INSIGHT

Data Science Laboratory
Federal University of Ceará

OLÁ!

Sou Ticiana Linhares,

Sou professora no Instituto Universidade Virtual. Sou pesquisadora no Insight Data Science Lab.
Sou doutora em Ciência da Computação.

Você pode me encontrar em
ticianalc@insightlab.ufc.br

AGENDA

1. Perceptron
2. Adaline
3. Multilayer Perceptron
4. CNN

Antes de começar, copie a pasta para seu Drive/ou baixe os dados...

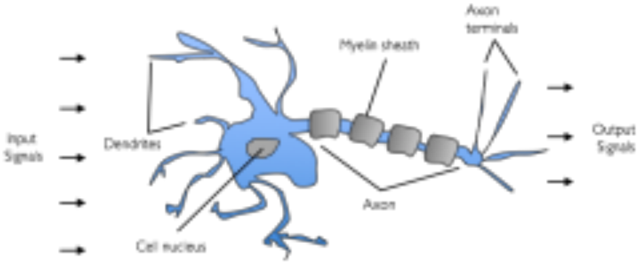
https://drive.google.com/drive/folders/167FVABYrtyTIlmCfkaWXlwK1y3LDoyMC?usp=sharing

1. Perceptron

1

Perceptron

- Em 1957, Frank Rosenblatt publicou pela primeira vez o conceito de Perceptron baseado no funcionamento dos neurônios humanos.
- O algoritmo automaticamente aprende os valores ótimos dos coeficientes que são features de entrada para tomada de decisão sobre um neurônio ser ativado ou não.
- No contexto de aprendizagem supervisionada e classificação, tal algoritmo poderia ser então usado para prever se uma amostra pertence a uma classe ou a outra.



Schematic of a biological neuron.

2

Perceptron

- Classificação Binária: positivo (1) e negativo (-1).
- Defina uma função de ativação *φ(z)* ou uma função que seja combinação linear dos valores de entrada *x* e um vetor de pesos *w*.
- Obtenha o produto escalar de *x* por *w*.
- z = w1x1 + …. + wmxm

$$\mathbf{w} = \begin{bmatrix} w^{(1)} \\ w^{(2)} \\ \vdots \\ w^{(m)} \end{bmatrix}, \mathbf{x} = \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(m)} \end{bmatrix}$$

Perceptron

- Se $\varphi(z)$ é acima de um limiar, retorna 1.
- Caso contrário, retorna -1.
- Heaviside Step Function

- $$\varphi(z) = \begin{cases} 1, \text{ se } z >= \theta \\ -1, \text{ caso contrário} \end{cases}$$

4

Heaviside Step Function Simplificada

- Coloque o limiar θ no lado esquerdo da equação e defina um peso com índice 0 como w0= - θ, dessa forma z pode ser escrito de forma mais compacta:

$$z = w0x0 + w1x1 + \ldots + wmxm = wTx$$

- $\varphi(z) = \begin{cases} 1, \text{ se } z >= 0 \\ -1, \text{ caso contrário} \end{cases}$
- w0= -θ é chamado de *bias unit*.

Produto Escalar

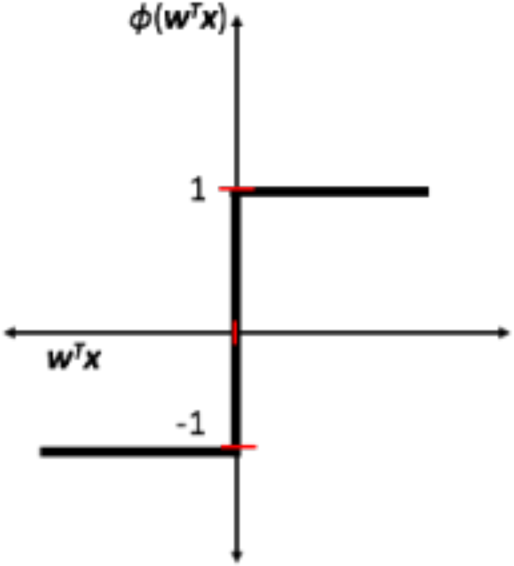
$$z = \mathbf{w}^T \mathbf{x} = \sum_{j=0}^{m} \mathbf{w_j x_j}$$

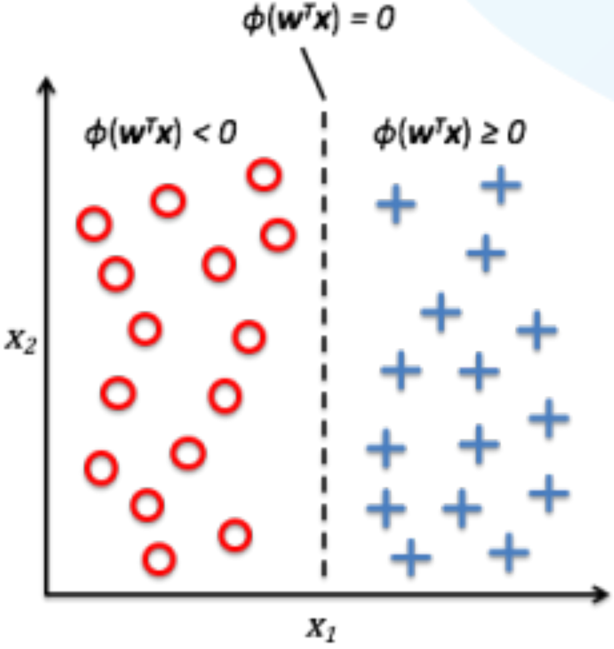
$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \times \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} = 1 \times 4 + 2 \times 5 + 3 \times 6 = 32.$$

- Neste caso, o perceptron diria que a instância pertence a classe positiva.

6

Perceptron





Algoritmo Rosenblatt Perceptron

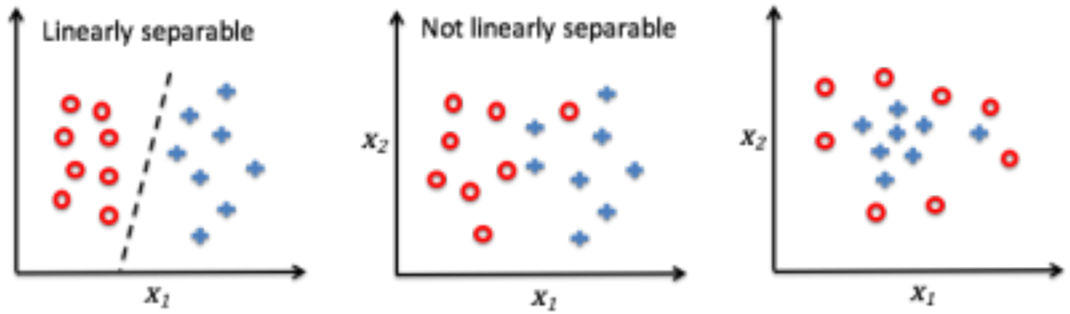
- Inicialize os pesos com 0 ou escolha números randômicos pequenos.
- Para cada amostra de treinamento x(i), execute os seguintes passos:
  - Compute o valor do vetor de saída $\hat{y}$.
  - Atualize os pesos.
- Regra para atualizar os pesos:
  - $wj = wj + \Delta wj$
- sendo
  - $\Delta wj = \eta \, (y(i) - \hat{y}(i)) \, x(i)j$
- Onde η é a taxa de aprendizagem (entre 0 e 1), y(i) é a classe correta de x(i), $\hat{y}(i)$ é a classe predita, e x(i) é a amostra de treinamento.

Atualização em dados bi-dimensionais

- $\Delta w0 = \eta\ (y(i) - \hat{y}(i)\ )$
- $\Delta w1 = \eta\ (y(i) - \hat{y}(i)\ )\ x(i)1$
- $\Delta w2 = \eta\ (y(i) - \hat{y}(i)\ )\ x(i)2$

Convergência

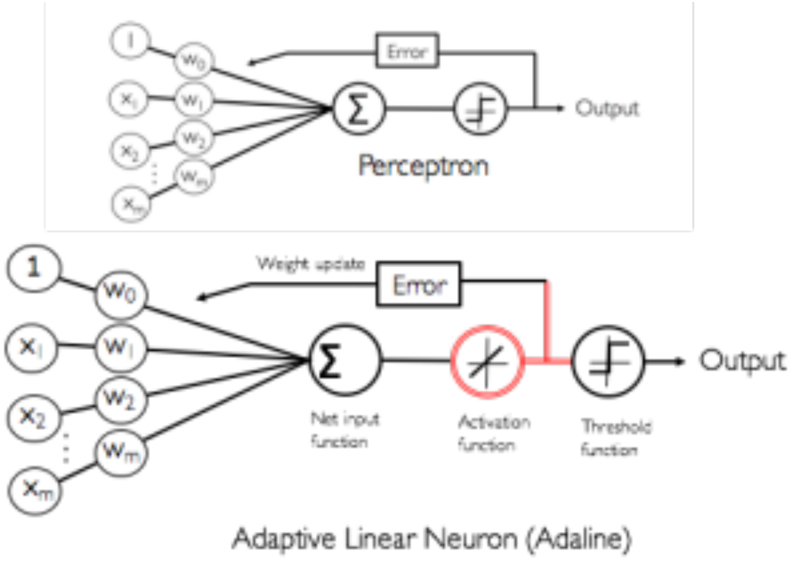
- A convergência é garantida se
  - Duas classes são linearmente separáveis.
  - Taxa de aprendizagem η deve ser suficientemente pequena.
- Se as classes não podem ser (rapidamente) separadas:
  - Aumente o número de épocas.
  - Modifique o valor de θ



2.
Adaline

Adaline (ADAptative LInear NEuron)

- Pesos são atualizados baseados em uma função linear de ativação.
- Na Adaline, $\varphi(z)$ é a função identidade aplicada a entrada:
  - $\varphi(wT\,x\,) = wT\,x$
- Um limiar é utilizado para prever a label, compara-se tal limiar com o valor $\varphi(wT\,x\,)$.



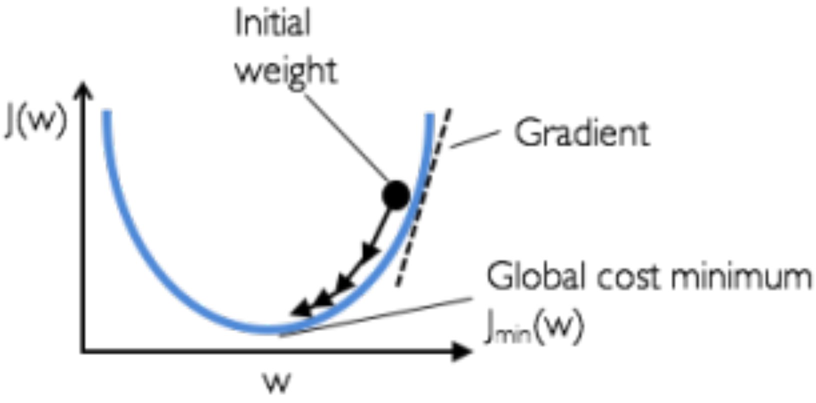
Função de Custo

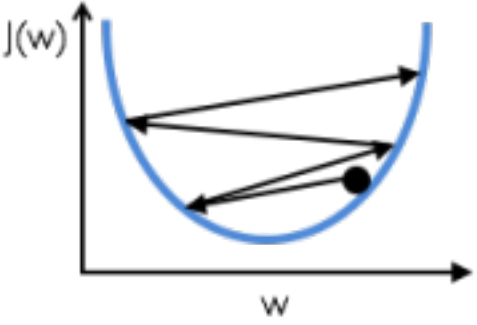
- Adaline usa uma função de custo J(.) e tenta aprender o valor dos pesos minimizando a função J(.).
- J(.) pode ser qualquer função, considere como sendo a Soma dos Erros Quadrados.
- Esta estratégia utiliza gradiente descendente para minimizar a função de custo J(.)

$$J(\mathbf{w}) = \frac{1}{2}\sum_i \left(y^{(i)} - \phi\left(z^{(i)}\right)\right)^2$$

12

Gradiente Descedente





Algoritmo Adaline

- Inicialize os pesos com 0 ou escolha números randômicos pequenos.
- Para todas as amostras de treinamento x, execute os seguintes passos:
    - Compute o valor do vetor de saída ŷ.
    - Atualize os pesos.
- Regra para atualizar os pesos:
    - $w = w + \Delta w$
- sendo
    - $\Delta w = - \eta \nabla J(w)$

$$\Delta w_j = -\eta \frac{\partial J}{\partial w_j} = \eta \sum_i \left(y^{(i)} - \phi\left(z^{(i)}\right) \right) x_j^{(i)}$$

- A atualização pega direção contrária ao gradiente $\nabla J(w)$, sendo este computado a partir da derivada da função de custo para cada peso wj.

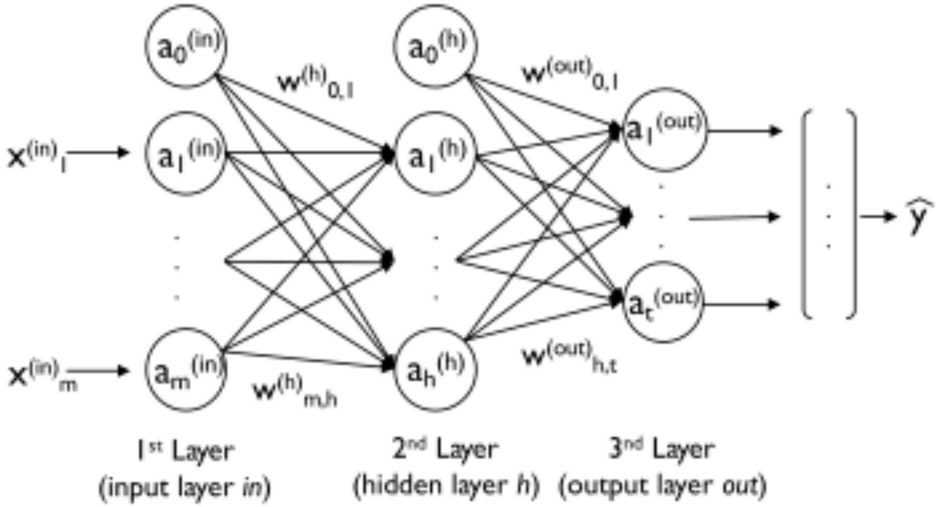
Adaline versus Perceptron

- Quais diferenças?
  - *φ(z(i))* sendo z(i) = wT x(i) um número real na Adaline, no Perceptron é a label {1, -1}.
  - Os pesos são atualizados usando todas as amostras de treinamento na Adaline, já no Perceptron, eles são atualizados para cada amostra.
  - Existe a versão para larga escala (grandes conjunto de dados) do algoritmo Adaline, que usa gradiente descendente estocástico.

3.
MultiLayer Perceptron

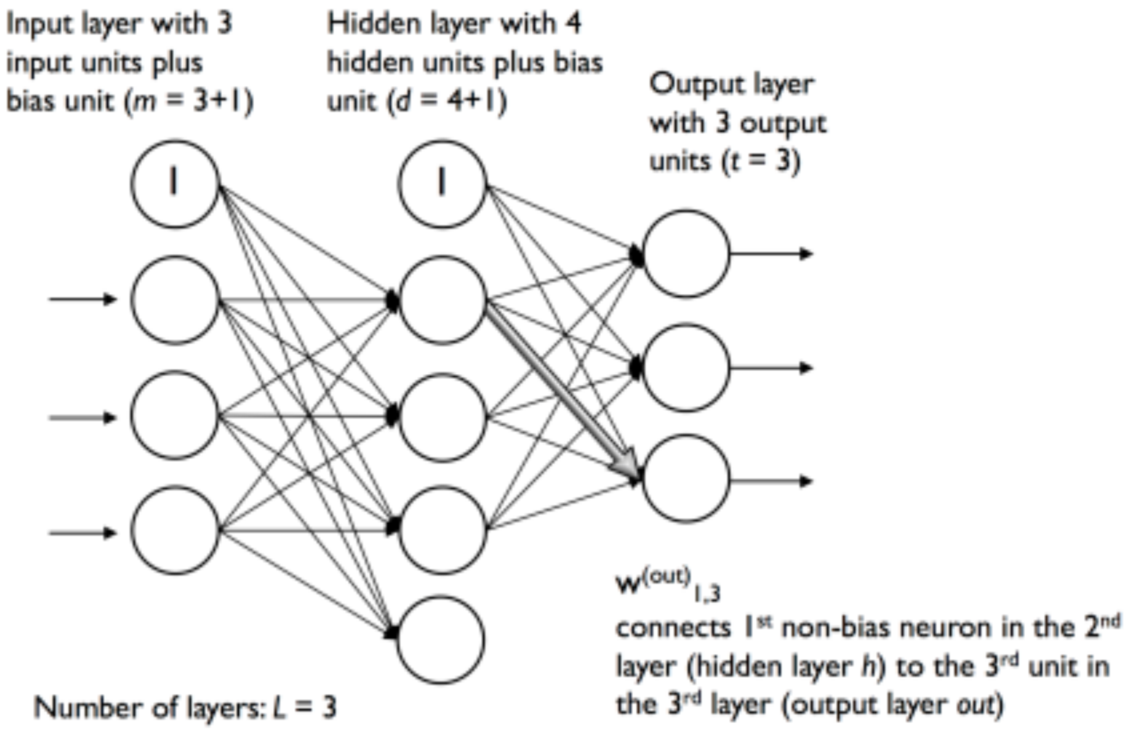
Multilayer Perceptron (MLP)

- Uma camada (layer) de entrada, uma de saída e uma camada oculta, chamada hidden layer.
- Considere apenas uma camada oculta. Esta é totalmente conectada a entrada, e a camada de saída também é totalmente conectada a oculta.
- As redes profundas tem mais que uma camada oculta.



$$\mathbf{a}^{(i)} = \begin{bmatrix} a_0^{(1)} \\ a_1^{(1)} \\ \vdots \\ a_m^{(1)} \end{bmatrix} = \begin{bmatrix} 1 \\ x_1^{(i)} \\ \vdots \\ x_m^{(i)} \end{bmatrix}$$

Multilayer Perceptron (MLP) - em suma.



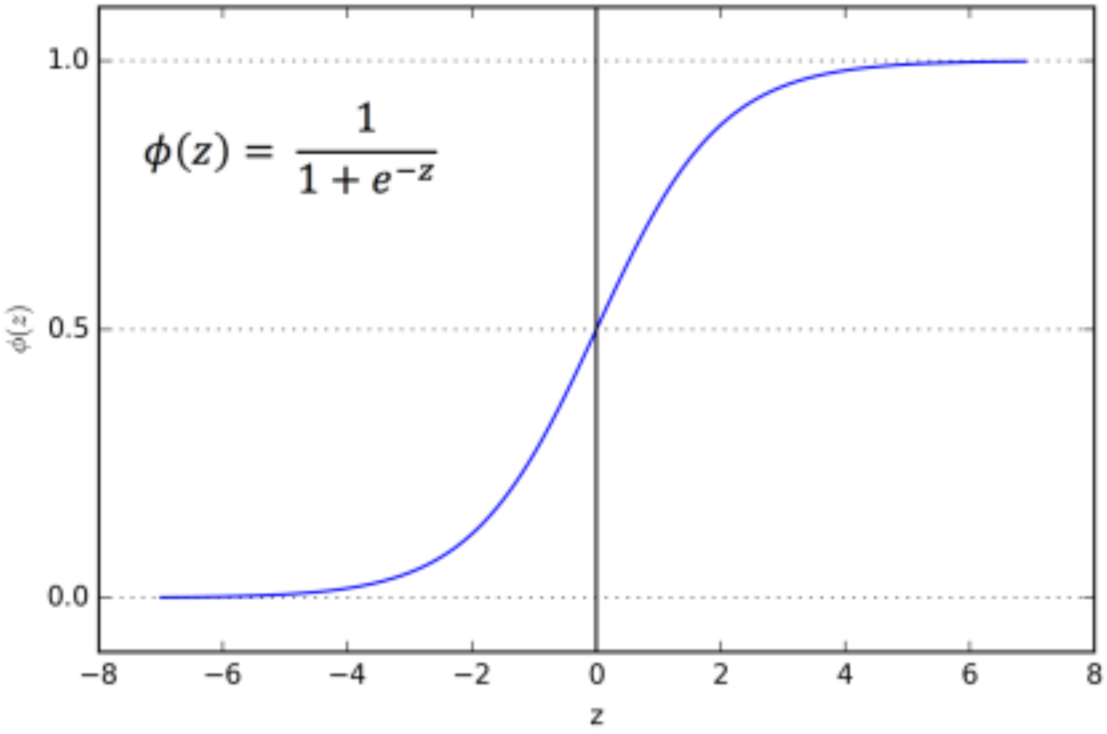
Algoritmo MLP

- Assuma que a entrada tem *m* dimensões.
- Cada unidade da camada oculta é conectada a todas as unidades da camada de entrada. Primeiro, nós calculamos a unidade de ativação, por exemplo a(2)1:
  - *z(2)1 = a(1)0 w(1)1,0 + a(1)1 w(1)1,1 +.... + a(1)m w(1)1,m*
- Computa a ativação para a primeira unidade na camada oculta:
  - *a(2)1 = φ(z(2)1)*
- *φ(.)* é uma função de ativação que deve ser diferenciável e os pesos devem ser atualizados usando gradiente descendente.
- Mas se o problema é classificação de imagens, utiliza-se uma função não linear. Em geral, a sigmoid.

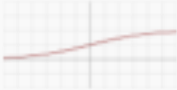
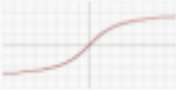
$$\phi(z) = \frac{1}{1 + e^{-z}},$$

17

Função Sigmoid Descendente



Funções de Ativação

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

Algoritmo MLP

- O objetivo é minimizar o valor/peso dos neurônios que contribuem mais para o erro.
- Utiliza-se uma função de perda (ou erro) na saída e propaga esse valor de volta na rede (*back propagation*). Neste algoritmo, vamos usar a função de custo da Regressão Logística:

$$J(\mathbf{w}) = -\sum_{i=1}^{n} y^{(i)} \log\left(a^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - a^{(i)}\right)$$

- No *back propagation*, o erro é minimizado usando o gradiente (derivada da função J(.)) com relação a cada peso das camadas da rede.
- Os pesos são atualizados para minimizar o erro resultante em cada neurônio.
- Cada rodada *forward* e *back propagation* é conhecida como uma iteração no treinamento, ou ainda, época.

Algoritmo MLP

- Pode haver um outro termo na função de erro J(.), chamado de regularização.

$$L2 = \lambda \|\mathbf{w}\|_2^2 = \lambda \sum_{j=1}^{m} w_j^2$$

- A atualização de pesos se dá da seguinte forma:
  - Δwh = (δwh + λwh)
  - Δbh = (δbh) (não há regularizador para o bias)
  - wh -= η(Δwh)
  - bh -= η(Δbh)
  - Δwout = (δwout + λwout)
  - Δbout = (δbout) (não há regularizador para o bias)
  - wout -= η(Δwout)
  - bout -= η(Δbout)

One Hot

- Como estamos implementando MLP para classificação em várias classes, temos que comparar com um vetor tx1 que indicará qual a classe correta (indicada pelo valor 1).
- No exemplo abaixo, a amostra pertence a classe 2.

$$a^{(3)} = \begin{bmatrix} 0.1 \\ 0.9 \\ \vdots \\ 0.3 \end{bmatrix}, \; \mathbf{y} = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

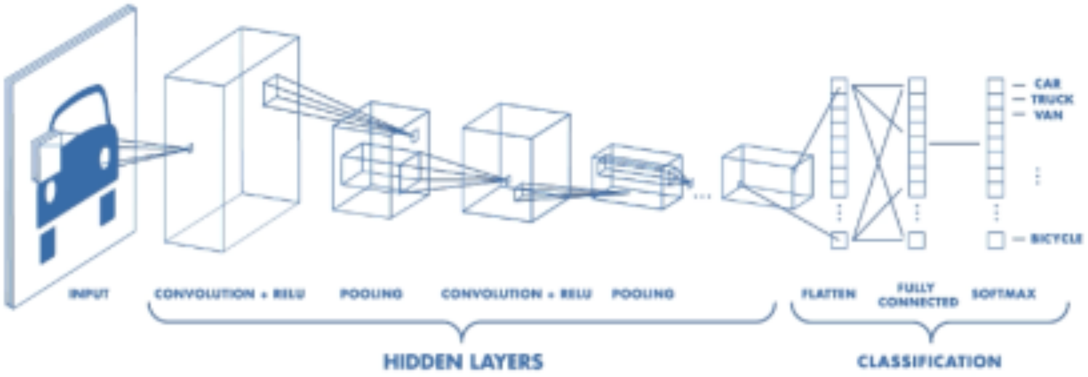
4. Convolutional Neural Networks

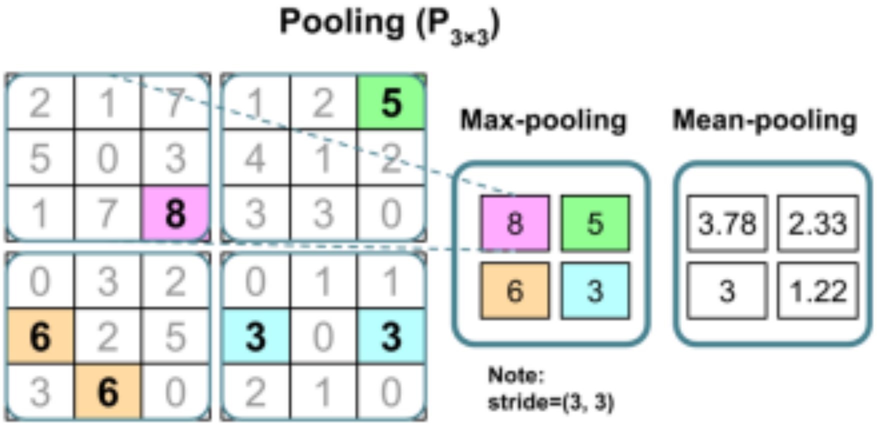
Redes Neurais Convolucionais (CNNs)

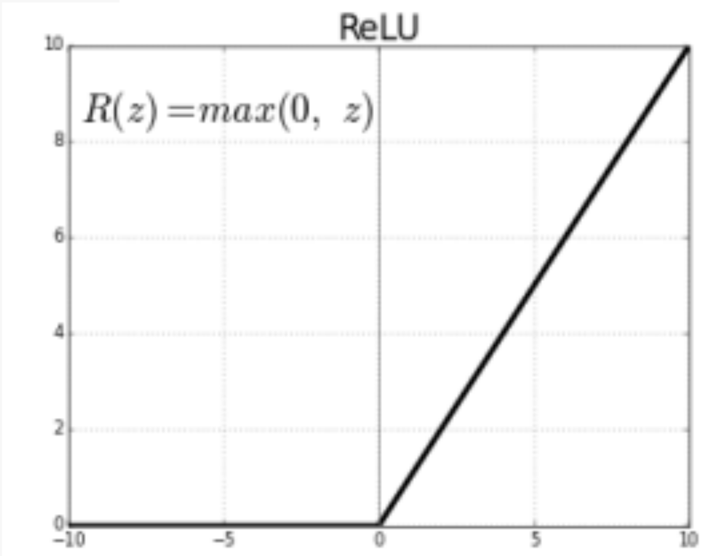
- MLP e CNNs profundas constroem uma hierarquia de features combinando features baixo nível em uma camada para reproduzir features de alto nível.
- Em imagens, as features de baixo nível são arestas e bordas, e são extraídas das primeiras camadas.
- As features de alto nível são formato dos objetos, tais como um prédio, carro ou cachorro.
- As CNNs são compostas por várias camadas convolucionais e sub-amostrais (Pooling), seguida por uma ou mais camadas totalmente conectadas.

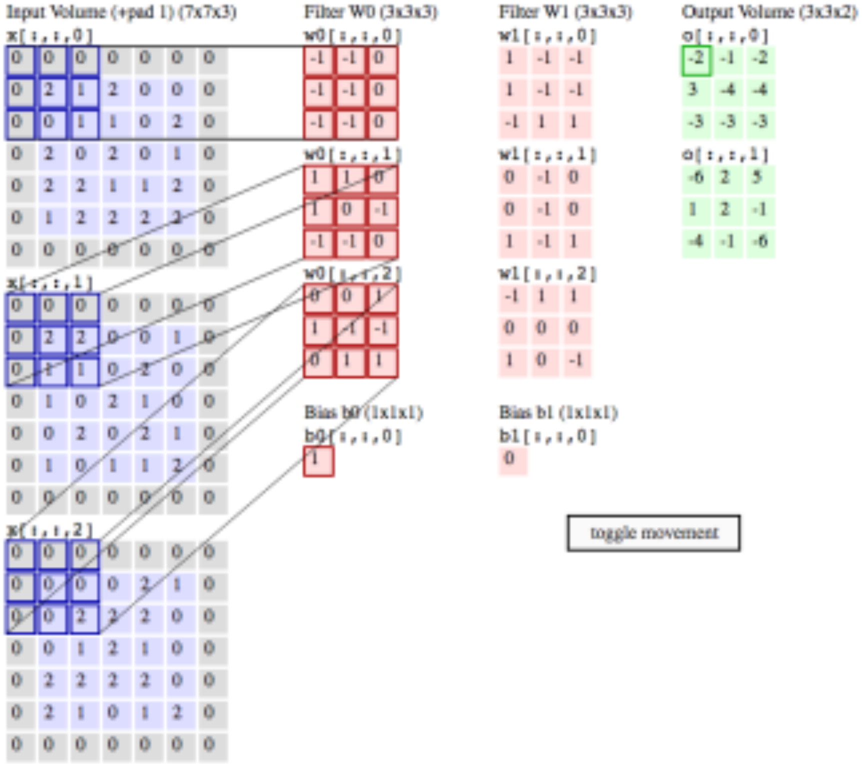
Camadas da CNN

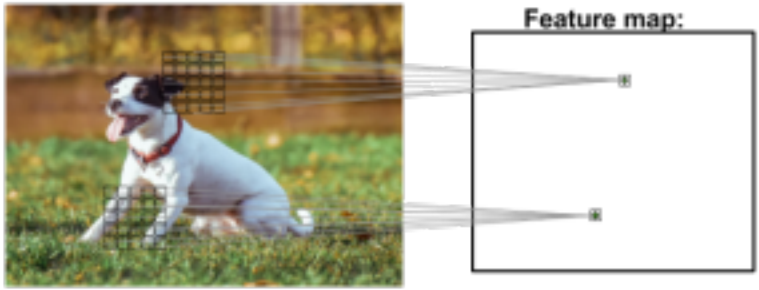
- Pooling: diminui a dimensionalidade, o que ajuda a reduzir o número de features e o overfitting.
- Convolução: Filtros (matrizes). Podem ser vistos como vários Perceptrons que se conectam a toda a imagem.
- A camada Flatten achata a saída das camada CNN em um array 2D.
- Relu e Softmax são funções de ativação.




Pooling (P3×3)
Max-pooling
Mean-pooling
8 5
6 3
3.78 2.33
3 1.22
Note:
stride=(3, 3)

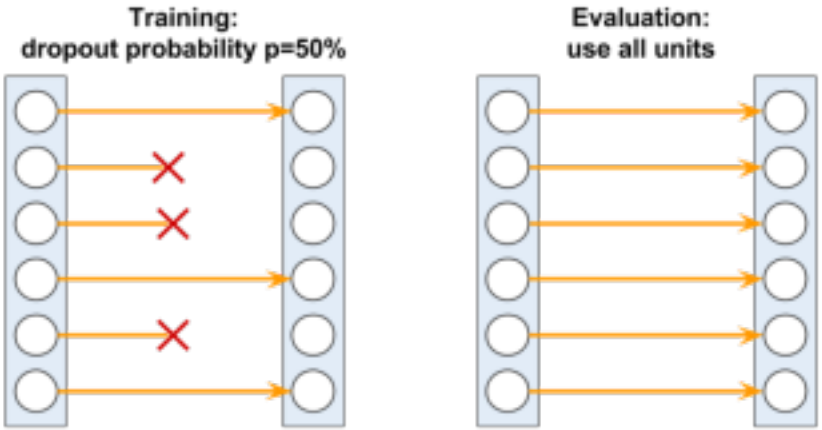

ReLU
R(z) = max(0, z)


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
x[:,:,1]
x[:,:,2]
Filter W0 (3x3x3)
w0[:,:,0]
w0[:,:,1]
w0[:,:,2]
Filter W1 (3x3x3)
w1[:,:,0]
w1[:,:,1]
w1[:,:,2]
Output Volume (3x3x2)
o[:,:,0]
o[:,:,1]
Bias b0 (1x1x1)
b0[:,:,0]
Bias b1 (1x1x1)
b1[:,:,0]
toggle movement


Feature map:

Parâmetros da CNN

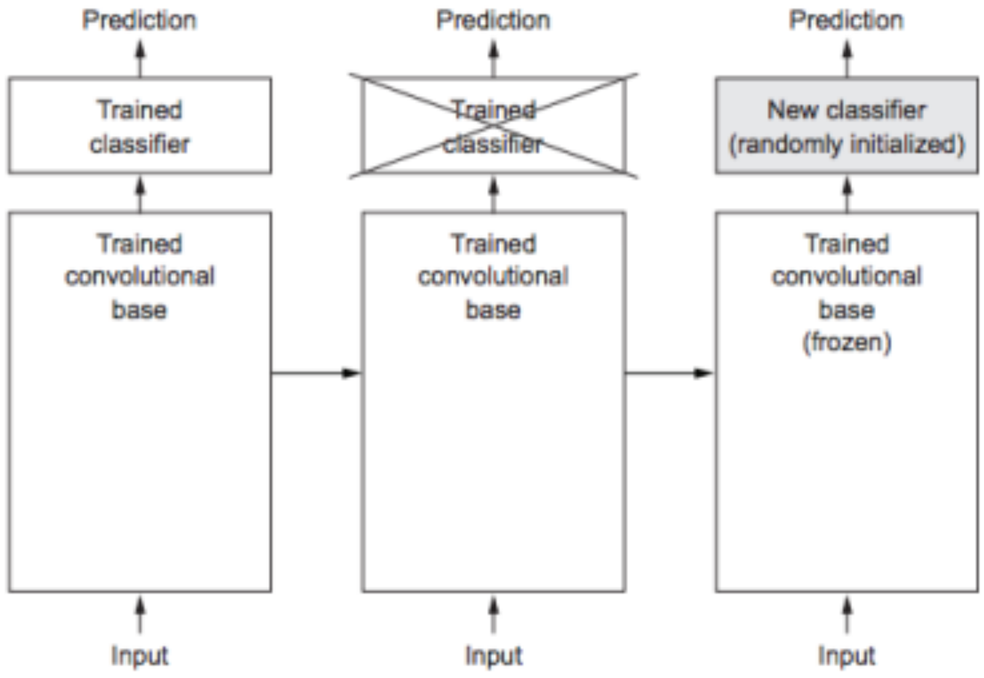
- Tamanho do Kernel (ou seja, o tamanho da matriz de pesos)
- Número de filtros
- The stride (quais são os *steps* do filtro)
- Padding
- Dropout (probabilidade de desativar alguns neurônios nas *hidden layers*)

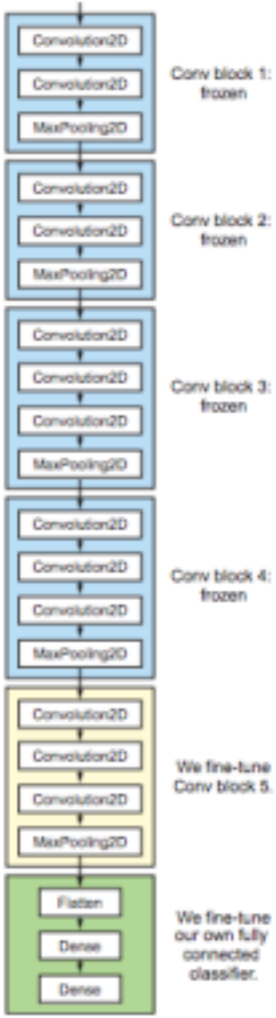


Usando uma rede pré-treinada

- Considere uma rede convolucional treinada sobre o ImageNet (1,4 milhões de imagens e 1,000 labels diferentes).
- ImageNet contém muitas classes, incluindo diferentes espécies de gatos, cachorros, entre outros.
- Existem duas formas de usar uma rede pre-treinada: feature extraction e fine tunning.
- Vários modelos estão presentes em bibliotecas como Keras e TensorFlow:
  - XCeption, Inception V3, ResNet50, VGG16, VGG19 e MobileNet.

Usando uma rede pré-treinada





Feature Extraction a esquerda e Fine-tunning a direita

OBRIGADO!

Dúvidas?

Você pode me encontrar em

- ticianalc@insightlab.ufc.br