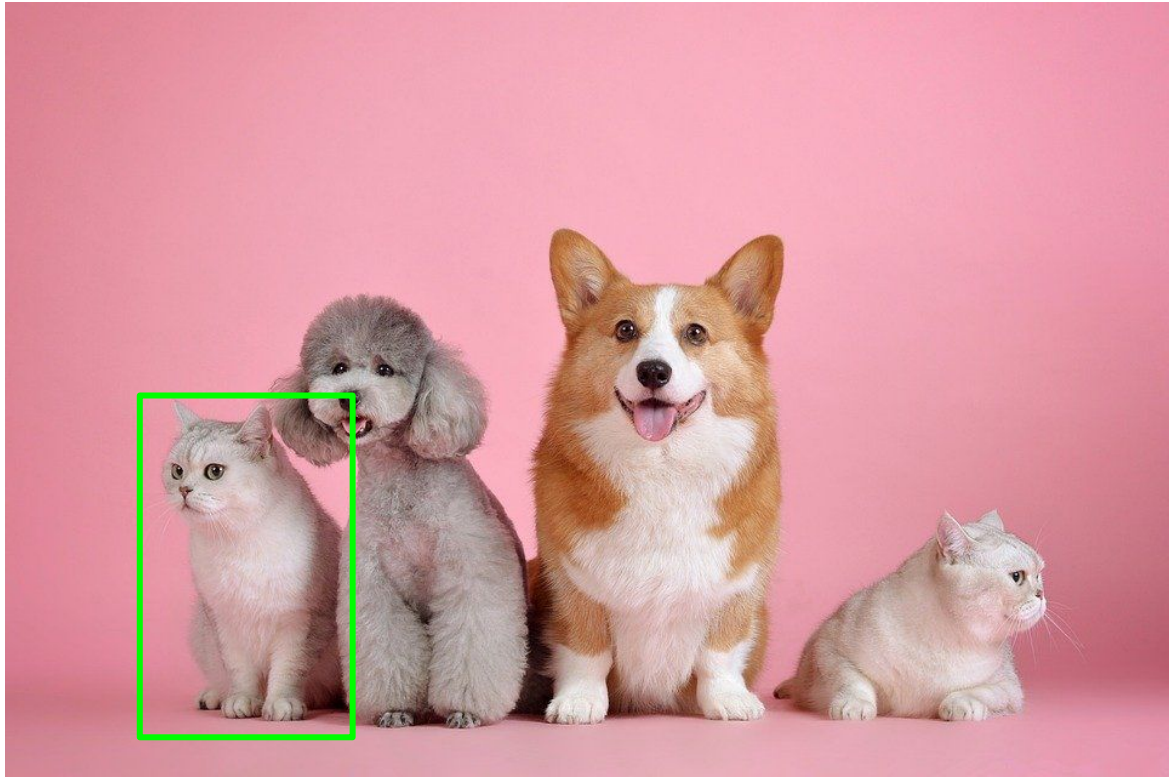
Janelas deslizantes

A estratégia mais simples para tratar o problema é realizar a classificação utilizando a ideia de janelas deslizantes:

Gato: sim

Cachorro: não

Backgound: não



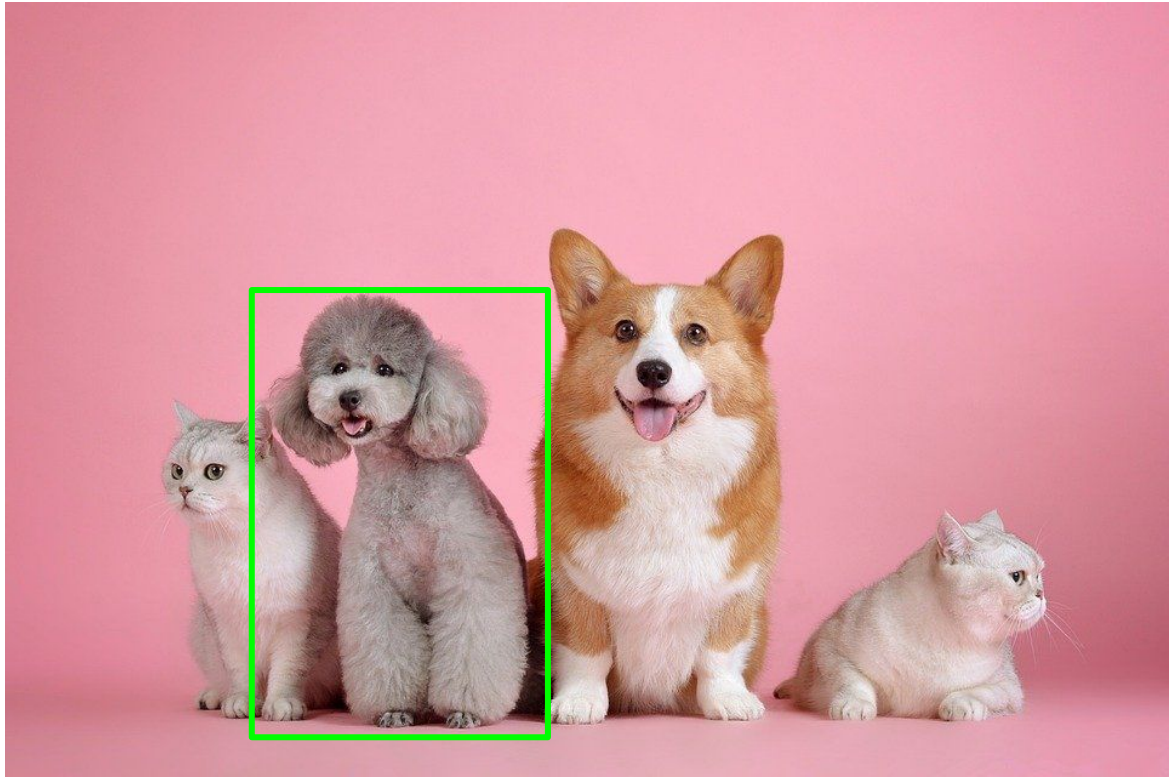
Janelas deslizantes

A estratégia mais simples para tratar o problema é realizar a classificação utilizando a ideia de janelas deslizantes:

Gato: não

Cachorro: sim

Backgound: não



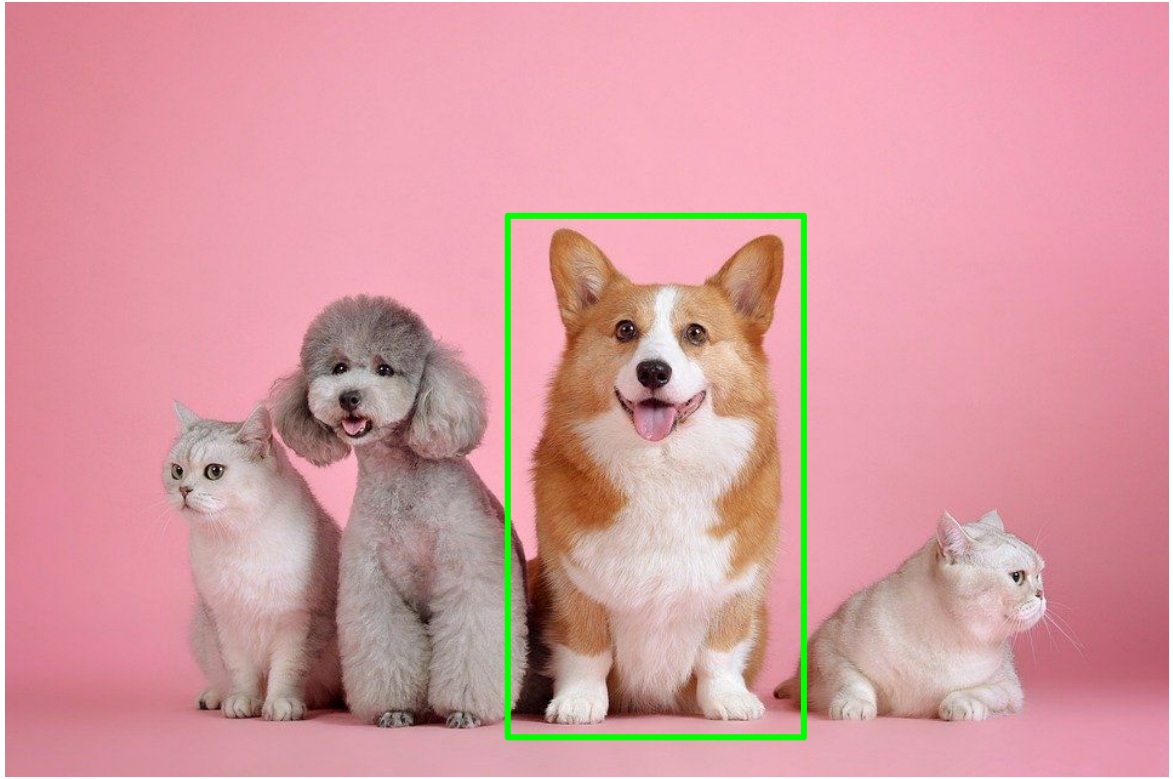
Janelas deslizantes

A estratégia mais simples para tratar o problema é realizar a classificação utilizando a ideia de janelas deslizantes:

Gato: não

Cachorro: sim

Backgound: não



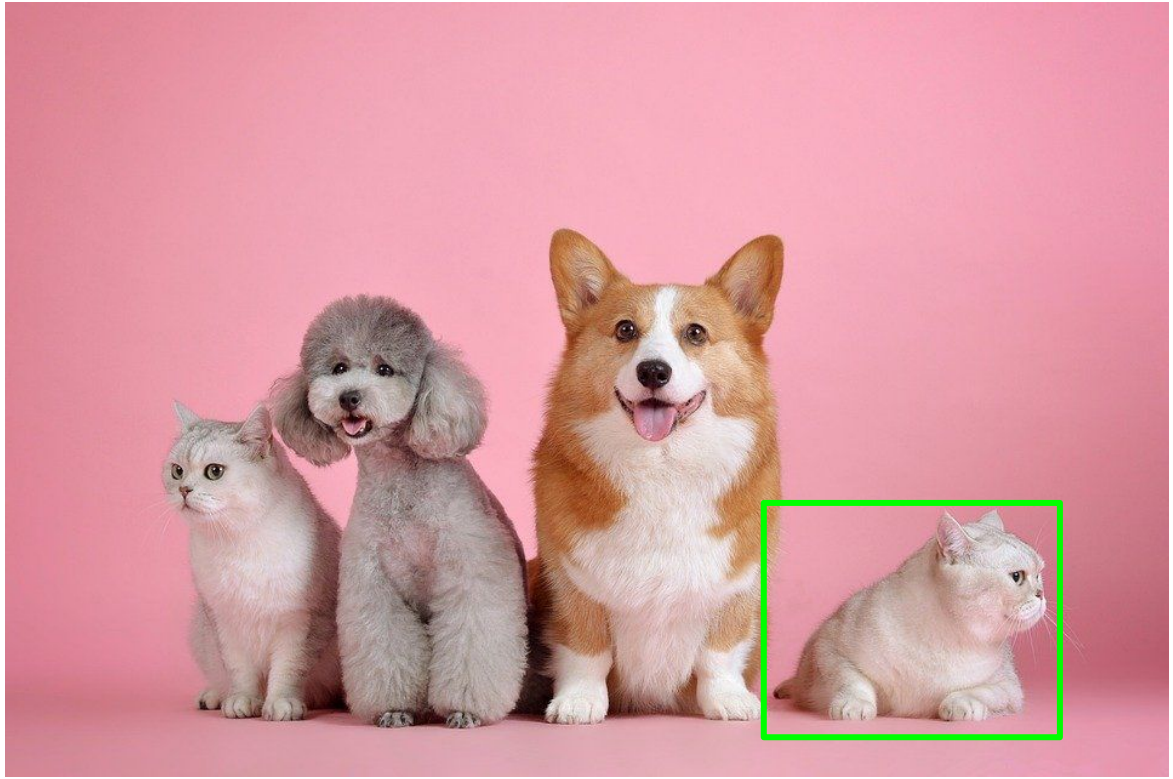
Janelas deslizantes

A estratégia mais simples para tratar o problema é realizar a classificação utilizando a ideia de janelas deslizantes:

Gato: sim

Cachorro: não

Backgound: não

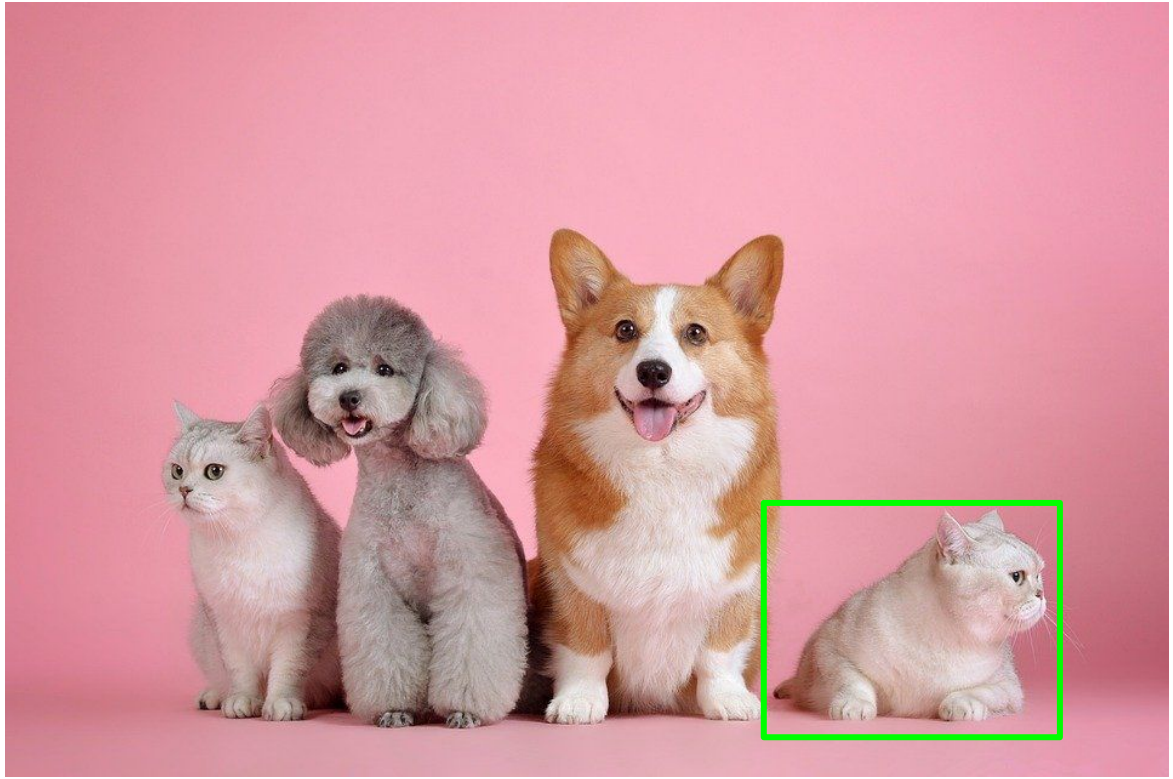


Janelas deslizantes

Problema: como definir os bounding boxes?

- aparecem em tamanhos e formatos diferentes
- podem estar em qualquer lugar da imagem

Realizar testes com força bruta faria com que o tempo de inferência se tornasse inviável.



Proposta de Regiões

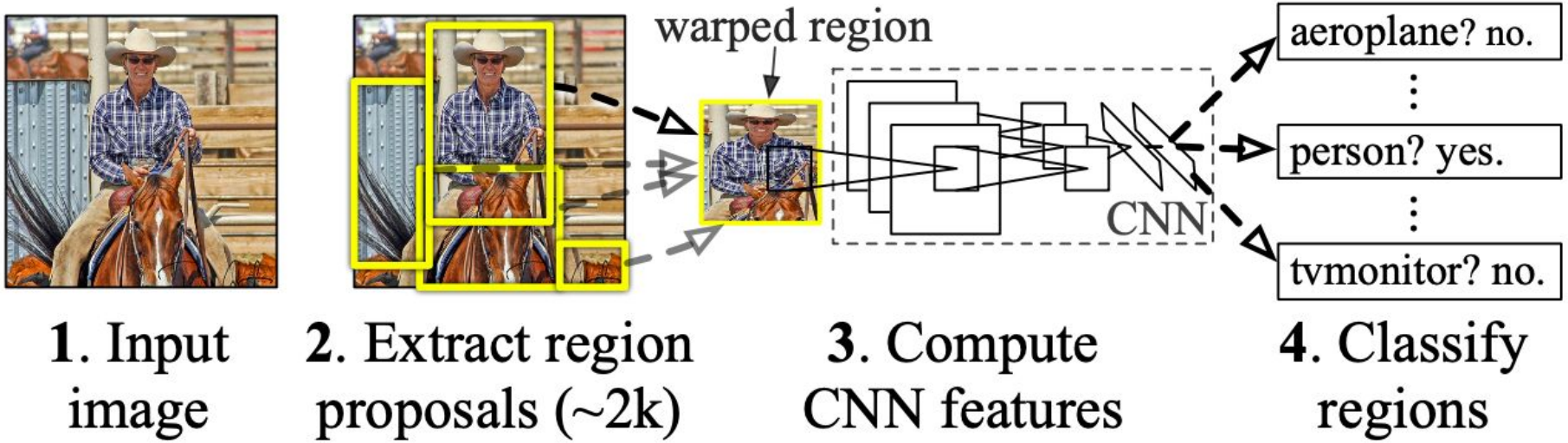
Como solução para esse problema, existe uma linha de pesquisa volta para construir propositores de regiões (region proposals):

- encontrar regiões na imagem que tem maior probabilidade de conter objetos
- relativamente rápido para encontrar regiões
- em geral, usam técnicas mais clássicas de visão computacional

A partir disso, aplicar o classificador nas regiões propostas.

Selective Search

R-CNN

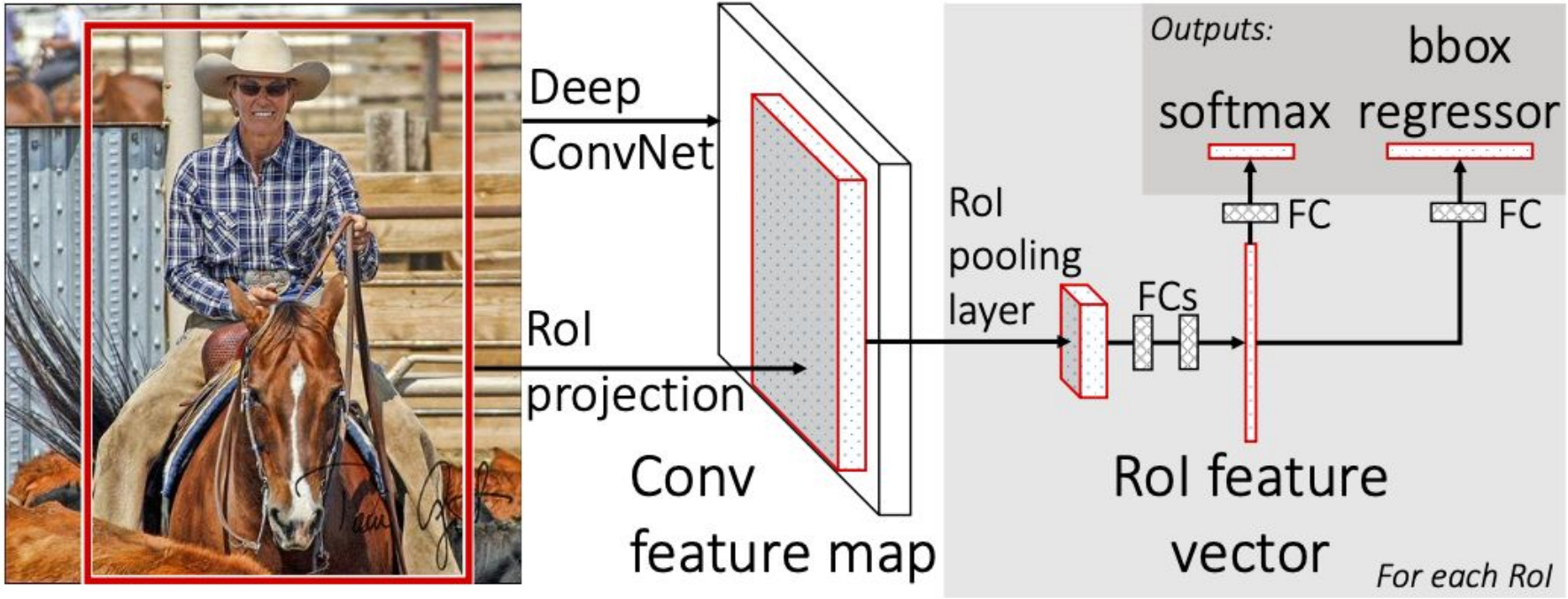

warped region
CNN
aeroplane? no.
person? yes.
tvmonitor? no.
1. Input image
2. Extract region proposals (~2k)
3. Compute CNN features
4. Classify regions

R-CNN

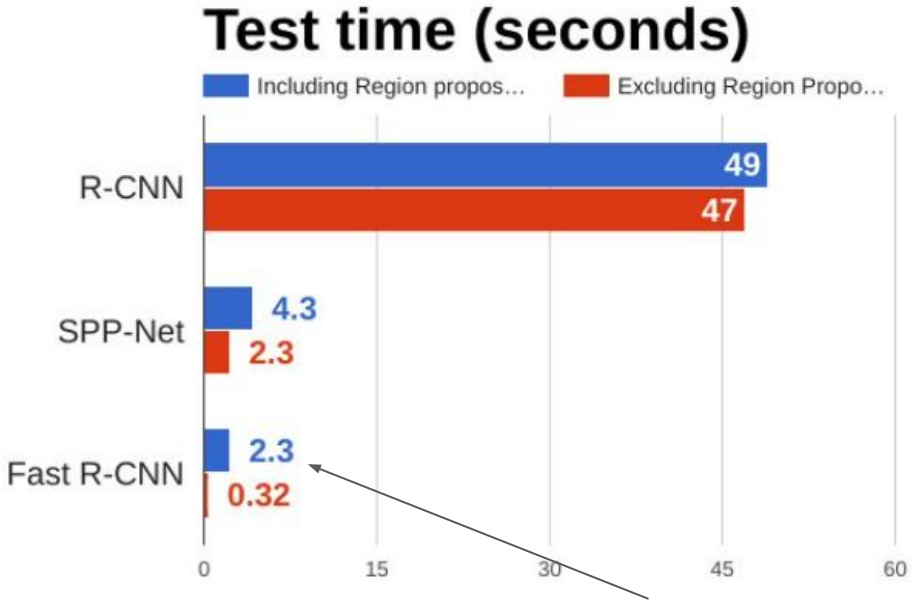
O método possui alguns problemas:

- treinamento é lento
- tempo de inferência é lento
- depender de outro algoritmo para determinar as regiões de interesse
- requer muito espaço em disco

Fast R-CNN


Deep
ConvNet
RoI
projection
Conv
feature map
RoI
pooling
layer
FCs
FC
FC
Outputs:
softmax
bbox
regressor
RoI feature
vector
For each RoI

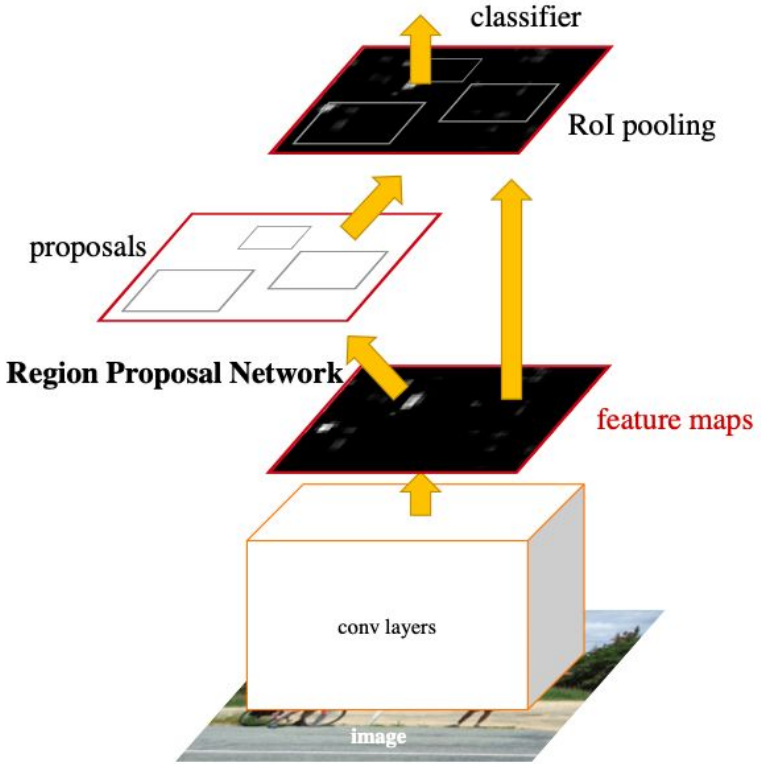
Fast R-CNN é mais rápida



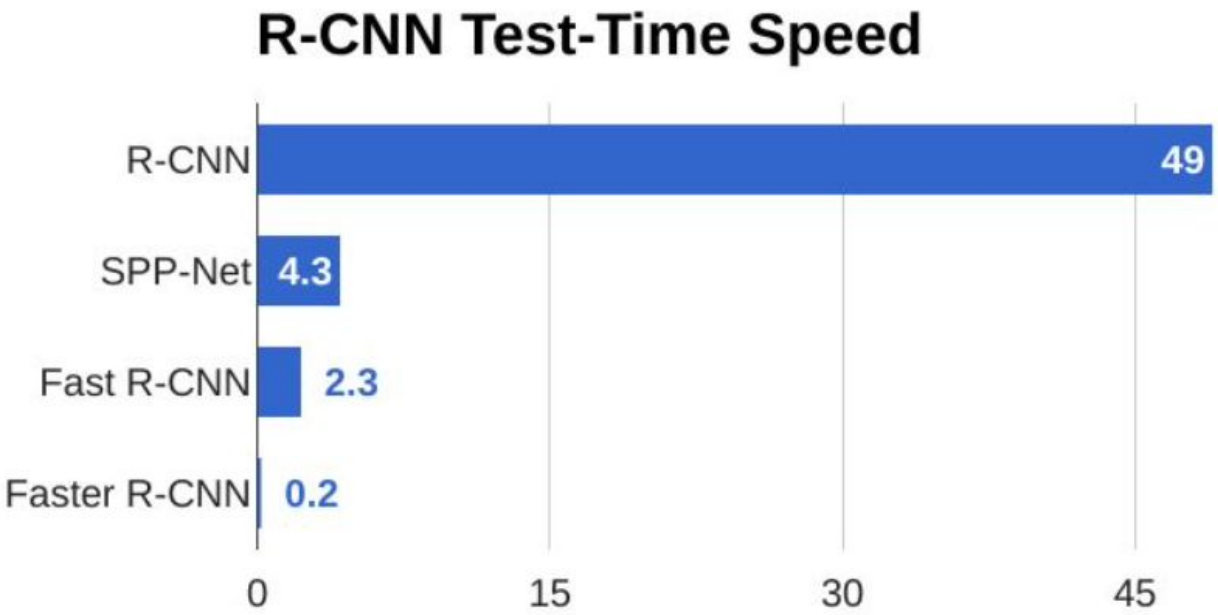
A maior parte do tempo é para proposição de regiões.

Faster R-CNN

Insere uma rede para propor regiões de interesse a partir de features.



Faster R-CNN é mais rápida ainda



Detecção sem proposição de regiões

Alguns métodos não realizam a detecção usando duas etapas (proposição de regiões e classificação). Ao invés disso, utilizam uma grande rede convolucional para realizar a classificação em uma única "passada" .

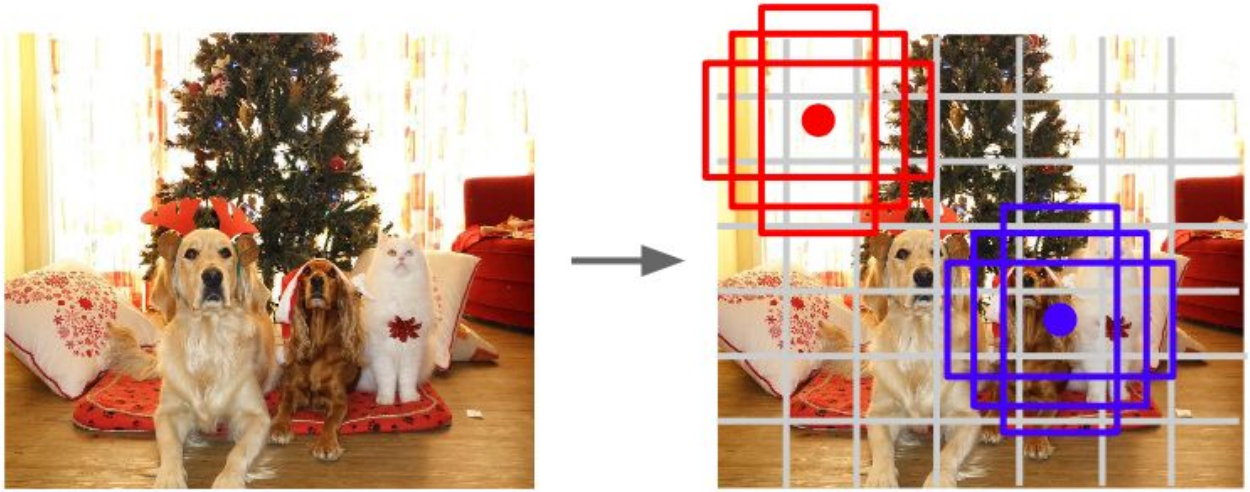
As redes partem de duas ideias básicas:

1. a imagem é dividida em um grid esparso (7x7, por exemplo)
2. usar bounding boxes base em cada célula para definir o bounding box final.

Nesse tipo de rede, o problema de encontrar o objeto é encarado como uma regressão dos bounding boxes básicos.

YOLO

A YOLO (You Only Look Once) é uma das principais representantes dessa categoria de rede.



YOLO

Em cada célula:

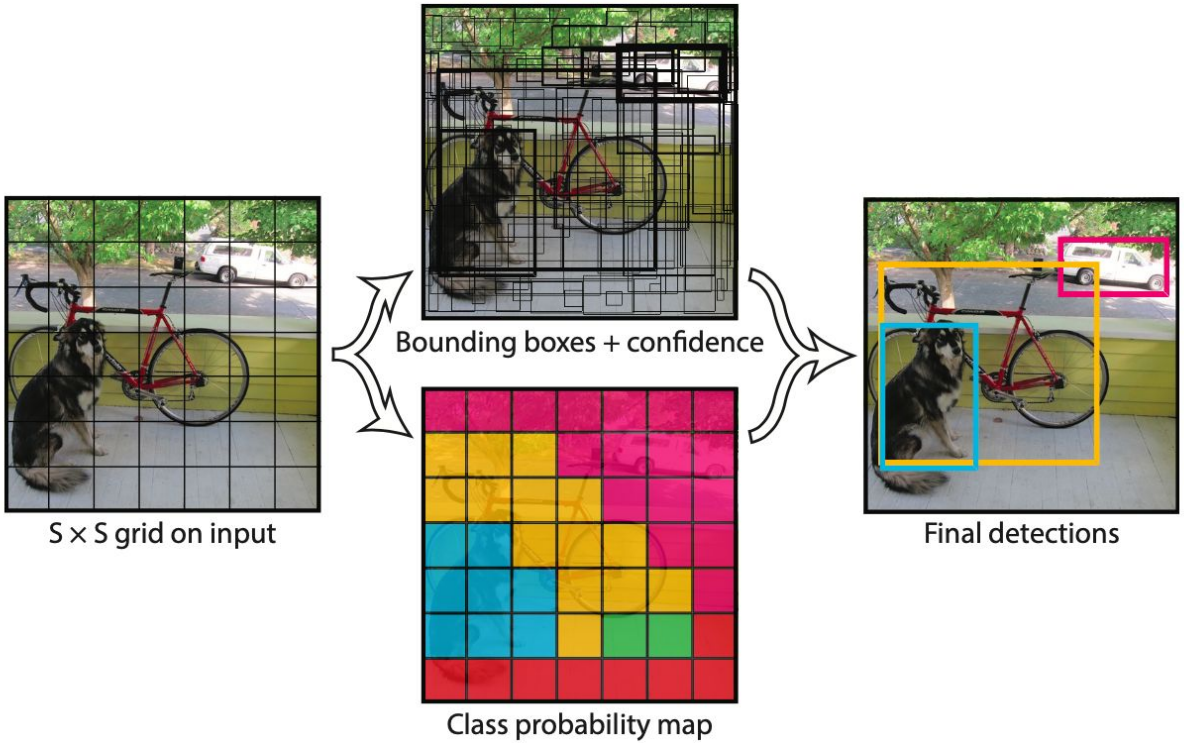
- realiza a regressão de cada um dos B bounding box base.
- o resultado da regressão é um vetor com 5 valores:
  - (x, y, h, w, confiança)
- prediz o score para cada classe C

A saída final é um tensor com o seguinte tamanho:

$$G \times G \times (5 * B + C)$$

G = dimensão do grid, B = número de bounding boxes base, C = número de classes

YOLO


Bounding boxes + confidence
S × S grid on input
Class probability map
Final detections

Fim