UNIVERSIDADE ESTADUAL DO CEARÁ

CENTRO DE CIÊNCIAS E TECNOLOGIA

CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Redes Neurais Artificiais – 2023/1

Exercício de Programação 1

Professor: Jose Everardo Bessa Maia

Aluno: Erick Santos do Nascimento

Matrícula: 1440434

Introdução

Este trabalho tem como objetivo programar o algoritmo de backpropagation para uma Rede Neural Multilayer Perceptron (MLP) de tamanho 4,5,3, representada na figura. Além disso, será utilizado o conjunto de dados Iris, composto por 90 padrões de treinamento e 60 de teste, selecionados aleatoriamente. O conjunto de dados será preparado para informação de classe "one hot". Duas abordagens serão implementadas, uma com funções de ativação sigmoide nas camadas de saída e escondida, e outra com a função de ativação sigmoide na camada de saída e gaussiana na camada escondida, também conhecida como Rede Neural Função Base Radial (RBFNN). Será apresentada a Matriz de Confusão de cada rede neural e os resultados de precisão, recall e F1-score serão comparados, além de diferentes números de épocas de treino, 100, 1000 e 100000

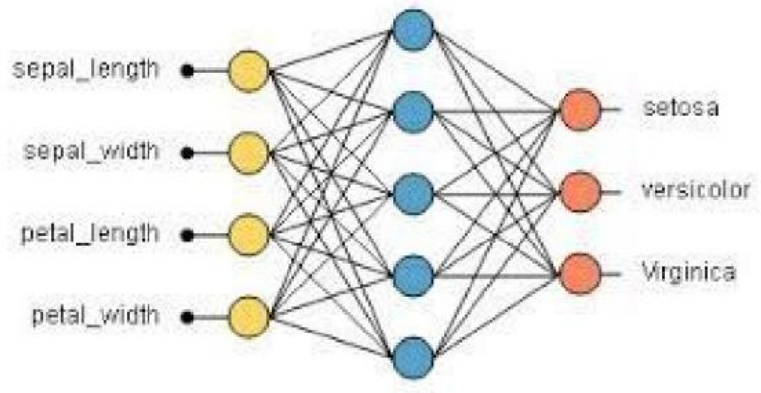


Figure 1: Arquitetura da Rede Neural MLP para treinar e testar com o data set **Iris**.

Taxa de aprendizado: `0.001`

Resultados

Para MLP:

-Com 100 épocas de treino

Accuracy: 0.333

Precision: 0.111

Recall: 0.333

F1-score: 0.167

Confusion matrix:

[[0 0 20]

[0 0 20]

[0 0 20]]

Classification report train:

Accuracy: 0.333

Precision: 0.111

Recall: 0.333

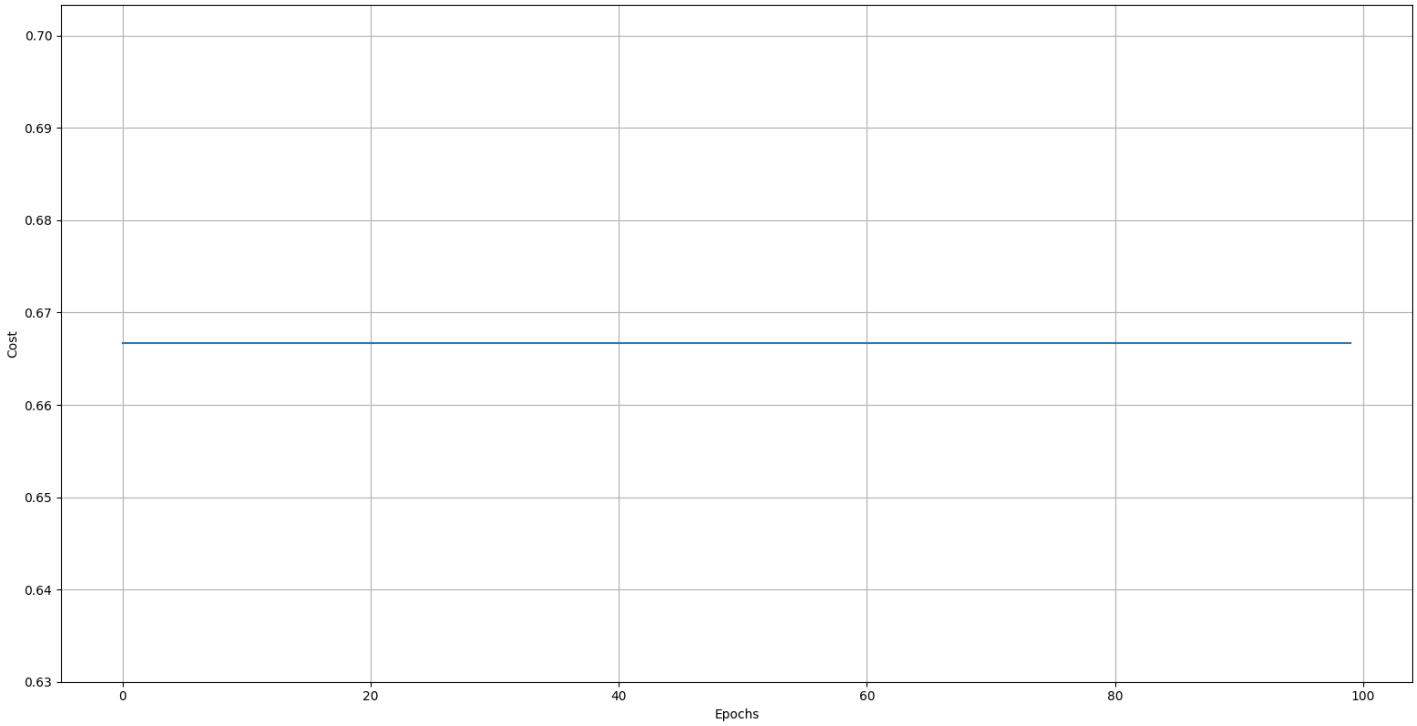
F1-score: 0.167

Confusion matrix:

[[0 0 30]

[0 0 30]

[0 0 30]]



-Com 1000 épocas de treino

Accuracy: 0.667

Precision: 0.500

Recall: 0.667

F1-score: 0.556

Confusion matrix:

[[20 0 0]

[0 0 20]

[0 0 20]]

Classification report train:

Accuracy: 0.667

Precision: 0.500

Recall: 0.667

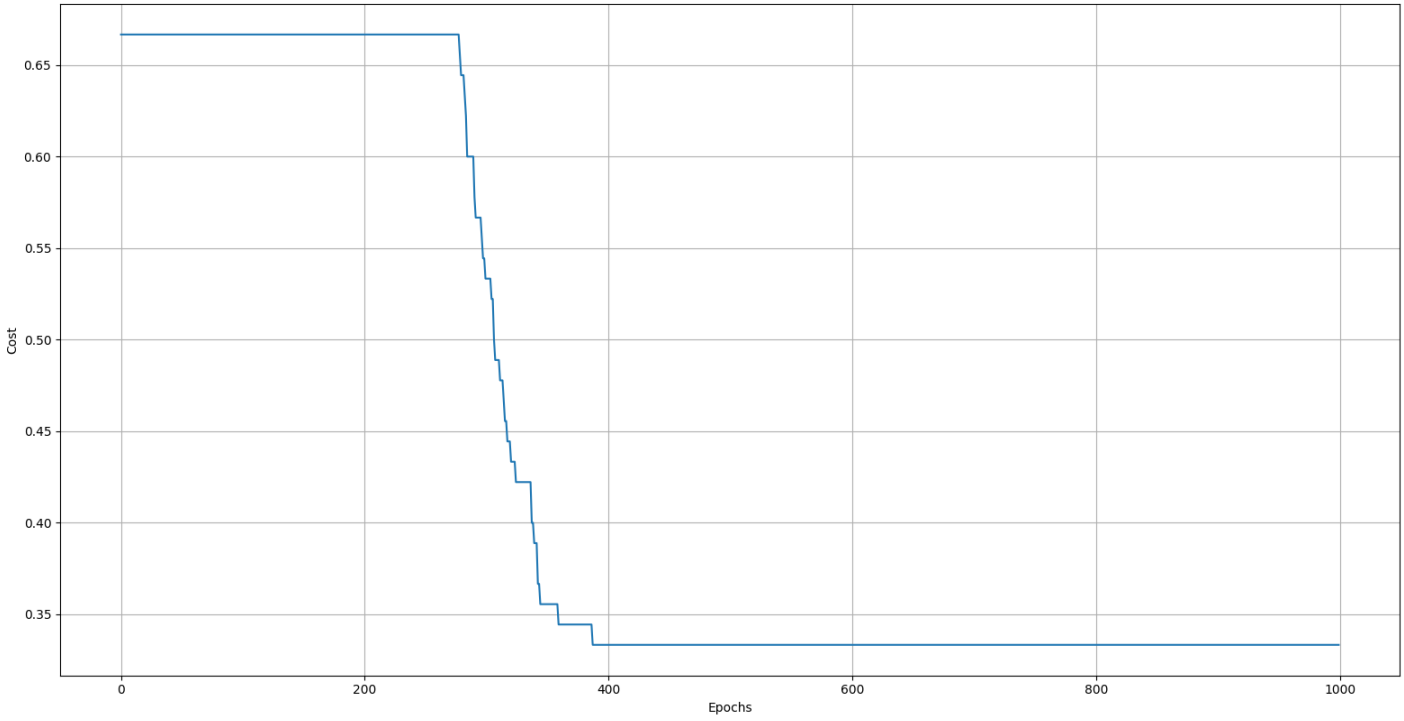
F1-score: 0.556

Confusion matrix:

[[30 0 0]

[0 0 30]

[0 0 30]]



-Com 100000 épocas de treino

Classification report test:

Accuracy: 0.950

Precision: 0.957

Recall: 0.950

F1-score: 0.950

Confusion matrix:

[[20 0 0]

[0 17 3]

[0 0 20]]

Classification report train:

Accuracy: 0.956

Precision: 0.961

Recall: 0.956

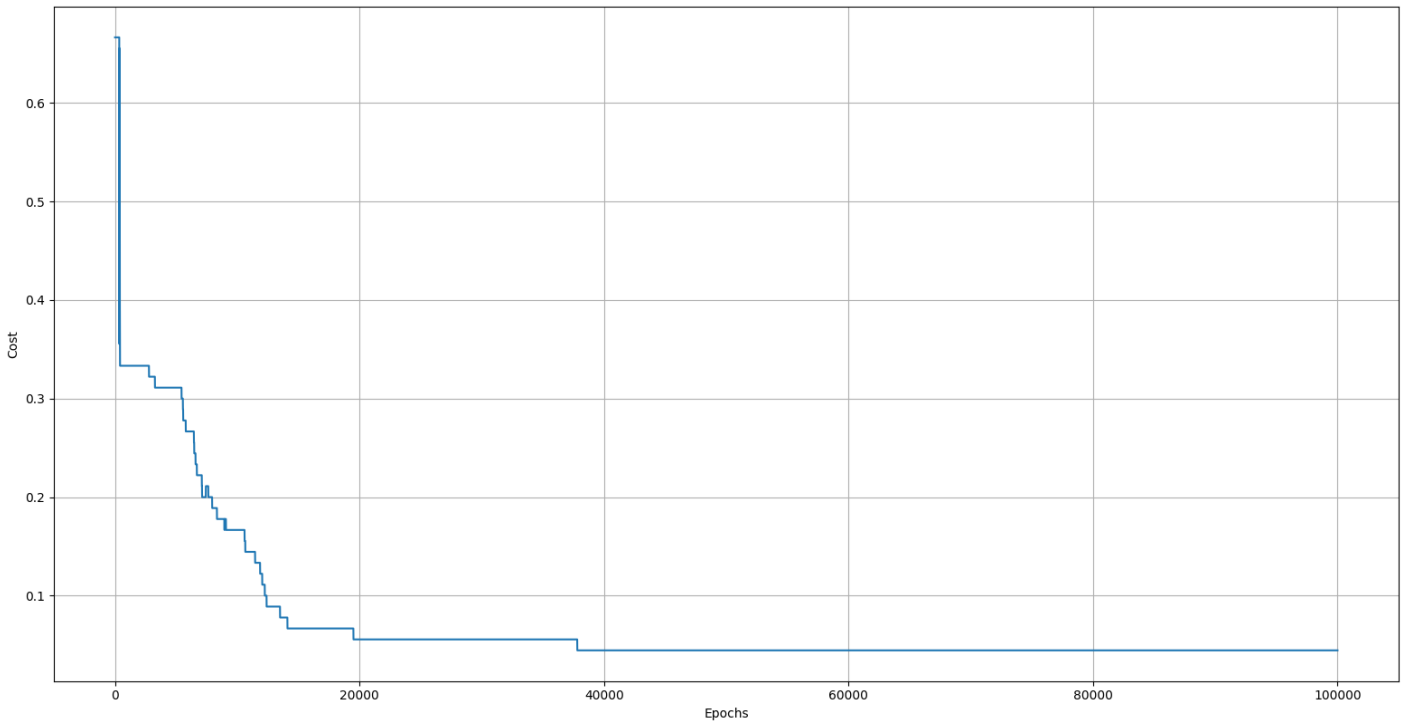
F1-score: 0.955

Confusion matrix:

[[30 0 0]

[0 26 4]

[0 0 30]]



Para Rede Neural Função Base Radial (RBFNN):

-Com 100 épocas de treino

Classification report test:

Accuracy: 0.367

Precision: 0.448

Recall: 0.367

F1-score: 0.232

Confusion matrix:

[[0 0 20]

[0 2 18]

[0 0 20]]

Classification report train:

Accuracy: 0.400

Precision: 0.452

Recall: 0.400

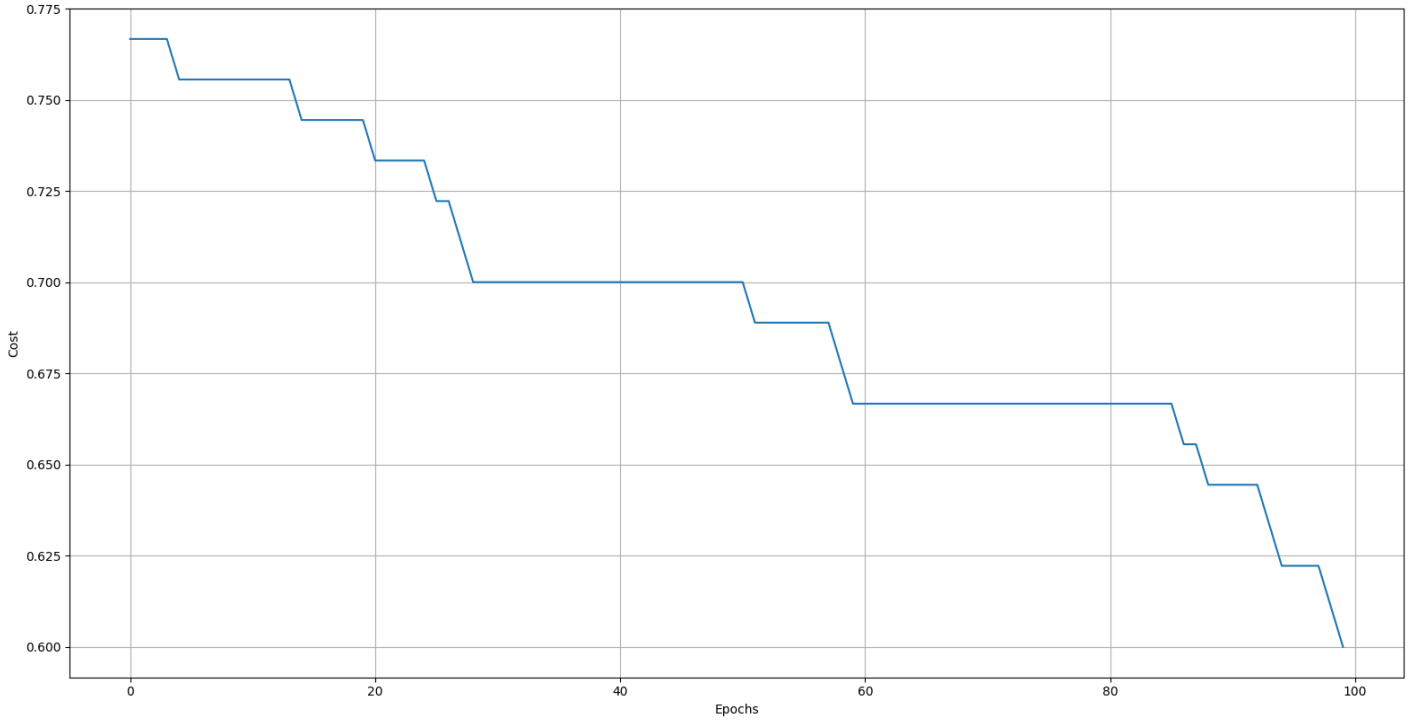
F1-score: 0.287

Confusion matrix:

[[0 0 30]

[0 6 24]

[0 0 30]]



-Com 1000 épocas de treino

Classification report test:

Accuracy: 0.950

Precision: 0.951

Recall: 0.950

F1-score: 0.950

Confusion matrix:

[[20 0 0]

[0 19 1]

[0 2 18]]

Classification report train:

Accuracy: 0.933

Precision: 0.935

Recall: 0.933

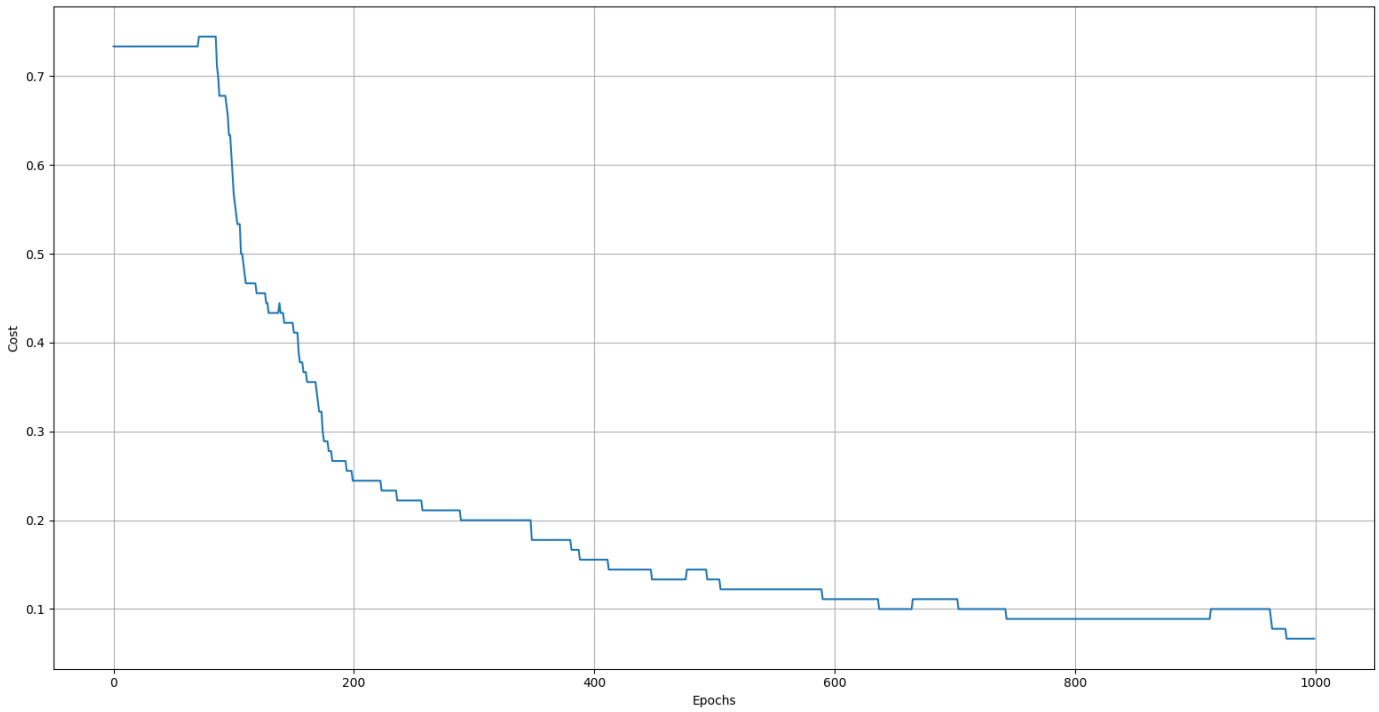
F1-score: 0.933

Confusion matrix:

[[30 0 0]

[0 28 2]

[0 4 26]]



-Com 100000 épocas de treino

Classification report test:

Accuracy: 0.950

Precision: 0.951

Recall: 0.950

F1-score: 0.950

Confusion matrix:

[[20 0 0]

[0 19 1]

[0 2 18]]

Classification report train:

Accuracy: 0.944

Precision: 0.947

Recall: 0.944

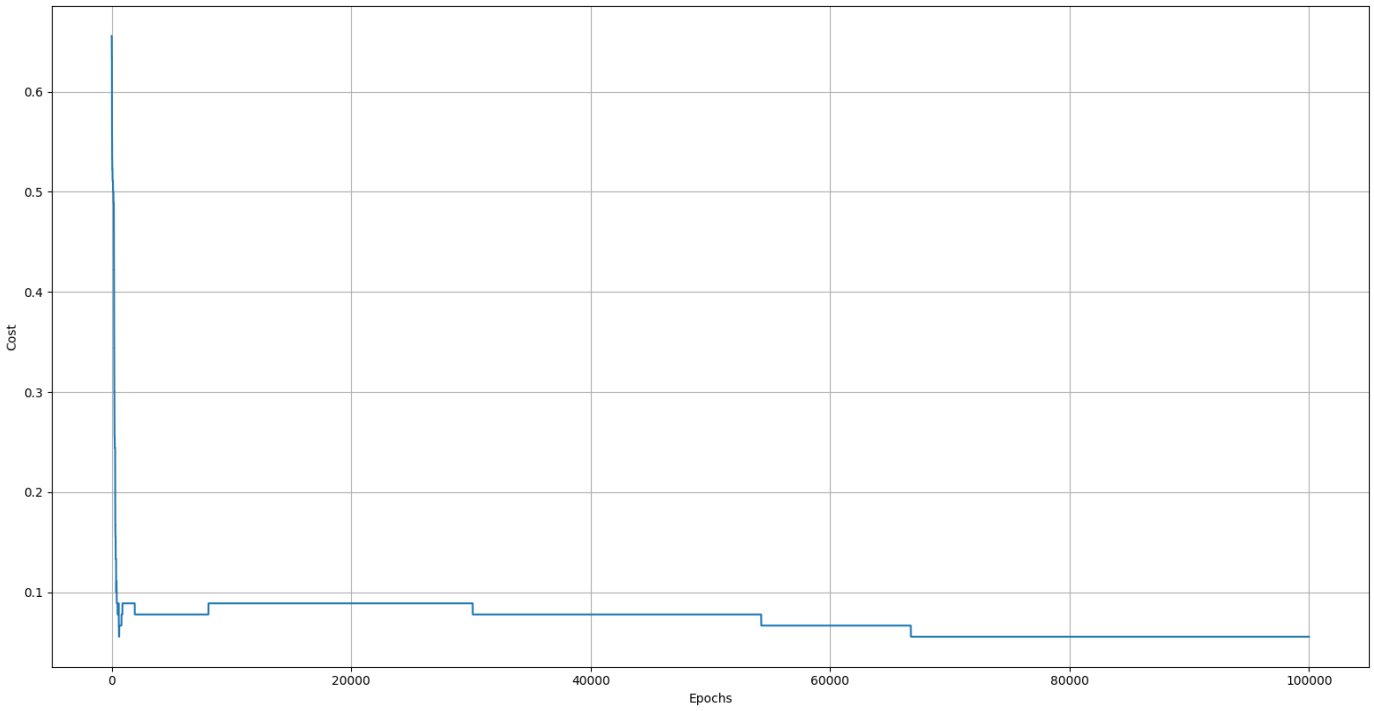
F1-score: 0.944

Confusion matrix:

[[30 0 0]

[0 29 1]

[0 4 26]]



Comentários

Ao analisar os resultados obtidos nos experimentos realizados com as redes neurais MLP e RBFNN, observou-se que a MLP apresentou desempenho ruim com 100 e 1.000 épocas, sem demonstrar qualquer comportamento de convergência com 100 épocas. Por outro lado, com 100.000 épocas, a MLP apresentou resultados satisfatórios. Foi possível perceber também que a MLP foi mais afetada pela redução do conjunto de treinamento para a divisão em classes.

No caso da RBFNN, com 100 épocas, a rede apresentou um desempenho ruim, mas já mostrou um comportamento de convergência. Porém, com 1.000 e 100.000 épocas, a RBFNN obteve resultados positivos e demonstrou uma convergência mais rápida do que a MLP. Esse resultado pode ser consequência da etapa de pré-treino dos neurônios gaussianos, que contribui para a melhoria do desempenho da rede.

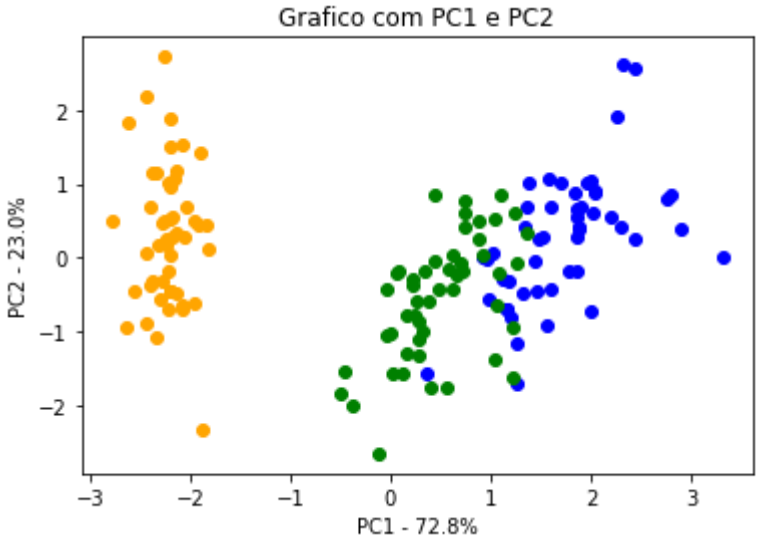
Com apenas 100 épocas de treinamento, ambos os modelos MLP e RBFNN não foram capazes de atingir uma taxa de acerto satisfatória. Isso indica que a taxa de aprendizado utilizada foi pequena e não foi suficiente para permitir que o modelo aprendesse o suficiente em tão poucas épocas. Assim, seria necessário um número maior de épocas para que o modelo pudesse convergir para uma boa taxa de acerto.

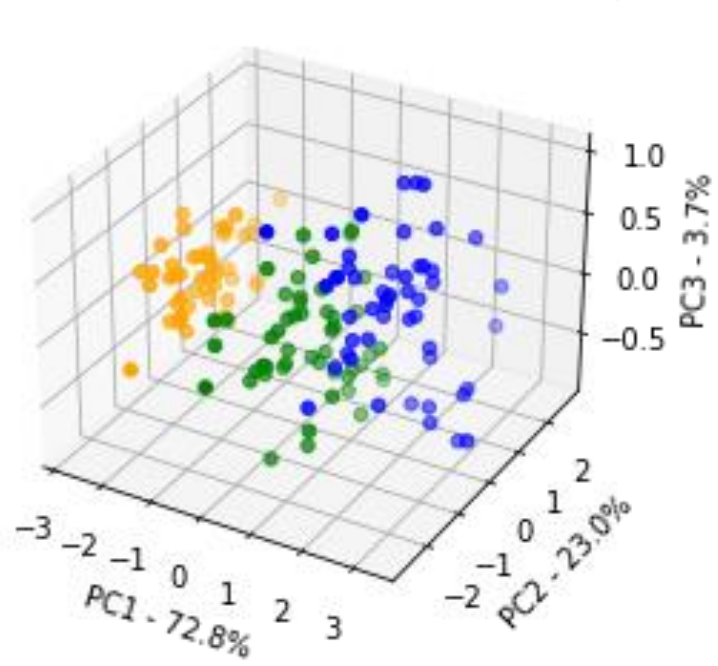
No caso do dataset usado (iris), apliquei PCA para observar o dataset

Em azul as amostras Iris-virginica

Em laranja as amostras de Iris-setosa

Em verde as amostras de Iris-versicolor





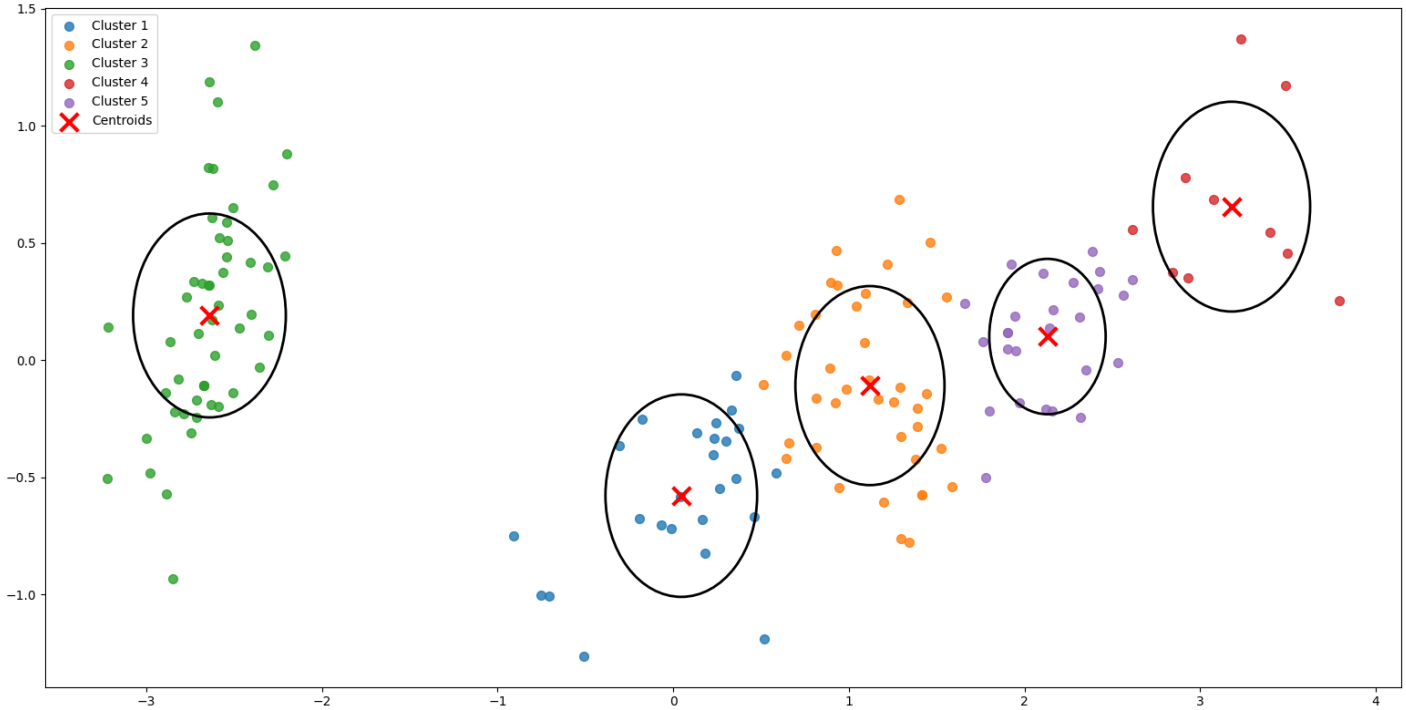
Pouca diferença em 3D

Aqui se demostra que a verdadeira tarefa é separar as amostras de Iris-virginica e Iris-versicolor

Tarefa que não parece ser muito fácil nem para as retas discriminadoras geradas pelo MLP nem pelos campos perceptivos das RBFNNs

Na imagem acima fica claro que mesmo que com 4 retas (já que uma provavelmente é usada para separar a Iris-setosa das demais) seria talvez impossível separar as 2 classes.

Aqui são os centros selecionados pelos neurônios gaussianos



Aqui podemos ver que o cluster na cor laranja é o mais afetado pela mistura das classes, claramente contém amostras das 2 classes o que pode ter gerado os erros da rede que obteve os melhores resultados.

Para acessar todo código: https://github.com/Birunda3000/Disciplina-Redes-Neurais-Artificiais e https://github.com/Birunda3000/PCA

Código

MLP

```
import os



import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import utils as ut
from IPython.display import display
from sklearn.model_selection import train_test_split
from tqdm import tqdm
```

```
# Load df
path = os.path.join(os.path.dirname(__file__), "..", "_DATA_", "Iris.csv")
df = pd.read_csv(path)
df = ut.fn_cat_onehot(df)
df.insert(0, "Bias", 1)
display(df.head())


# select the feature columns
features = df[["SepalLengthCm", "SepalWidthCm", "PetalLengthCm", "PetalWidthCm"]]
features_array = np.array(features)


# select the label columns
labels = df[
    ["Species_Iris-setosa", "Species_Iris-versicolor", "Species_Iris-virginica"]
]
labels_array = np.array(labels)


# Split data into train and test sets with equal proportion of each class
X_train, X_test, y_train, y_test = train_test_split(
    features_array, labels_array, test_size=0.4, stratify=labels_array, random_state=42,
shuffle=True
)


# Parameters
N = 0.001
EPOCHS = 100000
cost = []


# Network architecture
```

```python
input_neurons = features_array.shape[1]  # 4
hidden_neurons = 5
output_neurons = labels_array.shape[1]  # 3


# Random weights
w_hidden_1 = np.random.uniform(size=(input_neurons, hidden_neurons))
w_output = np.random.uniform(size=(hidden_neurons, output_neurons))


# Treinamento
for i in tqdm(range(EPOCHS)):
    # Feedforward
    activation_hidden_1 = ut.sigmoid(np.dot(X_train, w_hidden_1))
    activation_output = ut.sigmoid(np.dot(activation_hidden_1, w_output))
    cost.append(ut.classification_error(y_true=y_train, y_pred=activation_output))


    # Backpropagation
    delta_output = (y_train - activation_output) * ut.sigmoid_derivative(
        activation_output
    )
    delta_hidden_1 = delta_output.dot(w_output.T) * ut.sigmoid_derivative(
        activation_hidden_1
    )


    # Update weights
    w_output += np.dot(activation_hidden_1.T, delta_output) * N
    w_hidden_1 += np.dot(X_train.T, delta_hidden_1) * N


# Plot
print("Final classification error: ", cost[-1])
```

```
# Test
activation_hidden_test = ut.sigmoid(np.dot(X_test, w_hidden_1))
activation_output_test = ut.sigmoid(np.dot(activation_hidden_test, w_output))


y_true = np.argmax(y_test, axis=1)
y_pred = np.argmax(activation_output_test, axis=1)


print("Classification report test:")
ut.get_classification_metrics(y_true=y_true, y_pred=y_pred)


# Training
activation_hidden_train = ut.sigmoid(np.dot(X_train, w_hidden_1))
activation_output_train = ut.sigmoid(np.dot(activation_hidden_train, w_output))


y_true = np.argmax(y_train, axis=1)
y_pred = np.argmax(activation_output_train, axis=1)


print("Classification report train:")
ut.get_classification_metrics(y_true=y_true, y_pred=y_pred)


plt.plot(cost)
plt.xlabel("Epochs")
plt.ylabel("Cost")
plt.grid()
plt.show()
```

RBFNN

```
import os


import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import utils as ut
from sklearn.model_selection import train_test_split
from tqdm import tqdm


# Load df
path = os.path.join(os.path.dirname(__file__), "..", "_DATA_", "Iris.csv")
df = pd.read_csv(path)
df = ut.fn_cat_onehot(df)


# select the feature columns
features = df[["SepalLengthCm", "SepalWidthCm", "PetalLengthCm", "PetalWidthCm"]]
features_array = np.array(features)


# select the label columns
labels = df[
    ["Species_Iris-setosa", "Species_Iris-versicolor", "Species_Iris-virginica"]
]
labels_array = np.array(labels)


# Split data into train and test sets with equal proportion of each class
X_train, X_test, y_train, y_test = train_test_split(
```

```
    features_array, labels_array, test_size=0.4, stratify=labels_array, random_state=42,
shuffle=True
)


# Parameters
N = 0.001
EPOCHS = 100000
cost = []


# Network architecture
input_neurons = features_array.shape[1]  # 4 features
hidden_neurons = 5  # 5 neurons with gaussian activation
output_neurons = labels_array.shape[1]  # 3 neurons with sigmoid activation



# Random weights
w_output = np.random.uniform(size=(hidden_neurons, output_neurons))
b_output = np.random.uniform(size=(1, output_neurons))


# mu and sigma
mu, sigma = ut.get_kmeans_centers_for_rbf(df=features, n_clusters=hidden_neurons)
# sigma = np.ones((hidden_neurons))


# Training
for epoch in tqdm(range(EPOCHS)):
    # Forward propagation
    activation_hidden = ut.gaussian(x=X_train, mu=mu, sigma=sigma)
    activation_output = ut.sigmoid(np.dot(activation_hidden, w_output) + b_output)
    cost.append(ut.classification_error(y_true=y_train, y_pred=activation_output))
```

```
    # Backpropagation
    delta_output_w = (y_train - activation_output) * ut.sigmoid_derivative(
        activation_output
    )
    delta_output_b = delta_output_w.sum(axis=0, keepdims=True)


    # Update weights
    w_output += N * activation_hidden.T.dot(delta_output_w)
    b_output += N * delta_output_b


# Plot
'''corretos, errados = ut.calc_accuracy(true_labels=labels_array,
pred_labels=activation_output)
print("Previsões corretas: {}, Previsões erradas: {}".format(corretos, errados))


plt.plot(cost)
plt.title("Taxa de aprendizado: {}".format(N))
plt.xlabel("Épocas")
plt.ylabel("Custo")
plt.show()'''


# Plot
print("Final Error: {}".format(cost[-1]))


# Test
activation_hidden_test = ut.gaussian(x=X_test, mu=mu, sigma=sigma)
activation_output_test = ut.sigmoid(np.dot(activation_hidden_test, w_output) + b_output)
```

```
y_true = np.argmax(y_test, axis=1)
y_pred = np.argmax(activation_output_test, axis=1)

print("Classification report test:")
ut.get_classification_metrics(y_true=y_true, y_pred=y_pred)

# Training
activation_hidden_train = ut.gaussian(x=X_train, mu=mu, sigma=sigma)
activation_output_train = ut.sigmoid(
    np.dot(activation_hidden_train, w_output) + b_output
)

y_true = np.argmax(y_train, axis=1)
y_pred = np.argmax(activation_output_train, axis=1)

print("Classification report train:")
ut.get_classification_metrics(y_true=y_true, y_pred=y_pred)

plt.plot(cost)
plt.xlabel("Epochs")
plt.ylabel("Cost")
plt.grid()
plt.show()
```

Utils

```python
import numpy as np
import pandas as pd
from sklearn import metrics
from sklearn.cluster import KMeans
from sklearn.preprocessing import OneHotEncoder


def fn_cat_onehot(df):
    """Generate onehoteencoded features for all categorical columns in df"""
    # print(f"df shape: {df.shape}")
    # NaN handing
    nan_count = df.isna().sum().sum()
    if nan_count > 0:
        print(f"NaN = **{nan_count}** will be categorized under feature_nan columns")


    model_oh = OneHotEncoder(handle_unknown="ignore", sparse=False)
    for c in list(df.select_dtypes("category").columns) + list(
        df.select_dtypes("object").columns
    ):
        print(f"Encoding **{c}**")  # which column
        matrix = model_oh.fit_transform(
            df[[c]]
        )  # get a matrix of new features and values
        names = model_oh.get_feature_names_out()  # get names for these features
        df_oh = pd.DataFrame(
            data=matrix, columns=names, index=df.index
        )  # create df of these new features


        # display(df_oh.plot.hist())
```

```
        df = pd.concat([df, df_oh], axis=1)  # concat with existing df


        df.drop(
            c, axis=1, inplace=True
        )  # drop categorical column so that it is all numerical for modelling


    # print(f"#### New df shape: **{df.shape}**")
    return df


# Activation function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))


# Derivative of the activation function
def sigmoid_derivative(x):
    return x * (1 - x)


# MSE
def MSE(Y_target, Y_pred):
    return np.mean((Y_target - Y_pred) ** 2)


# np.where(output > 0.5, 1, 0)


def classification_error(y_true, y_pred):
```

```
    y_true = np.argmax(y_true, axis=1)
    y_pred = np.argmax(y_pred, axis=1)
    incorrect = np.sum(y_true != y_pred)
    total = y_true.shape[0]
    error_rate = incorrect / total
    return error_rate


def calc_accuracy(true_labels, pred_labels):
    # transforma as saídas preditas em um array binário
    pred_labels_bin = np.zeros_like(pred_labels)
    pred_labels_bin[np.arange(len(pred_labels)), pred_labels.argmax(axis=1)] = 1
    # calcula o número de acertos e erros
    correct = (pred_labels_bin == true_labels).all(axis=1).sum()
    incorrect = len(true_labels) - correct
    return correct, incorrect


def gaussian_1(x, mu, sigma):
    output = []
    x = np.array(x)
    mu = np.array(mu)
    sigma = np.array(sigma)
    for sample in x:
        output.append(np.exp(-((np.linalg.norm(sample - mu)) ** 2) / (2 * sigma**2)))
    return np.array(output)


def gaussian(x, mu, sigma):
    output = []
    for neuron in range(len(mu)):
```

```
        output.append(gaussian_1(x=x, mu=mu[neuron], sigma=sigma[neuron]))
    return np.array(output).T


def gaussian_derivative_x(x, mu, sigma):
    return -np.exp(-((x - mu) ** 2) / (2 * sigma**2)) * (x - mu) / sigma**2


def gaussian_derivative_mu(x, mu, sigma):
    return np.exp(-((x - mu) ** 2) / (2 * sigma**2)) * (x - mu) / sigma**2


def gaussian_derivative_sigma(x, mu, sigma):
    return np.exp(-((x - mu) ** 2) / (2 * sigma**2)) * (x - mu) ** 2 / sigma**3


def get_kmeans_centers_for_rbf(df, n_clusters):
    # Initialize a KMeans model with the desired number of clusters
    kmeans = KMeans(n_clusters=n_clusters)

    # Fit the model to the data
    kmeans.fit(df)

    # Get the centers of each cluster as a numpy array
    centers = kmeans.cluster_centers_

    # Get the standard deviation of each cluster
    stds = np.zeros(n_clusters)
    mean_distance = np.zeros(n_clusters)
```

```
    for i in range(n_clusters):
        cluster_points = df[kmeans.labels_ == i]
        mean_distance[i] = np.linalg.norm(cluster_points - centers[i], axis=1).mean()

    # Return the centers, standard deviation, and mean distance of each cluster
    return centers, mean_distance


def get_classification_metrics(y_true, y_pred):
    accuracy = metrics.accuracy_score(y_true, y_pred)
    precision = metrics.precision_score(y_true, y_pred, average="weighted")
    recall = metrics.recall_score(y_true, y_pred, average="weighted")
    f1_score = metrics.f1_score(y_true, y_pred, average="weighted")
    confusion_matrix = metrics.confusion_matrix(y_true, y_pred)
    print("Accuracy: {:.3f}".format(accuracy))
    print("Precision: {:.3f}".format(precision))
    print("Recall: {:.3f}".format(recall))
    print("F1-score: {:.3f}".format(f1_score))
    print("Confusion matrix:\n", confusion_matrix)
    return accuracy, precision, recall, f1_score, confusion_matrix
```