TÉCNICO LISBOA

Computação Inteligente para a Internet das Coisas

Mestrado Engenharia Electrónica

Projeto 1 – Neural Networks – Localization error estimation in WSN

Autores:

João Bernardo Silva (97004) joao.garces.silva@tecnico.ulisboa.pt
Victor Macedo (1105095) victor.macedo@tecnico.ulisboa.pt

Grupo 4

2022/2023 – 2º Semestre, P4

Conteúdo

1 Introdução

O objetivo deste projeto consiste em treinar uma rede neural para calcular o erro de localização expectável segundo um grupo de parâmetros, que se encontram em Lab6-Proj1_Dataset.csv, o mesmo possui os seguintes dados:

- Node Density (numero de sensores por m^2)
- Anchor Ratio (numero de ancoras pelo numero de nodes total)
- Transmission Range (m)
- Step Size (parâmetro no algoritmo usado para estimar a localização do sensor)
- Iterations (numero de interações usadas no algoritimo para estimar a localização do sensor)
- Estimated Sensor Location Error – ELSE (m)

Para poder ser feita uma boa estimativa, primeiro foi necessário fazer um pré-processamento dos dados, normalizar valores e retirar outliers.

Feito o tratamento de dados, é treinada a rede neural, usando o MLP Regressor [4].

2 Metodologia

2.1 Pré processamento

Antes de modelar e treinar o modelo de rede neural é necessário efetuar uma análise prévia do dataset que será utilizado, usando de ferramentas gráficas como o *matplotlib* plotou-se o seguindo gráfico com os dados originais.

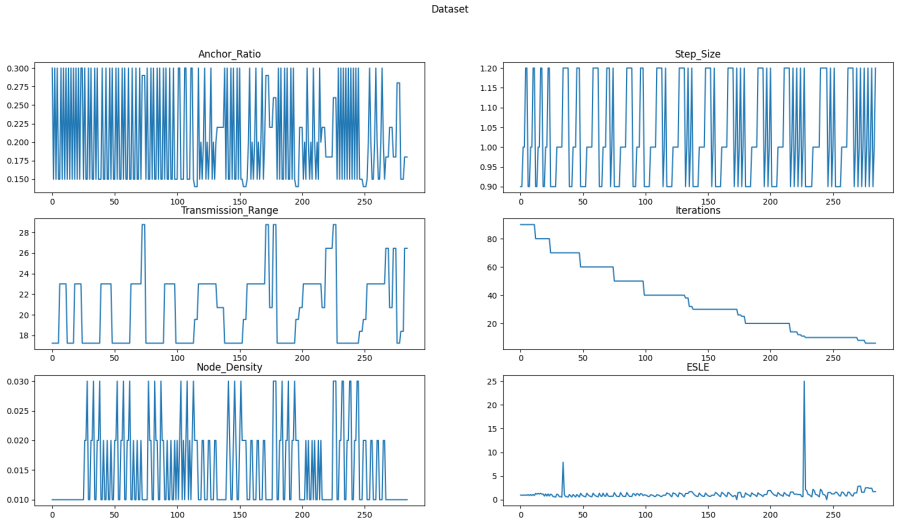

Figura 1: Valores do Dataset original

Analisando a figura 1 percebe-se que há outliers, principalmente nos dados **ESLE**, e os valores não estão normalizados, sendo assim deve ser feito um pré processamento para garantir um melhor funcionamento da rede.

2.1.1 Outlier

Para remoção dos outliers utilizou um fator k de 1.5, ou seja valores 50% acima ou abaixo da média devem ser identificados como outliers e para a abordagem foram testados 3 diferentes métodos: interpolação dos valores, remoção do valor e uso do valor prévio.

2.1.2 Padronização

Uma rede neural obtém melhores resultados de forma mais rápida se os valores do dataset estiverem padronizados. Por tanto para atingirmos melhores resultados faz-se necessário padronizar os mesmos. Foram testados 2 diferentes abordagens: normalização e o Z-score. Para efetuar a normalização utilizou-se a equação 1

$$X_{Norm} = \frac{x - min(x)}{max(x) - min(x)} \tag{1}$$

E para o calculo do Z-score:

$$X_z = \frac{x - media(x)}{desvio(x)} \tag{2}$$

2.1.3 Divisão do dataset

O dataset a ser utilizado tem que ser dividido em pelo menos 2 classes: treino e teste, para que a rede neural consiga ajustar seus parâmetros e predizer a saída baseada na entrada, há também um 3° conjunto chamado validação o qual não é usado no treino da rede porém é muito útil para testa-la ao final do treinamento. Como visto nas aulas teóricas [1] as divisões mais comuns de dataset são:

- 70% Treino; 15% Validação; 15% Teste
- 80% Treino; 10% Validação; 10% Teste
- 60% Treino; 20% Validação; 20% Teste

Logo devem ser realizados testes com as 3 opções citadas acima.

2.2 Rede Neural

2.2.1 Modelo

O modelo de rede neural utilizado foi MLPRegressor (Multi-Layer Perceptron Regressor), usando a livraria *sklearn.neural_network.MLPRegressor* [4].

2.2.2 Função de ativação

Função de ativação corresponde à função utilizada na hidden layer, no caso deste modelo existem quatro funções:identidade; logística; tangente hiperbólica; unidade linear retificada.

Identidade corresponde a uma função linear do tipo $f(x) = x$.

Logística corresponde a uma função sigmoide: $f(x) = \frac{1}{1+e^{-x}}$

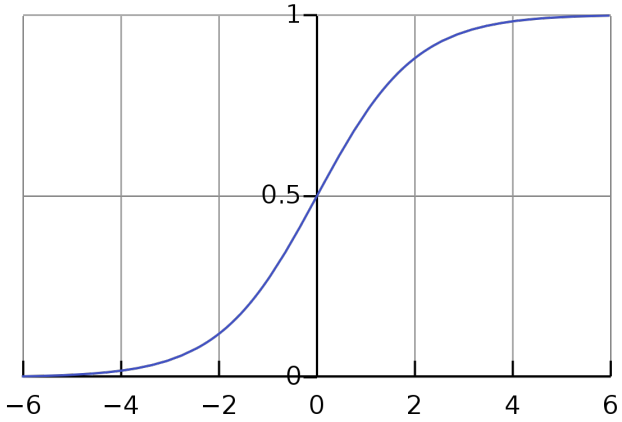

Figura 2: Função logística sigmoide

Unidade linear retificada corresponde a uma função nula até $x = 0$, que cresce linearmente para $x \geq 0$.

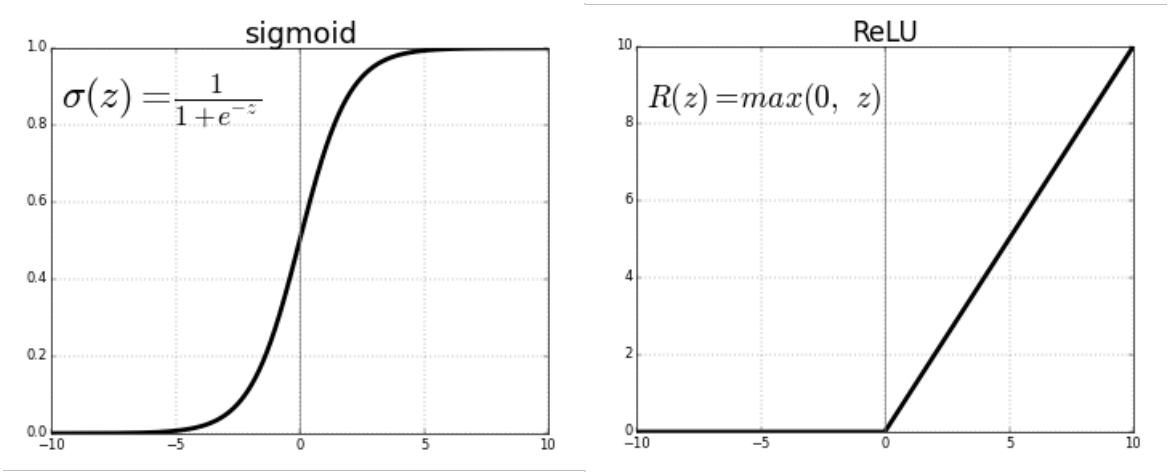

Figura 3: Comparação entre função de ativação sigmoide e unidade linear retificada

Tangente hiperbólica é representada graficamente da seguinte forma:

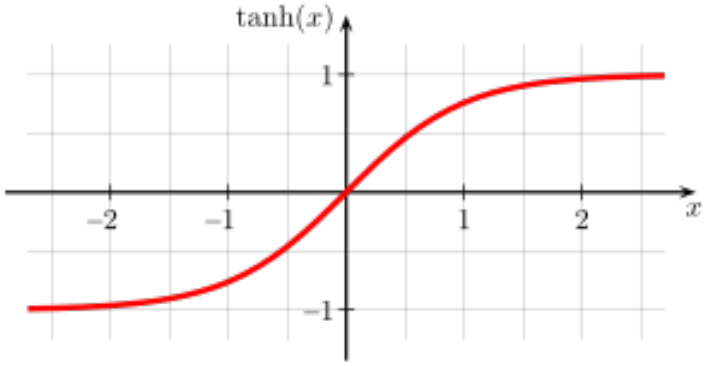

Figura 4: Representação Gráfica da Tangente Hiperbólica

2.2.3 Solver

O *solver* especifica o algoritmo de otimização utilizado para treinar a rede neural, ou seja, determina como os pesos da rede são atualizados durante o processo de treino.

existem três algoritmos:

- lfbgs - usa o algoritmo de memória limitada Broyden-Fletcher-Goldfarb-Shanno (L-BFGS), é aconselhável usar para datasets pequenos;
- sgd - descida do gradiente estocástico. Este algoritmo atualiza os pesos da rede neural usando uma única amostra ou um mini-lote de amostras por vez. É adequado para problemas em grande escala e pode lidar com dados em *streaming*. No entanto, pode ser sensível à taxa de aprendizagem e pode exigir ajuste cuidadoso dos hiperparâmetros.
- adam - Estimação Adaptativa de Momento. É uma extensão do sgd e incorpora momento e taxas de aprendizagem adaptativas.É normalmente usado como escolha padrão, pois tem um bom desempenho numa ampla variedade de problemas.

3 Resultados

3.1 Pré processamento

A principio utilizou-se uma rede MLPRegressor, com 10 hidden layers, função de ativação *identity*, solver *lbfgs* e learning rate *adaptive*, variou-se os parametros afim de identificar quais resultavam em menor Root Mean Square Error (RMSE), consequentemente o melhor desempenho.

3.1.1 Outlier

O algoritmo identificou 16 outliers para um $k = 1.5$, aplicando as diferentes metodologias obteve-se o resultado da tabela 1.

Metodologia	RMSE	Melhora RMSE (%)
Interpolation	0.178	67,34
Remove	0.311	42,93
Previous	0.301	44,77
Nenhum	0.545	-

Tabela 1: Resultado metodologias outlier

Utilizando a interpolação obteve-se resultados próximo de 70% menor se comparado com a não remoção dos outliers, logo a abordagem utilizada deverá ser utilizada no algoritmo, resultado no seguinte dataset:

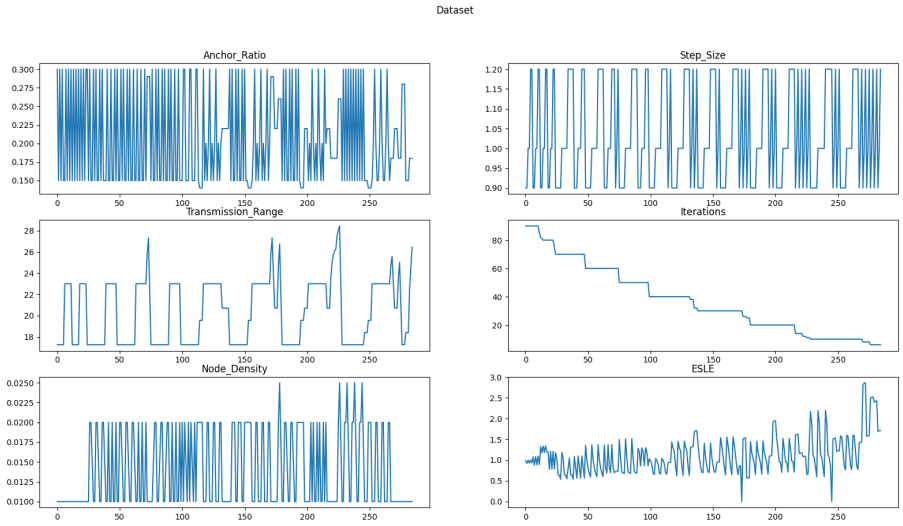

Figura 5: Dataset após a interpolação

3.1.2 Padronização

Os testes com os algoritmos da padronização foram feitos utilizando a interpolação dos outlier e resultaram na tabela 2:

Metodologia	RMSE	Melhora RMSE (%)
Normalização	0.0623	65
Z-Score	0.3503	-96
Nenhum	0.178	-

Tabela 2: Resultado metodologias padronização

O algoritmo Z-score além de apresentar um resultado muito pior que a normalização mostrou-se ineficiente pois gerou resultados piores do que ao não aplicar a padronização. Sendo assim deve ser utilizado a normalização que resulta no dataset da figura 6

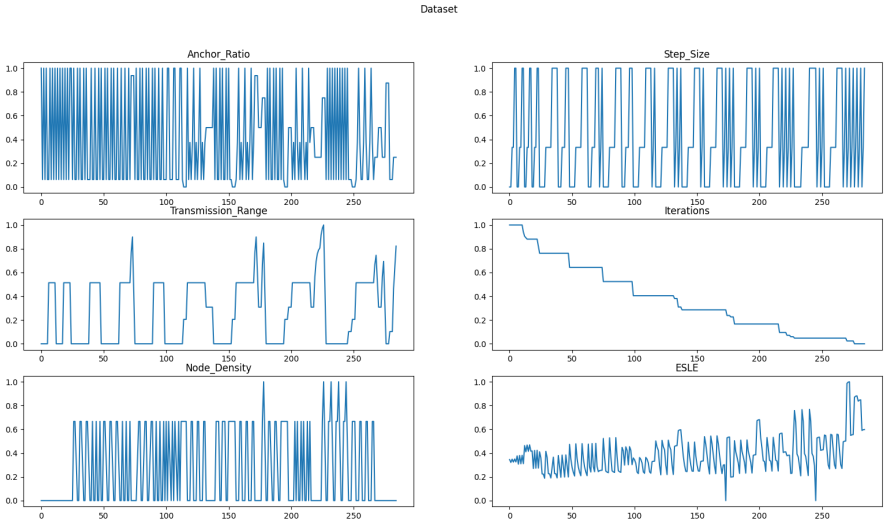

Figura 6: Dataset após a normalização

3.1.3 Divisão do dataset

Utilizando as diferente opções de divisão do dataset os métodos citados acima obteve-se os resultados da tabela 3:

Metodologia	RMSE	% em relação ao melhor
70/15/15	0.0927	-8,93
80/10/10	0.0851	-
60/20/20	0.1029	-20,92

Tabela 3: Resultado divisão dataset

A divisão 80/10/10 se mostrou mais eficiente e deve ser utilizada no algoritmo.

3.2 Rede Neural

3.2.1 Função de ativação

Utilizando as diferente opções de funções de ativação obteve-se os resultados da tabela 4:

Activation	RMSE	% em relação ao melhor
Identity	0.0851	57,77
Logisitc	0.0845	58,18
Tanh	0.0684	71,89
Relu	0.04920	-

Tabela 4: Resultado das funções de ativação

Como pode ser observado, os melhores resultados são obtidos com reLU.

3.2.2 Solver

Solver	RMSE	% em relação ao melhor
lbfgs	0.0492	-
sgd	0.1327	37,08
adam	0.1179	41,73

Tabela 5: Resultado de RMSE utilizando os diferentes solvers

Como foi dito anteriormente, lbfgs é aconselhado para datasets de pequenas dimensões, logo é o que obtém melhores resultados para a situação em questão.

4 Conclusão

O modelo de rede neural com o melhor desempenho para o problema em questão utiliza os seguintes parâmetros: função de ativação Relu, Solver lbfgs e divisão de dataset 80/10/10.

O RMSE obtido ao final da rede neural foi de 0.04920, para mensurar se o resultado obtido é satisfatório é preciso comparar com a precisão de um GPS. A ferramenta Maps da empresa Google possui uma precisão de 20m [3] e os GPS da empresa Garmin cerca de 5m a 10m de precisão [2] , sendo assim é possível afirmar que o algoritimo desenvolvido neste trabalho possui um ótimo desempenho.

Referências

[1] Joao Paulo Carvalho. *Slides Computação Inteligente para a Internet das Coisas*. 2023.

[2] Garmin. *GPS Accuracy*, (accessed: 29.05.2023). Disponivel em: https://support.garmin.com/pt-PT/?faq=aZc8RezeAb9LjCDpJplTY7.

[3] Google. *Encontre e melhore a precisão da sua localização*, (accessed: 29.05.2023). Disponivel em: https://support.google.com/maps/answer/2839911?hl=pt&co=GENIE.Platform%3DAndroid.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Apêndices

A Código main

```
import math
import pickle
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn.model_selection as sk
from joblib import dump
from sklearn import metrics
from sklearn.neural_network import MLPRegressor



def find_out(df, coluna, k):      # Funcao para detetar os outliners, retorna vetor com os index outliners
    u = 0
    v = 0
    out = []            # Vetor com os index outliners
    for i in range(0, len(df.index)):
        u += df[coluna][i]     # Calculo da media
    u = u/len(df.index)
    for i in range(0, len(df.index)):
        v += (df[coluna][i]-u)**2           # Calculo da somatoria da variancia
    v = np.sqrt(v/len(df.index))             # Calculo final variancia
    for i in range(0, len(df.index)):
        if abs(df[coluna][i]-u) > k*v:
            out.append(i)
    return out


def remove_out(df, coluna, k):     # Funcao para remover os outliners
    out = find_out(df, coluna, k)
    for i in out:
        df = df.drop(i)
    return df


def previous_out(df,coluna,k):    # Funcao para substituir o outliner pelo valor anterior
    out = find_out(df,coluna,k)
    for i in out:
        if i != 0:
            df[coluna][i] = df[coluna][i-1]
        #else:
         #   df[coluna][i] = df[coluna][i+1]
    return df


def interpolation_out(df,coluna,k):    # Funcao para substituir o outliner
```

```
    pelo valor interpolado
    out = find_out(df,coluna,k)
    for i in out:
        if i < 1:
            i = 1
        elif i > len(df.index)-2:
            i = len(df.index)-2
        df[coluna][i] = (df[coluna][i+1] + df[coluna][i-1])/2
    return df


def plot_dados(df,col):
    n=0
    l=0
    coluna = df_original.columns
    fig, axs = plt.subplots(int(math.ceil(len(coluna)/col)),col)
    fig.suptitle('Dataset')
    for i in coluna:
        axs[n,l].plot(df.loc[:,i])
        axs[n,l].set_title(i)
        if n == (len(coluna)/col - 1):
            n = 0
            l += 1
        else:
            n += 1

    print(len(df.index))
    plt.show()


df_original = pd.read_csv("../CI4Iot/Projeto_1/Dataset/Lab6-Proj1_Dataset.
    csv")

### Pre processamento do dataset
## Loop para remover os outliers do dataset
k = 1.5
for colunas in df_original.columns:
    df = interpolation_out(df_original, colunas, k)


##Loop para padronizar os dados
for colunas in df_original.columns:
    df[colunas] = (df[colunas] -  df[colunas].min()) / (df[colunas].max() -
    df[colunas].min()) #Normalizacao
    #df[colunas] = df[colunas] - df[colunas].mean() / df[colunas].std()  #Z-
    score



##Divisao do Dataset
#Train de 80%, test de 10% e validation de 10%
X_train ,X_test, Y_train, Y_test = sk.train_test_split(df.loc[:,[df.columns
    [0],df.columns[1],df.columns[2],df.columns[3],
                                                                    df.
    columns[4]]],df.loc[:,df.columns[5]],test_size= 0.2, random_state=42)
```

```
X_test ,X_val, Y_test, Y_val = sk.train_test_split(X_test,Y_test,test_size=
    0.5, random_state=42)


rede = MLPRegressor(random_state=1,hidden_layer_sizes=(10,),activation='relu
    ',solver ='lbfgs',learning_rate = 'adaptive', max_iter=10000).fit(X_train
    , Y_train)

save = pickle.dumps(rede)
dump(rede, 'rede.joblib')
print("Accuracy test: ", rede.score(X_test,Y_test))
print("Accuracy validation: ", rede.score(X_val,Y_val))

Y_pred = rede.predict(X_val)

print("RMSE: ", np.sqrt(metrics.mean_squared_error(Y_val,Y_pred)))
```

B Código para teste

```
##Algoritimo para teste da rede
import numpy as np
import pandas as pd
import sklearn.model_selection as sk
from joblib import load
from sklearn import metrics


def find_out(df, coluna, k):      # Funcao para detetar os outliners,
    retorna vetor com os index outliners
    u = 0
    v = 0
    out = []             # Vetor com os index outliners
    for i in range(0, len(df.index)):
        u += df[coluna][i]     # Calculo da media
    u = u/len(df.index)
    for i in range(0, len(df.index)):
        v += (df[coluna][i]-u)**2           # Calculo da somatoria da
    variancia
    v = np.sqrt(v/len(df.index))             # Calculo final variancia
    for i in range(0, len(df.index)):
        if abs(df[coluna][i]-u) > k*v:
            out.append(i)
    return out



def interpolation_out(df,coluna,k):    # Funcao para substituir o outliner
    pelo valor interpolado
    out = find_out(df,coluna,k)
    for i in out:
        if i < 1:
            i = 1
        elif i > len(df.index)-2:
```

```
            i = len(df.index)-2
        df[coluna][i] = (df[coluna][i+1] + df[coluna][i-1])/2
    return df


df_original = pd.read_csv("../CI4Iot/Projeto_1/Dataset/Lab6-Proj1_TestSet.csv")

### Pre processamento do dataset
## Loop para remover os outliers do dataset
k = 0.5
for colunas in df_original.columns:
    df = interpolation_out(df_original, colunas, k)
##Loop para normalizar os dados
for colunas in df_original.columns:
    df[colunas] = (df[colunas] -  df[colunas].min()) / (df[colunas].max() - df[colunas].min())

X = df.copy()
X = X.drop([colunas], axis =1)

Y = df.loc[:,df.columns[5]]
rede = load('rede.joblib')

print("Accuracy test: ", rede.score(X,Y))

Y_pred = rede.predict(X)

print("RMSE: ", np.sqrt(metrics.mean_squared_error(Y,Y_pred)))
```