Aprendizado Profundo – Conceitos básicos

Prof. Clodoaldo A M Lima

Sumário

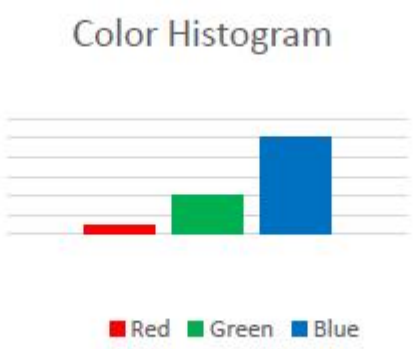
- Etapas de um processo de Aprendizado de máquina
- Generalização
- Regularização
- Como evitar sobre-ajuste
- Revisão sobre Processamento de Sinais
  - Definiação de Convolução 1D
- Compartilhamento de pesos
- Correlação
- Convolução 2D
- Padding
- Pooling

Aprendizado de Máquina

- 1) Coletar dados e extrair características
- 2) Construir o modelo: escolha classe de hipótese $\mathcal{H}$ e da função de perda ℓ
- 3) Otimização: minimizar a perda empírica

Extração de Características

Extrai as características



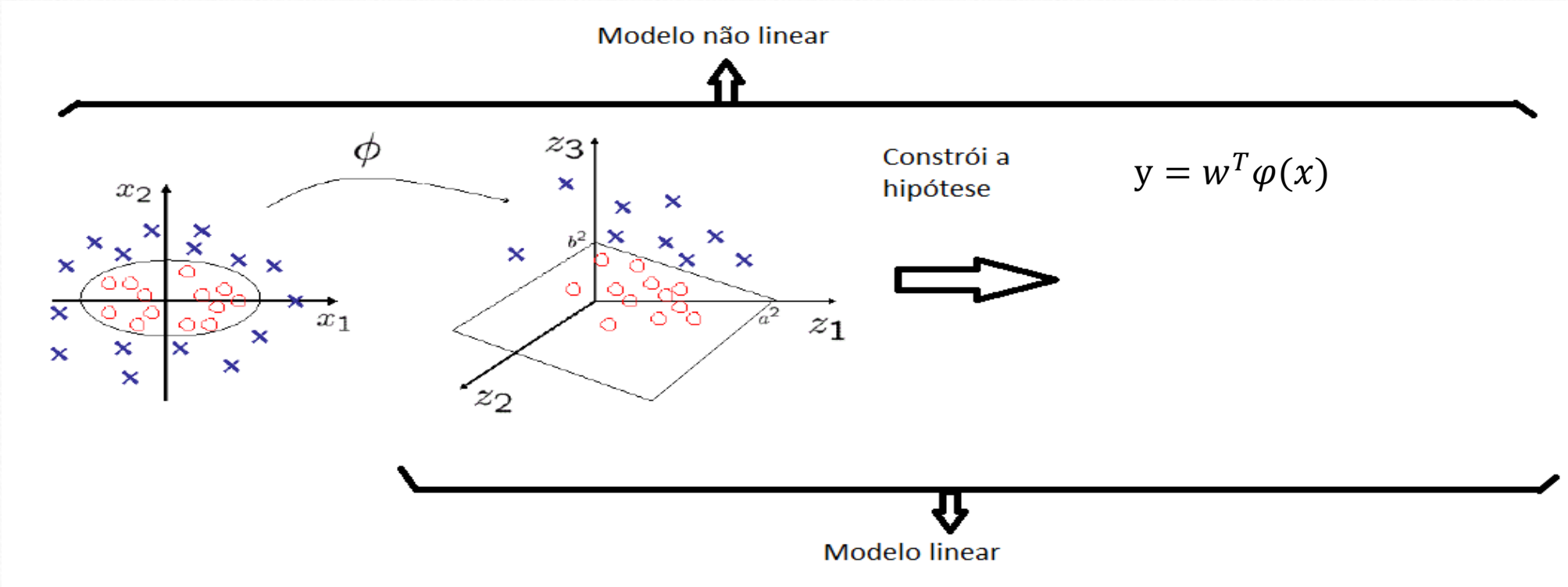
Constrói a hipotese

$$y = w^T \varphi(x)$$

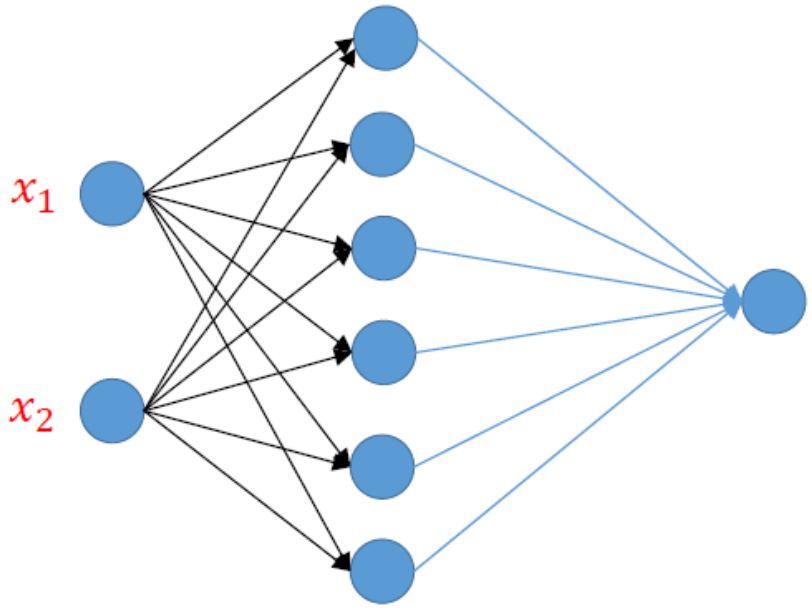
Extração de características

- Problemas:
  - Nem sempre as características relevantes para a classificação são extraídas
  - É dependente do contexto
  - Muitas vezes requer um especialista para selecionar as características para determinada área

Características – Parte do modelo (SVM)



Exemplo – SVM com Kernel Polinomial



$\varphi(x)\ fixado$

$$y = sign(w^T \varphi(x) + b)$$

Motivação : Aprender a representação

- Por que não podemos aprender $\varphi(x)$?

x

$Aprender$

$\varphi(x)$

$\phi(x)$

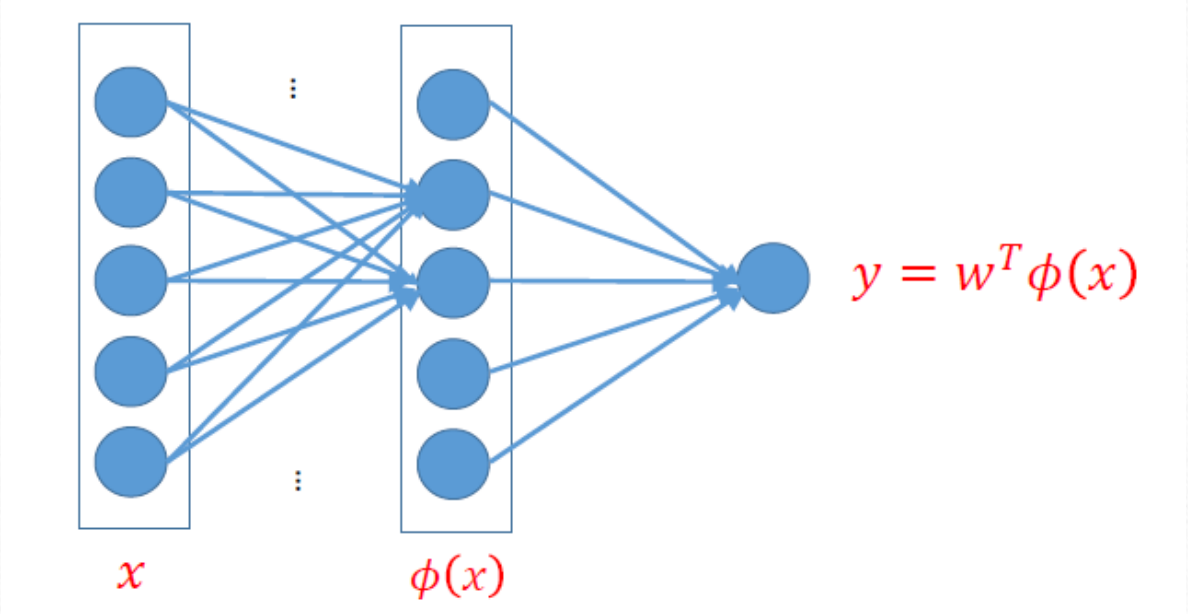
$Aprender$

w

$y = w^T \varphi(x)$

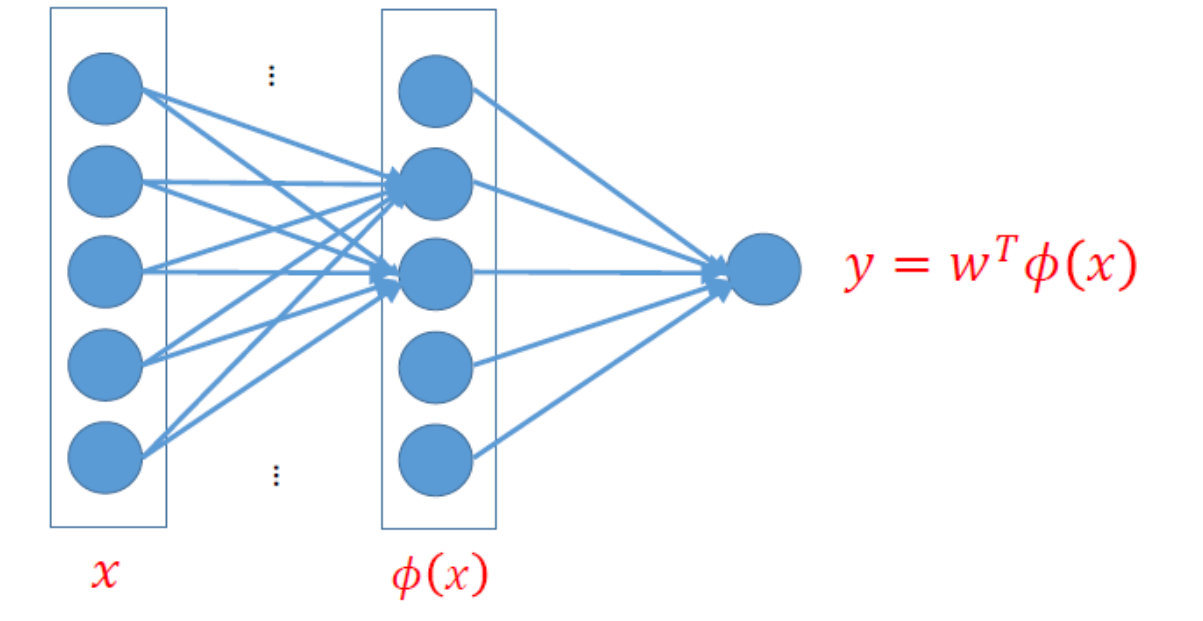
Redes Neurais Feedforward

- Ver cada dimensão de $\varphi(x)$ como algo a ser aprendido



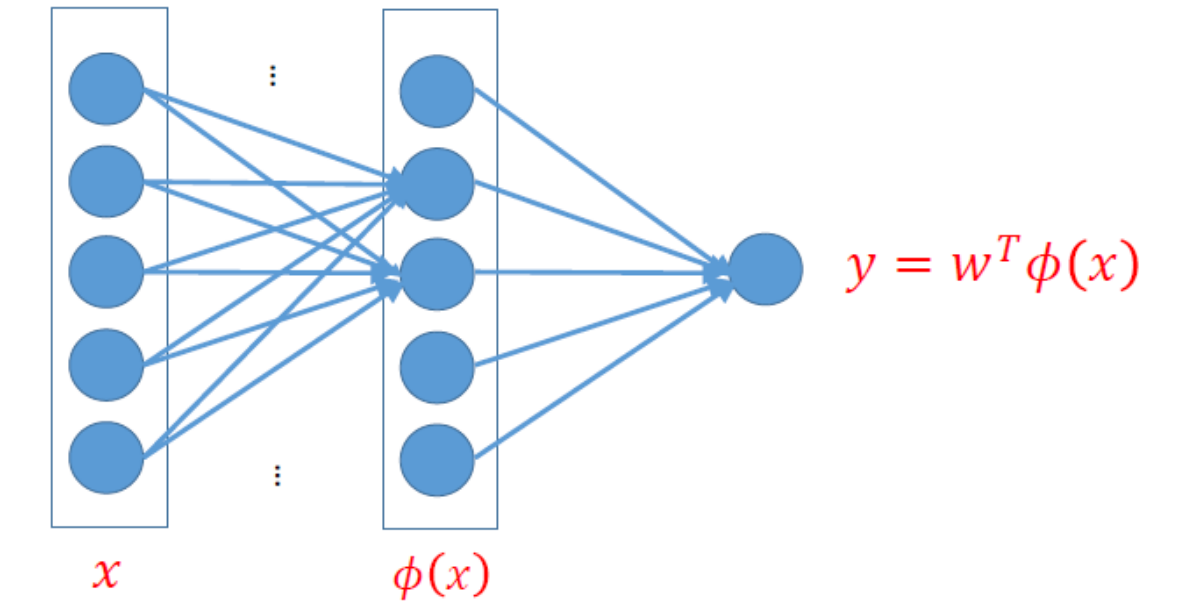
Redes Neurais FeedForward

- Funções lineares $\varphi(x) = \theta^T x$ não funcionam: necessita de alguma não linearidade



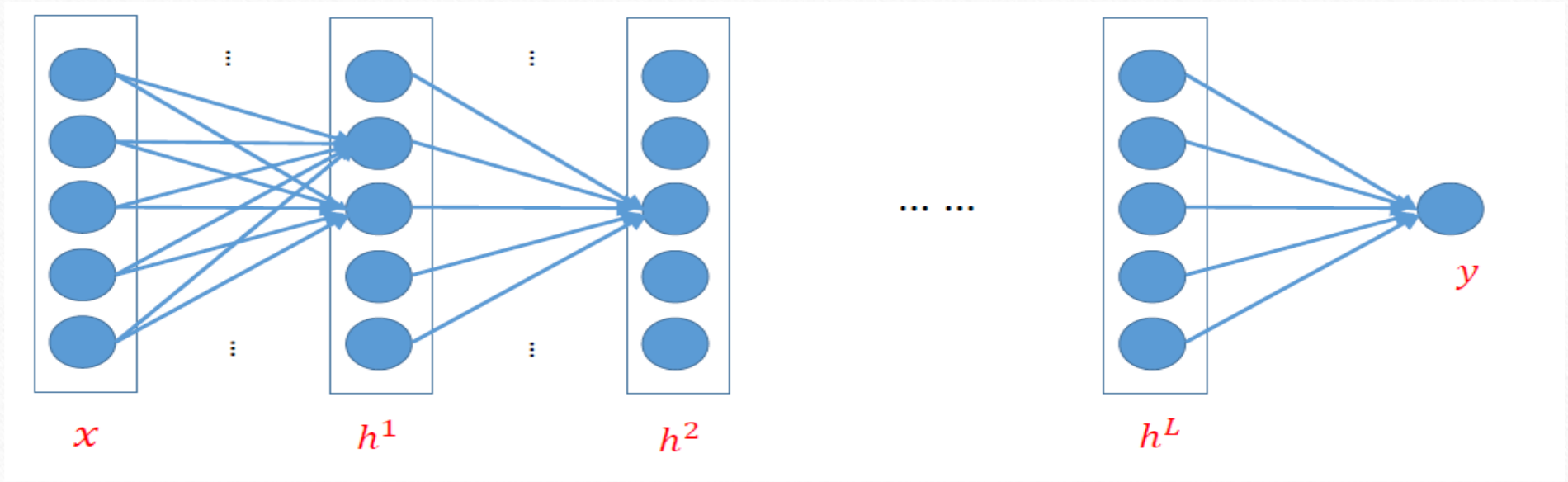
Redes Neurais FeedForward

- Tipicamente, $\varphi(x) = f(\theta^T x)$, onde f é alguma função não linear



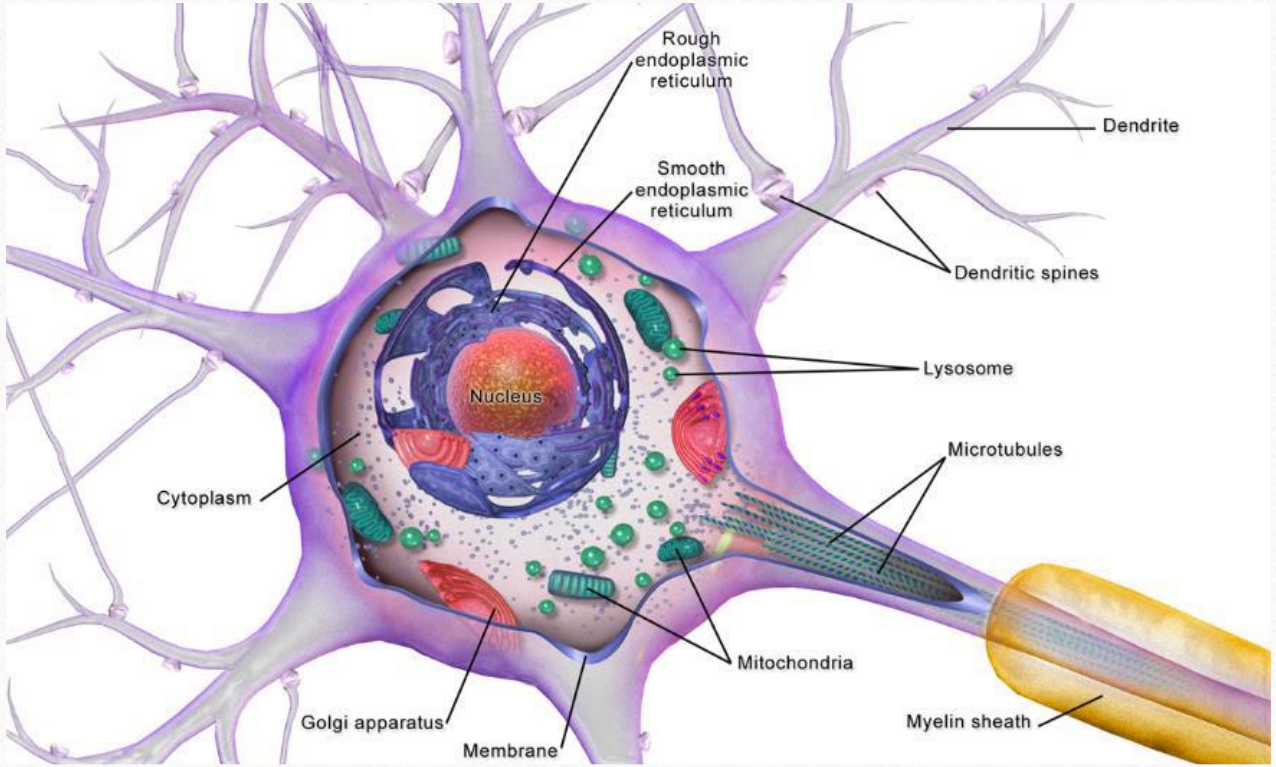
Redes Neurais Feedforward Profunda

- Podemos adicionar camadas



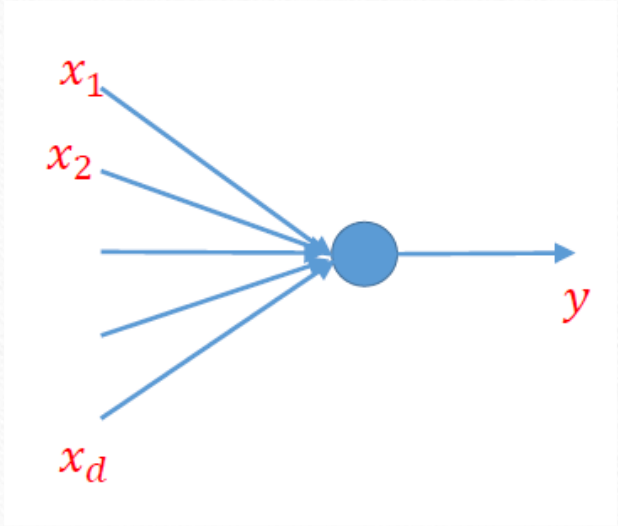
Relembrando: Neurônio

- Fonte wikipedia



Modelo abstrato do Neurônio

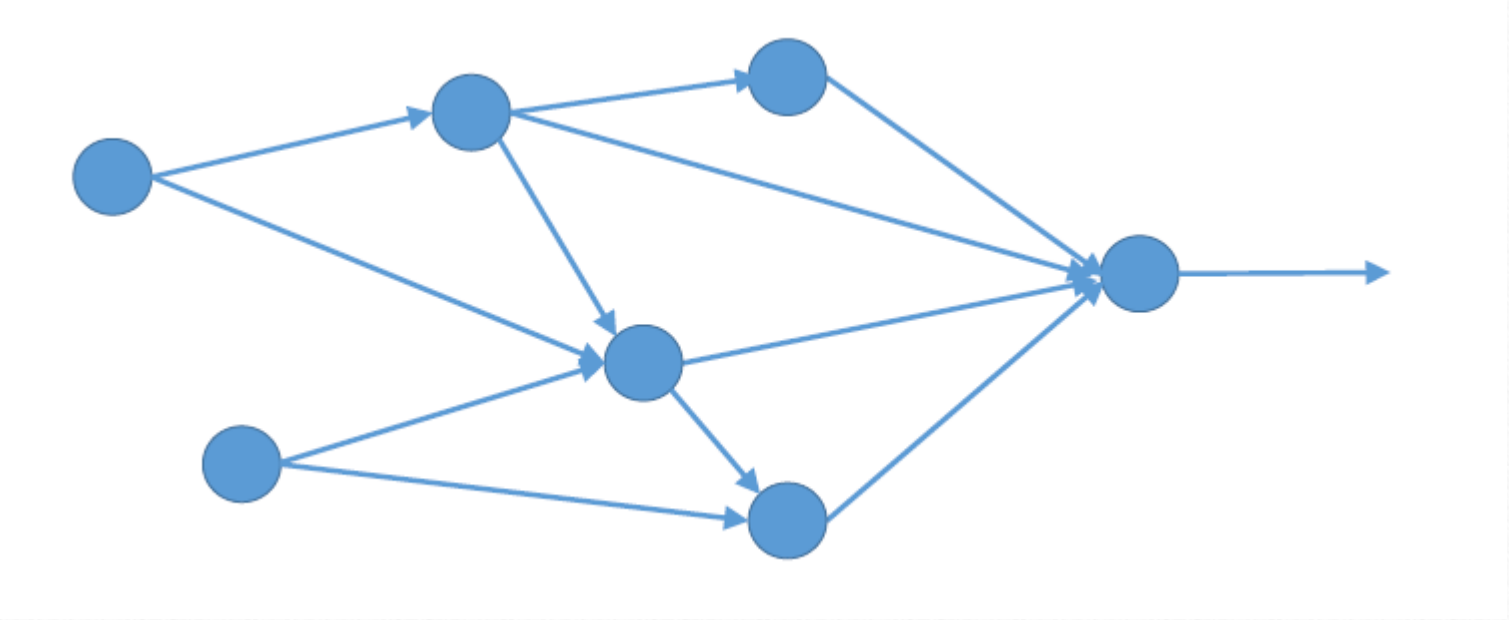
- Neurônio ativado quando a correlação entre a entrada e um vetor de pesos θ excede algum limite b



$$y = f(\theta^T x - b)$$

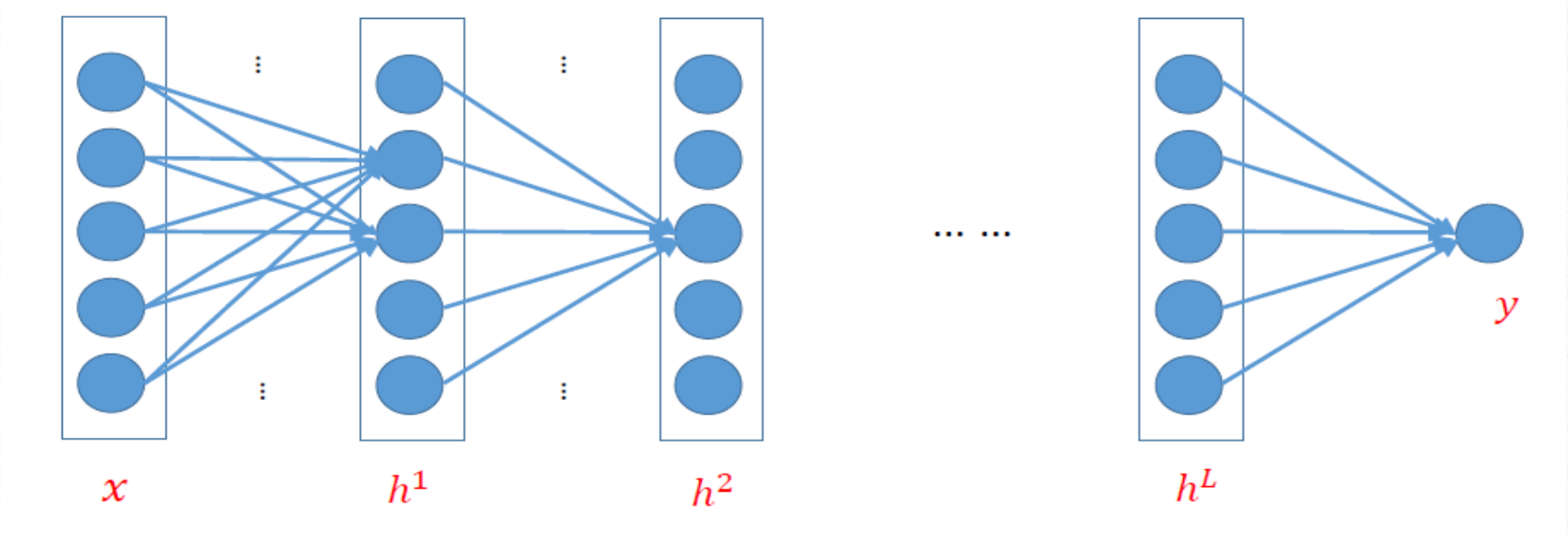
- f é função de ativação

Rede Neural Artificial Feedforward



Redes Neurais Artificiais

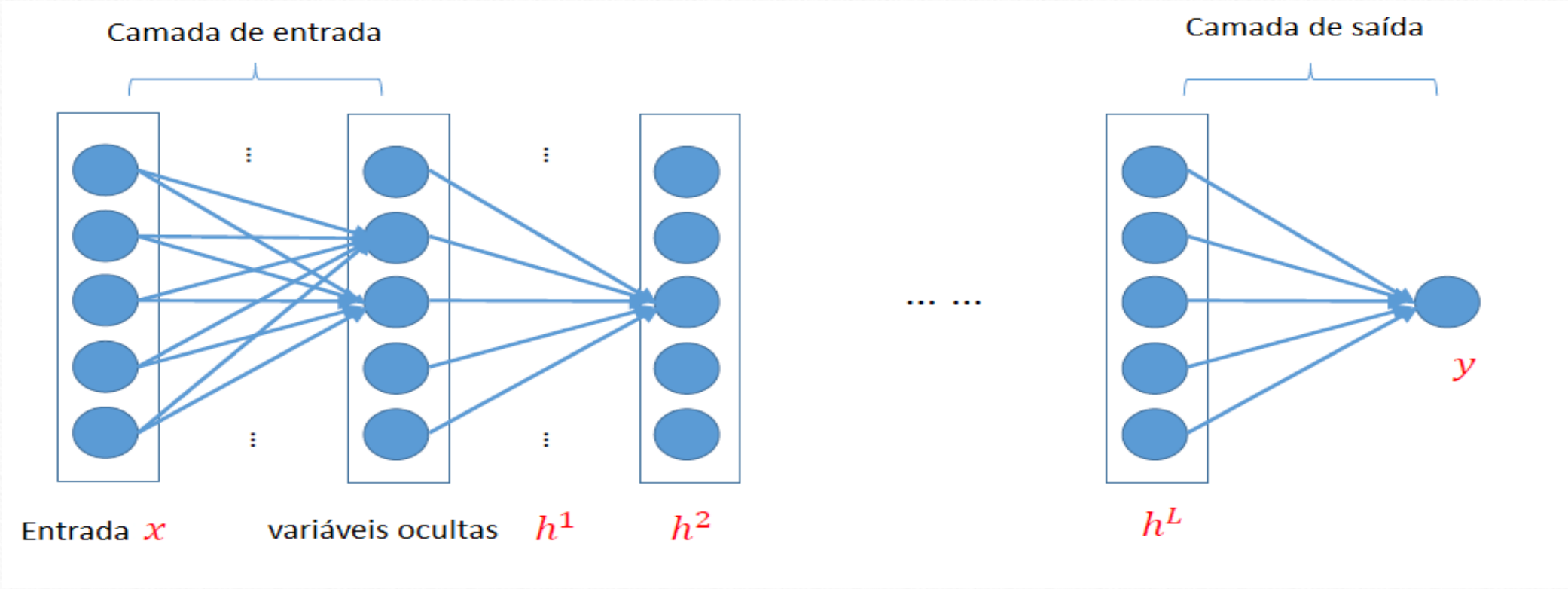
- Colocar em camadas: redes Neurais profundas
  - Muitos parâmetros a serem sintonizados



Componentes de uma Rede Neural

- Representações:
  - Entrada
  - Variáveis ocultas
- Camadas / pesos:
  - Camadas ocultas
  - Camada de saída

Componentes



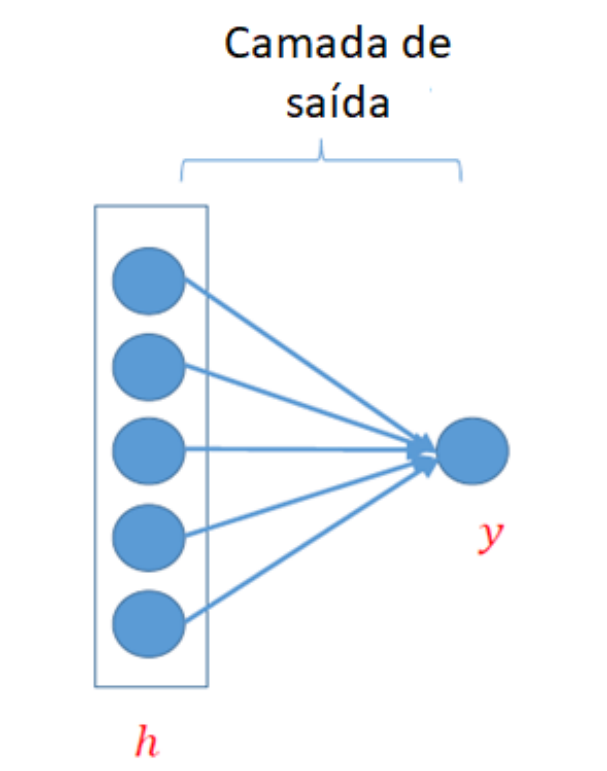
Entrada

- Representado como um vetor
- Às vezes exigem algum pré-processamento, por exemplo,
  - Subtrair média
  - Normalizar para [-1,1]

- EXPANDE

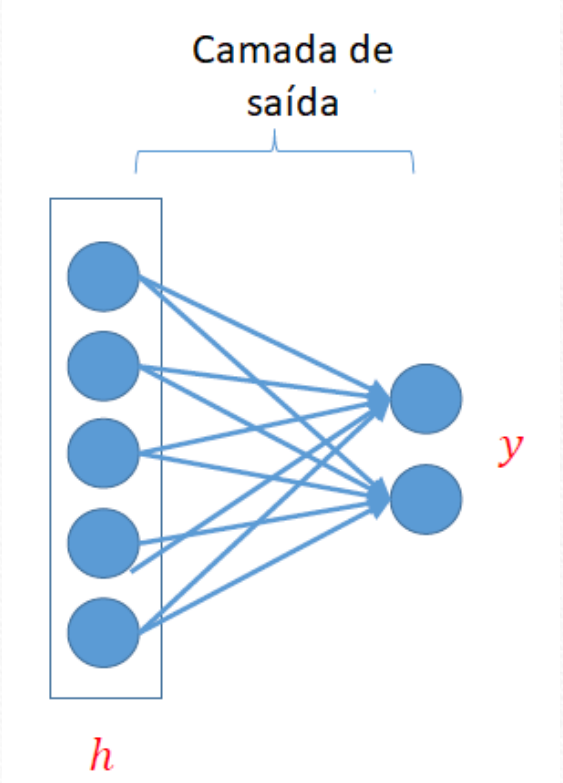
Camada de Saída

- Regressão $y = w^T h + b$
- Geralmente Unidades lineares:
  - Nenhuma não linearidade



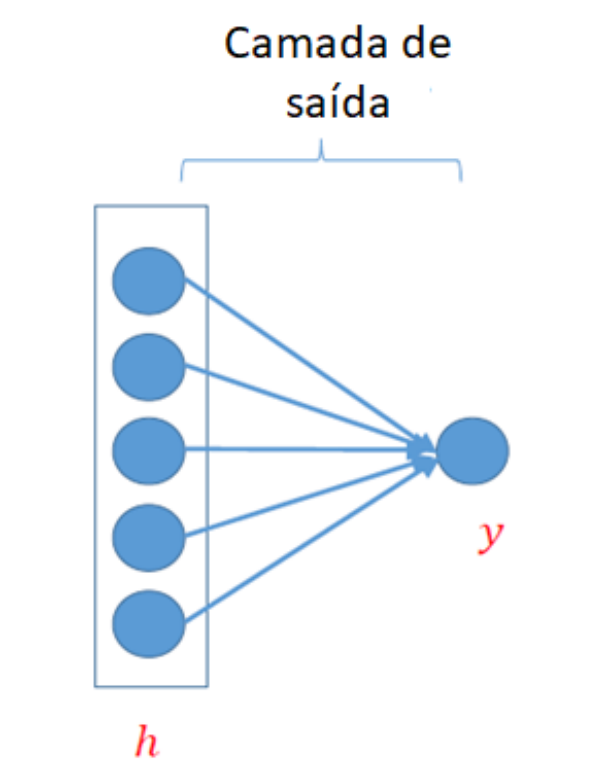
Camada de Saída

- Regressão Multi-dimensional $y = w^T h + b$
- Geralmente Unidades lineares:
  - Nenhuma não linearidade



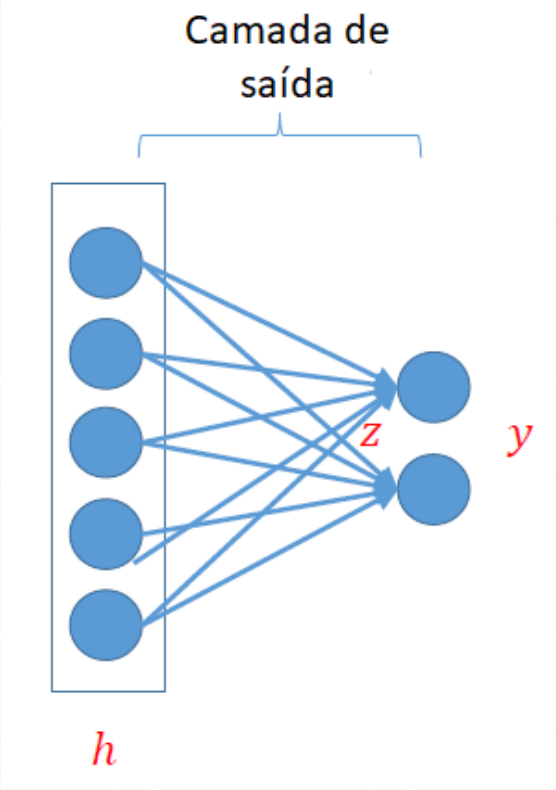
Camada de Saída

- Classificação binária: $y = \sigma(w^T h + b)$
- Corresponde a regressão logística sobre h



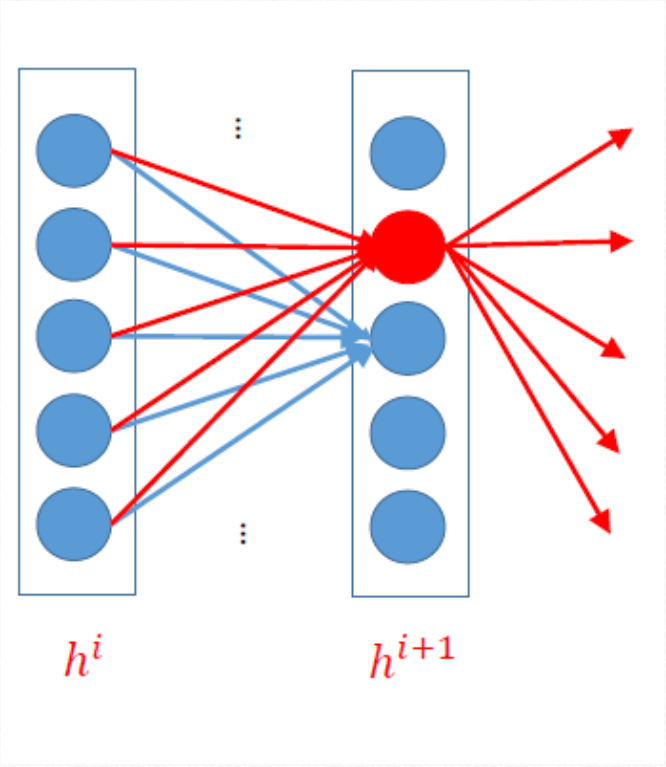
Camada de Saída

- Classificação múltiplas Classes
- Função softmax $y(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^{h} \exp(z_j)} \quad z_i = w_i^T h$
- Corresponde a usar regressão logística múltiplas classes



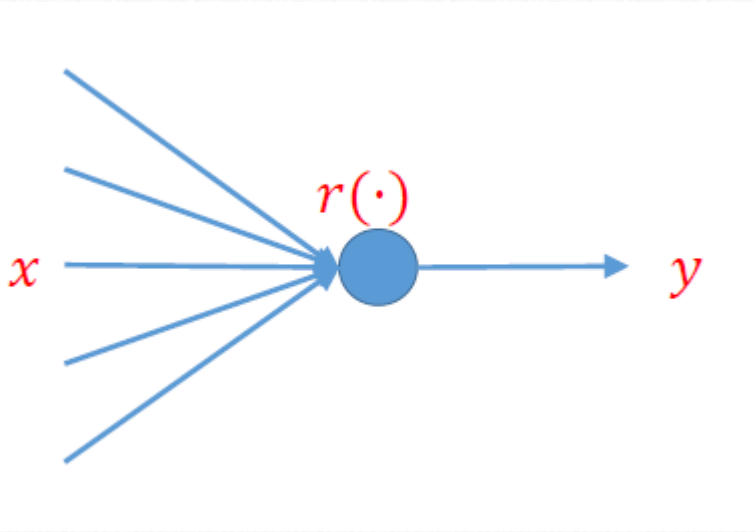
Camada escondida

- Cada Neurônio realiza uma combinação linear ponderada da saída das camadas anteriores
- Produz um valor para a próxima camada



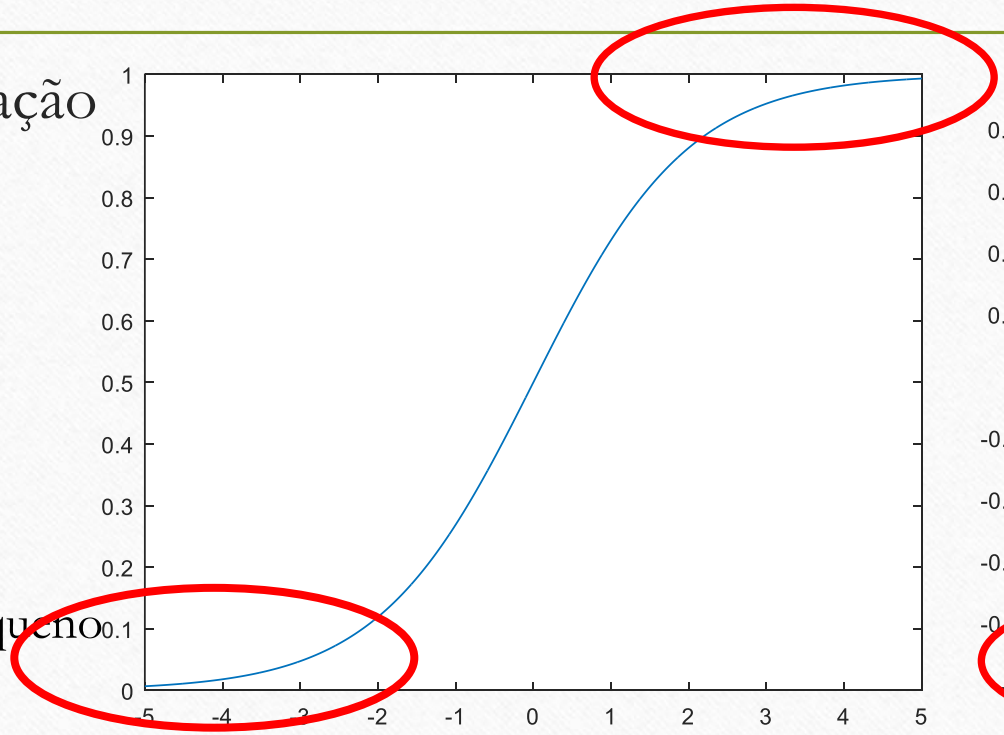
Camada escondida

- $y = \sigma(w^T x + b)$
- Função ativação σ
- Sigmoide $\sigma(x) = \frac{1}{1+\exp(x)}$
- Tangente Hiperbólica $\sigma(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$

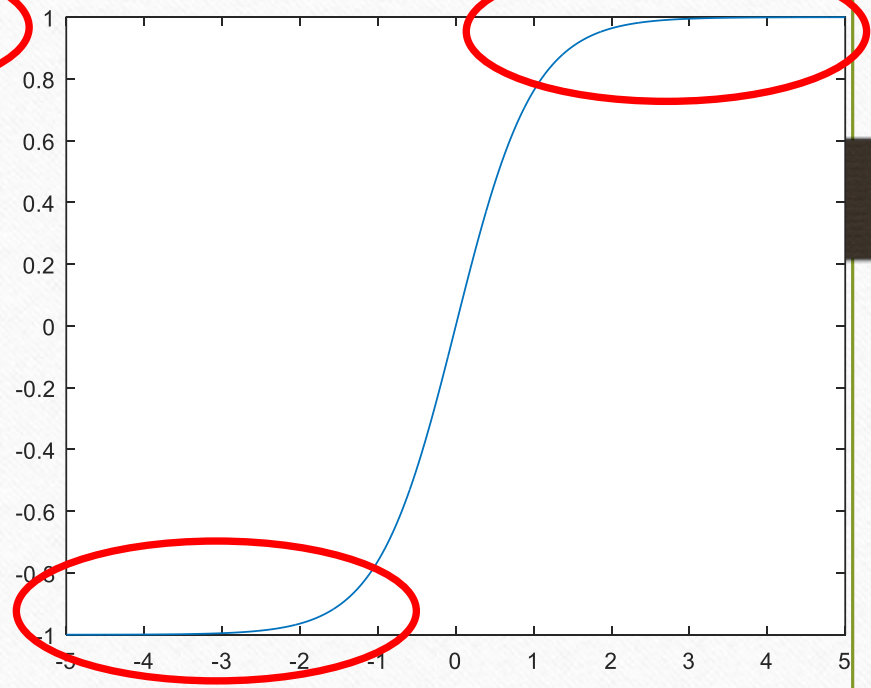


Camada escondida

- Problema saturação
- Gradiente muito pequeno



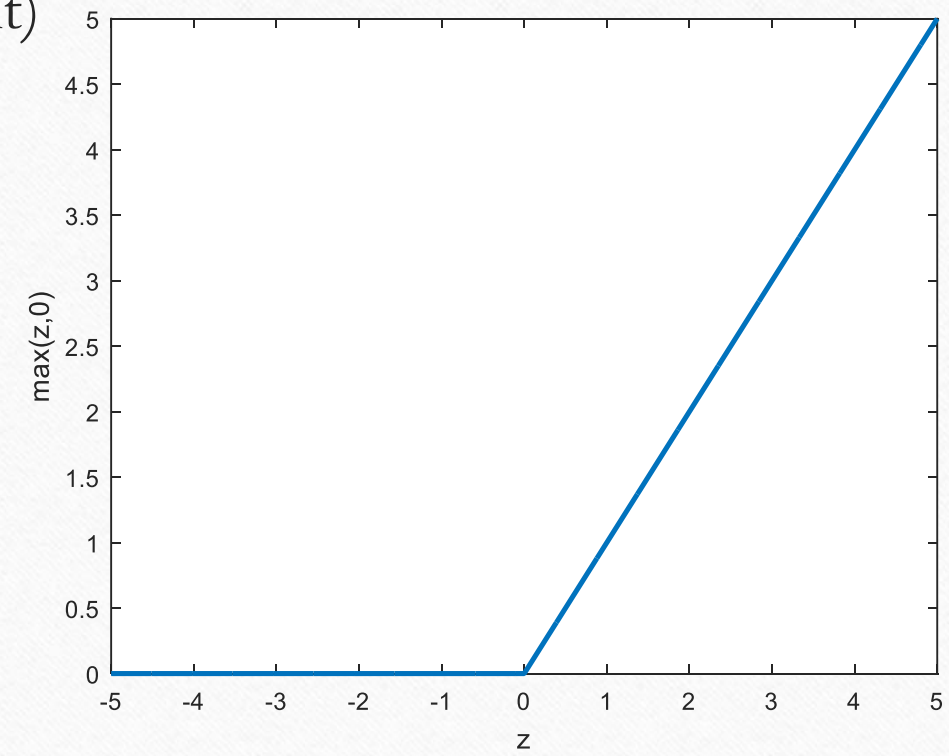
- FUNÇÃO SIGMOIDE



- FUNÇÃO TANGENTE HIPERBÓLICA

Camada escondida

- Função de ativação ReLU (Rectified Linear Unit)
  - $ReLU(z) = \max(z, 0)$

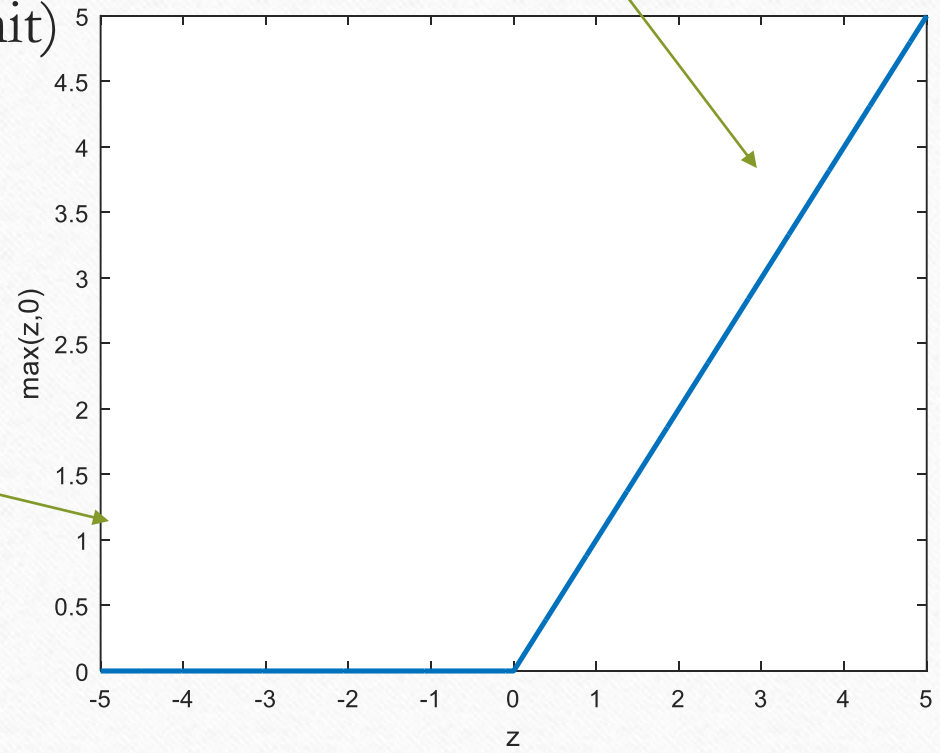


Camada escondida

- GRADIENTE IGUAL A 1

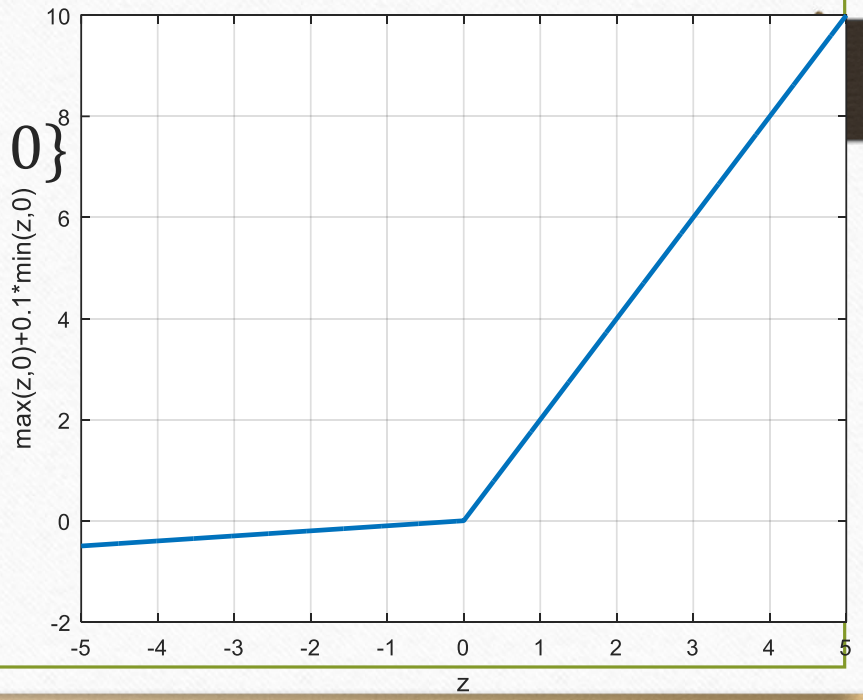
- Função de ativação ReLU (Rectified Linear Unit)
  - $ReLU(z) = \max(z, 0)$

- GRADIENTE IGUAL A 0



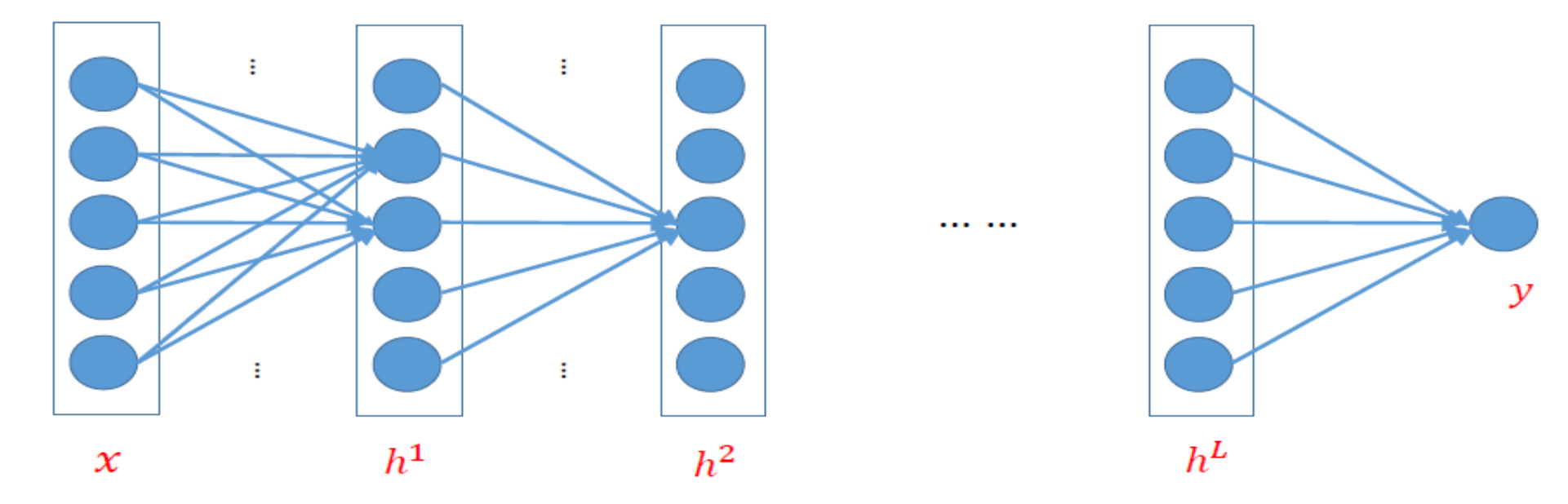
Camada escondida

- Generalizações da Relu $gReLU(z) = \max\{z, 0\} + \alpha\min\{z, 0\}$
- Leaky-ReLU $\mathrm{LR}eLU = \max\{z, 0\} + 0{,}001\min\{z, 0\}$
- Parametric-Relu(z): α *a ser aprendiddo*



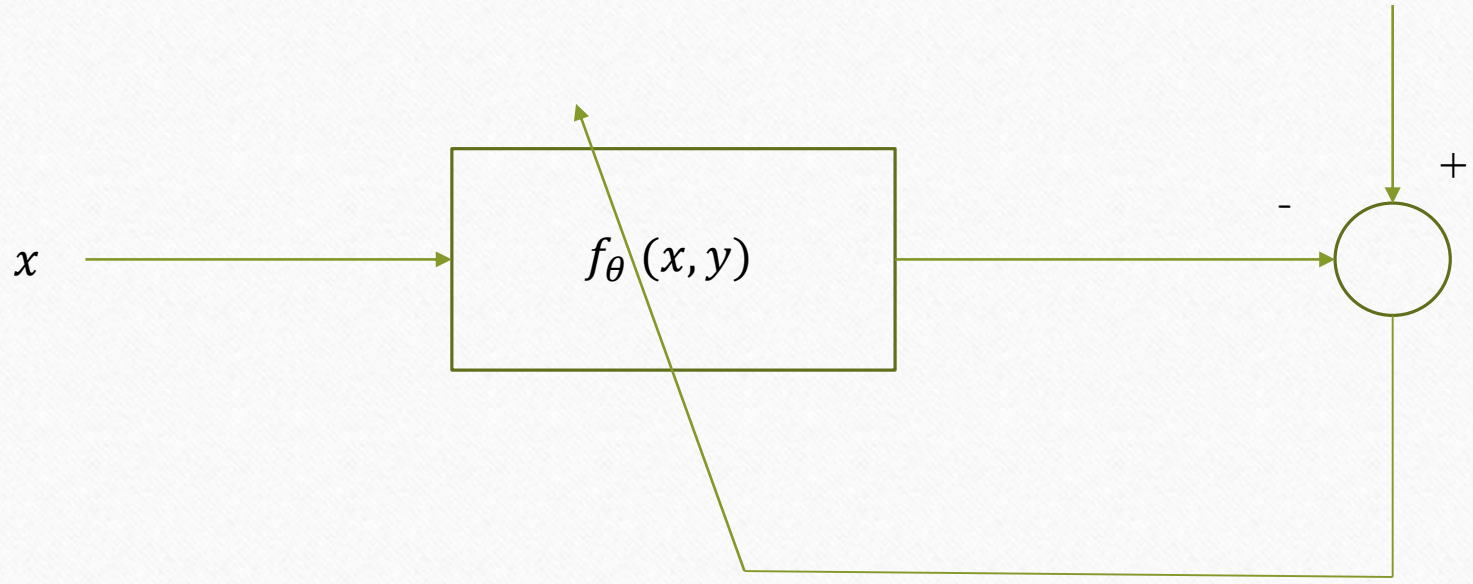
Algoritmo Backpropagation

- Como treinar uma rede neural?



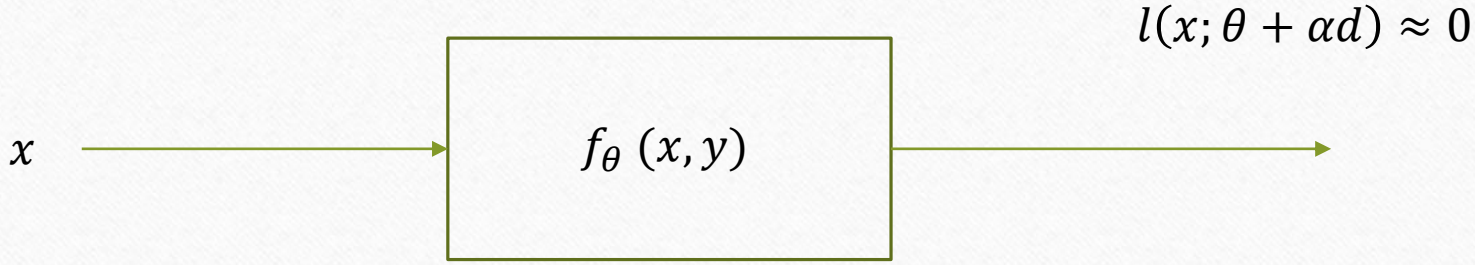
Algoritmo Backpropagation

- Defina uma função de perda $l(x;\theta) = l(f_\theta, x, y)$



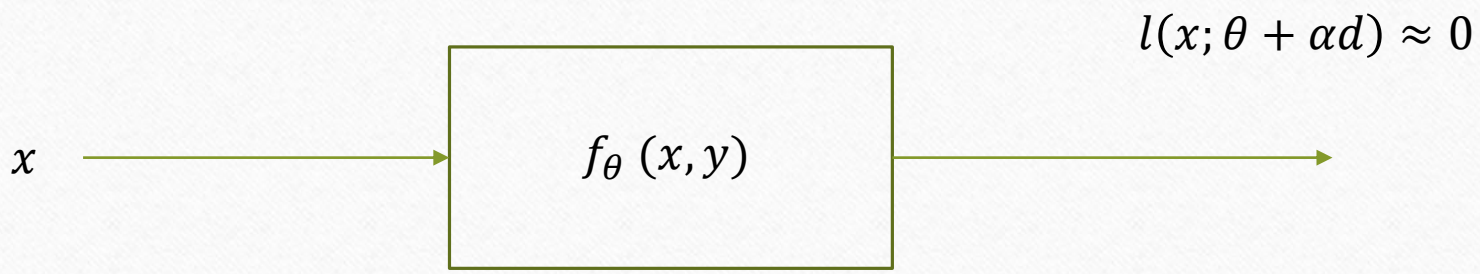
Algoritmo Backpropagation

- Encontrar a direção d



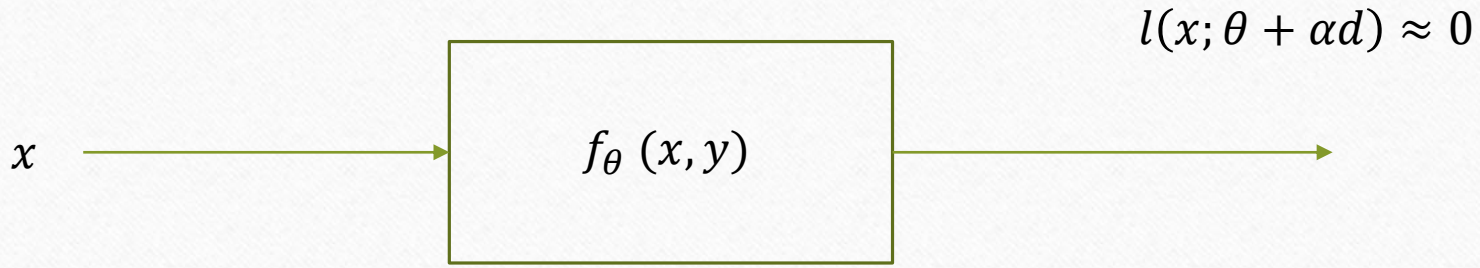
Algoritmo Backpropagation

- Encontrar a direção d: $l(x;\theta+\alpha d) = l(x;\theta) + \nabla l(x;\theta)\ \alpha d$ para escalar α pequeno



Algoritmo Backpropagation

- Conclusão; move θ em direção $-\nabla l(x;\theta)$ para escalar α pequeno



Gradiente Descendente

- Minimiza a perda $\hat{l}(\theta)$, onde a hipótese é parametrizada por θ
- Gradiente Descendente
- Inicialize θ
- Enquanto $\left\| \nabla \hat{l}\ (\theta_t) \right\| > \varepsilon$
  - $\theta_{t+1} = \theta_t + \alpha\ \nabla \hat{l}\ (\theta_t)$

Vanishing Gradient Problem

- Redes Neurais com muitas camadas: transformações sucessivas da informação de entrada;
- Transformam a representação em um nível, começando da camada de entrada, em uma representação em um nível maior e ligeiramente mais abstrato;
- Problema: perda da informação do gradiente;
- Utilizando o Backpropagation, a informação do gradiente é propagada para trás;
- Entretanto, quanto mais se distancia da camada de saída, mais a informação do gradiente diminui;

Vanishing Gradient Problem

- Este problema, aliado ao crescimento do número de parâmetros ajustáveis(pesos), são os principais motivos de redes neurais convencionais utilizarem poucas camadas;
- O problema é ainda pior quando são utilizadas funções de ativação tradicionais como sigmoida e tangente hiperbólica;
- O uso da regra da cadeia tem o efeito de multiplicar vários números pequenos produzidos pelas funções de ativação, fazendo com que o gradiente diminua exponencialmente.

Gradiente descendente Estocástico

- Na prática, o cálculo do custo e do gradiente para todo o conjunto de treinamento pode ser muito lento e às vezes intratável em uma única máquina se o conjunto de dados for grande demais para caber na memória principal.
- Outro problema com os métodos de otimização em lote é que eles não fornecem uma maneira fácil de incorporar novos dados em uma configuração 'online'.
- O gradiente descendente estocástico (SGD) aborda esses dois problemas, seguindo o gradiente negativo da função objetivo, depois de ver apenas um ou alguns exemplos de treinamento.
- O uso do SGD na configuração de rede neural é motivado pelo alto custo da propagação de retorno ao longo de todo o conjunto de treinamento.
- A SGD pode superar esse custo e ainda levar a uma convergência rápida.

Gradiente descendente Estocástico

- Suponha que os pontos de dados cheguem um por um
- $\hat{l}(\theta) = \frac{1}{n}\sum_{t=1}^{n} l(\theta, x_t, y_t)$, mas nós conhecemos somente $l(\theta, x_t, y_t)$ no instante t
- Ideia: simplesmente faça o que puder com base nas informações locais
- Inicialize θ
- Enquanto$\|\nabla l(\theta_t, x_t, y_t)\| > \varepsilon$
  - $\theta_{t+1} = \theta_t + \alpha \nabla l(\theta_t, x_t, y_t)$

Mini-batch

- Em vez de uma amostra dos dados, podemos trabalhar com um pequeno lote de b pontos
- $(x_{tb+1}, y_{tb+1}), \cdots, (x_{tb+b}, y_{tb+b})$

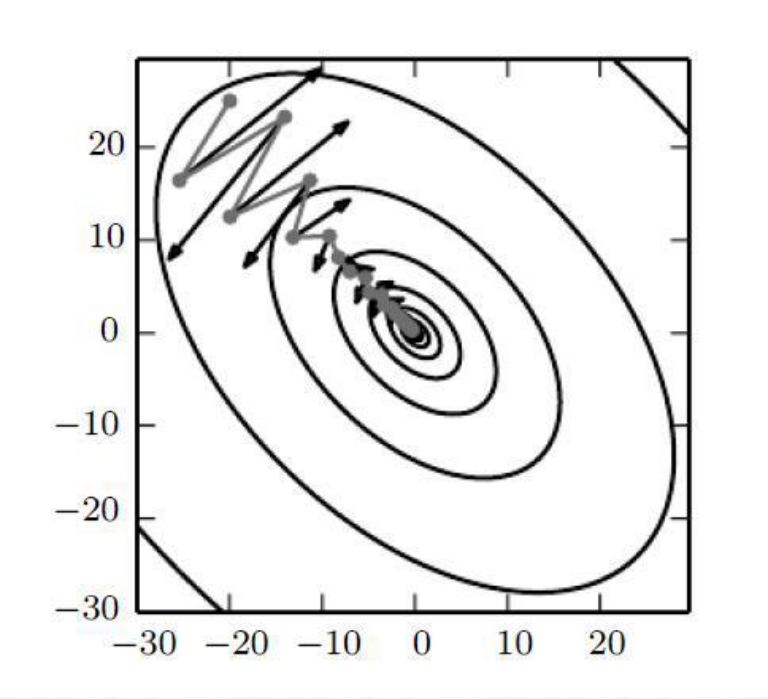
- Regra de atualização
- $\theta_{t+1} = \theta_t + \alpha_t \nabla \left(\frac{1}{b} \sum_{1 \le i \le b} l(\theta_t, x_{tb+i}, y_{tb+i}) \right)$

- Tamanho típico do lote: $b = 132$

Momento

- Desvantagem do SGD: pode ser lenta quando o gradiente é pequeno
- Observação: quando o gradiente é consistente em etapas consecutivas, pode dar passos maiores
- Metáfora: rolando uma bola em declive suave

Momento



Momento

- Trabalhe com um pequeno lote de pontos

$$(x_{tb+1}, y_{tb+1}), \ldots, (x_{tb+b}, y_{tb+b})$$

- Mantenha uma variável de momento vt e defina uma taxa de decaimento α

- Regra de atualização

$$v_t = \alpha v_{t-1} - \eta_t \nabla \left(\frac{1}{b} \sum_{1 \le i \le b} l(\theta_t, x_{tb+i}, y_{tb+i}) + R(\theta_t) \right)$$

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

Momento

- Mantenha uma variável de momento vt e defina uma taxa de decaimento α
- Regra de atualização

$$v_t = \alpha v_{t-1} - \eta_t \nabla \left(\frac{1}{b} \sum_{1 \le i \le b} l(\theta_t, x_{tb+i}, y_{tb+i}) + R(\theta_t) \right)$$

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

- Guia prático: α é definido como 0,5 até o aprendizado inicial estabilizar e depois é aumentado para 0,9 ou superior.

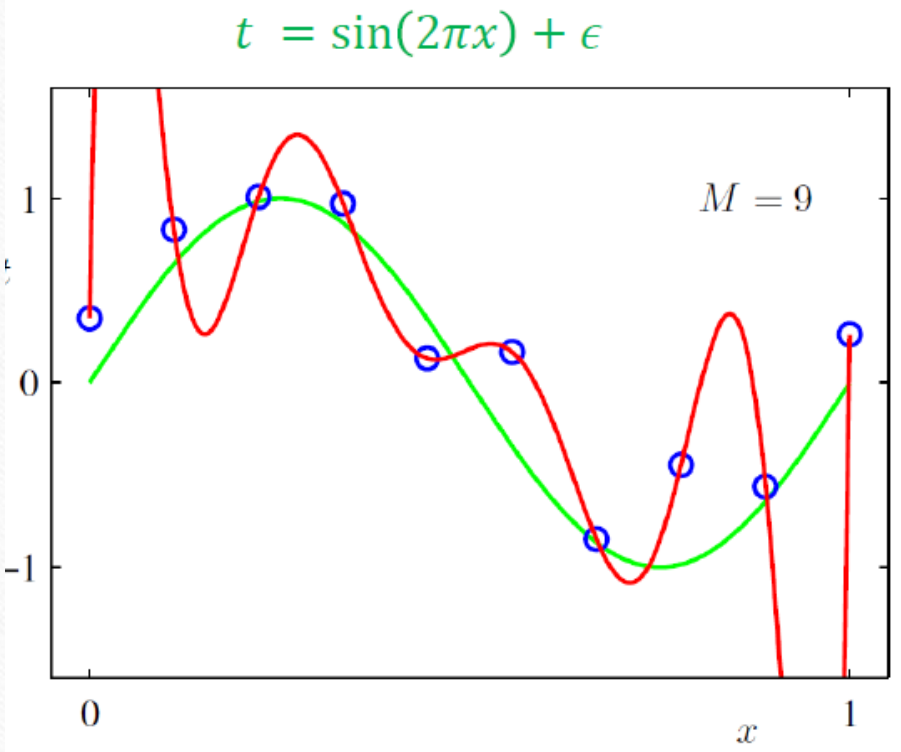
Capacidade de Generalização

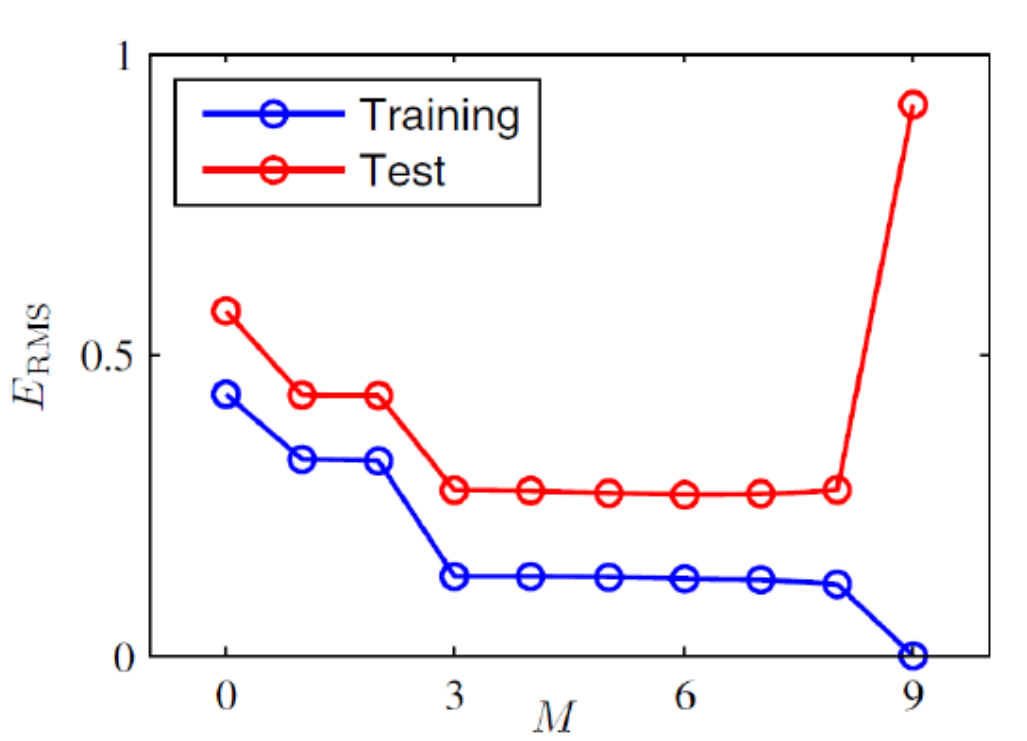
- O desafio central do aprendizado de máquina é que nosso algoritmo deve ter um bom desempenho em entradas novas e inéditas - não apenas naquelas em que nosso modelo foi treinado.
- A capacidade de ter um bom desempenho em entradas não observadas anteriormente é chamada de generalização.
- Modelos com baixa capacidade podem ter dificuldades para se ajustar ao conjunto de treinamento. Os modelos com alta capacidade podem sobre-ajustar memorizando as propriedades do conjunto de treinamento que podem não ser adequadas para o conjunto de teste.

O que é Regularização

- Em geral: qualquer método para evitar sobre-ajuste ou ajudar na otimização
- Especificamente: termos adicionais no objetivo de otimização do treinamento para evitar sobre-ajuste ou ajudar na otimização

Exemplo de sobre-ajuste: regressão usando polinômios





Sobre-ajuste

- Risco empírico e Risco esperado são diferentes
- Quanto menor o conjunto de dados, maior a diferença entre os dois
- Maior a classe de hipóteses, mais fácil encontrar uma hipótese que produza maior diferença entre os dois
  - Assim, possui pequeno erro de treinamento, mas grande erro de teste (sobre-ajuste)

Como evitar sobre-ajuste

- Conjunto de dados grande pode ajudar
- Descartar hipóteses inúteis também ajudar
- Regularização clássica: tem como objetivo restringir a quantidade de hipóteses
- Outros tipos de regularização: aumento de dados, parada antecipada, etc.

Regularização

- Regularização é qualquer modificação que fazemos a um algoritmo de aprendizado que visa reduzir seu erro de generalização, mas não seu erro de treinamento.
- A regularização é uma das preocupações centrais do campo de aprendizado de máquina, rivalizada em sua importância apenas pelos métodos de otimização.
- O teorema No Free lunch deixou claro que não existe melhor algoritmo de aprendizado de máquina e, em particular, a melhor forma de regularização.
- Em vez disso, devemos escolher uma forma de regularização que seja adequada à tarefa específica que queremos resolver.

Regularização com restrição hard

- Objetivo do treinamento

$$min_f \hat{L}(f) = \frac{1}{n}\sum_{i=1}^{n} l(f, x_i, y_i)$$

$$sujeito\ f \in \mathcal{H}$$

- Quando parametrizado

$$min_\theta \hat{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n} l(\theta, x_i, y_i)$$

$$sujeito\ \theta \in\ \Omega$$

Regularização com restrição hard

- Quando Ω medida por uma quantidade R

$$min_\theta \hat{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n} l(\theta, x_i, y_i)$$

$$sujeito\ R(\theta) \leq r$$

- Exemplo: Regularização l^2

$$min_\theta \hat{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n} l(\theta, x_i, y_i)$$

$$sujeito\ \|R(\theta)\|_2^2 \leq r^2$$

Regularização com restrição soft

- A otimização da restrição hard é equivalente à restrição soft
- Para algum parâmetro $\lambda^* > 0$

$$min_\theta \hat{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n} l(\theta, x_i, y_i) + \lambda^* \mathrm{R}(\theta)$$

- Exemplo: Regularização l^2

$$min_\theta \hat{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n} l(\theta, x_i, y_i) + \lambda^* \|\theta\|_2^2$$

Regularização com restrição soft

- Método de multiplicado lagrangeano λ
  - $\mathcal{L}(\theta, \lambda) = \hat{L}(\theta) + \lambda[R(\theta) - r]$
- Supondo θ^* é o ótimo para otimização com restrição hard
  - $\theta^* = argmin_\theta max_{\lambda \geq 0} \mathcal{L}(\theta, \lambda) = \hat{L}(\theta) + \lambda[R(\theta) - r]$
- Supondo λ^* é o ótimo, este correspondente ao máximo
  - $\theta^* = argmin_\theta \, \mathcal{L}(\theta, \lambda^*) = \hat{L}(\theta) + \lambda^*[R(\theta) - r]$

Regularização usando abordagem bayesiana a priori

- Visão bayesiana: tudo é uma distribuição
- A priori sobre as hipóteses: $p(\theta)$

- A posteriori sobre as hipóteses: $p(\theta|\{x_i y_i\})$
- Likelihood: $p(\{x_i y_i\}|\theta)$

- Regra bayesiana: $p(\theta|\{x_i y_i\}) = \frac{P(\theta)P(\{x_i,y_i\}|\theta)}{P(\{x_i,y_i\})}$

Regularização usando abordagem bayesiana a priori

- Regra bayesiana

$$p(\theta|\{x_i y_i\}) = \frac{P(\theta)P(\{x_i, y_i\}|\theta)}{P(\{x_i, y_i\})}$$

- Máximo a posteriori (MAP)
  - $max_\theta \, log \, p(\theta|\{x_i, y_i\}) = max_\theta \, logp(\theta) + logp(\{x_i, y_i\}|\theta)$

Regularização usando abordagem bayesiana a priori

- Exemplo: perda l^2 com regularização l^2
  - $min_\theta \widehat{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n}(f_\theta(x_i) - y_i)^2 + \lambda^* \|\theta\|_2^2$

- Corresponde para likelihood normal $p(x, y|\theta)$ e uma a priori normal $p(\theta)$

Três possibilidades

- Escolha típica para otimização - restrição soft
- $min_\theta \widehat{L_R}(\theta) = \hat{L}(\theta) + \lambda R(\theta)$
- Restrições rígidas e visão bayesiana: conceitual; ou usado para derivação

Três possibilidade

- Restrição Hard preferida se
  - Conhecermos o limite explícito $R(\theta) \leq r$
  - Restrições soft causam uma retenção em mínimos locais para valores pequenos de θ
  - A projeção de volta ao conjunto factível leva à estabilidade do problema de otimização
- Possibilidade bayesiana preferida se
  - Conheçermos a distribuição prévia

Regularização Clássica

- Norma da penalidade
  - Regularização l_2
  - Regularização l_1
- Robustez a ruído

Regularização l_2

- Efeito na descida do gradiente (estocástico)

$$min_\theta \, \widehat{L}(\theta) = \widehat{L}(\theta) + \lambda^* \|\theta\|_2^2$$

- Efeito na solução ideal

Efeito na descida do gradiente

- Gradiente de objetivo regularizado
  - $\nabla \widehat{L_R}(\theta) = \nabla L(\theta) + \alpha\theta$
- Atualização do gradiente descendente
  - $\theta \leftarrow \theta - \eta \nabla \widehat{L_R}(\theta)$
  - $\theta \leftarrow \theta - \eta(\nabla L(\theta) + \alpha\theta)$
  - $\theta \leftarrow (1 - \eta\alpha)\theta - \eta \nabla L(\theta)$
- Terminologia: decaimento do peso

Efeito sobre solução ótima

- Considere a aproximação quadrática em torno de θ^*

$$\hat{L}(\theta) \approx \hat{L}(\theta^*) + (\theta - \theta^*)^T \nabla \hat{L}(\theta^*) + \frac{1}{2}(\theta - \theta^*)^T H(\theta - \theta^*)$$

- Uma vez que θ^* é ótimo, $\nabla \hat{L}(\theta) = 0$

$$\hat{L}(\theta) \approx \hat{L}(\theta^*) + \frac{1}{2}(\theta - \theta^*)^T H(\theta - \theta^*)$$

$$\nabla \hat{L}(\theta) \approx H(\theta - \theta^*)$$

Efeito sobre solução ótima

- Gradiente do função objetiva regularizada

$$\nabla \hat{L}_R(\theta) \approx H(\theta - \theta^*) + \alpha\theta$$

- Na solução ótima θ_R^*

$$0 = \nabla \hat{L}_R(\theta_R^*) \approx H(\theta_R^* - \theta^*) + \alpha\theta_R^*$$

$$\theta_R^* \approx (H + \alpha I)^{-1} H\theta^*$$

Efeito sobre solução ótima

- O ótimo

$$\theta_R^* \approx (H + \alpha I)^{-1} H\theta^*$$

- Supondo H tem decomposição em autovetor $H = Q\Lambda Q^T$

$$\theta_R^* \approx (H + \alpha I)^{-1} H\theta^* = Q(\Lambda + \alpha I)^{-1}\Lambda Q^T\theta^*$$

- Efeito: Reescala os autovetores de H

Regularização l_1

$$\min_{\theta} \hat{L}_R(\theta) = \hat{L}(\theta) + \alpha||\theta||_1$$

- Efeito sobre gradiente descendente (estocástico)
- Efeito sobre solução ótima

Efeito sobre Gradiente descendente

- Gradiente da função objetiva regularizada

$$\nabla \hat{L}_R(\theta) = \nabla \hat{L}(\theta) + \alpha \operatorname{sign}(\theta)$$

- onde sign aplica para cada elemento de θ

- Atualiza o gradiente

$$\theta \leftarrow \theta - \eta \nabla \hat{L}_R(\theta) = \theta - \eta \nabla \hat{L}(\theta) - \eta\alpha \operatorname{sign}(\theta)$$

Efeito sobre a solução ótima

- Considere a aproximação quadrática em torno de θ^*

$$\hat{L}(\theta) \approx \hat{L}(\theta^*) + (\theta - \theta^*)^T \nabla \hat{L}(\theta^*) + \frac{1}{2}(\theta - \theta^*)^T H(\theta - \theta^*)$$

- Uma vez que θ^* é ótimo, $\nabla \hat{L}(\theta) = 0$

$$\hat{L}(\theta) \approx \hat{L}(\theta^*) + \frac{1}{2}(\theta - \theta^*)^T H(\theta - \theta^*)$$

Efeito sobre solução ótima

- Além disso, assuma que H é diagonal e positivo ($H_{ii} > 0, \forall i$)
  - não é verdade em geral, mas assuma por ter alguma intuição
- O objetivo regularizado é (ignorando constantes)

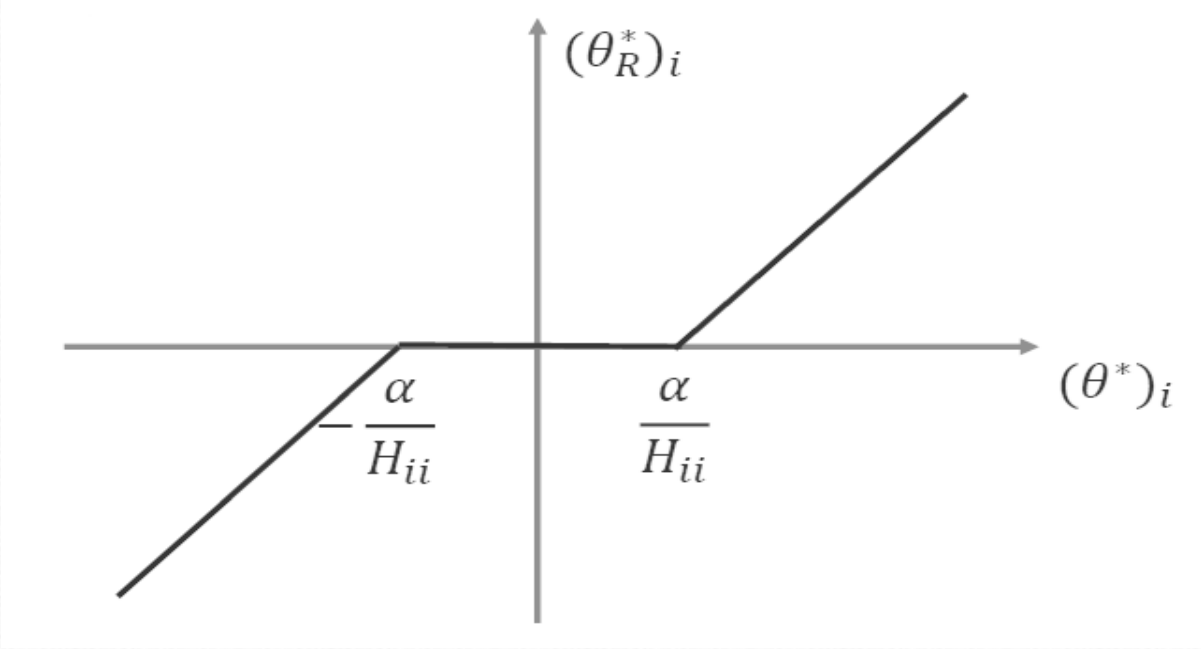
$$\hat{L}_R(\theta) \approx \sum_i \frac{1}{2} H_{ii}(\theta_i - \theta_i^*)^2 + \alpha\,|\theta_i|$$

- Na solução ótima θ_R^*

$$(\theta_R^*)_i \approx \begin{cases} \max\left\{\theta_i^* - \frac{\alpha}{H_{ii}}, 0\right\} & \text{if } \theta_i^* \geq 0 \\ \min\left\{\theta_i^* + \frac{\alpha}{H_{ii}}, 0\right\} & \text{if } \theta_i^* < 0 \end{cases}$$

Efeito sobre solução ótima

- Efeito: induz esparsidade



Efeito sobre solução ótima

- Além disso, assuma que H é diagonal
- Expressão compacta para o melhor θ_R^*

$$(\theta_R^*)_i \approx \text{sign}(\theta_i^*)\max\{|\theta_i^*| - \frac{\alpha}{H_{ii}}, 0\}$$

Possibilidade Bayesiana

- A regularização l_1 corresponde a Laplaciano anterior

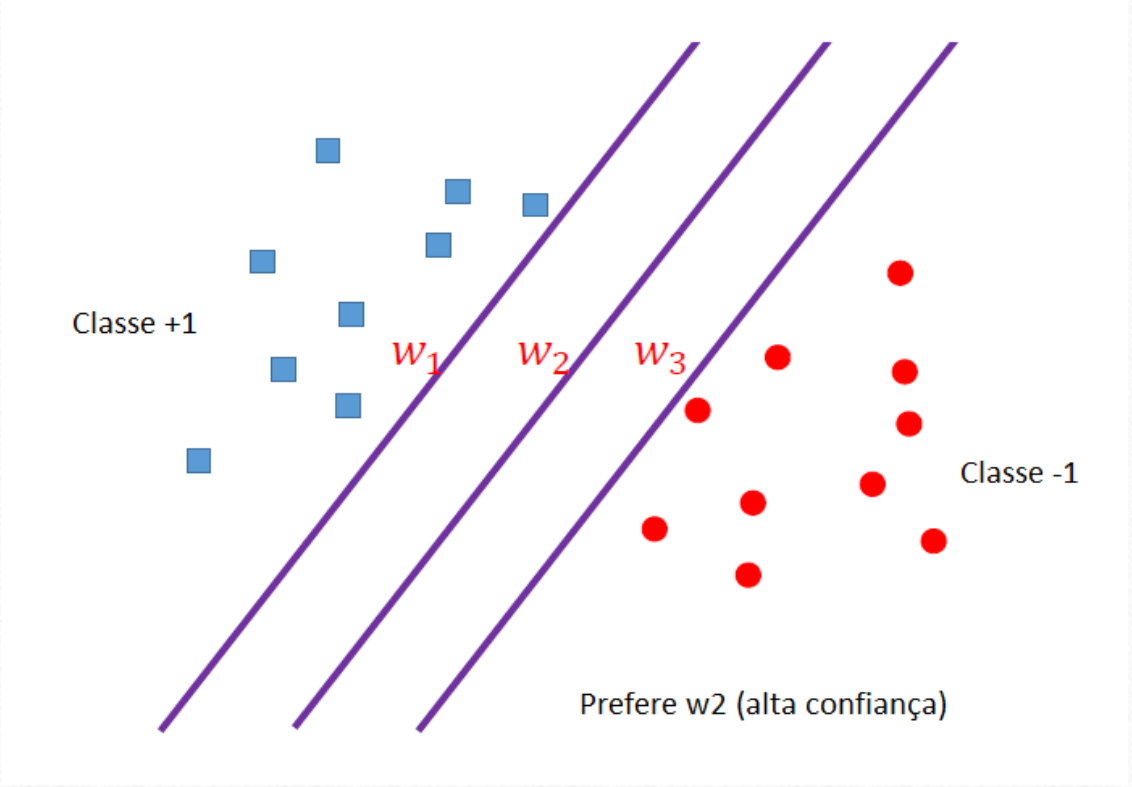
$$p(\theta) \propto \exp(\alpha \sum_i |\theta_i|)$$

$$\log p(\theta) = \alpha \sum_i |\theta_i| + \text{constant} = \alpha||\theta||_1 + \text{constant}$$

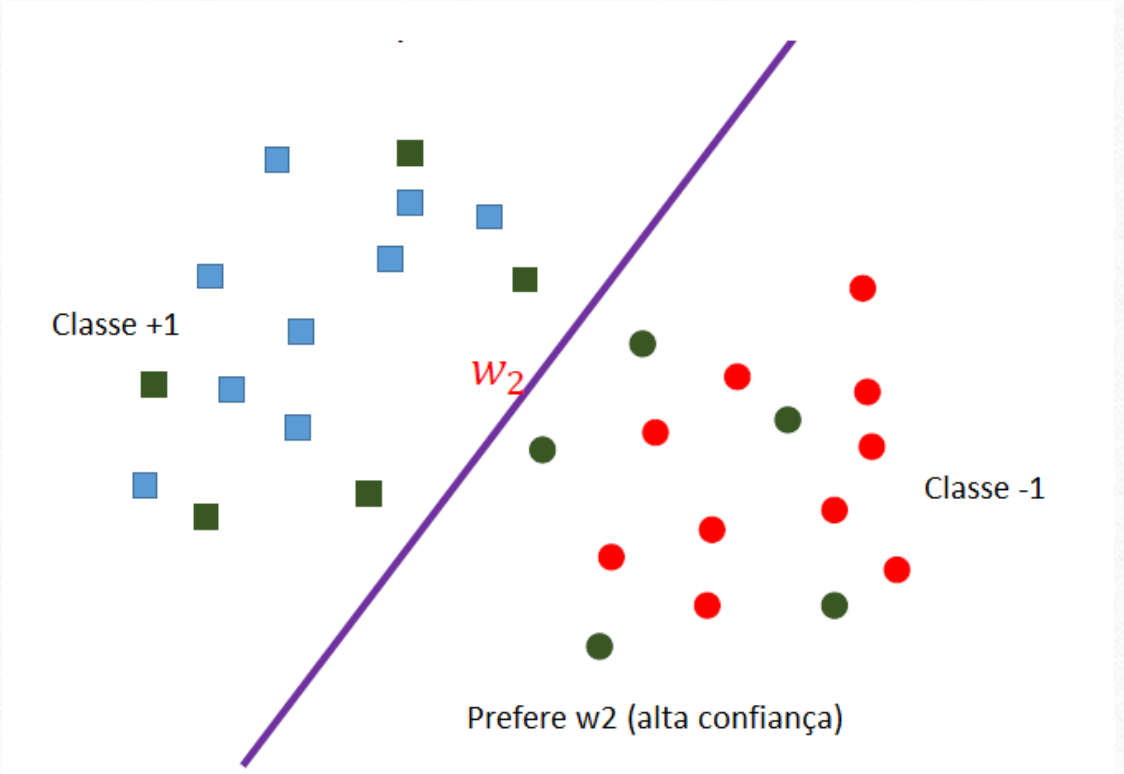
Outros tipos de regularizações

- Robustez ao ruído
  - Ruído na entrada
  - Ruído nos pesos
  - Ruído na saída
- Aumento de dados
- Parada antecipada – Early stopping
- Dropout

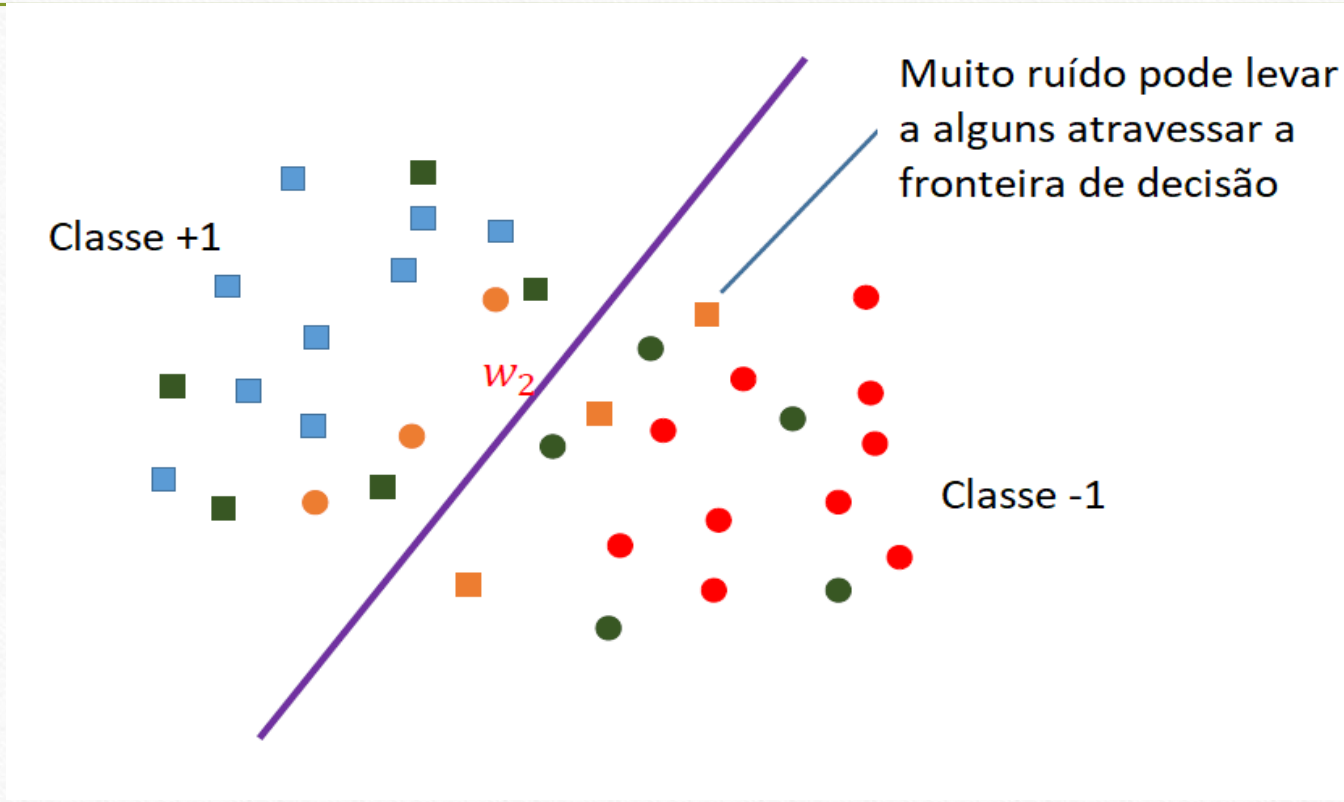
Múltiplas Soluções



Adicionar ruído a entrada



Cuidado: Muito ruído



$$\mathrm{E}[\mathrm{x}] = \sum_{i=1}^{\infty} x_i p(x_i)$$

Equivalência para decaimento dos pesos

- Suponha que a hipótese seja $f(x) = w^T x$, o ruído é $\epsilon \sim N\ (0, \lambda I)$
- Depois de adicionar ruído, a perda é

$$L(f) = \mathbb{E}_{x,y,\epsilon}[f(x+\epsilon) - y]^2 = \mathbb{E}_{x,y,\epsilon}[f(x) + w^T\epsilon - y]^2$$

$$L(f) = \mathbb{E}_{x,y,\epsilon}[f(x) - y]^2 + 2\mathbb{E}_{x,y,\epsilon}[w^T\epsilon(f(x) - y)] + \mathbb{E}_{x,y,\epsilon}[w^T\epsilon]^2$$

$$L(f) = \mathbb{E}_{x,y,\epsilon}[f(x) - y]^2 + \lambda||w||^2$$

Adicionar ruído aos pesos

- Para a perda em cada ponto de dados, adicione um termo de ruído aos pesos antes de calcular a previsão

$$\epsilon \sim N(0, \eta I),\ w' = w + \epsilon$$

- Predição: $f_{w'}$ ao invés de f_w
- Perda torna-se

$$L(f) = \mathbb{E}_{x,y,\epsilon}[f_{w+\epsilon}(x) - y]^2$$

Adicionar ruído aos pesos

- Perda torna-se $L(f) = \mathbb{E}_{x,y,\epsilon}[f_{w+\epsilon}(x) - y]^2$
- Para simplifica usando expansão de Taylor
  - $f_{w+\epsilon}(x) \approx f_w(x) + \epsilon^T \nabla f(x) + \frac{\epsilon^T \nabla^2 f(x) \epsilon}{2}$
- Logo

$$L(f) \approx \mathbb{E}[f_w(x) - y]^2 + \eta \mathbb{E}[(f_w(x) - y)\nabla^2 f_w(x)] + \eta \mathbb{E}||\nabla f_w(x)||^2$$

Pequeno, pode ser ignorado — termo de regularização

Aumentar os dados

Aumentar os dados

- Adicionando ruído à entrada: um tipo especial de aumento
- Cuidado com a transformação aplicada:
  - Exemplo: classificando "b" e "d“
  - Exemplo: classificando '6' e '9'

Parada antecipada

- Ideia: não treine a rede até que alcance erros de treinamento muito pequenos

- Lembre-se do sobre-ajuste: Maior a classe de hipóteses, mais fácil encontrar um hipótese que se encaixa na diferença entre os dois

- Evitar o sobre-ajuste: não exagere na hipótese; usar erro de validação para decidir quando parar

Parada Antecipada

- Ao treinar, também calcular erro de validação
- Sempre que erro de validação diminua, guarde uma cópia dos pesos
- Quando erro de validação não melhorar por algum tempo, pare
- Retornar a cópia dos pesos armazenados

Parada Antecipada

- seleção de hiperparâmetro: etapa de treinamento é o hiperparâmetro
- Vantagem
  - Eficiente: junto com o treinamento; armazene apenas uma cópia extra de pesos
  - Simples: nenhuma alteração no modelo / algo
- Desvantagem: precisa de dados de validação

Parada Antecipada

- Estratégia para se livrar da desvantagem
  - Após uma parada precoce da primeira execução, treine uma segunda execução e reutilize os dados de validação

- Como reutilizar dados de validação
  - Comece de novo, treine com dados de treinamento e dados de validação até o número anterior de épocas
  - Comece com os pesos na primeira corrida, treine com os dados de treinamento e os dados de validação útil a perda de validação <a perda de treinamento no ponto de parada inicial

Dropout

- Selecione aleatoriamente os pesos para atualizar
- Mais precisamente, em cada etapa da atualização
  - Amostre aleatoriamente uma máscara binária diferente para todas as unidades de entrada e ocultas
  - Múltiple os bits de máscara com as unidades e faça a atualização como de costume
- Probabilidade típica dropout: 0,2 para entrada e 0,5 para unidades ocultas

Quais regularizações são usadas com frequência?

- Regularização ℓ_2
- Parada antecipada
- Dropout

- Aumento de dados se as transformações são conhecidas / fáceis de implementar

Revisão de Processamento de Sinais

Prof. Clodoaldo A M Lima

Sistema Linear Invariante no Tempo (LTI)

- Um sistema é dito ser invariante no tempo se, para todo n_0, a sequência de entrada com valores $x_1[n] = x[n - n_o]$ produz a sequências saída com valores $y_1[n] = y[n - n_o]$

- Sistema linear → Se $y_1[n]$ e $y_2[n]$ são as respostas do sistema quando $x_1[n]$ e $x_2[n]$ são as respectivas entrada, então o sistema é linear se e somente se
  - $T[x_1[n] + x_2[n]] = T[x_1[n]] + T[x_2[n]] = y_1[n] + y_2[n]$
  - $T[\alpha x_1[n]] = \alpha T[x_1[n]] = \alpha y_1[n]$

Sistema Linear Invariante no Tempo (LTI)

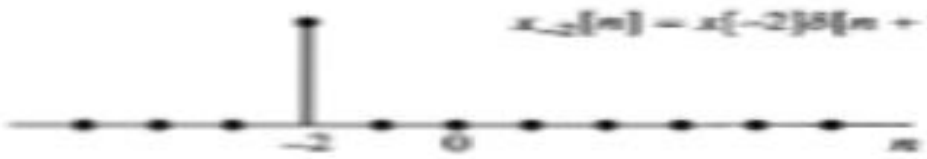
- LTI → consiste de sistemas que são lineares e invariante no tempo.
- Sabemos que
  - $x[n] = \sum_{k=-\infty}^{\infty} x[k]\delta[n-k]$
- Seja $h_k[n]$ a resposta para o sistema $\delta[n-k]$, um impulso ocorrendo em n = k.
  - y$[n] = T[x[n]] = T\{\sum_{k=-\infty}^{\infty} x[k]\delta[n-k]\}$
- Considerando a propriedade de linearidade. De acordo com o principio de superposição, nos podemos escrever que
  - y$[n] = T\{\sum_{k=-\infty}^{\infty} x[k]\delta[n-k] = \sum_{k=-\infty}^{\infty} x[k]T[\delta[n-k]\} = \sum_{k=-\infty}^{\infty} x[k]h_k[n]$

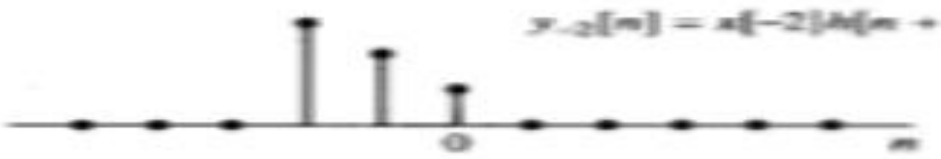
Sistema Linear Invariante no Tempo (LTI)

- A resposta do sistema para qualquer entrada pode ser expressada em termos da resposta do sistema para a sequência $\delta[n-k]$
- Se somente a linearidade é imposta, $h_k[n]$ dependerá de ambos n e k, a equação tem utilidade computacional limitada.
- Se considerarmos a propriedade de invariância, isto é, se $h[n]$ é a resposta para x[n], então $h[n-k]$ é a resposta para $\delta[n-k]$. Com esta restrição, podemos escrever que
  - $y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]$
- Consequência: um sistema LTI é completamente caracterizado por sua resposta impulsiva h[n] no sentido que, dado h[n], é possível usar a equação acima para calcular a saída y[n] devido a qualquer entrada x[n].
- A equação acima é comumente chamada de soma convolucional e representado por
  - $y[n] = x[n] * h[n]$

Interpretação

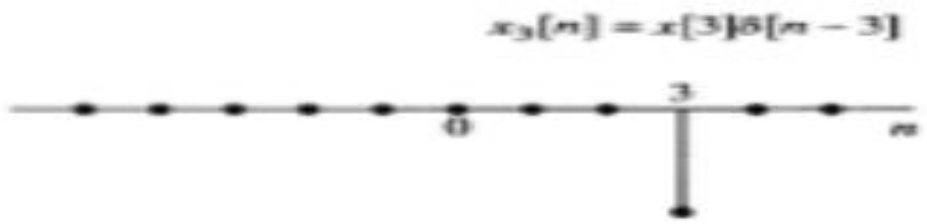
- Considere a figura abaixo, que mostra uma resposta impulsiva, uma simples sequência de entrada tendo três amostras diferentes de zero, a saída individual devido a cada amostra, e a saída composta devido a todas as amostras na sequência de entrada.
- $x[n]$ pode ser decomposta como a soma de três sequências $x[-2]\delta[n+2]$, $x[0]\delta[n]$, $x[3]\delta[n-3]$, representado os três valores diferente de zero na sequência $x[n]$.
- A seqüência $x[-2]h[n+2]$, $x[0]h[n]$ e $x[3]h[n-3]$ são as respostas do sistema para $x[-2]\delta[n+2]$, $x[0]\delta[n]$ e $x[3]\delta[n-3]$ respectivamente
- A resposta para $x[n]$ é então a soma de três respostas individuais

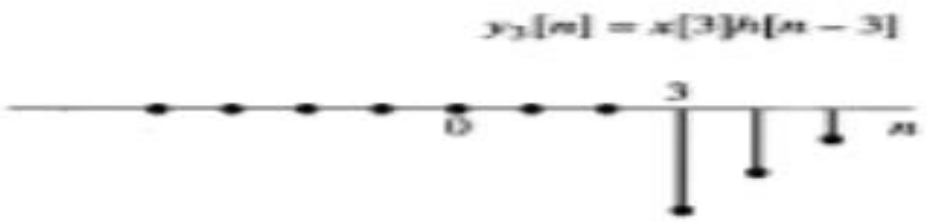

$x_{-2}[n] = x[-2]\delta[n+2]$
-2
0
n

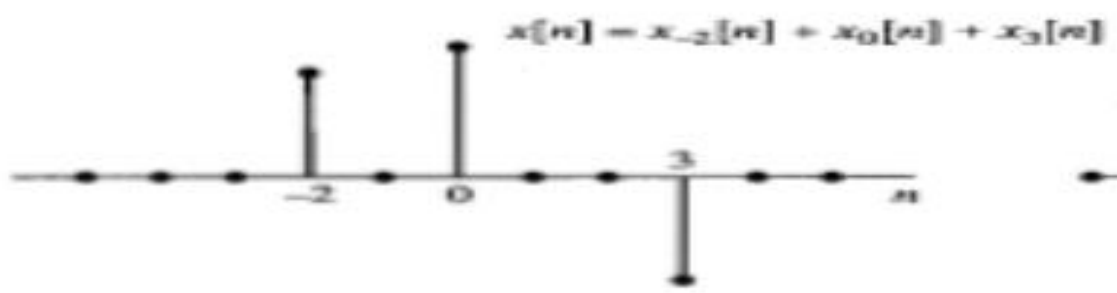

$y_{-2}[n] = x[-2]h[n+2]$
0
n

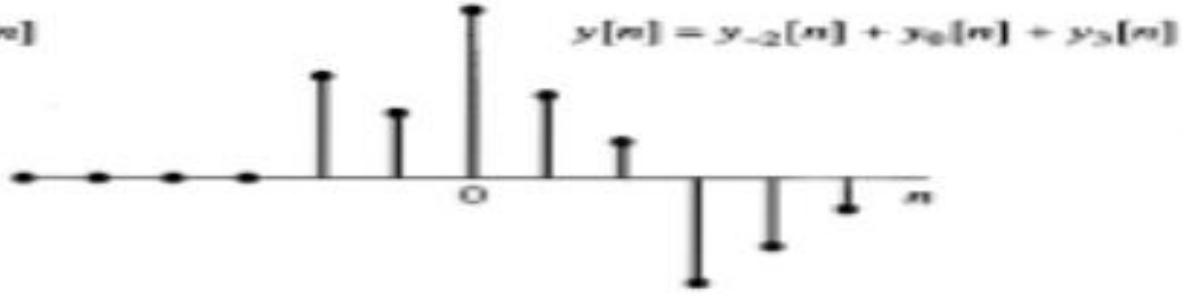

$x_0[n] = x[0]\delta[n]$
0
n


$y_0[n] = x[0]h[n]$
0
n


$x_3[n] = x[3]\delta[n-3]$
0
3
n


$y_3[n] = x[3]h[n-3]$
0
3
n


$x[n] = x_{-2}[n] + x_0[n] + x_3[n]$
-2
0
3
n


$y[n] = y_{-2}[n] + y_0[n] + y_3[n]$
0
n

Cálculo da soma convolucional

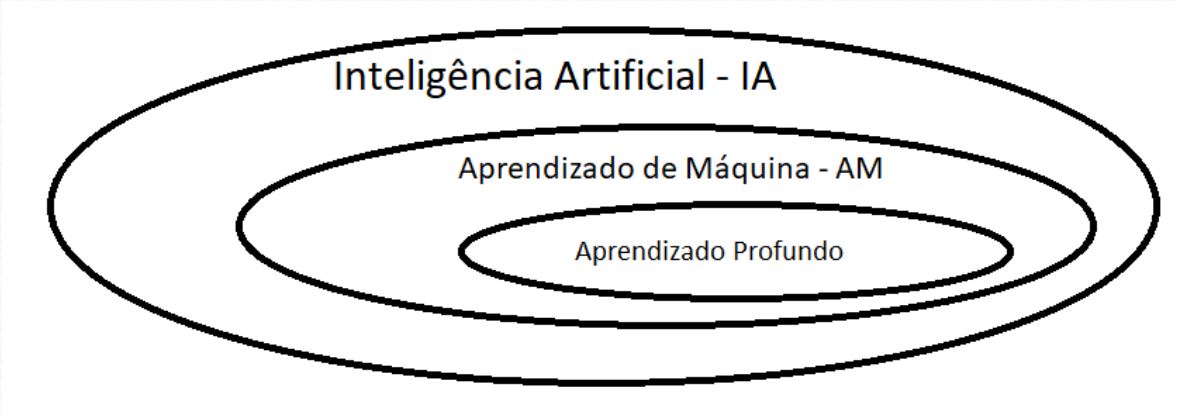
- Supondo que h[k] é a sequência mostrada abaixo, desejamos encontrar h[n-k] = h[-(k-n)].
- Vamos define $h_1[k]$ como h[-k], que é mostrado na figura.
- A seguir definimos $h_2[k]$ sendo $h_1[k]$ atrasado, por n amostras ,sobre o eixo k, isto, $h_2[k] = h_1[n-k]$ (figura (c)).
- Usando a relação entre $h_1[k]$ e h[k], podemos mostrar que $h_2[k] = h_1[n-k]$=h[-(k-n)] = h[n-k] e assim, a figura inferior é a saída desejada.

Cálculo da soma convolucional

- Resumindo, o calculo de h[n-k] a partir de h[k], primeiro espelhamos h[k] em torno de k =0 e então atrasamos o sinal espelhado por n amostras
- Para implementar a convolução de tempo discreto, as sequências x[k] e h[n-k] são multiplicadas para ∞ <k<-∞, e os produtos são somados para calcular a amostra de saída y[n]. Para obter outra amostra da saída, a origem da sequência h[-k] é deslocada para a nova posição e o processo é repetido. Este procedimento aplica se a cálculos realizadas numericamente sobre dados amostrados ou analiticamente com sequência para a qual os valores das amostras tem formulas simples.

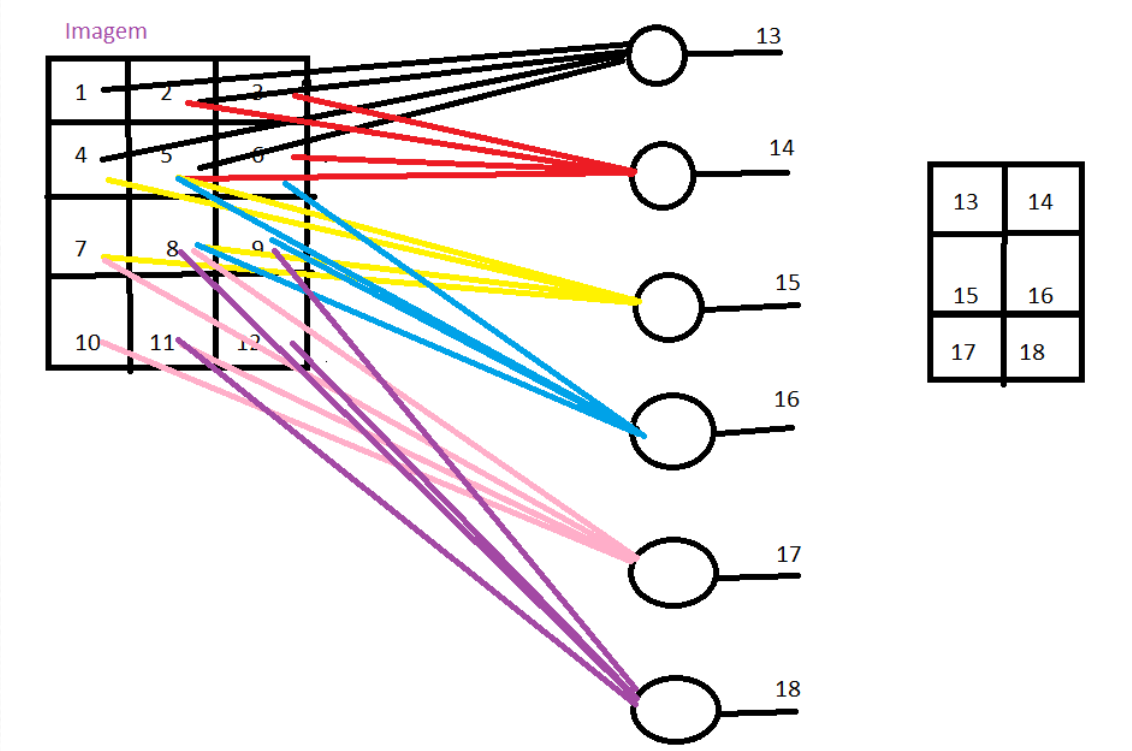
Aprendizado Profundo

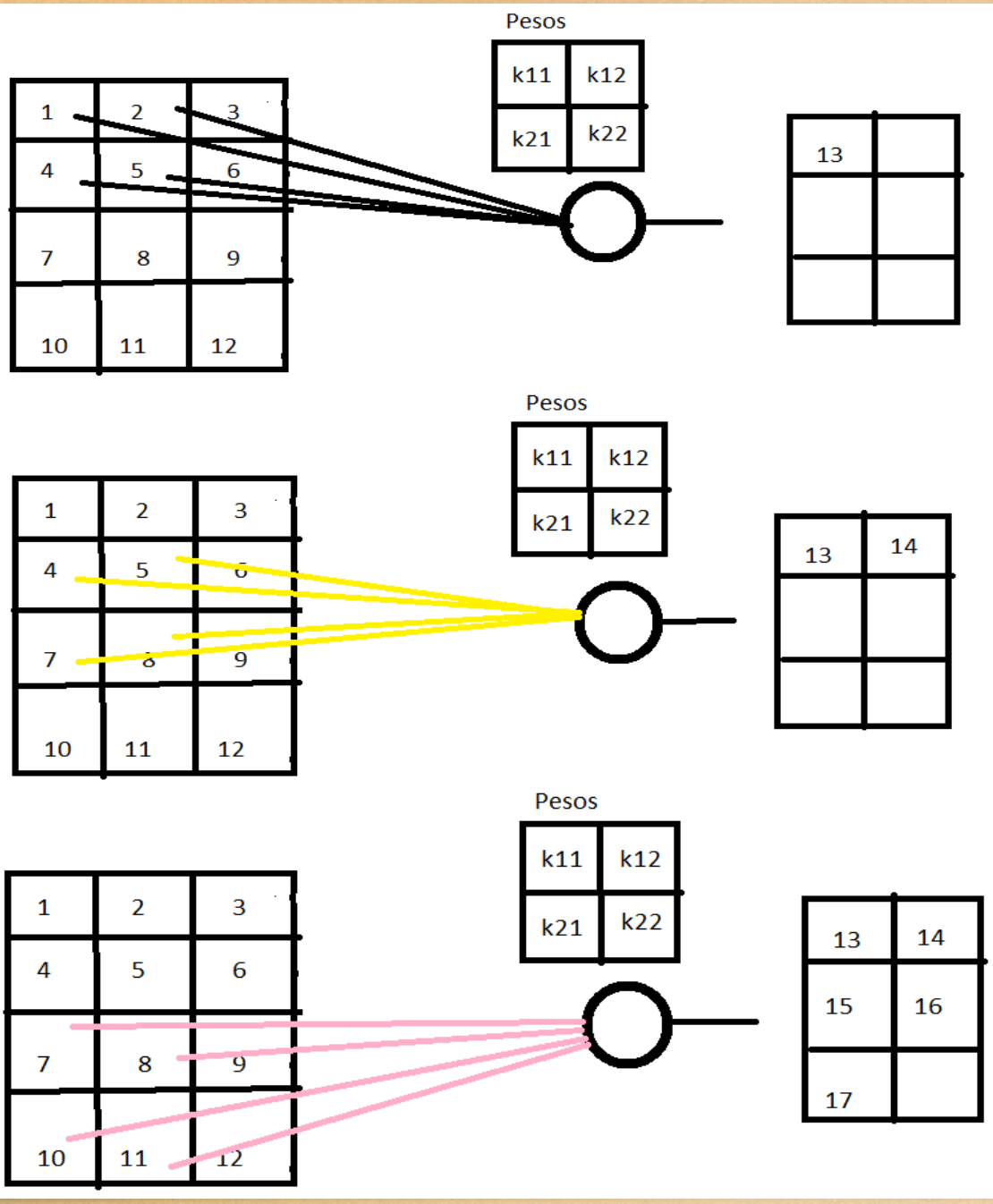
O que é aprendizado Profundo

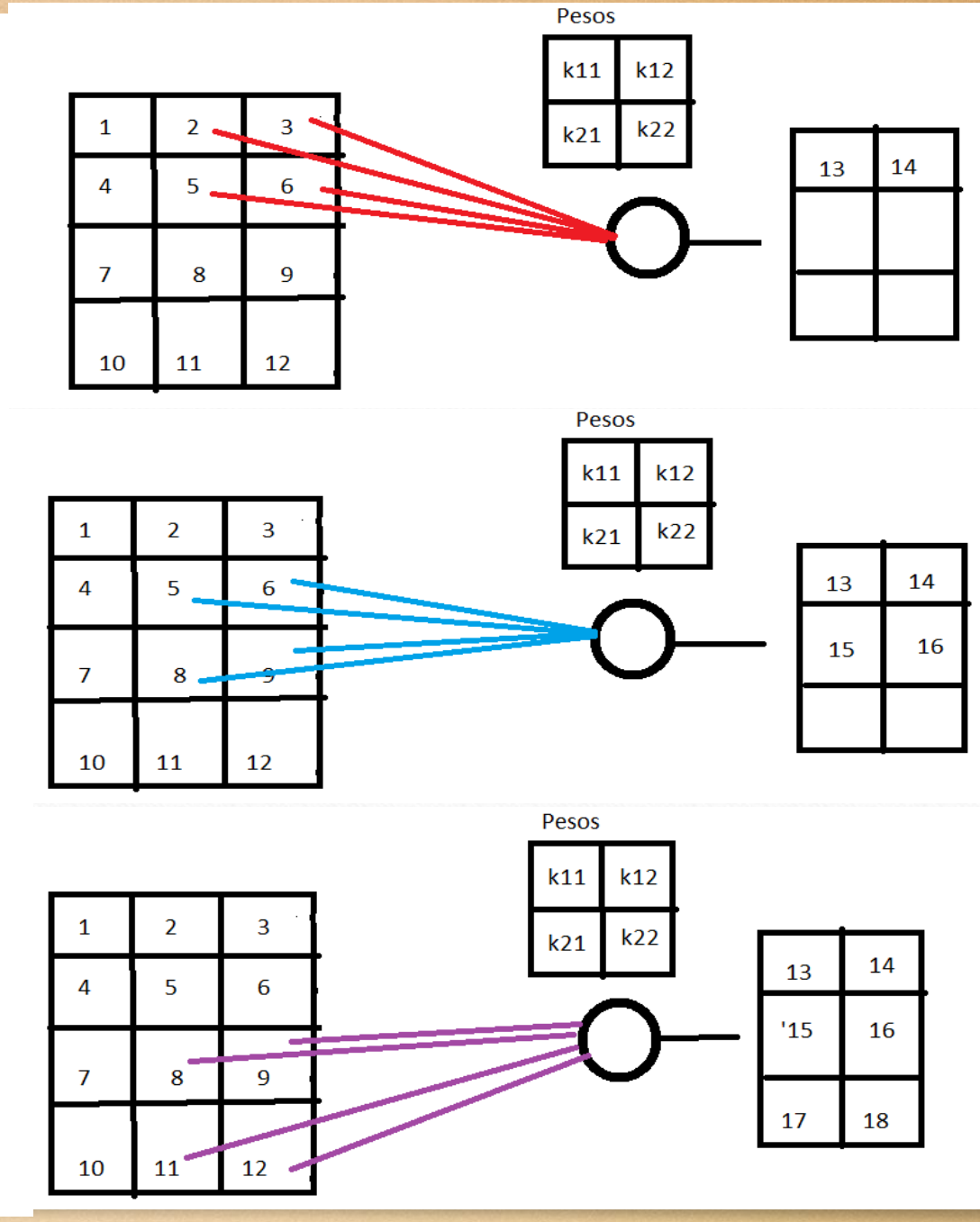


- IA: o termo mais amplo, aplicável a qualquer técnica que permita aos computadores imitar a inteligência humana.
- ML: subconjunto de IA com o objetivo de otimizar um critério de desempenho usando exemplo dados ou experiências anteriores, mas sem instruções explícitas.
- DL : Um subconjunto de ML que visa entender representações de alto nível de dados usando uma estrutura mais profunda de várias camadas de processamento

Compartilhamento Pesos




Pesos
k11
k12
k21
k22
1
2
3
4
5
6
7
8
9
10
11
12
13


Pesos
k11
k12
k21
k22
1
2
3
4
5
6
7
8
9
10
11
12
13
14

Pesos
k11
k12
k21
k22
1
2
3
4
5
6
7
8
9
10
11
12
13
14

Pesos
k11
k12
k21
k22
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

Pesos
k11
k12
k21
k22
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17

Pesos
k11
k12
k21
k22
1
2
3
4
5
6
7
8
9
10
11
12
13
14
'15
16
17
18

Correlação Cruzada

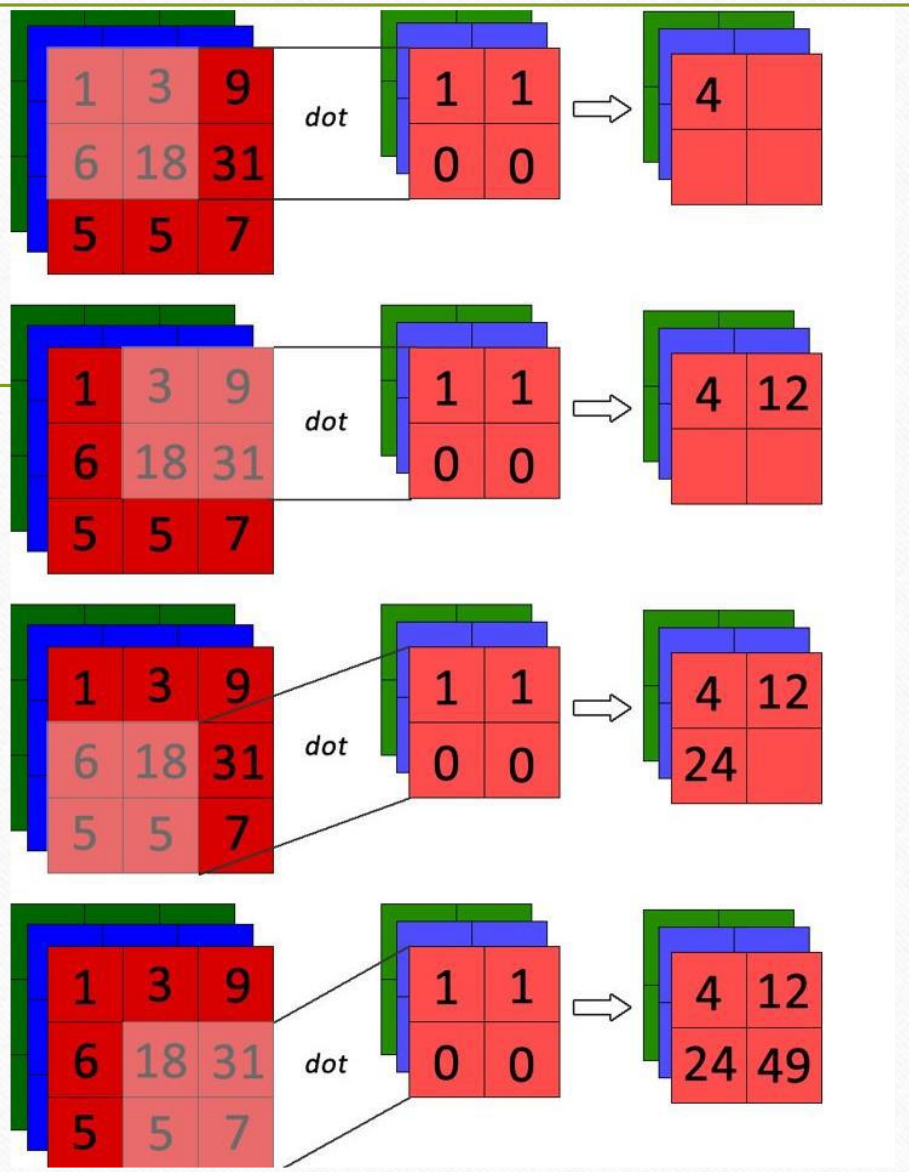
- Dada uma imagem de entrada *I* e um filtro (kernel) *K* de dimensões $k_1 \times k_2$, a operação de correlação cruzada é dada por:

$$(I \otimes K)_{ij} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} I(i+m, j+n) K(m,n)$$

Correlação

Kernel

1	1
0	0



Convolução - Imagem

- Conforme mencionado anteriormente, matematicamente, uma convolução é uma operação linear representada por (*) que a partir de duas funções, gera uma terceira (*featuremap*);
- No contexto de imagens, podemos entender esse processo como um filtro/*kernel* que transforma uma imagem de entrada;
- Um *kernel* aqui é entendido como uma matriz utilizada para uma operação de multiplicação de matrizes;
- Esta operação é aplicada diversas vezes em diferentes regiões da imagem;
- A cada aplicação, a região é alterada por um parâmetro conhecido como*stride;*
- Normalmente o strider possui o valor 1, o que significa que a transformação será aplicada em todos os *pixels* da imagem;
- A matriz resultante possui dimensionalidade menor que a imagem de origem.

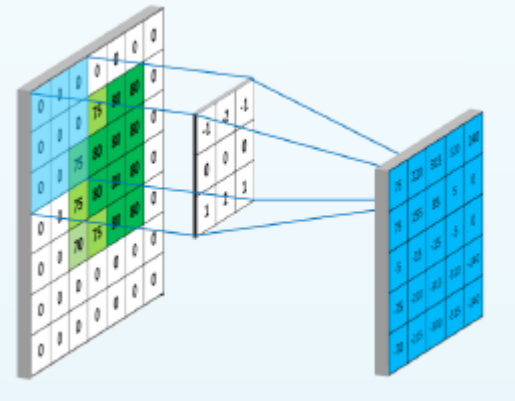
Kernel

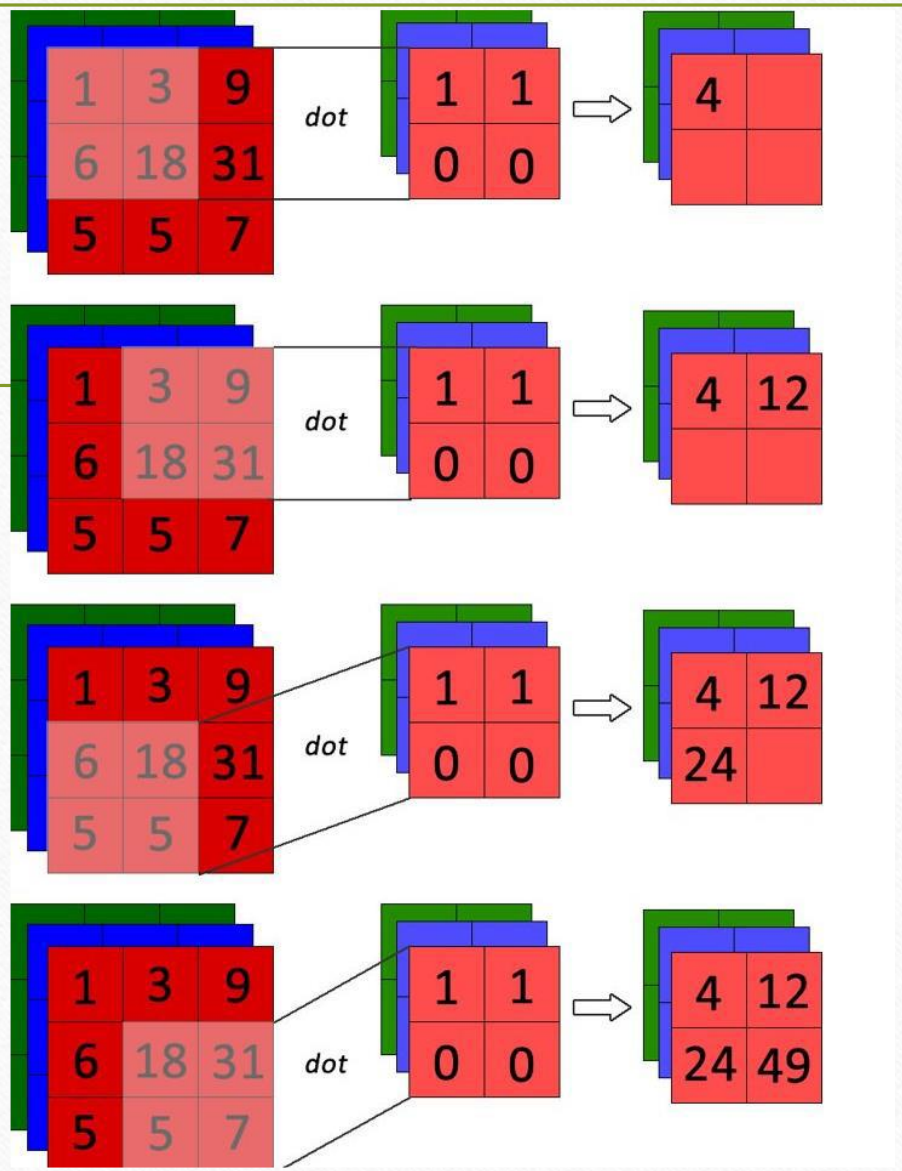
0	0
1	1

Convolução

$$(I * K)_{ij} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} I(i-m, j-n) K(m,n)$$

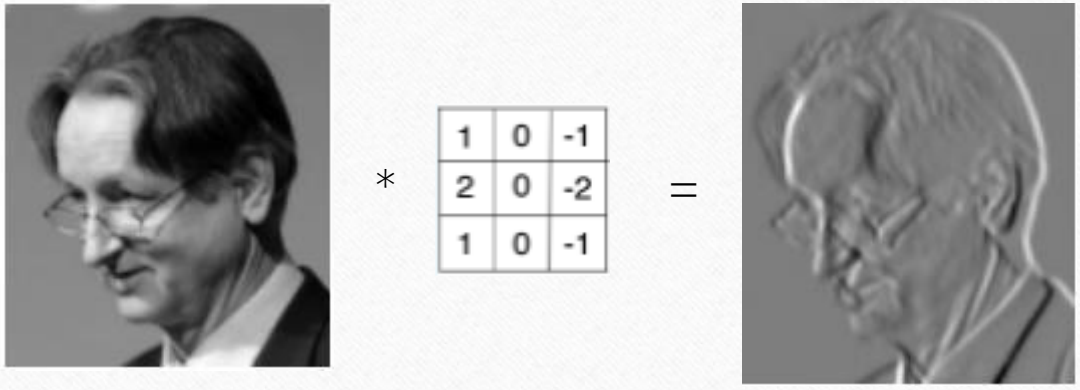
$$= \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} I(i+m, j+n) K(-m,-n)$$





Kernels

- Alguns filtros de convolução mais utilizadas com imagens:

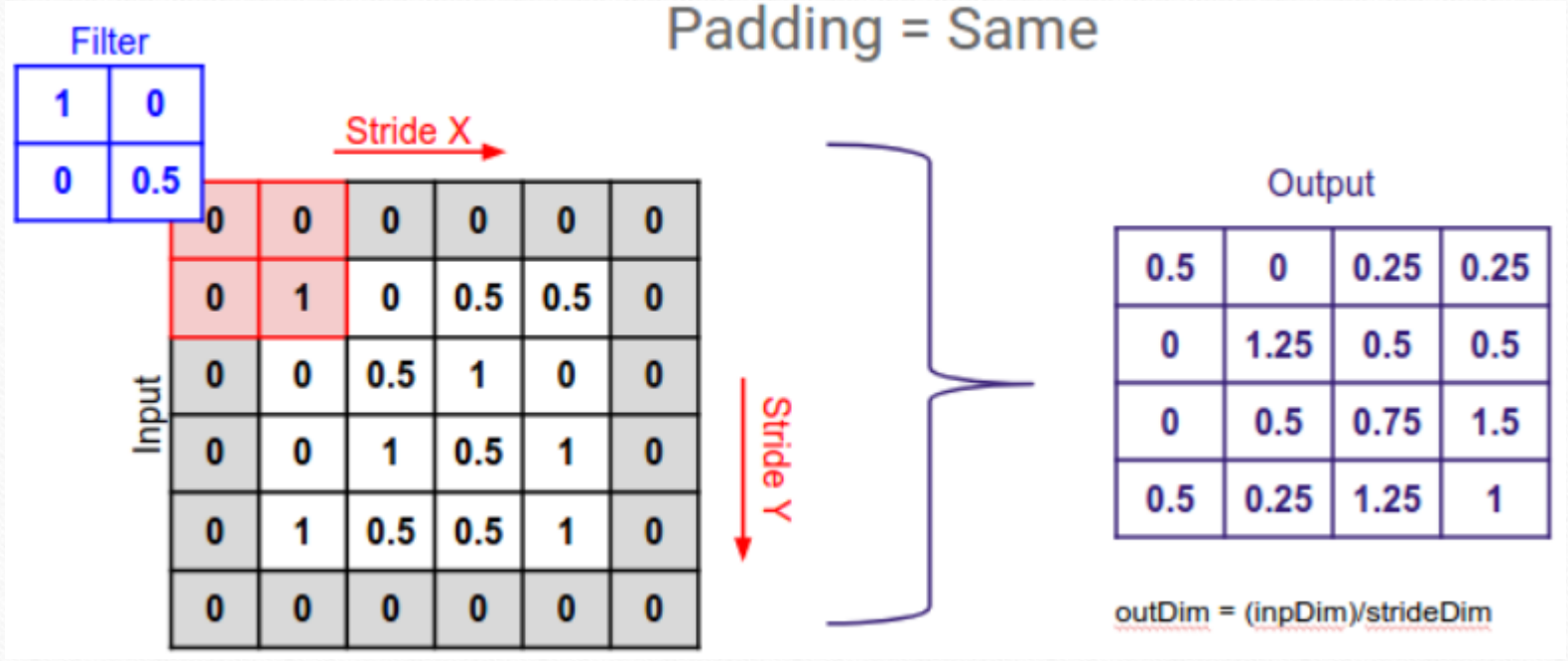


Operation	Kernel ω	Image result g(x,y)
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	

Padding

- Muitas convoluções podem impactar na assertividade da CNN se o tamanho da imagem for muito reduzido;
- Para contornar esse cenário, normalmente é utilizado o conceito de *Padding*–como acrescentar zeros ao redor da imagem;
- *Padding*é um processo em que alguns pixels são adicionados ao redor da imagem antes da operação de convolução, de forma a manter a dimensionalidade na imagem resultante durante a operação;
- Esse processo é utilizado porque essas imagens resultantes podem conter elementos que facilitam a identificação da classe alvo para a rede;
- Utilizado nas camadas iniciais.

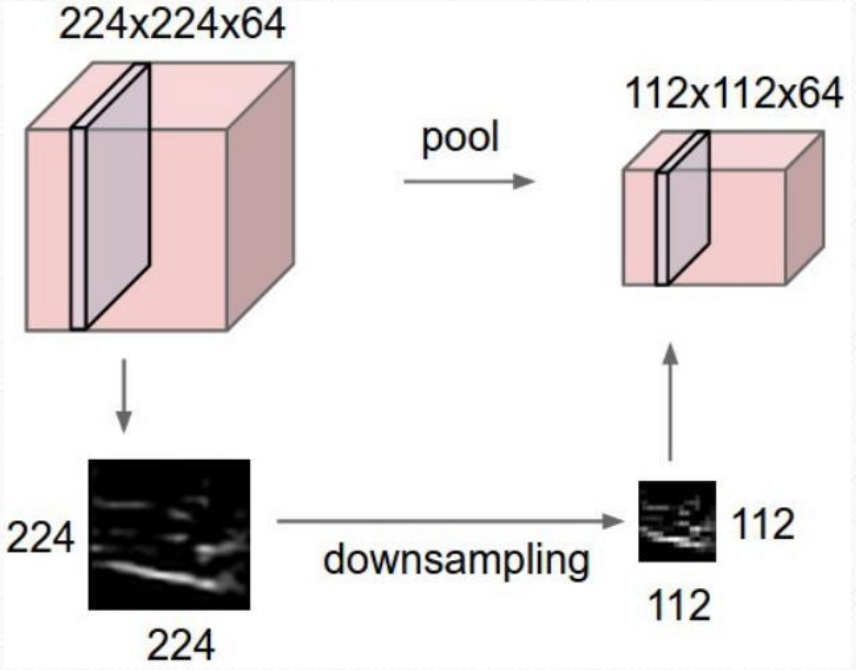
Padding

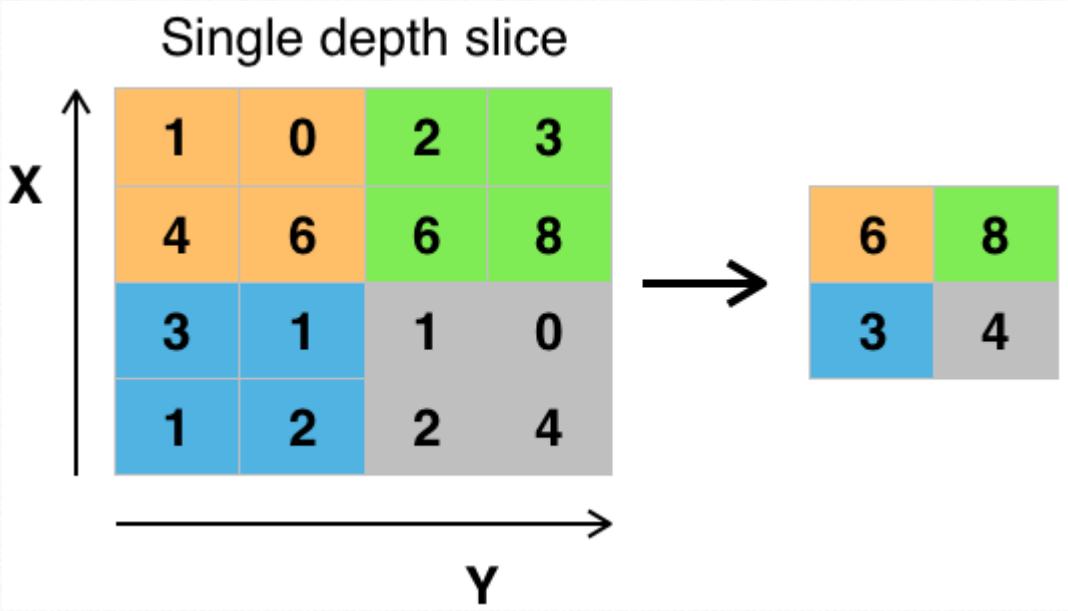

Padding = Same
Filter
1
0
0
0.5
Stride X
Stride Y
Input
0 0 0 0 0 0
0 1 0 0.5 0.5 0
0 0 0.5 1 0 0
0 0 1 0.5 1 0
0 1 0.5 0.5 1 0
0 0 0 0 0 0
Output
0.5 0 0.25 0.25
0 1.25 0.5 0.5
0 0.5 0.75 1.5
0.5 0.25 1.25 1
outDim = (inpDim)/strideDim

Pooling

- Pooling é um processo de redução de amostragem (*downsamping*);
- Realiza, assim, redução da dimensionalidade/*features maps*;
- De forma grosseira, pode-se entender essa transformação como uma redução do tamanho da imagem;
- A principal motivação dessa operação no modelo é de diminuir sua variância a pequenas alterações e também de reduzir a quantidade de parâmetros treinados pela rede.

Pooling

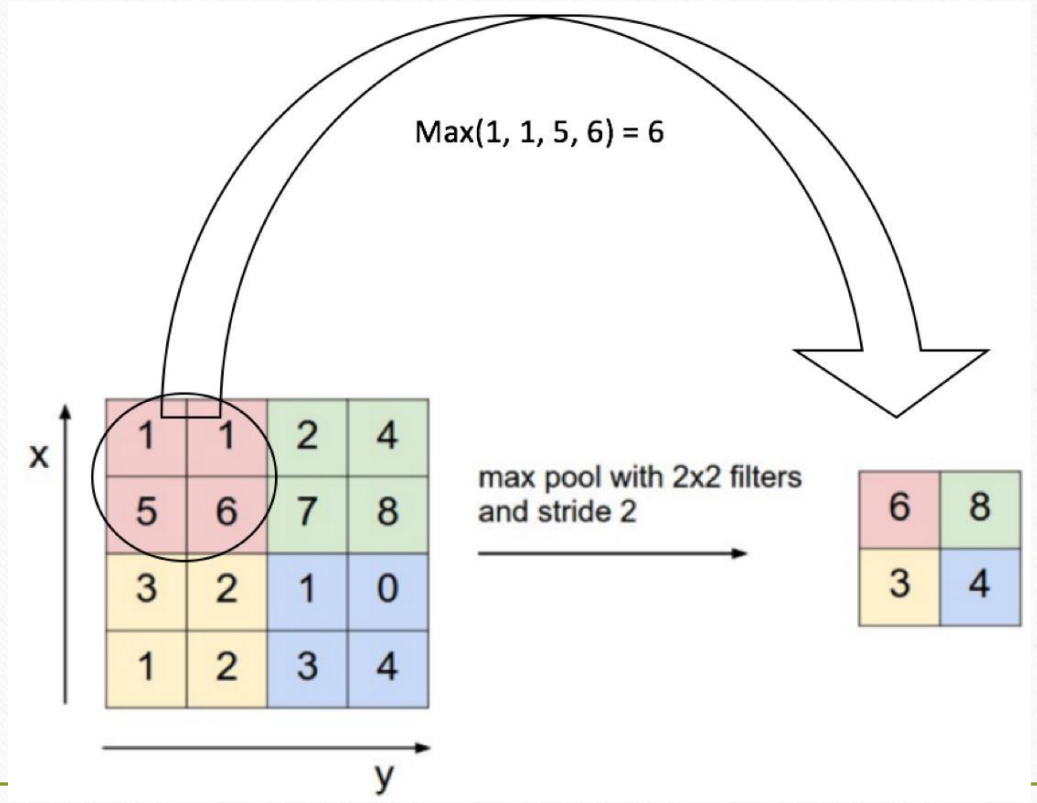

224x224x64
pool
112x112x64
224
224
downsampling
112
112


Single depth slice
X
Y
1 0 2 3
4 6 6 8
3 1 1 0
1 2 2 4
6 8
3 4

Pooling

- Existem 3 operações básicas de *Pooling:*
  - *Max Pooling*(mais utilizada)
  - Sum Pooling;
  - Avarage Pooling;
- Todas elas seguem o mesmo princípio e só se diferem na forma como calculam o valor final
- A operação de Max Pooling retira o maior elemento de determinada região da matriz considerando o tamanho do pool aplicado;
- Particiona a imagem de entrada em um conjunto de retângulos não-sobrepostos e, para cada sub-região, retorna o máximo

Pooling


Max(1, 1, 5, 6) = 6
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