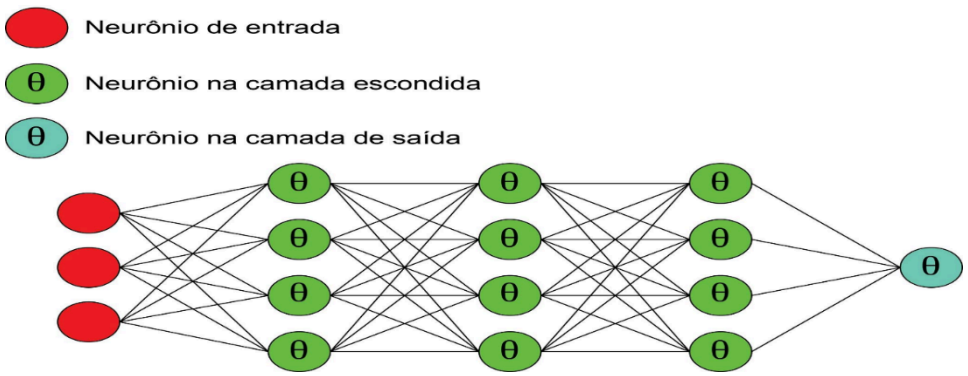
Praticando Redes Neurais

As redes neurais artificiais são um modelo de aprendizagem de máquina inspirado nas redes neurais biológicas. Na década de 1980 houve um ressurgimento na pesquisa de redes neurais. Uma rede perceptron em múltiplas camadas (MLP) é uma classe de redes sem ciclos ou direta (*feed forward*). É composta de pelo menos 3 camadas de perceptrons: uma de entrada, uma escondida (ou oculta) e uma de saída. Exceto os nós de entrada, cada nó é um neurônio que usa uma função de ativação não-linear.

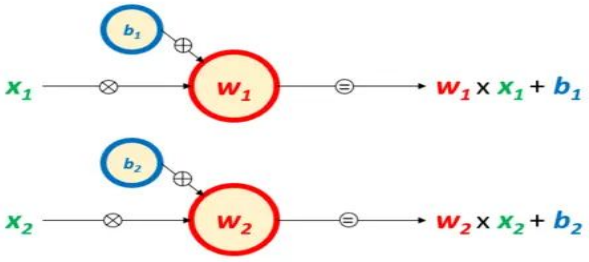
MLP usa uma técnica para treinamento chamada *backpropagation*; ela consegue distinguir dados não linearmente separáveis.

As redes MLP são formadas por:

- Camadas (layers), que possuem:
  - Quantidade k>0 de neurônios
  - Uma função de ativação θ, a qual é aplicada para cada neurônio da camada.
- Cada conexão entre 2 neurônios tem um valor chamado peso (*weight*) (W). Cada conexão é ilustrada por uma aresta.



As redes neurais promovem transformações matemáticas nos dados que recebem para processar. Em cada neurônio de cada camada, elas multiplicam o valor de entrada pelo peso do neurônio correspondente, somam com o bias, b, associado ao neurônio (valor que não depende da entrada), e passam esse valor adiante.



Essas operações são lineares, de forma que, por mais complexa que seja a rede neural, ela só pode captar relações lineares entre as variáveis de entrada e a variável de saída.

Para torná-las capazes de modelar também relações não-lineares, os resultados de saída de cada camada passaram a ser processados pelas chamadas funções de ativação.

Isso é especialmente importante nas camadas escondidas da rede neural. Nas camadas de saída, as funções de ativação podem ter finalidades específicas, dependendo do problema que a rede neural está tentando resolver.

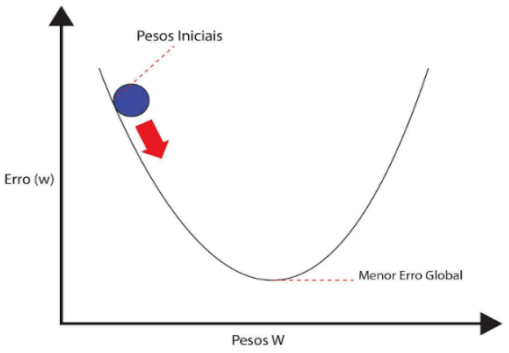
Mas, qual função de ativação usar? Pelo seu desempenho, a função de ativação mais utilizada nas camadas escondidas é a *ReLU*, tendo a preferência na abordagem inicial do problema. Quando ela não é eficiente, pode-se recorrer às outras funções, considerando-se caso a caso.

Para as camadas de saída, podemos usar a função adequada à natureza do problema a ser resolvido:

- classificação binária, podemos recorrer à função sigmóide, com a tangente hiperbólica também sendo uma possível solução;
- classificação não-binária, podemos aplicar a função *softmax*;
- regressão, usamos a ativação linear.

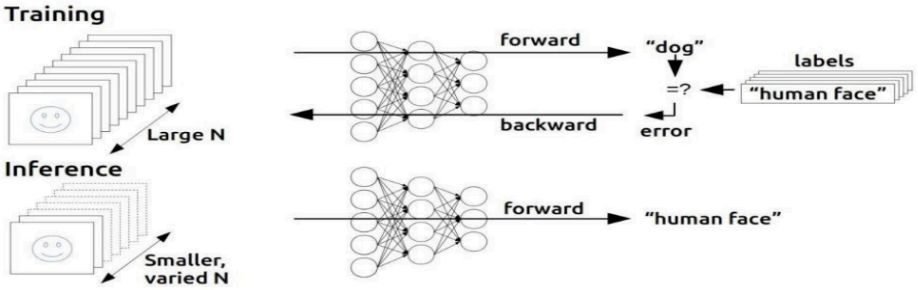
Treinamento

Cada conexão entre neurônios representa um peso. O treinamento de redes neurais envolve descobrir valores de pesos que minimizam o erro.



Mesmo para um modelo com poucos neurônios, há infinitas permutações possíveis.

Os algoritmos mais eficientes para ajustar os pesos de um modelo de rede neural utilizam a ideia do gradiente descendente. A função que calcula o erro do modelo calcula a diferença entre a saída esperada e a saída atual. A derivada de uma função de custo permite calcular o gradiente para cada peso W, o que permite ajustá-los para um valor que minimiza o erro.



Experimentos: Veja
https://colab.research.google.com/drive/13pc4OMAnWNO9b1NAMteLiBmeR4a4-tjF

1) Código de um único Perceptron em python puro:

```
#  Treino (função booleana OU):
#    a b    saída
#    0 0    0
#    0 1    1
#    1 0    1
#  Teste:
#    1 1    1

from numpy import exp, array, random, dot
import numpy as np

entradas_treinamento = array([[0, 0], [0, 1], [1, 0]])
saidas_treinamento = array([[0, 1, 1]]).T  #determina a matriz transposta

random.seed(1)

pesos_sinapticos = 2 * random.random((2, 1)) - 1 #pesos aleatórios entre -1 e 1

for epoca in range(100):
    saida = 1 / (1 + exp(-(dot(entradas_treinamento, pesos_sinapticos)))) #saídas do neurônio
    mudanca_pesos = dot(entradas_treinamento.T, (saidas_treinamento - saida) * saida * (1 - said
a))
    pesos_sinapticos += mudanca_pesos
    print('Mudança nos pesos:', np.linalg.norm(mudanca_pesos))

#Testando com 1 e 1, esperando que dê 1 na saída
print (1 / (1 + exp(-(dot(array([1, 1]), pesos_sinapticos)))))
```

Atividade:

- A previsão relacionada ao ponto (1,1) está correta?

- Testem a função booleana XOR (OU-EXCLUSIVO):

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

O perceptron consegue fazer previsões sobre esta função? Por que é assim?

2) Código de um único Perceptron em Python puro:

Usando a técnica de aprendizado supervisionado, seja o seguinte problema: fazer com que o programa identifique se um determinado animal é bípede ou quadrúpede. Caso a resposta da rede não seja a esperada como resposta, um ajuste de pesos é realizado de forma que o algoritmo consiga encontrar um conjunto de pesos ideais para que assim a rede classifique corretamente as entradas (**Teorema de Convergência do Perceptron de Rosenblatt**).

Para isso adotamos os seguintes padrões:

Saídas:

- quadrúpede = 1
- bípede = -1

Vetores de treinamento

- Cão [1 -1 1 1]
- Gato [1 1 1 1]
- Cavalo [1 1 -1 1]
- Homem [-1 -1 -1 1]

E a saída esperada para cada animal é:

- Cão = 1 (quadrúpede)
- Gato = 1 (quadrúpede)
- Cavalo = 1 (quadrúpede)
- Homem = -1 (bípede)

Atribuímos o valor zero a todos os pesos e bias (b) e o valor um para o threshold (valor limiar) e taxa de aprendizado.

```
# aplicativo para verificar se o ser vivo eh quadrupede ou bipede
# quadrupede = 1, bipede = -1
# cao = [-1,-1,1,1] | resposta = 1
# gato = [1,1,1,1] | resposta = 1
# cavalo = [1,1,-1,1] | resposta = 1
# homem = [-1,-1,-1,1] | resposta = -1

# pesos (sinapses)
w = [0,0,0,0]
# entradas
x = [[-1,-1,1,1],
     [1,1,1,1],
     [1,1,-1,1],
     [-1,-1,-1,1]]
# respostas esperadas
t = [1,1,1,-1]
# bias (ajuste fino)
b = 0
#saida
y = 0
# numero maximo de interacoes
max_int = 10
# taxa de aprendizado
taxa_aprendizado = 1
#soma
soma = 0
#theshold
threshold = 1
# nome do animal
animal = ""
# resposta = acerto ou falha
resposta = ""

# dicionario de dados
d = {'-1,-1,1,1' : 'cao',
     '1,1,1,1' : 'gato',
     '1,1,-1,1' : 'cavalo',
     '-1,-1,-1,1' : 'homem' }

print("Treinando")

# funcao para converter listas em strings
def listToString(list):
    s = str(list).strip('[]')
```

```
    s = s.replace(' ', '')
    return s

# inicio do algoritmo
for k in range(1,max_int):
    acertos = 0
    print("INTERACAO "+str(k)+"-------------------------")
    for i in range(0,len(x)):
        soma = 0

        # pega o nome do animal no dicionário
        if listToString(x[i]) in d:
            animal = d[listToString(x[i])]
        else:
            animal = ""

        # para calcular a saida do perceptron, cada entrada de x eh multiplicada
        # pelo seu peso w correspondente
        for j in range(0,len(x[i])):
            soma += x[i][j] * w[j]

        # a saida eh igual a adicao do bias com a soma anterior
        y_in = b + soma
        #print("y_in = ",str(y_in))

        # funcao de saida eh determinada pelo threshold
        if y_in > threshold:
            y = 1
        elif y_in >= -threshold and y_in <= threshold:
            y = 0
        else:
            y = -1

        # atualiza os pesos caso a saida nao corresponda ao valor esperado
        if y == t[i]:
            acertos+=1
            resposta = "acerto"
        else:
            for j in range (0,len(w)):
                w[j] = w[j] + (taxa_aprendizado * t[i] * x[i][j])
            b = b + taxa_aprendizado * t[i]
            resposta = "Falha - Peso atualizado"

        #imprime a resposta
        if y == 1:
            print(animal+" = quadrupede = "+resposta)
        elif y == 0:
            print(animal+" = padrao nao identificado = "+resposta)
        elif y == -1:
            print(animal+" = bipede = "+resposta)

    if acertos == len(x):
        print("Funcionalidade aprendida com "+str(k)+" interacoes")
        break;
    print("")
print("Finalizado")
```

3) Exemplo básico de MLP (Multi-layer Perceptron) usando a biblioteca Sklearn:

```
from sklearn.neural_network import MLPClassifier

X = [[0., 0.], [0., 1.], [1., 0.]]
y = [0., 1., 1.]

clf = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(2), random_state=1)
clf.fit(X, y) #treino

clf.predict([[1.,1.]]) #aplicação/teste
```

Atividade:

- A previsão relacionada ao ponto (1,1) está correta?

- Testem a função booleana XOR (OU-EXCLUSIVO). A rede MLP consegue fazer previsões sobre esta função?

4) MLP usando Sklearn com a base IRIS:

```
import numpy as np
import pandas as pd
import seaborn as sns # visualization
from sklearn.neural_network import MLPClassifier # neural network
from sklearn.datasets import load_iris

iris = load_iris()

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.25)

#transformando a base em um dataframe def sklearn_to_df(sklearn_dataset):
df_iris = pd.DataFrame(iris.data, columns=iris.feature_names)
df_iris['target'] = pd.Series([iris['target_names'][iris['target'][i]] for i in range(150)])

sns.pairplot( data=df_iris, vars=('sepal length (cm)','sepal width (cm)','petal length (cm)','petal width (cm)'), hue='target' )
df_iris.describe()

clf = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(3, 3), random_state=1)
clf.fit(X_train, y_train)
prediction = clf.predict(X_test)

from sklearn.metrics import classification_report

print('Relatório:\n',classification_report(y_test,prediction))
print("\nMatriz de confusão detalhada:\n",
      pd.crosstab(y_test, prediction, rownames=['Real'], colnames=['Predito'],
      margins=True, margins_name='Todos'))
```

Teste:

https://colab.research.google.com/drive/13pc4OMAnWNO9b1NAMteLiBmeR4a4-tjF

Atividade:

- Analise o relatório de avaliação. A rede trabalhou adequadamente?

- Teste a rede MLP com a base CENSUS - Kaggle