Processamento e Análise de Imagens

Redes Neurais Artificiais

Felipe Augusto Lima Reis

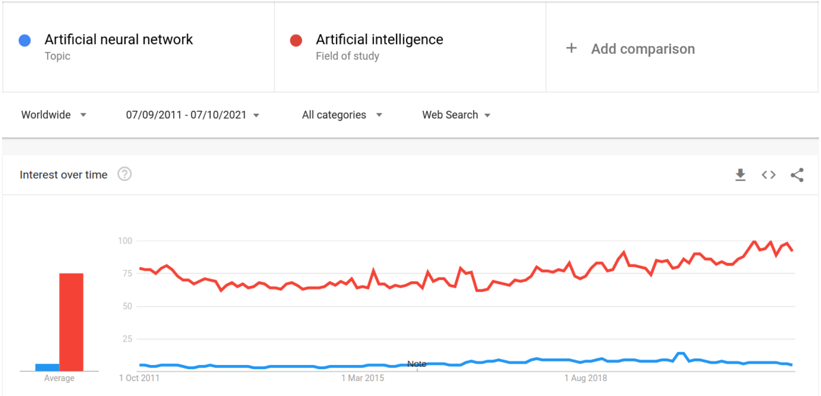
PUC Minas

Agenda

Contexto

Contexto

- Redes neurais sempre foram um tópico de interesse em Inteligência Artificial.



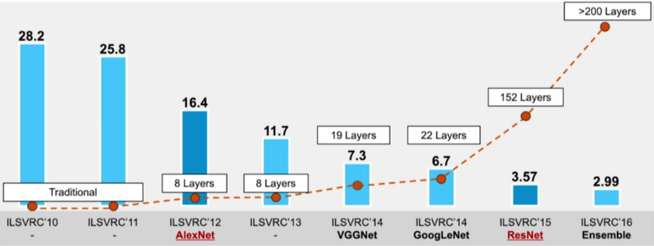
Fonte: [Google Trends, 2020]

Contexto

- O crescimento da capacidade das redes neurais nos últimos anos está relacionado aumento do poder computacional, especialmente das GPUs
  - Como veremos no decorrer da disciplina, o treinamento de redes neurais pode ser muito caro computacionalmente;
  - O uso de processamento paralelo, principalmente em GPUs, pode acelerar consideravelmente o aprendizado.

Contexto

- Nos últimos anos, um dos eventos mais importantes na área de redes neurais foi o desempenho da rede AlexNet na detecção e classificação de objetos no ILSVRC[1] 2012
  - A rede diminuiu o erro das top-5 classes previstas em mais de 10% [Goodfellow et al., 2016] [Krizhevsky et al., 2012].



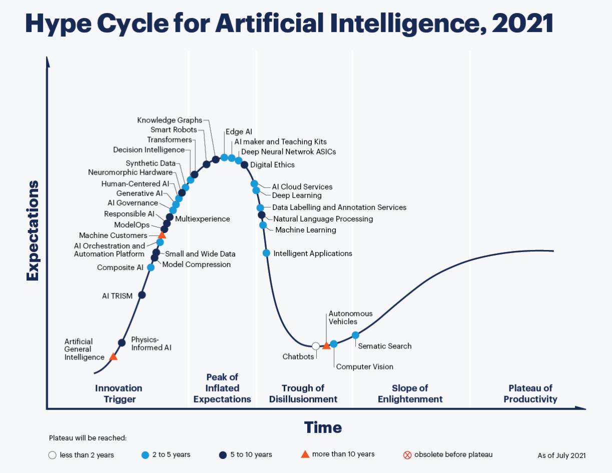
Fonte: [Sadek Alaoui - SQLML, 2017]

[1] ImageNet Large Scale Visual Recognition Challenge (ILSVRC).

Contexto

- O aumento do desempenho das redes neurais possibilitou o surgimento de diversas aplicações:
  - Assistentes virtuais;
  - Processamento de linguagem natural;
  - Ferramentas de tradução automática;
  - Predição de desastres naturais;
  - Suporte ao diagnóstico médico;
  - Sistemas de recomendações;
  - Veículos autônomos;
  - ...

Contexto



Fonte: [Laurence Goasduff - Gartner, 2021]

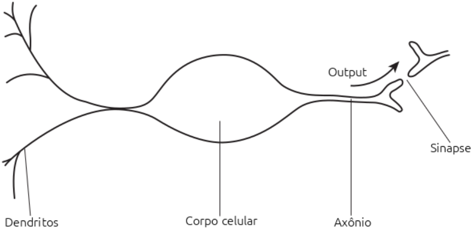
O que estudaremos de redes neurais?

- O que veremos nesta seção da disciplina:
  - Inspiração para primeiros modelos de redes neurais;
  - Perceptron e Perceptron Multicamadas;
  - Tipos de aprendizado;
  - Redes supervisionadas, não supervisionadas, recorrentes e convolucionais.
- O que não veremos nesta disciplina:
  - Arquiteturas de redes neurais;
  - Regularização;
  - Inicialização de pesos;
  - Otimizadores;
  - Pré-processamento e aumento artificial de dados;
  - ...

Inspiração Biológica

Neurônios biológicos

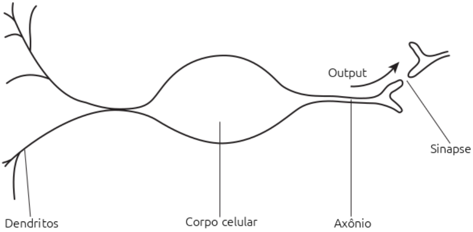
- Neurônios biológicos são células especializadas responsáveis por gerar e transmitir impulsos nervosos, com capacidade para responder a estímulos do meio;
- Seu funcionamento é baseado em alterações na diferença de potencial elétrico em sua membrana [Montanari, 2016];



Fonte: Adaptado de [Coppin, 2004]

Neurônios biológicos

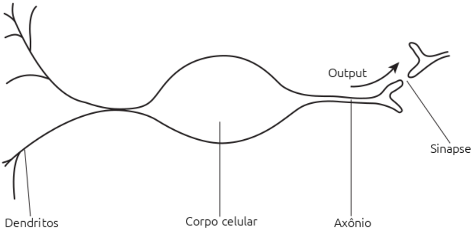
- Os neurônios formam uma rede de conexões capaz de captar informações dos receptores sensoriais, processá-las, originar memórias e/ou sinais apropriados [Montanari, 2016];
- Os locais de comunicação entre dois neurônios, ou entre um neurônio e a célula efetora (célula que esponde ativamente a um estímulo), são chamados de sinapses [Montanari, 2016].



Fonte: Adaptado de [Coppin, 2004]

Neurônios biológicos

- Um neurônio pode ser subdividido nas seguintes partes:
  - Dendritos: captam estímulos de outros neurônios ou do meio;
  - Corpo celular (soma): processam estímulos, produzindo um potencial de ativação capaz de disparar um impulso elétrico;
  - Axônio: conduz impulsos para outros neurônios conectores ou ligados a um tecido muscular.



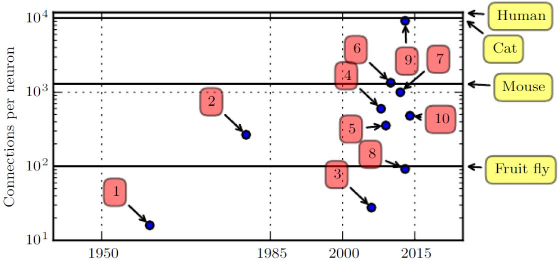
Fonte: Adaptado de [Coppin, 2004]

Cérebro humano

- Constituído por mais de 10 bilhões de neurônios e cerca de 60 trilhões de sinapses [Coppin, 2004];
- Possui como propriedade a plasticidade:
  - Neurônios podem mudar a natureza e o número de conexões, em resposta a eventos, possibilitando o aprendizado;
  - Ao aprender, teoricamente, o cérebro:
    - Reforça conexões que levam a soluções corretas;
    - Enfraquece conexões que levam a soluções incorretas;
  - A intensidade da sinapse, determina influência sobre os neurônios aos quais está conectado;
  - Caso uma conexão seja enfraquecida, ela terá menor influência em respostas a estímulos [Coppin, 2004].

Neurônios artificiais

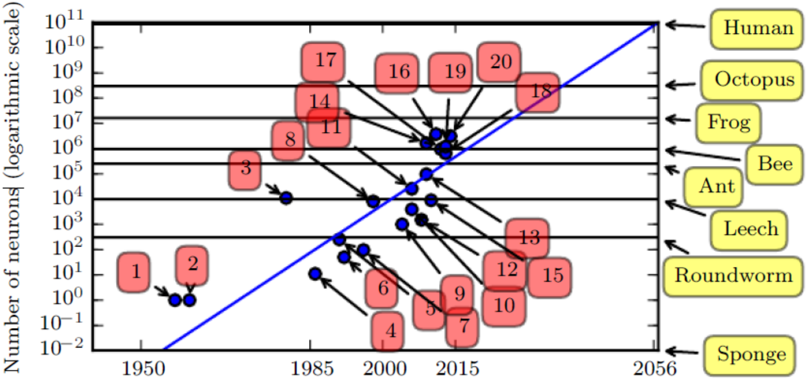
- Inspirados por neurônios e sistemas nervosos biológicos, foram criadas as primeiras redes neurais artificiais;
- Redes neurais artificiais, em geral, são menores e possuem significativamente menos conexões que o cérebro humano.



Fonte: [Goodfellow et al., 2016]

(9) COTS HPC unsupervised convolutional network (Coates et al., 2013)

Neurônios artificiais



Fonte: [Goodfellow et al., 2016]

---

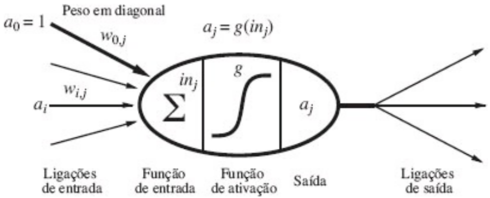
(1) Perceptron 1 (Rosenblatt, 1958, 1962)

(19) COTS HPC unsupervised convolutional network (Coates et al., 2013)

(20) GoogLeNet 2 (Szegedy et al., 2014a)

Neurônios artificiais

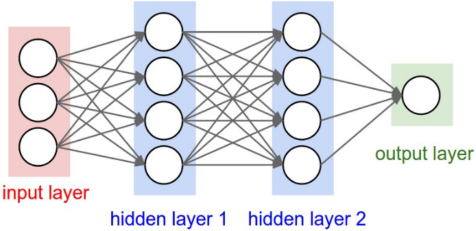
- Neurônios artificiais correspondem a um modelo matemático simples, desenvolvido McCulloch e Pitts, em 1943 [Russel and Norvig, 2013] [Coppin, 2004];
- Eles disparam uma combinação linear de suas entradas quando algum limiar é excedido [Russel and Norvig, 2013].



Fonte: [Russel and Norvig, 2013]

Redes neurais artificiais

- Uma rede neural é apenas uma coleção de neurônios
  - Suas propriedades são determinadas pela topologia e pelos próprios neurônios [Russel and Norvig, 2013].



Fonte: Adaptado de [Li et al., 2021]

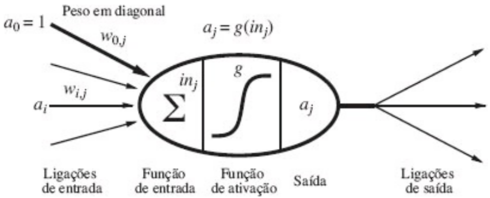
Neurônios Artificiais

Neurônios Artificiais

- O modelo de neurônio proposto por McCulloch e Pitts (1943) possui semelhanças com neurônios biológicos [da Silva, 2014]:
  - Entrada: sinais $\{x_1, x_2, ..., x_n\}$ oriundos do ambiente externo;
  - Pesos Sinápticos (w_i): utilizado para definição de "importância" ou "relevância" das entradas para o neurônio;
  - Bias (w_b): entrada extra, utilizada para aumentar o grau de liberdade dos ajustes dos pesos;
  - Corpo: soma os produtos das entradas e pesos $(x_i \times w_i)$ com o bias (w_b) e aplica a função de ativação;
  - Função de Ativação: controla o comportamento do sinal da saída $f(x)$ de um neurônio limitando o intervalo de valores assumidos.

Funcionamento de um Neurônio Artificial

- Funcionamento de um neurônio:
  1. Cada neurônio possui um sinal de entrada x_i;
  2. Esse neurônio i liga-se a outro neurônio j e é capaz de propagar um valor de ativação x_i;
  3. Cada ligação tem um peso numérico $w_{i,j}$ associado;



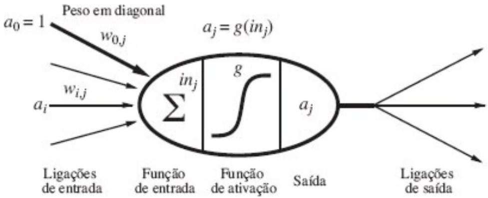
Fonte: [Russel and Norvig, 2013]

Baseado em [da Silva, 2014] e [Russel and Norvig, 2013]

Funcionamento de um Neurônio Artificial

- Funcionamento de um neurônio:
  4. O potencial de ativação de um neurônio é dado pela soma ponderada dos sinais de entrada, somado ao bias (w_b);

$$in_j = \sum_{i=1}^{n} (x_i \cdot w_i) + w_b$$



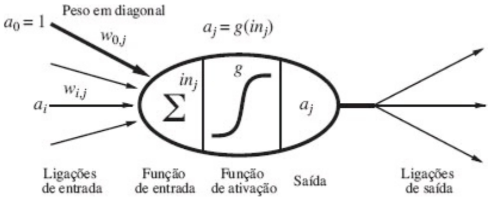
Fonte: [Russel and Norvig, 2013]

Baseado em [da Silva, 2014] e [Russel and Norvig, 2013]

Funcionamento de um Neurônio Artificial

- Funcionamento de um neurônio:
  5. Em seguida, é aplicada uma função de ativação g a essa soma, com objetivo de limitar o sinal de saída.

$$x_j = g\left(\sum_{i=1}^{n}(x_i \cdot w_i) + w_b\right)$$

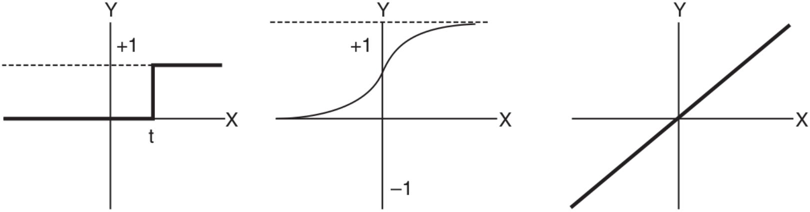


Fonte: [Russel and Norvig, 2013]

Baseado em [da Silva, 2014] e [Russel and Norvig, 2013]

Funções de ativação

- Funcões de ativação são aplicadas sobre o neurônio para limitar ou obter o nível de ativação na saída.



(a) Função degrau **(b)** Função sigmoide **(c)** Função linear

Fonte: Adaptado de [Coppin, 2004]

Perceptrons

Perceptrons

- Perceptrons são neurônios simples, propostos por Frank Rosenblatt, em 1958 [Coppin, 2004];
- Esses neurônios são utilizados para classificação de entradas em duas categorias (classificador binário);
  - Indicam se uma entrada pertence a um conjunto ou não;
  - Podem aprender operações booleanas, como AND ou OR;
  - São considerados como um tipo de classificador linear;
  - Introduziram o conceito de treinamento de neurônios e/ou redes neurais.

Perceptrons

- Um Perceptron utiliza uma função degrau (*step*) que retorna $+1$ se a soma de pesos da entrada X for maior que um limiar t e 0 se X é menor ou igual a t [Coppin, 2004]

$$Y = \begin{cases} +1 & for\ X > t \\ 0 & for\ X \leq t \end{cases} \qquad Step(X) = \begin{cases} +1 & for\ X > t \\ 0 & for\ X \leq t \end{cases}$$

Fonte: [Coppin, 2004]

Processo de Aprendizado

- Para que um Perceptron possa aprender, primeiramente ele deve passar por uma fase de treinamento [Coppin, 2004];
- Nessa fase, os pesos correspondentes às entradas são ajustados
  - Pesos podem ser considerados como indicadores de importância ou relevância das entradas.

Processo de Aprendizado

- Para início do aprendizado, pesos aleatórios são definidos (em geral, entre -0.5 e +0.5) [Coppin, 2004];
- Uma entrada é dada ao Perceptron, que produz uma saída
  - A saída do Perceptron é comparada com a saída esperada;
  - Caso a saída seja incorreta, os pesos são ajustados e uma nova tentativa é feita
    - A taxa de ajuste dos pesos é chamada de taxa de aprendizado, com valor típico entre 0 e 1;
  - Cada ciclo de ajuste de pesos (tentativa e avaliação de resultados) é chamada de época.

Processo de Aprendizado

- Formalmente, a fórmula de treinamento proposta por Rosenblatt (1960) é dada por [Coppin, 2004];

$$\boxed{w_i \leftarrow w_i + (\eta \times x_i \times e)}$$

  - Onde, η corresponde à taxa de aprendizado (*learning rate*) e e corresponde ao erro/perda (*error*).

- A equação é conhecida como Regra de Treinamento do Perceptron (*Perceptron Training Rule*).

Processo de Aprendizado

- Considere o treinamento[2] de um Perceptron para aprendizado da operação lógica OR.

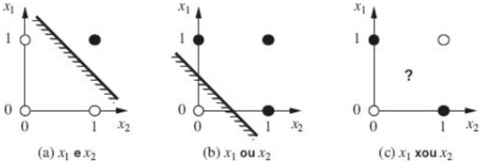
Época	w_1	w_2	x_1	x_2	Val. Esperado	Val. Atual	Erro	Ação
1	-0.2	0.4	0	0	0	0	0	-
1	-0.2	0.4	0	1	1	1	0	-
1	-0.2	0.4	1	0	1	0	1	Atual. Pesos
1	**0**	**0.4**	1	1	1	1	0	-
2	0	0.4	0	0	0	0	0	-
2	0	0.4	0	1	1	1	0	-
2	0	0.4	1	0	1	0	1	Atual. Pesos
2	**0.2**	**0.4**	1	1	1	1	0	-
3	0.2	0.4	0	0	0	0	0	-
3	0.2	0.4	0	1	1	1	0	-
3	0.2	0.4	1	0	1	1	0	-
3	0.2	0.4	1	1	1	1	0	-

Fonte: Adaptado de [Coppin, 2004]

[2] Pesos iniciais aleatórios, intervalo [-0.5, 0.5]. Limiar função degrau: $t = 0$. Taxa de aprendizado: $\eta = 0.2$.

Processo de Aprendizado

- Perceptrons são considerados classificadores lineares, ou seja, somente podem aprender modelos linearmentes separáveis;
  - Dessa forma, são incapazes de identificar conjuntos que não possam ser divididos por uma reta (ex. operação XOR).



Fonte: [Russel and Norvig, 2013]

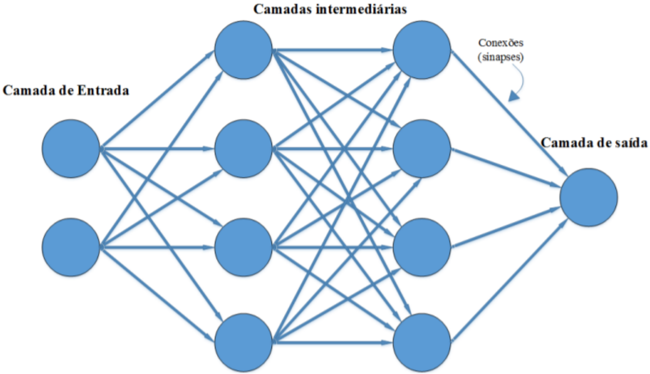
Redes Neurais

Redes Neurais

- Redes neurais são agrupamentos de neurônios artificiais
  - Cada um dos neurônios provê um mecanismo simples de processamento;
  - No contexto de redes, neurônios são chamados de nós.
- Redes podem ser arranjadas em camadas (*layers*)
  - A primeira camada, de entrada de dados, é chamada de camada de entrada;
  - As m camadas intermediárias são chamadas de camadas intermediárias ou ocultas;
  - A última camada, de saída da rede, é chamada de camada de saída;

Redes Neurais

- Um grafo de uma rede neural pode ser vista na imagem abaixo.



Fonte: [Antonelli and Neitzel, 2015]

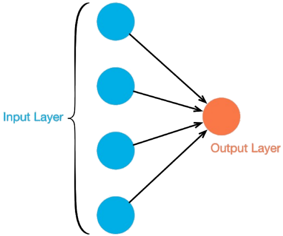
Funcionamento das Redes Neurais

- Funcionamento básico das redes neurais:
  1. Neurônios na camada de entrada recebem entradas para serem classificadas;
  2. As entradas causam o disparo de alguns neurônios (retornam valor acima de um determinado limiar, de acordo com a função de ativação);
  3. Neurônios das camadas iniciais transmitem o sinal para os neurônios das camadas seguintes;
  4. O processo se repete até o final da rede, quando a rede retorna um ou múltiplos valores de saída[3].

[3] Múltiplos valores de saída podem ser utilizados, por exemplo, no processamento de imagens, onde cada saída corresponde a um *pixel* de resultado.

Classificação das Redes Neurais

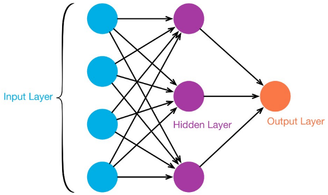
- Redes de Camada Única
  - Extremamente simples, usadas para classificação de padrões e filtragem linear;
  - Possuem somente uma camada de entrada e outra de saída;
  - O fluxo de informação é unidirecional;
  - São exemplos as redes Perceptron e Adaline.



Fonte: [Nahua Kang - Towards Data Science, 2017]

Classificação das Redes Neurais

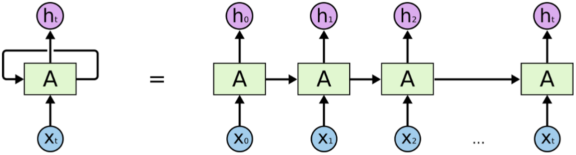
- Redes de Múltiplas Camadas
  - Redes mais utilizadas, com múltiplas camadas intermediárias;
  - Utilizadas para classificação de padrões, otimização, robótica, controle de processos, etc.
  - Possuem somente uma camada de entrada, *m* camadas intermediárias e uma camada de saída;
  - São exemplos as redes Perceptron Multicamadas (MLP).



Fonte: [Nahua Kang - Towards Data Science, 2017]

Classificação das Redes Neurais

- Redes Recorrentes
  - Redes com múltiplas camadas intermediárias que realimentam neurônios com valores a partir das saídas;
  - Utilizada em séries temporais, otimização e controle de processos dinâmicos;
  - Buscam estabilidade de comportamento, de forma a evitar oscilações e comportamentos caóticos;
  - São exemplos as Redes de Hopfield e as redes Perceptron Multicamadas com Realimentação.



Fonte: [Matheus Facure, 2017]

Feed-forward Neural Network

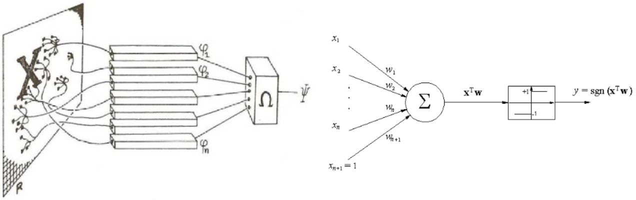
- Uma rede com alimentação para a frente (*feed-forward network*) é aquela que tem conexões somente em uma direção, isto é, forma um grafo acíclico dirigido [Russel and Norvig, 2013];
- Cada nó recebe valores de entrada das camadas anteriores e fornece resultados para nós em camadas subsequentes;
  - Não há laços (auto-*loop*);
- Não possuem estado interno que não sejam os próprios pesos.

Redes Perceptron e Multilayer Perceptron

Redes Perceptron

Redes Perceptron

- As primeiras redes perceptron eram formados por um único neurônio;
- Uma estrutura pré-processava as entradas para, em seguida, entregar os valores ao neurônio [da Silva, 2014];
- A Rede Perceptron pode ser considerada uma rede de camada única com alimentação para frente [Russel and Norvig, 2013].



Fonte: [da Silva, 2014]

Redes Perceptron

- Redes Perceptron com m saídas possuem o equivalente a m redes separadas
  - Cada peso afeta apenas uma das saídas;
  - São necessários m processos de treinamento separados [Russel and Norvig, 2013];
- O aprendizado usa a Regra de Aprendizagem Perceptron
  - Cada Perceptron é treinado separadamente, com a seguinte fórmula[4]:

$$\boxed{w_i \leftarrow w_i + (\eta \times x_i \times e)}$$

[4] Onde e corresponde ao erro entre a saída e o valor esperado e η corresponde a taxa de aprendizado.

Treinamento Redes Perceptron

- O treinamento das redes Perceptron podem ser sumarizados (e expandidos) na seguinte equação:

$$\boxed{w_j(t+1) = w_j(t) + ((\eta \times x_j^{(i)})(y^{(i)} - \hat{f}(x^{(i)}))}$$

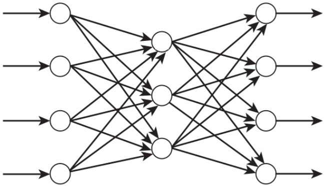
onde

- $w_j(t)$: peso da j-ésima conexão de entrada no instante t;
- η: taxa de aprendizado (*learning rate*);
- $x_j^{(i)}$: valor do j-ésimo atributo do vetor de entrada x (i);
- $\hat{f}(a^{(i)}$: saída produzida pela rede no instante de tempo t para o vetor $a(i)$;
- $y(i)$: saída desejada pela rede para o vetor $a(i)$;
- $(y^{(i)} - \hat{f}(a^{(i)})$: erro e para uma determinada saída.

Redes Perceptron Multicamadas (MLP)

Redes MLP

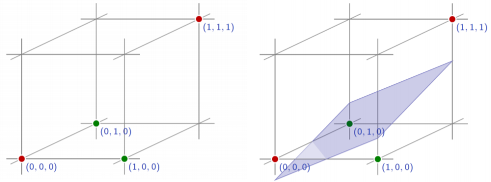
- Redes Perceptron Multicamadas (*Multilayer Perceptron*)) correspondem à adição de camadas intermediárias à rede Perceptron;
- As camadas intermediárias possibilitam a representação de funções contínuas (não-linearmente separáveis).



Fonte: [Coppin, 2004]

Redes MLP

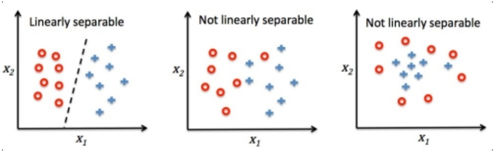
- Redes MLP são capazes de modelar problemas mais complexos que as redes Single-Layer Perceptron;
  - Podem solucionar, por exemplo, a operação lógica XOR;
    - Essa operação pode ser decomposta em função de NAND, OR e NOR, que são linearmente separáveis;
    - Com isso, a operação XOR pode ser aprendida por um perceptron multicamada;



Fonte: Francois Fleuret at EPFL apud [Lee, 2020]

Redes MLP

- Outras funções que também não são linearmente separáveis podem ser solucionadas por redes MLP.



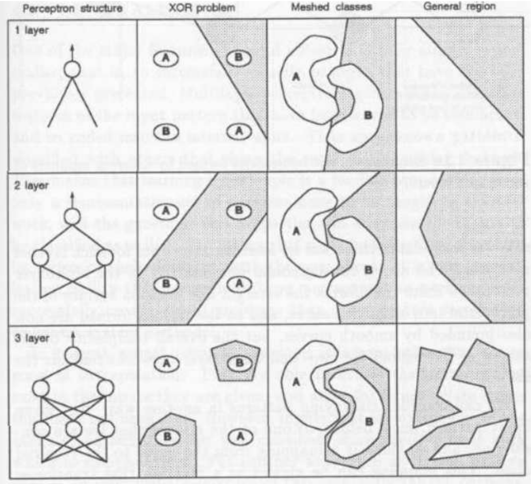
Fonte: [Raschka, 2015]

Redes MLP

- A especialização de uma rede MLP de 3 camadas, pode, por exemplo, seguir o seguinte modelo[5]:
    - Camada 1: neurônio aprende a função correspondente a representação de um hiperplano;
    - Camada 2: combina grupos de hiperplanos definidos pela camada anterior, formando regiões convexas;
    - Camada 3: combina conjuntos de regiões convexas em regiões de formato arbitrário.

[5] [Faceli et al., 2011] apud [da Silva, 2014]

Redes MLP

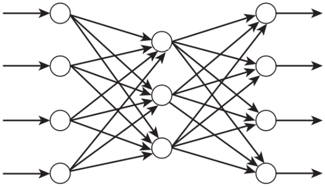


Fonte: [da Silva, 2014]

Treinamento de Redes Multicamadas

Treinamento de Redes MLP

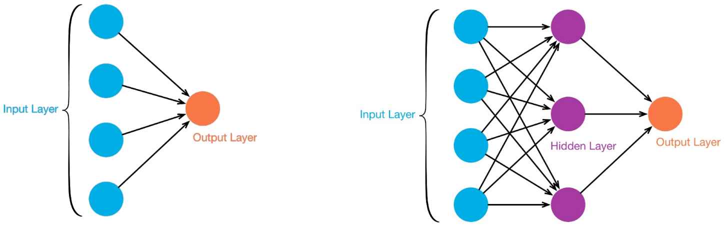
- Apesar das possibilidades de generalização das redes MLP, ninguém sabia como treinar a rede [Russel and Norvig, 2013]
  - Redes de múltiplas camadas possuem múltiplos pesos associados às entradas;
  - O número de pesos a serem ajustados é extremamente alto [Coppin, 2004].



Fonte: [Coppin, 2004]

Treinamento de Redes MLP

- Em redes de uma camada, o ajuste de pesos de um neurônio não altera o valor de outro
  - No entanto, em redes multicamadas, a alteração de pesos em camadas iniciais impactam diretamente nas camadas finais;
  - Alterar pesos de camadas iniciais significa alterar valores das camadas subsequentes [Russel and Norvig, 2013].



Fonte: [Nahua Kang - Towards Data Science, 2017]

Treinamento de Redes MLP

- Para solucionar esses problemas, o treinamento da rede é dividido em 2 fases
  - Fase Forward
    - Ativações dos neurônios (valores x pesos ativados) são propagadas da entrada para saída;
  - Fase Backward
    - A erro/perda entre o valor produzido pela rede e o valor correto é propagado para trás, a fim de modificar pesos e valores de *bias*;
- As fase de propagação para frente e para trás são dependentes [Zhang et al., 2020].

Forward Propagation

- O treinamento em redes com alimentação para frente ocorre, tradicionalmente, para frente ao longo da rede;
- Por esse motivo, esta fase é denominada Forward Propagation ou Forward Pass (propagação para frente) [Zhang et al., 2020] [Goodfellow et al., 2016];
- Na Fase Forward, a propagação para frente é utilizada para cálculo dos pesos
    - Cada neurônio computa seus valores e passa a informação para o neurônio seguinte, até a saída da rede;
    - Os valores gerados pela rede são comparados aos valores de treinamento e a diferença entre eles é a perda (erro).

Backpropagation

- O algoritmo Backpropagation (retropropagação) foi desenvolvido por Rumelhart, Hinton & Williams, em 1986 [Goodfellow et al., 2016];
- Esse algoritmo é vastamente utilizado na Fase Backward, onde a ordem de cálculo é invertida em relação a fase forward [Zhang et al., 2020];
- O algoritmo permite que informações voltem através da rede, a fim de calcular o gradiente de erro e ajustar os pesos [Goodfellow et al., 2016] [Coppin, 2004].

Backpropagation - Conceitos

Backpropagation - Função Sigmoide

- Antes de detalharmos o algoritmo e os cálculos, vamos analisar a função sigmoide:

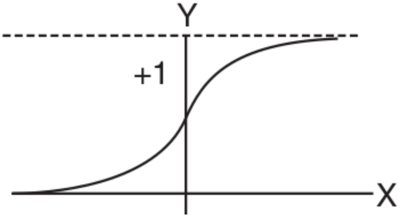
$$\boxed{\sigma(x) = \frac{1}{1 + e^{-x}}}$$

- Sua derivada é dada por:

$$\boxed{\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x))}$$

Backpropagation - Função Sigmoide

- A função, ainda, é bem comportada, no intervalo entre 0 e 1.



Fonte: Adaptado de [Coppin, 2004]

Backpropagation - Regra da Cadeia

- Suponha duas funções, $y = f(x)$ e $z = g(y)$;
- Usando a regra da cadeia, podemos calcular a derivada como:

$$\boxed{(f \circ g)'(x) = (f(g(x)))' = f'(g(x))g'(x)}$$

- Alternativamente, usando z em relação a x temos:

$$\boxed{\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x}}$$

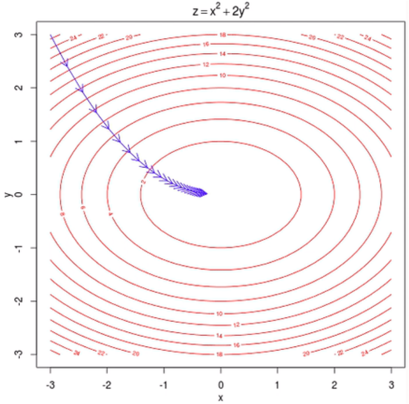
Backpropagation - Método do Gradiente

- O Método do Gradiente[6] ou Método do Máximo Declive é o método numérico usado para minimização do erro esperado;
- O método busca seguir o caminho mais íngreme na superfície que representa a função de erro, de modo a encontrar o mínimo de erro [Coppin, 2004].

[6] Frequentemente utiliza-se o nome em inglês, *Gradient Descent* (GD).

Backpropagation - Método do Gradiente

- Considere a curva de nível abaixo:
  - O método seguirá o declive máximo da curva em cada trecho.



Fonte: [Hoang Duong - Hoang Duong blog, 2015]

Link: versão animada do método do gradiente.

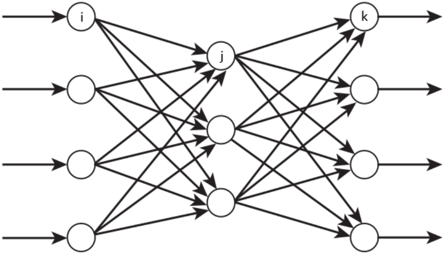
Backpropagation - Processo

Backpropagation

- O método percorre a rede em ordem reversa, de acordo com a Regra da Cadeia (Cálculo) [Zhang et al., 2020];
- O algoritmo armazena variáveis intermediárias (derivadas parciais) para cálculo do gradiente [Zhang et al., 2020].
- Para que seja possível calcular o gradiente, a função deve ser contínua e diferenciável em todos os pontos
  - Com isso, a função degrau não pode ser utilizada, uma vez que não é contínua;
  - Outras funções, como a sigmoide e a tangente hiperbólica são utilizadas (em especial, a sigmoide).

Backpropagation

- Para entender o algoritmo de backpropagation, vamos considerar uma rede neural com 3 camadas;
- Consideremos a notação: (i) nós de entrada, (j) nós intermediários e (k) nós de saída.
- O peso w_{ij} corresponde a conexão entre nós i e j.



Fonte: Adaptado de [Coppin, 2004]

Backpropagation

- A função usada para derivar a saída do nó j é dada por:

$$\boxed{Y_j = \frac{1}{1 + e^{-X}}} \quad \text{, sendo} \quad \boxed{X_j = \sum_{i=1}^{n} x_i \cdot w_{ij} - \theta_j}$$

onde

- n: número de entradas do nó j;
- w_{ij}: pesos da conexão entre o nó i e j;
- θ: limiar do nó j (valor aleatório entre 0 e 1);
- x_i: valor de entrada para o nó de entrada i;
- y_j: valor produzido para o nó j;

Backpropagation

- O erro entre o valor gerado pela rede e o valor esperado é dado por:

$$\boxed{e_k = \frac{1}{2}\left(d_k - y_k\right)^2}$$

- O gradiente de erro é dado pela fórmula:

$$\boxed{\delta_k = \frac{\partial y_k}{\partial x_k} \cdot e_k}$$

- Onde ∂y_k corresponde a derivada da sigmoide, gerando:

$$\boxed{\delta_k = y_k \cdot (1 - y_k) \cdot e_k}$$

Backpropagation

- A derivada do erro para uma camada intermediária é dada por:

$$\boxed{\delta_j = y_j \cdot (1 - y_j) \sum_{k=1}^{n} w_{ij} \cdot \delta_k}$$

- Esses cálculos produzem:

$$\boxed{w_{ij} \leftarrow w_{ij} + (\eta \cdot x_i \cdot \delta_j)}$$

$$\boxed{w_{jk} \leftarrow w_{jk} + (\eta \cdot y_j \cdot \delta_k)}$$

Onde η corresponde a taxa de aprendizado (*learning rate*).

Backpropagation

- Sumarizando (1/2)

$$\boxed{w_{jl}(t+1) = w_{jl}(t) + (\eta \cdot x^{j} \cdot \delta_{l})}$$

onde,

- $w_j l$: peso entre neurônio l e o j-ésimo atributo de entrada ou a saída do j-ésimo neurônio da camada anterior;
- δ_l: erro associado ao l-ésimo neurônio;
- x^j: entrada recebida pelo l-ésimo neurônio (j-ésimo atributo de entrada ou saída do j-ésimo neurônio da camada anterior);
- η: taxa de aprendizado;

Adaptado de [da Silva, 2014]

Backpropagation

- Sumarizando (2/2)

$$\boxed{\delta_l = \begin{cases} f'_a e_l & \text{se camada de saída} \\ f'_a \sum w_{kl}\delta_k & \text{se camada intermediária} \end{cases}}$$

onde,

- f'_a: derivada parcial da função de ativação do neurônio;
- e_l: erro cometido pelo neurônio em relação à saída esperada

$$\boxed{e_l = \frac{1}{2}\sum_{q=1}^{k}(y_q - \hat{f}_q)^2}$$

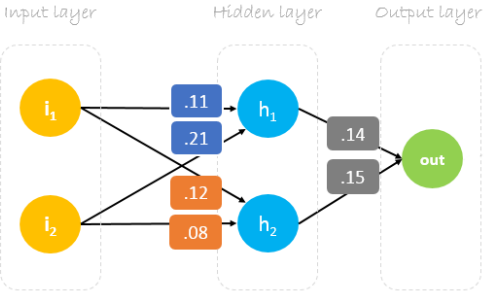
Adaptado de [da Silva, 2014]

Cálculo do Método Backpropagation [7]

[7] Baseado em [Hani M. K., 2019]

Pesos Iniciais

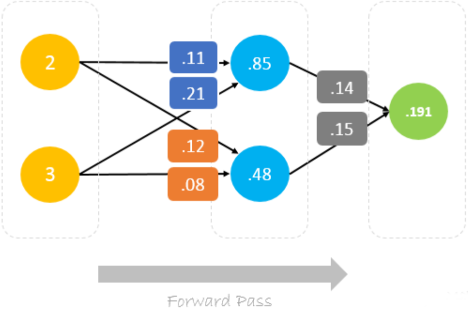
- Exemplo adaptado de [Hani M. K., 2019];
- Vamos considerar uma rede neural de 3 camadas, com os pesos iniciais já definidos.



Fonte: [Hani M. K., 2019]

Forward Pass

- Consideremos valores iniciais de entrada $i_1 = 2$ e $i_2 = 3$;
- Consideremos que o valor esperado na saída da rede é $z = 1$.



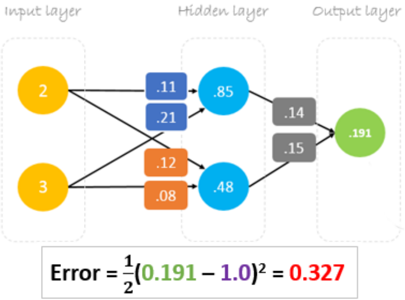
$$\begin{bmatrix} 2 & 3 \end{bmatrix} \cdot \begin{bmatrix} 0.11 & 0.12 \\ 0.21 & 0.08 \end{bmatrix} = \begin{bmatrix} 0.85 & 0.48 \end{bmatrix} \cdot \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} = \begin{bmatrix} 0.191 \end{bmatrix}$$

Fonte: [Hani M. K., 2019]

Cálculo do Erro

- Podemos cálcular o erro entre a saída esperada e a saída da rede usando a fórmula:

$$\boxed{e_l = \frac{1}{2}\sum_{q=1}^{k}(y_q - \hat{f}_q)^2}$$



Fonte: [Hani M. K., 2019]

Cálculo da Derivada do Erro

- O erro entre o valor gerado pela rede e o valor esperado, para uma única saída, é dado por:

$$\boxed{e_k = \frac{1}{2}(d_k - y_k)^2}$$

- Podemos cálcular a derivada do erro para um nó da rede, fazemos:

$$\boxed{\delta_l = \frac{\partial e_l}{w_l} = 2 \times \frac{1}{2}(d_k - y_k)}$$

Backward Pass

- Para cálculo do backpropagation, devemos usar a equação:

$$\boxed{w_{jl}(t+1) = w_{jl}(t) + (\eta \cdot x^{j} \cdot \delta_{l})}$$

onde,

- $w_j l$: peso entre neurônio l e o j-ésimo atributo de entrada ou a saída do j-ésimo neurônio da camada anterior;
- δ_l: erro associado ao l-ésimo neurônio;
- x^j: entrada recebida pelo l-ésimo neurônio (j-ésimo atributo de entrada ou saída do j-ésimo neurônio da camada anterior);
- η: taxa de aprendizado;

Backward Pass

- Considerando o erro da saída, temos:

$$\delta_k = (d_k - y_k) = (0.191 - 1) = -0.809$$

- Considerando uma taxa de aprendizado empírica $\eta = 0.05$, temos:

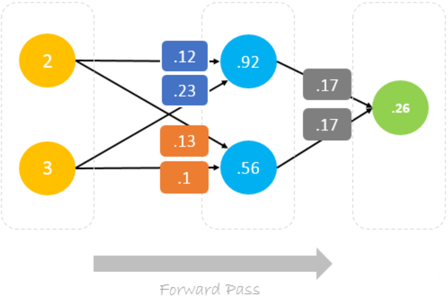
$$\begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - 0.05(\text{-}0.809) \begin{bmatrix} 0.85 \\ 0.48 \end{bmatrix} = \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - \begin{bmatrix} \text{-}0.034 \\ \text{-}0.019 \end{bmatrix} = \begin{bmatrix} 0.17 \\ 0.17 \end{bmatrix}$$

$$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - 0.05(-0.809) \begin{bmatrix} 2 \\ 3 \end{bmatrix} . [0.14 \quad 0.15] = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - \begin{bmatrix} \text{-}0.011 & \text{-}0.012 \\ \text{-}0.017 & \text{-}0.018 \end{bmatrix} = \begin{bmatrix} .12 & .13 \\ .23 & .10 \end{bmatrix}$$

Fonte: [Hani M. K., 2019]

Forward Pass

- Um novo *forward pass* é executado.



$$[2 \quad 3] \cdot \begin{bmatrix} 0.12 & 0.13 \\ 0.23 & 0.10 \end{bmatrix} = [0.92 \quad 0.56] \cdot \begin{bmatrix} 0.17 \\ 0.17 \end{bmatrix} = [0.26]$$

Fonte: [Hani M. K., 2019]

Critério de Parada

- O algoritmo executa novamente até atingir um critério de parada:
  - Atingir um número pré-determinado de épocas de treinamento;
  - Não melhorar seu desempenho por um determinado número de épocas;
  - Atingir um determinado valor de perda, acurácia, etc.

Referências I

Antonelli, G. and Neitzel, I. (2015).
Aplicação de redes neurais artificiais na indústria de fios de algodão: Determinação do Índice de fibras imaturas.
Revista Gestão Industrial, 11.

Coppin, B. (2004).
Artificial Intelligence Illuminated.
Jones and Bartlett illuminated series. Jones and Bartlett Publishers, 1 edition.

da Silva, D. M. (2014).
Inteligência Artificial - Slides de Aula.
IFMG - Instituto Federal de Minas Gerais, Campus Formiga.

Faceli, K., Lorena, A. C., Gama, J., and Carvalho, A. C. P. L. F. (2011).
Inteligência Artificial - Uma abordagem de Aprendizado de Máquina.
Editora LTC, 1 edition.

Goodfellow, I., Bengio, Y., and Courville, A. (2016).
Deep Learning.
MIT Press.
http://www.deeplearningbook.org.

Google Trends (2020).
Artificial neural network vs artificial intelligence.
[Online]; acessado em 25 de Agosto de 2020. Disponível em: https://trends.google.com.br/trends/explore?date=2011-09-07%202021-10-07&q=%2Fm%2F05dhw,%2Fm%2F0mkz.

Referências II

Hani M. K. (2019).
Backpropagation step by step.
[Online]; acessado em 03 de Setembro de 2020. Disponível em:
https://hmkcode.com/ai/backpropagation-step-by-step/.

Hoang Duong - Hoang Duong blog (2015).
Gradient descent and variants - convergence rate summary.
[Online]; acessado em 03 de Setembro de 2020. Disponível em:
http://hduongtrong.github.io/2015/11/23/coordinate-descent/.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012).
Imagenet classification with deep convolutional neural networks.
In Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1, NIPS'12, pages 1097–1105, Red Hook, NY, USA. Curran Associates Inc.

Laurence Goasduff - Gartner (2021).
The 4 trends that prevail on the gartner hype cycle for ai, 2021.
[Online]; acessado em 06 de Outubro de 2021. Disponível em: https://www.gartner.com/en/articles/the-4-trends-that-prevail-on-the-gartner-hype-cycle-for-ai-2021.

Lee, S. (2020).
(artificial) neural networks: From perceptron to mlp.
Disponível em: https://github.com/i-Systems/tutorial/blob/gh-pages/KIM/slides/05_ANN.pdf.

Referências III

Li, F.-F., Krishna, R., and Xu, D. (2021).
Convolutional neural networks (cnns / convnets).
[Online]; acessado em 26 de Janeiro de 2021. Disponível em:
https://cs231n.github.io/convolutional-networks/.

Matheus Facure (2017).
Redes neurais recorrentes.
[Online]; acessado em 01 de Setembro de 2020. Disponível em:
https://matheusfacure.github.io/2017/09/12/rnn/.

Montanari, T. (2016).
Histologia: texto, atlas e roteiro de aulas práticas.
Ed. da autora, 3 edition.
Disponível em: http://www.ufrgs.br/livrodehisto.

Nahua Kang - Towards Data Science (2017).
Multi-layer neural networks with sigmoid functionâ deep learning for rookies.
[Online]; acessado em 01 de Setembro de 2020. Disponível em: https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f.

Raschka, S. (2015).
Python Machine Learning, 1st Edition.
Packt Publishing, 1 edition.

Referências IV

Russel, S. and Norvig, P. (2013).
Inteligência artificial.
Campus - Elsevier, 3 edition.

Sadek Alaoui - SQLML (2017).
Convolutional neural network.
[Online]; acessado em 25 de Agosto de 2020. Disponível em:
http://sqlml.azurewebsites.net/2017/09/12/convolutional-neural-network/.

Zhang, A., Lipton, Z. C., Li, M., and Smola, A. J. (2020).
Dive into Deep Learning.
https://d2l.ai.