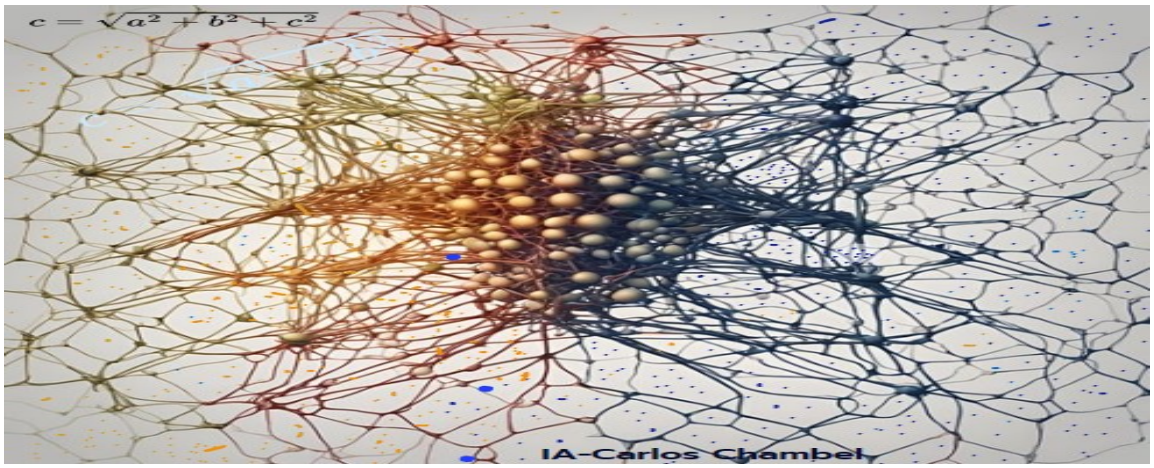
Notas sobre inteligência artificial



A inteligência artifical é uma área da ciência da computação que simula a inteligência humana num computador, dando a este a capacidade de poder processar informações e tomar decisões futuras normalmente executadas por pessoas, utilizando técnicas de Machine Learning (aprendizagem de máquina), Deep Learning (aprendizagem profunda) e muitas outras abordagens, e com a capacidade de aprenderem com os processos realizados, perceberem as diferenças e se ajustarem a elas, sem intervenção directa do ser humano.

Actualmente esta é uma realidade presente em muitas áreas da nossa vida, por exemplo, nos chatbots (assistentes virtuais) que simulam o atendimento humano, interagindo com os clientes fazendo perguntas e apresentando soluções, algumas directamente na resolução de problemas diversos, de equipamentos e instalações, marcações nas agendas de consultas, com a vantagem de estarem sempre disponíveis 7_{d}x24_{h} com o aumento da produtividade e a redução de

custos laborais e fiscais, e também nos diagnósticos de ECD, na análise de imagens médicas (radiografias e ressonâncias magnéticas), tecnologias genética, condução autónoma dos automóveis, televisores, telemóveis, acção militar(drones, mísseis, veículos autónomos de combate, escudos de defesa, simulação e planeamento, etc.), robótica, aviões, internet (por exemplo nos mecanismos de pesquisas, nas plataformas de email, comércio electrónico), etc..
A inteligência artificial tem por base o processamento e análise de grandes quantidades de dados em tempo real e nas diversas tecnologias tais como a machine learning, a deep learning, o processamento da linguagem humana (PLN), reconhecimento facial, visão artificial computacional, etc..

Assim, e cada vez mais integrada no nosso cotidiano, a inteligência artificial tem o potencial de transformar disruptivamente o modo como interagimos com o mundo e impulsionará as mudanças para o futuro em todos os elementos sociais.

Apesar do "Teste de Turing" ter sido formulado há setenta e quatro anos, durante as quatro décadas seguintes pouco se falava da inteligência artificial, devido em grande parte às limitações do wardware.

Sendo uma das forças mais poderosas da nossa era, o actual impacto da inteligência artificial na economia, no emprego, nas empresas, na área da saúde, do ensino, da cultura, etc. é, nesta data, a ponta do iceberg do que está ainda por vir.

Esta recente explosão no desenvolvimento da inteligência artificial deve-se sobretudo a três factores:

1. O enorme incremento nos dados disponíveis e capacidades de armazenamento;
2. A evolução do hardware com CPU´s com mais núcleos e velocidade (por exemplo I9-12900 com 16 núcleos, 24 threads e clockbase 2,4Ghz) , a disponibilidade de GPU´s com muitas centenas de núcleos (por exemplo, a placa gráfica RTX3050-8G com 2560 núcleos) que possibilitam ainda mais o processamento paralelo, sistemas de armazenamento com tecnologias SSD NVME, etc.;
3. O desenvolvimento de novas linguagens de programação (Python, Java, C#, etc.) SDK´s (software development kit) e outros recursos de desenvolvimento, muito deles disponíveis em opensource.

Uma das muitas dificuldades em entender a inteligência artificial reside no alto nível de complexidade dos algoritmos de machine learning, o que só é possível com a ajuda de ferramentas de abstração, isto é, de matemática, mais concretamente de álgebra linear, cálculo diferencial, cálculo integral e estatística. O exemplo clássico é a compreensão das dimensões espaciais: é fácil entender uma forma de duas ou três dimensões, isto é, uma recta ou um cubo, respectivamente. Mas acrescentar mais uma dimensão (não uma dimensão "temporal"), passando para quatro dimensões,

provavelmente estará muito para além da capacidades de análise do comum dos mortais... Entretanto, num simples algoritmo de machine learning pode ser necessário saber "caminhar" num sistema com mais de cem ou mil dimensões.

Inteligência Artificial, Machine Learning e Deep Learning

Machine Learning é um sub-conjunto da Inteligência Artificial e Deep Learning um sub-conjunto de Machine Learning.

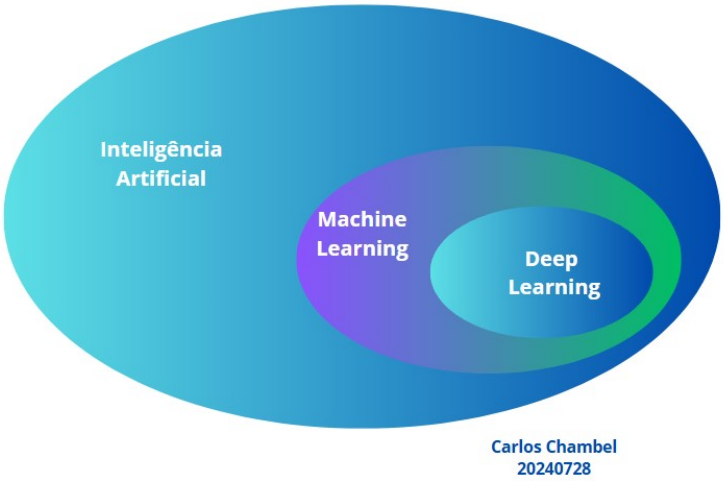


Fig.1

Rede neural artificial vs cérebro humano

O cérebro humano tem cerca de 171000000000000 neurónios e células coadjuvantes e cada neurório tem cerca de 10000 sinapses.

Apenas só os neurónios, dá um total de cerca de 86000000000000000 circuitos e, ainda, considerando a sua neuroplasticidade, potenciais químicos e eléctricos, etc., temos uma complexidade impossível de reproduzir em

qualquer máquina criada pelo homem e para o seu mapeamento seriam necessárias centenas de anos de investigação.

O neurónio de uma rede neural artificial inspira-se no neurónio biológico, conforme figura abaixo.
A camada de entrada, a função soma e a função de activação, e a camada de saída, têm a sua analogia com os dendritos, corpo celular, axónios e sinapses de um neurónio biológico, respectivamente.

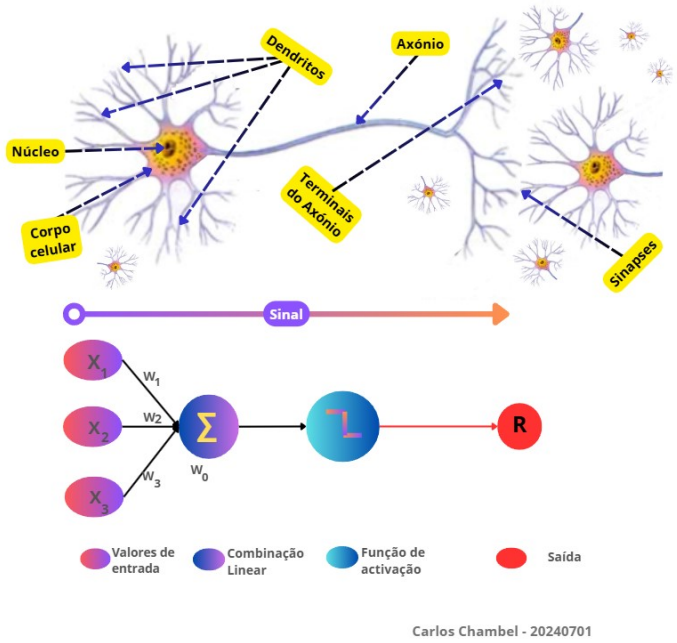


Fig.2

Machine Learning

Machine Learning (aprendizagem de máquina) é simplesmente um subconjunto da inteligência artificial. É a matemática e estatística utilizada nas ciências de

programação dos computadores para se conseguir o processamento de informações e tomada de decisão a partir de dados inicialmente fornecidos, sem codificar as regras em máquinas e programas para dar resultados finais. Para tal são necessários:

- Dados de entrada, em quantidade, quantos mais melhor, por exemplo, números, textos, sons, imagens, etc.
- Exemplos de saída do que se espera obter: sons gerados por humanos, animais, etc, fotos de objectos esperados, etc.
- Processo de medição e aferição do algoritmo, isto é, a distância entre a saída actual do algoritmo e da sua saída esperada.
- Realimentar periodicamente o modelo com a introdução de novos dados, obtendo valores para retreinar o mesmo.

Um sistema de programação clássico entra com a informação e as regra, faz o processamento e com a saída das respostas.

Fig.3

Um sistema de machine learning é treinado em vez de explicitamente programado, e faz o processamento dessas

informações directamente a partir dos dados relevantes para uma tarefa conseguindo apresentar uma estrutura lógica nos mesmos e encontrar representações úteis e regras com uma orientação a partir de um sinal de feedback.

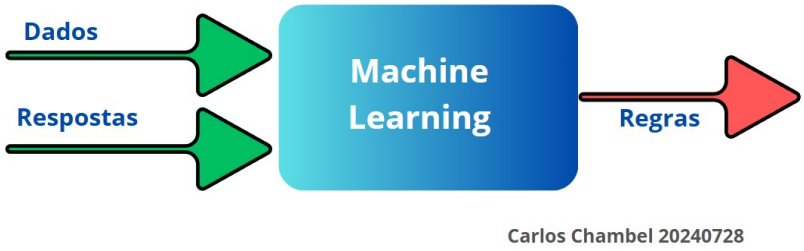


Fig.4

No Machine Learning existem diversas abordagens para o tratamento dos dados:

1. Aprendizagem supervisionada. Neste, o sistema é treinado com dados de entrada e respectivos rótulos, tendo cada um uma resposta associada. O sistema aprende a partir desses exemplos e do mapeamento correspondente e é capaz de realizar tarefas semelhantes quando em presença de novos dados;
2. Aprendizagem não supervisionada. Neste, o sistema é exposto a conjuntos de dados não rotulados e deve encontrar padrões e estruturas dentro desses dados, agrupando-os ou descobrindo relações entre eles.
3. Aprendizagem por reforço. Treina os computadores por tentativa e erro aplicando um sistema de recompensas (pontos) para alcançar a melhor recompensa final.

Redes neurais artificiais

Rede neural artificial é uma referência à neurobiologia com uma inspiração nas redes neurais biológicas dos animais, muito embora se deva afirmar que os mecanismos dos modelos de aprendizagem profunda não são implementados como algo parecido com a complexidade existente nos milhões de neurónios e sinapses do cérebro humano, com outras células de suporte, os processos químicos e eléctricos, etc.

Uma rede neural artificial é um conjunto de neurónios, ou nós artificiais conectados em camadas, com os respectivos *pesos* e *bias*, cada um dos quais processa um conjunto de entradas e produz uma saída que, por sua vez, é enviada a outros neurónios da camada seguinte, se o valor dessa saída se encontrar acima de um valor limiar específico. Se não atingir esse limiar, nenhuma informação desse neurónio é passada para a próxima camada.

Existem diversos tipos de redes neurais artificais, consoante a aplicação que se pretende, por exemplo classificação e reconhecimento de rostos de pessoas, criação de histórias artificiais, detecção e classificação de objectos, geração de novas imagens, aprendizagem de padrões, o processamento de linguagem natural, etc..

Alguns exemplos de arquitecturas de redes neurais articiais utilizadas actualmente são as convolucionais, as profundas, as generativas, as recorrentes, etc..

As redes neurais artificiais podem ser de diversos tipos, consoante as tarefas para que são projectadas, por exemplo:

1. *redes neurais artificiais de alimentação directa feedforward* (FNNs) ou perceptrons de múltiplas camadas (MLPs), quando a informação é alimentada sempre numa direcção, da entrada para a saída, sem loops. São redes neurais de neurónios sigmóides;
2. redes neurais artificiais recorrentes (RNNs) que são bidireccionais, sendo possível usar loops de feedback, em que a saída de alguns neurónios afecta a entrada dos mesmos e estimula outros neurónios;
3. redes neurais artificiais convolucionais (CNNs) onde as camadas ocultas executam funções matemáticas, álgebra linear, consoante o objectivo da rede, utilizadas em visão computacional, detecção de obtectos, etc.

Perceptron

O Percepton é uma estrutura fundamental de construção de uma rede neural artificial. Tem várias entradas e pesos que devem ser aplicados a essas entradas e utiliza uma função de activação para assim produzir uma saída binária.

Um perceptron é uma rede neural artificial com uma camada única. Uma rede neural artificial pode ter milhões destes neurónios e milhões dos respectivos *pesos* e *bias*.

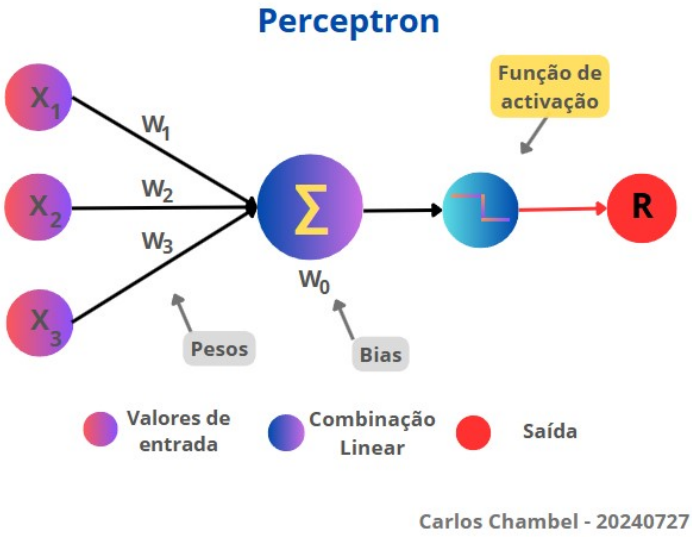


Fig.5

$$R = \begin{cases} 0, & (\sum_{i=1}^{n} w_i x_i \leq z), \\ 1, & (\sum_{i=1}^{n} w_i x_i > z) \end{cases}$$

Sendo z o valor limite (threshold)

Pesos: São valores numéricos associados às conexões entre os neurónios e determinam a força que essas conexões têm na saída de um neurónio sobre a entrada de outro neurónio. São ajustados iteractivamente para minimizar a diferença entre os valores da rede e os resultados reais. Quanto maior é o peso, mais importante é o respectivo sinal.

Bias: São valores acrescentados ao somatório dos produtos dos valores de cada neurónio pelo respectivo peso, com o objectivo de atingir as compensações essenciais para a introdução de flexibilidade no processo de aprendizagem, ajustando a facilidade com que um neurónio é activado, isto é uma saída igual a um.

A camada de entrada são os dados que alimentam a rede neural, um vector:

$$X = \begin{bmatrix} x_1 \\ . \\ . \\ . \\ x_m \end{bmatrix}$$

E os pesos, um vector:

$$W = \begin{bmatrix} w_1 \\ . \\ . \\ . \\ w_m \end{bmatrix}$$

A função soma ponderada faz o cálculo das entradas com os respectivos pesos

$$soma = \sum_{i=1}^{k} x_i w_i$$

O *Bias* associado a cada neurónio é adicionado a esta soma

$$w_0 + \sum_{i=1}^{k} x_i w_i$$

A função de activação aceita o resultado da soma ponderada depois de adicionado o *bias*.

$$Y = w_0 + \sum_{i=1}^{k} x_i w_i$$

Ou, em forma matricial, o produto e soma das matrizes:

$$\begin{bmatrix} x_{11} \dots x_{1m} \\ x_{21} \dots x_{2m} \\ .. \\ .. \\ .. \\ x_{n1} \dots x_{nm} \end{bmatrix} * \begin{bmatrix} w_0 \\ w_1 \\ .. \\ .. \\ .. \\ w_m \end{bmatrix} + \begin{bmatrix} \varepsilon_0 \\ \varepsilon_1 \\ .. \\ .. \\ .. \\ \varepsilon_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ .. \\ .. \\ .. \\ y_n \end{bmatrix}$$

A saída de uma camada torna-se a entrada da camada seguinte, repetindo-se o processo até à camada final, produzindo assim a previsão do sistema

Funções de activação em Redes neurais

A *função de activação* em redes neurais ou *função de transferência*, são funções matemáticas aplicadas na saída de um neurónio (ou camadas de neurónios) determinando o nível de activação desse neurónio, isto é, transformar a entrada ponderada somada num valor de saída que vai

alimentar a próxima camada ou como saída. Determina se uma unidade neural deve ser activada ou não.

Alguns exemplos:

1-activação linear

$$y = f(x)$$

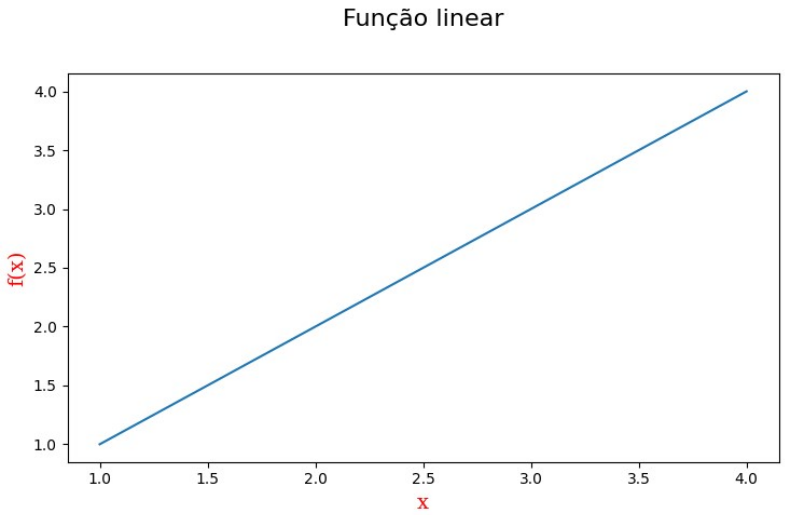


Fig.6 Carlos Chambel/20240723

Na regressão linear simples a função linear é definida por $y=ax+b+\varepsilon$ onde a e b são coeficientes desconhecidos,

a = declive da recta,

b = ponto de intersecção da recta com o eixo Y, e

ε = termo do erro .

2-Função de activação Sigmoid

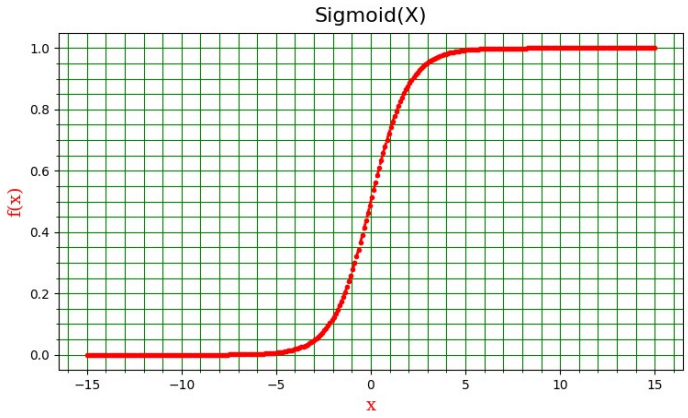


Fig.7 Carlos Chambel/20240723

É uma função de activação de redes neurais artificiais não-linear, com a qual é possível processar o modelo com retropropagação e empilhamento de várias camadas de neurónios.
Normalmente é usada para modelos para prever a probabilidade como uma saída entre zero (não activação) e um (activação).

Entradas muito negativas terminam próximas de zero, e entradas muito positivas terminam próximas de um.

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

ou, mais explicitamente:

$$y = \frac{1}{1 + e^{(-\sum w_j \ x_j - b)}}$$

O contradomínio da função é o intervalo]0,1[,isto é, não é zero ou um, mas quaisquer valor real entre zero e um.

É diferenciável e a sua derivada é:

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

Exemplo de programação para o cálculo da função sigmoid:

```
"""
        Carlos Chambel - 20240714
      machine learning - função sigmoid
              Python 3.11
"""
# fig. 5
import numpy as np
1 usage
def funcao_sigmoid(soma):
    return 1.0 / (1.0 + np.exp(-1 * soma))
x1 = 0.8
x2 = 0.6
x3 = 0.9
w1 = 2.1
w2 = .1
w3 = .31
b0 = 0.071
cS = x1 + w1 + x2 * w2 + x3 * w3 + b0
print("resultado da função: ", funcao_sigmoid(cS))
```

Fig.8

```
resultado da função:  0.9647702807940787

Process finished with exit code 0
```

Fig.9

Fig.10 Carlos Chambel/20240729

Na função de activação ReLu, todos os valores negativos de entrada são substituídos por zero e os não negativos permanecem inalterados

$$R(z) = \max(0, z)$$

$$R'(z) = \begin{cases} 1, se..z \geq 0 \\ 0, se..z < 0 \end{cases}$$

Exemplo com a biblioteca *PyTorch*:

```
"""
        Carlos Chambel - 20240523
          machine learning - função ReLu
              Python 3.11
"""
# fig. 3


import ...
icch = torch.tensor([[5., -4., 1., 3.],
                     [6., 2., 8., -3.],
                     [3., 11., -5., 11.],
                     [-177., 0., 24., 2.]])
print("Inicial")
print(icch)
Ch = torch.nn.ReLU()
Occh = Ch(icch)
print("Final ")
print(Occh)
```

```
Inicial
tensor([[   5.,   -4.,    1.,    3.],
        [   6.,    2.,    8.,   -3.],
        [   3.,   11.,   -5.,   11.],
        [-177.,    0.,   24.,    2.]])
Final
tensor([[ 5.,  0.,  1.,  3.],
        [ 6.,  2.,  8.,  0.],
        [ 3., 11.,  0., 11.],
        [ 0.,  0., 24.,  2.]])

Process finished with exit code 0
```

Fig.11

Quando se activa uma destas funções pretende-se encontrar valores diferentes para os *pesos* e *bias* por forma a minimizar a diferença entre os valores de entrada e os valores calculados na saída.

Função de perda

O cálculo do erro numa rede neural artificial é o processo fundamental para treinar a rede e determinar a precisão da correcção. A função de perda é uma função para tratar a diferença entre os valores previstos e os obtidos pela rede actuando nos parâmetros do sistema até que essa diferença seja o menor possível. O tipo de função a aplicar depende dos objectivos da implementação da rede neural.
Existem muitos processos para calcular o erro de uma rede neural artificial. Conforme o problema que está a ser analisado assim o processo do cálculo do erro deve ser tratado com a metodologia adequada. O erro é sempre positivo.

Erro = diferença simples entre os valores previstos e os obtidos

A perda geral é dada pela expressão:

$$L = \sum_{i=1}^{n} L_n$$

n = número de épocas

L_n = perda na época *n*

O objectivo é encontrar pesos e bias por forma a que $C(w,b) \approx 0$.

Cálculo do erro

Exemplos de métricas para o cálculo que minimiza o erro:

Erro quadrático médio (MSE):

$$C(w,b) = \frac{1}{2n} \sum_{i=1}^{n} (y_i - \lambda_i)^2$$

Onde w são os pesos da rede, *b são os bias,* y é o valor real e λ o valor da previsão da rede e n o número total de entradas de treinamento

Erro raiz quadrada do quadrático médio (RMSE):

$$C(w,b) = \sqrt{\frac{1}{2n} \sum_{i=1}^{n} (y_i - \lambda_i)^2}$$

Onde w são os pesos da rede, *b são os bias,* y é o valor real e λ o valor da previsão da rede e n o número total de entradas de treinamento

Descida de Gradiente

É uma técnica fundamental utilizada para uma dada função, de forma optimizada o seu gradiente, calcular o seu mínimo global, independentemente de na mesma função existirem muitos mínimos locais. No exemplo abaixo, o objectivo é ir movendo o ponto da função de duas variáveis *V1,V2* na direcção em que *K* alcance o seu mínimo. A inclinação da linha determina a direcção do ponto. Para inclinações negativas devem-se aumentar os pesos e vice-versa.

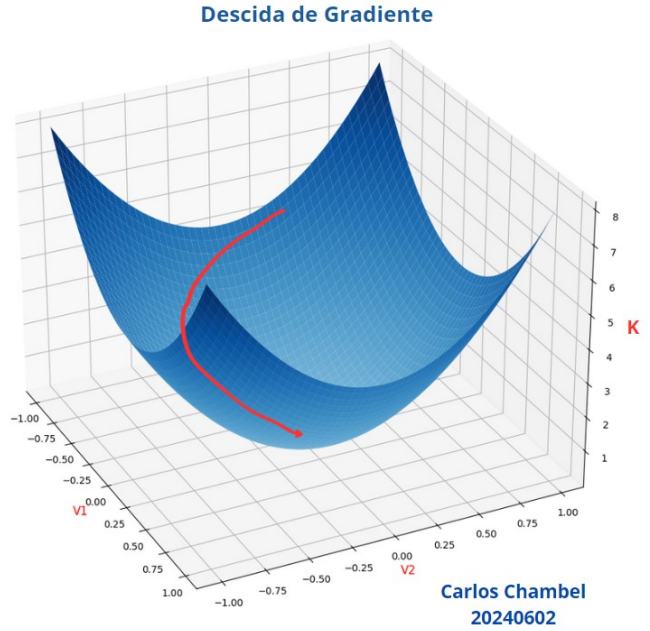


Fig.12

Taxa de aprendizagem

É um conceito fundamental nesta área, e representa a capacidade de optimização do sistema ajustando automaticamente, ou através de outra técnica qualquer, os valores da taxa durante o treino das redes neurais.

Aprendizagem Profunda
(Deep Learning)

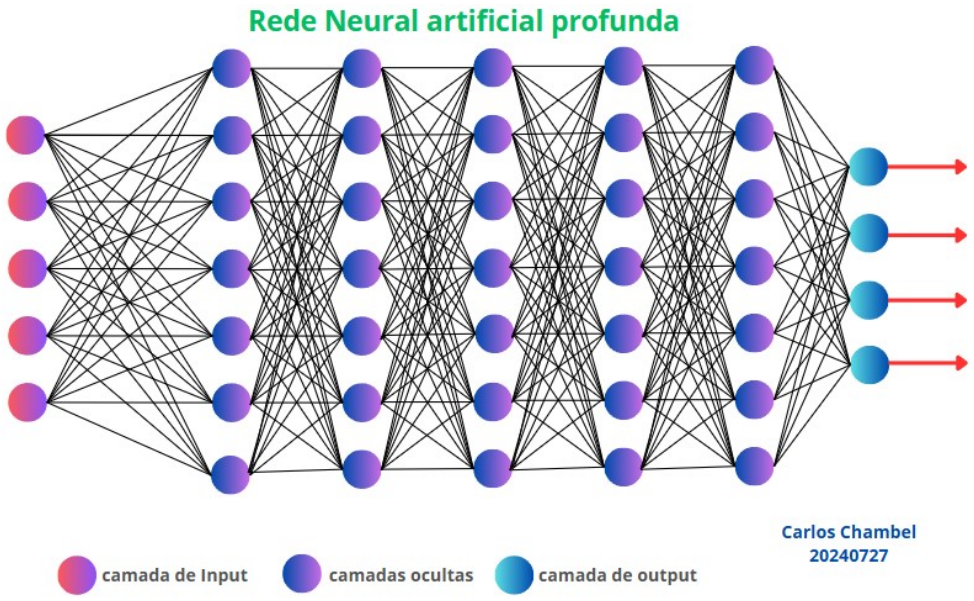


Fig. 13

A aprendizagem profunda (Deep Learning) é um subconjunto de machine learning que usa redes neurais artificiais com muitas camadas para processar grandes quantidades de dados não processados (quanto mais dados, melhor). "Profunda" não significa nada de profundo, mas uma estrutura matemática para os computadores aprenderem, sem serem explicitamente programados, representações a partir de dados e a profundidade do modelo é o número das sucessivas camadas de representações que constituem o sistema.

Elementos da rede neural artificial:

1. Camada de entrada: Esta camada recebe os dados que alimentam a rede. Nesta fase não é realizado qualquer cálculo.

2. Camadas ocultas: Executam todos os tipos de computação sobre a informação inserida na entrada e transfere o resultado para a camada de saída.
3. Camada de saída: Executa uma função de activação, diferente das usadas nas camadas ocultas, e apresenta o valor final como resultado.

Exemplos de aplicações construídas com Deep Learning:

1. Visão computacional (viaturas autónomas, reconhecimento facial, imagiologia médica, biometria, detecção e identificação de objectos, etc.)
2. Inteligência artificial generativa (geração de texto, novos conteúdos, etc.)
3. Automação (gerir espaços/obtectos, treinar robots, jogos, etc)
4. Medicina(análise MADs, equipamentos, detecção doenças, etc)
5. Processamento da linguagem natural-NLP (reconhecimento de fala, entender, decifrar, tradução, conversão de texto para fala, geração de texto pata texto, etc.)

No processo de aprendizagem pretende-se ajustar automaticamente os pesos na rede neural artificial, sem qualquer intervenção humana. Como é que a rede neural artificial aprende? Por tentativa e erro, partindo de uma previsão, verifica se essa previsão é muito alta ou muito baixa, vai ajustando os pesos, para mais ou para menos, até atingir com mais precisão o que está na entrada.

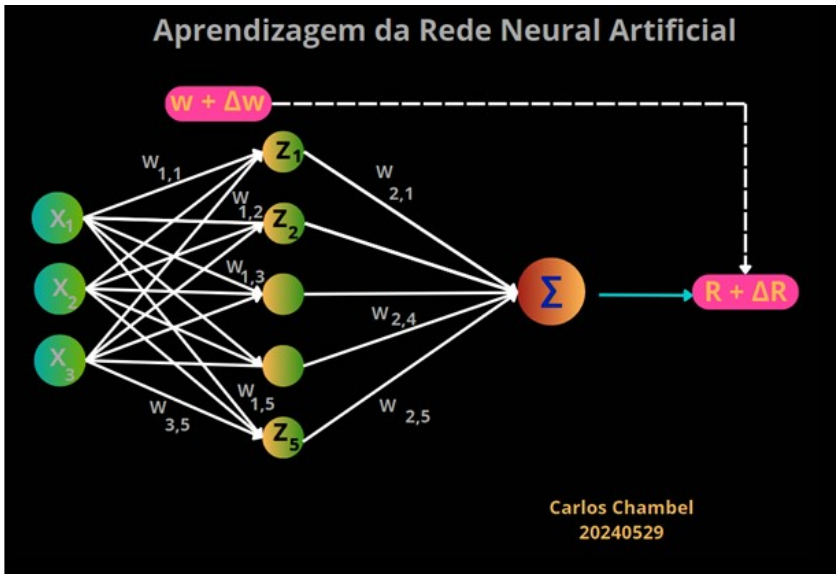


Fig.14

O objectivo do algoritmo de apendizagem é determinar quais são os melhores valores para os pesos, por forma a minimizar ao máximo a perda global do modelo.

Exemplo de um algoritmo de uma rede neural artificial densa com uma camada oculta, com a activação aplicando a função sigmoide:

Épocas (n)

1º Propagação (para a frente)

1. aplicar a função soma
2. aplicar a função de activação
3. cálculo do erros – função perda
4. cálculo do delta de saida

2º Retropropagação (para trás)

1.cálculo da derivada da perda
2.Actualização dos pesos

Fim

Exemplo de algoritmo propagação – cálculo da camada de entrada até à camada de saída

Inicializar pesos com valores aleatórios,pequenos e distintos
Inicializar valor número de épocas
Inicializar valor de bias igual a zero
Inicializar valor de limiar de activação (threshold)
Inserir valores reais esperados para cada neurónio
Inserir valor taxa de aprendizagem $m \in [0,1]$

- *Para cada época*
  - *Até que o erro> limiar do erro*
    - *Para cada neurónio da camada oculta*
      - *Para cada neurónio da camada de entrada*
        - *Função Soma:*
          - *multiplicar o valor dessa entrada pelo peso respectivo*
          - *Acumular os produtos*
        - *Função de Activação:*
          - *Adicionar bias*
          - *Calcular o valor de activação do neurónio corrente aplicando a função de activação sigmoide à soma dos produtos e depois de adicionado o bias.*
    - *Fim*
    - *Para o neurónio de saída*
      - *Para cada neurónio da camada oculta*
        - *Função Soma:*
          - *multiplicar o valor de activação pelo peso respectivo*
          - *Acumular os produtos*
        - *Função de Activação:*
          - *Adicionar Bias*
          - *Calcular o valor da previsão da rede neural aplicando a função de activação sigmoide à soma dos produtos e depois de adicionado o bias.*
        - *Calcular o erro:*
          - *erro=valor esperado - valor da previsão da rede neural*
          - *Delta = derivada da função sigmoide do valor da previsão da rede neural*
          - *DeltaSaída = erro * Delta*

- *Fim*
- *Para cada neurónio da camada oculta*
    - *Delta = derivada da função sigmoide do valor da função sigmoide calculado anteriormente para o neurónio corrente*
    - *DeltaCamadaOculta = Delta * DeltaSaída * peso*
- *Fim*
- *Para cada neurónio da camada oculta*
    - *DeltaCamadaOculta = Valor do neurónio corrente * DeltaCamadaSaída*
    - *Novo valor do peso do neurónio corrente = valor do peso do neurónio corrente * momento + entrada * DeltaCamadaOculta * taxa de aprendizagem*
- *Fim*
- *Para cada neurónio da camada de entrada*
    - *Para cada neurónio da camada oculta*
        - *DeltaCamadaEntrada = Valor do neurónio corrente * DeltaCamadaOculta*
        - *Novo valor do peso do neurónio corrente = valor do peso do neurónio corrente * momento + entrada * DeltaCamadaOculta * taxa de aprendizagem*
    - *Fim*
- *Fim*
- *Fim*
- *Fim*

Treinamento da rede neural artificial

No exemplo da fig 13, considerando a activação com a função Sigmoid, temos cinco neurónios de entrada, mais cinco camadas ocultas de sete neurónios cada, e quatro neurónios de saída. O total de *pesos* e *bias* , seria:

5x7+7x7+7x7+7x7+7x7+7x4 +7+7+7+7+7+5 = 284

os quais podemos alterar para que a rede se comporte como se pretende.

No treinamento, o sistema ajusta os pesos, bias e taxa de aprendizagem para minimizar o erro entre os valores previstos e os parâmetros correctos nos dados de treinamento.

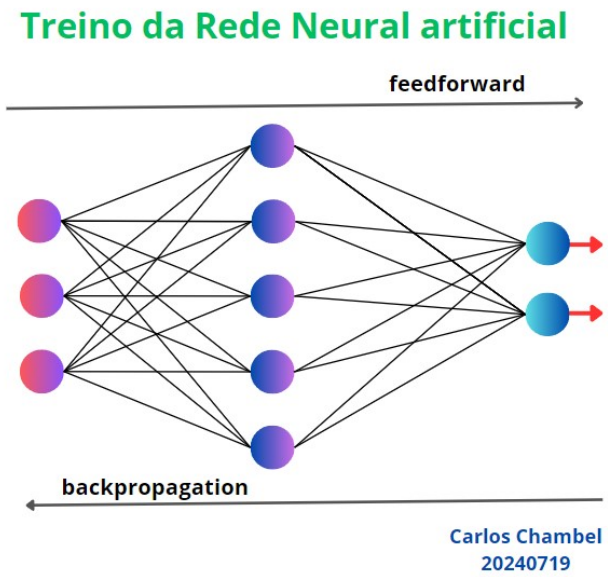


Fig. 15

Implementação de uma rede neural

Linguagem de programação: *Python*

Frameworks: *TensorFlow , PyTorch*, etc.

Exemplo:
Criação de uma rede neural com o recurso à linguagem de programação Puython (versão 3.11) e bibliotecas TensorFlow (versão 2.16.1) , Numpy (versão 1.26.4), etc.

1. Importar bibliotecas:

```
"""
        Carlos Chambel - 20240722
        Redes neurais artificiais
            Python 3.11
            TensorFlow 2.16.1
            Numpy 1.26.4
            Matplotlib  4.8.4

"""
"""..."""

import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt
```

2. Importar uma base de dados e dados de teste:

```
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
```

3. Preparação dados para a camada de entrada:

```
train_images = train_images / 255.0
test_images = test_images / 255.0
```

4. Definição do modelo da rede neural, criar primeira camada e camada oculta com 128 neurónios e camada de saída com dez neurónios com as respectivas funções de activação

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])
```

5. Compilar modelo

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

6. Treinar modelo com doze épocas

```
45    model.fit(x_train, y_train, epochs=epocas)
```

```
Epoch 1/12
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 720us/step - accuracy: 0.8655 - loss: 0.4792
Epoch 2/12
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 1s 709us/step - accuracy: 0.9553 - loss: 0.1536
Epoch 3/12
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 1s 766us/step - accuracy: 0.9676 - loss: 0.1082
Epoch 4/12
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 1s 730us/step - accuracy: 0.9723 - loss: 0.0875
Epoch 5/12
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 1s 729us/step - accuracy: 0.9768 - loss: 0.0757
Epoch 6/12
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 1s 706us/step - accuracy: 0.9804 - loss: 0.0644
Epoch 7/12
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 1s 710us/step - accuracy: 0.9810 - loss: 0.0581
Epoch 8/12
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 1s 709us/step - accuracy: 0.9838 - loss: 0.0488
Epoch 9/12
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 1s 710us/step - accuracy: 0.9843 - loss: 0.0484
Epoch 10/12
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 1s 707us/step - accuracy: 0.9850 - loss: 0.0452
Epoch 11/12
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 1s 705us/step - accuracy: 0.9872 - loss: 0.0390
Epoch 12/12
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 1s 705us/step - accuracy: 0.9875 - loss: 0.0380
```

7. Medir a precisão (accuracy) e perdas

```
53    score = model.evaluate(x_test,  y_test, verbose=2)
54    print("perdas teste: ", score[0])
55    print("precisão teste: ", score[1])
```

```
perdas teste:  0.0710158720612526
precisão teste:  0.9817000031471252
```

8. Salvar o modelo

```
61    model.save('Chambel_model_1.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()`

Process finished with exit code 0
```