```
# -*- coding: utf-8 -*-
"""
@author: Laura
"""
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix

base_dados = pd.read_csv('bd_tarefa5.csv', sep=';')
estatistica = base_dados.describe() #apenas de colunas numericas

# Separação das variáveis Previsoras e a Classe
previsores = base_dados.iloc[:, 0:16].values
classe = base_dados.iloc[:, 16].values

# NORMALIZAÇÃO DA BASE DE DADOS
#converter dados textuais em numericos para interpretação do programa
le = LabelEncoder()
for i in range(0, 16):
    previsores[:, i] = le.fit_transform(previsores[:, i])

# colocar as bases numericas em unidades proximas
scaler = StandardScaler()
previsores = scaler.fit_transform(previsores)

# DIVISAO DA BASE EM CONJUNTOS DE TREINAMENTO E TESTE
previsores_treino, previsores_teste, classe_treino, classe_teste = train_
test_split(
    previsores, classe, test_size=0.20, random_state=0)

# REDE NEURAL PARA CLASSIFICAÇÃO
"""
FOR para verificar eficiencia com diferentes funcoes e diferentes valores
para camada intermediaria
Valores camada intermediaria indo de 10 a 1000, em passos de 20
"""

funcoes = ['logistic', 'tanh', 'identity', 'relu'] #vetor com as funcoes
camadas = [4,9,16,25,36,49,64,81,100,121,144,169,196,225]

eficiencia_funcoes = []
eficiencia_camadas = []
eficiencia_total = [[]]
```

```
for j in range(len(funcoes)):
    print('j = ',funcoes[j])
    eficiencia_camadas.clear()
    for k in range(len(camadas)):
        print(camadas[k])
        nn = MLPClassifier(hidden_layer_sizes=(camadas[k],),
                           activation=funcoes[j], max_iter=1000, tol=0.001)

        treino = (nn.fit(previsores_treino, classe_treino))
        teste = nn.predict(previsores_teste)
        #MATRIZ DE CONFUSÃO
        matriz = confusion_matrix(classe_teste, teste)

        #vetor com as eficiencias para variação do num de camadas
        acerto = (matriz[0,0]+matriz[1,1])/len(classe_teste)*100
        eficiencia_camadas.append(acerto)

    eficiencia_funcoes.insert(j, eficiencia_camadas)
    eficiencia_total.append(list(eficiencia_funcoes[0]))

del(eficiencia_total[0]) #apagar colchetes iniciais

plt.plot(eficiencia_total[0], label="Logistic")
plt.plot(eficiencia_total[1], label="Tanh")
plt.plot(eficiencia_total[2], label="Identity")
plt.plot(eficiencia_total[3], label="Relu")
plt.title('Eficiencia com camda intermediaria variando de 4 a 225')
plt.legend(loc='lower right')
plt.show()
```

Selecionou-se trabalhar com Rede Neural de Classificação pois o objetivo é definir a categoria dos valores da base de dados (e comparar com a coluna *'y'*, as classificações originais).

A matriz de confusão foi utilizada para calcular a eficiência da rede, somando os acertos de 1 aos acertos de 0, e dividindo pelo tamanho da amostra (multiplicando por 100 para obter os valores em porcentagem).

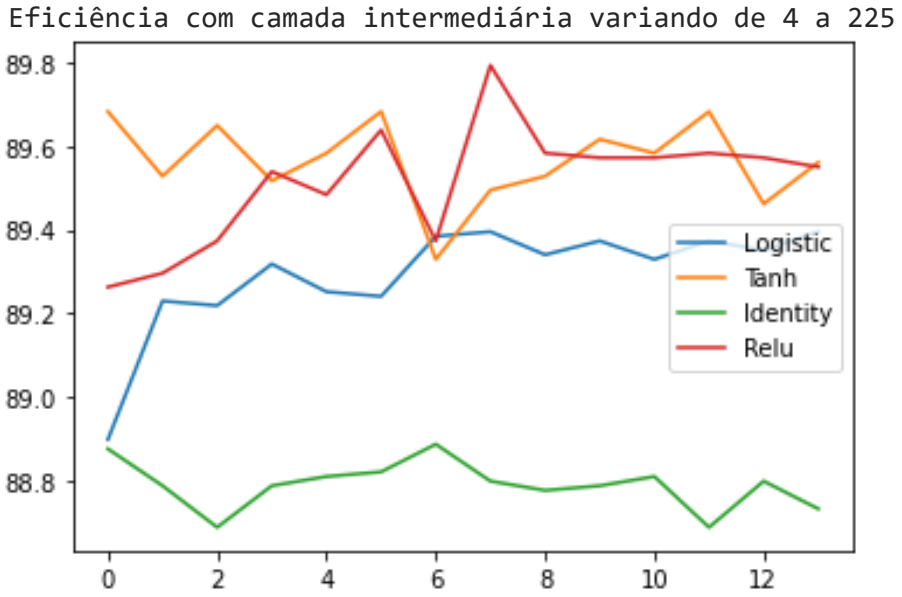
$$\frac{\text{matriz}[0,0] + \text{matriz}[1,1]}{\text{len(classe\_teste)}} * 100$$

Foram feitos testes para diferentes tamanhos e valores de camada intermediária, e para cada uma das funções disponíveis (logistic, tanh, identity, relu).

**Teste 01:**

Os valores de camadas intermediárias usados inicialmente foram: 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196 e 225.

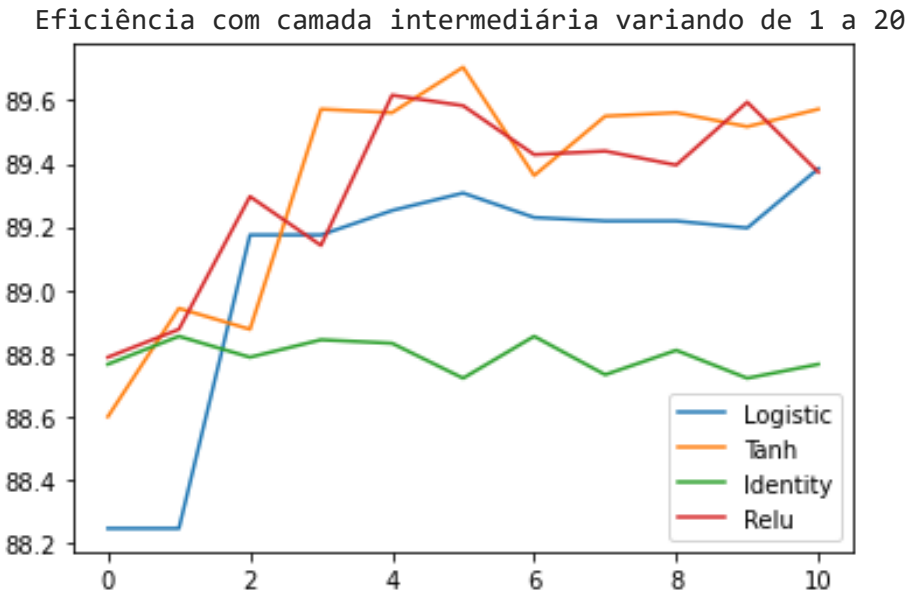
A maior eficiência registrada foi de (aproximadamente) 89.7932%, e ocorre com a função de ativação *Relu*, utilizando 81 camadas intermediárias.




**Teste 02:**

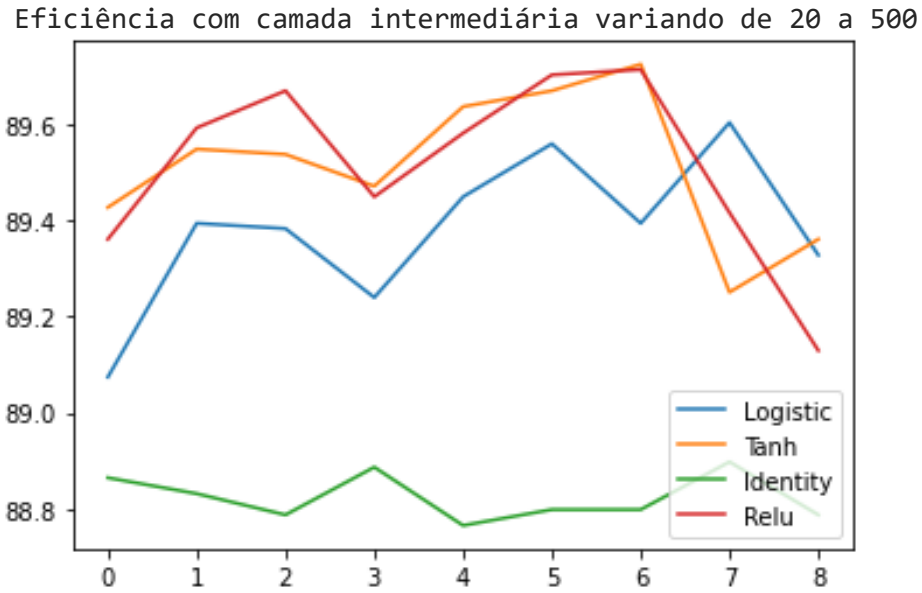
Executando o mesmo código, porém com menos camadas (1, 2, 4, 6, 8, 10, 12, 14, 16, 18 e 20), o gráfico obtido é apresentado abaixo.

Neste caso a melhor eficiência foi de 89.7047%, ocorrendo com 10 camadas intermediárias, na função de ativação *Tangente Hiperbólica*.



**Teste 03:**

Por fim testou-se as camadas com valores 20, 110, 200, 260, 320, 360, 400, 460 e 500. O gráfico gerado está registrado abaixo.




**Conclusões:**

Destaca-se que, a rede *Relu* foi a que tomou menos tempo de processamento, com em média 2 minutos (o tempo foi verificado pelo terminal do *Spyder*, visualmente), em todas as situações testadas.

No geral, para esta base de dados as melhores funções para utilizar são a *Relu* ou a *Tangente Hiperbólica*. Para a máquina utilizada, o *Relu* se mostrou melhor por ter a maior eficiência e o menor tempo de processamento.