UNIVERSIDADE FEDERAL DA PARAÍBA - UFPB
CENTRO DE ENERGIAS ALTERNATIVAS E RENOVÁVEIS - CEAR
CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Marcelo Miranda Camboim - 2016016785

Segunda Prova

João Pessoa

6 de Março de 2020

Marcelo Miranda Camboim - 2016016785

Segunda Prova

Relatório referente à realização da segunda prova da disciplina Automação Inteligente, com objetivo de obter aprovação parcial na disciplina, na UFPB, referente ao período 2019.2.

Prof. Dr. Juan Maurício Villanueva

Universidade Federal da Paraíba - UFPB
Centro de Energias Alternativas e Renováveis - CEAR
Curso de Graduação em Engenharia Elétrica

João Pessoa

6 de Março de 2020

Conteúdo

Lista de Figuras

1 Introdução

Uma subestação de energia é uma instalação elétrica de alta potência responsável por receber energia de alta tensão das usinas geradoras, reduzir esses níveis e entregar a energia para os consumidores industriais e/ou transmitir a energia em direção aos consumidores residenciais.

Uma subestação terá um limite máximo de potência que pode ser distribuída de acordo com as suas limitações físicas, desse modo, é de extrema importância conseguir estimar a demanda do consumidor, de modo que possam ser feitas expansões no sistema antes que ele chegue em seu limite.

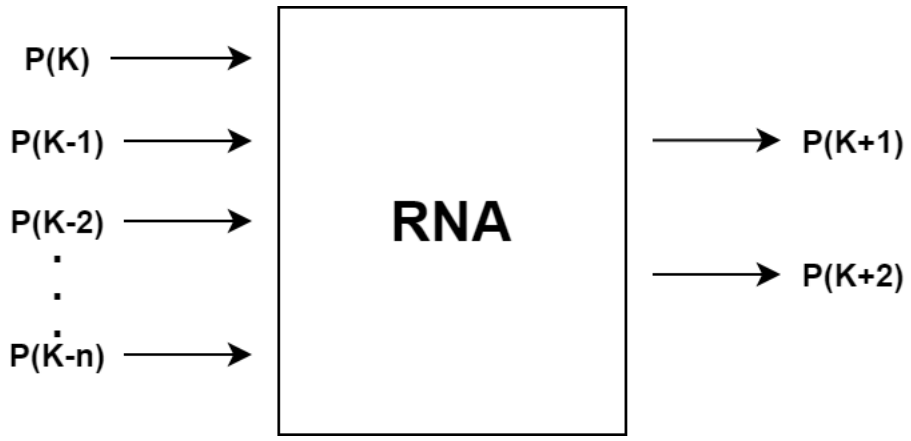

Figura 1: Rede Neural Proposta.

Nesse trabalho será apresentando um sistema de estimação de demanda de potências futuras a partir de demandas passadas utilizado redes neurais artificiais, um exemplo dessa rede é mostrado na Figura 1.

2 Redes Neurais Artificiais

Rede Neural é um tipo de algoritmo de inteligência artificial que visa imitar a capacidade de aprendizado do ser humano. Sua estrutura é formada a partir de neurônios artificiais, esses que são modelados como ilustra a Figura 2.

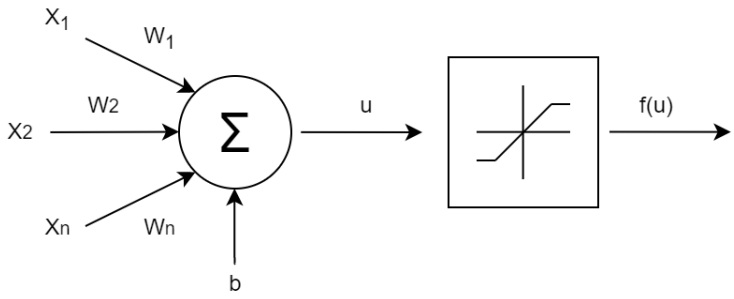

Figura 2: Modelo de um neurônio artificial.

Em que [X_1-X_n] são as entradas, [W_1-W_n] os pesos e b o bias, durante o aprendizado de nossa rede, os valores dos pesos e do bias são recalculados constantemente de modo a adequar a rede à solução desejada, esse processo é chamado de treinamento. Os pesos e o bias representam a forma da rede em armazenar conhecimento.

$$u = X_1 \cdot W_1 + X_2 \cdot W_2 + ... + X_n \cdot W_n + b \tag{1}$$

$$y = f(u) \tag{2}$$

A partir da Figura 2, definimos matematicamente o neurônio utilizando as Equações (1) e (2), em que a função f representa a função de ativação do nosso neurônio artificial.

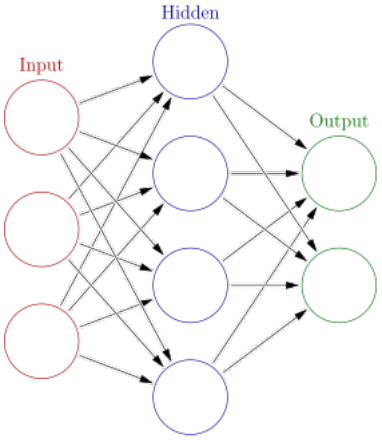

Figura 3: Rede neural com múltiplos neurônios.

Geralmente, nas redes neurais utilizamos topologias com múltiplos neurônios como é visto na Figura 3.

3 Carregamento e Tratamento dos Dados

Para iniciar nossa rede neural, inicialmente precisamos dispor de dados, os dados que serão utilizados na rede neural são referentes a uma subestação de energia localizada no estado da Paraíba. Esses dados mostram a potência fornecida pela subestação à cada 15 minutos entre os anos de 2008 e 2014.

Inicialmente, importou-se as bibliotecas necessárias para efetuar o carregamento e o tratamento dos dados, Pandas para o carregamento e manipulação, Numpy para efetuar contas de álgebra linear, Matplotlib para visualização dos dados e o módulo Datetime para manipulação de data e hora.

Listing 1: Importando as bibliotecas.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime
%matplotlib inline
```

Posteriormente, carregou-se os dados de potência utilizando a função do pandas para ler arquivos CSV, além disso, removeu-se as colunas vazias.

Listing 2: Carregando os dados.

```
df = pd.read_csv('dados_demanda.csv',sep=',',delimiter=';')
df.dropna(axis=1,inplace=True)
df.head(10)
```

A Figura 4 mostra as 10 primeiras linhas dos dados que foram carregados a partir do arquivo CSV.

Para utilizar algumas funções do Pandas de filtragem de dados a partir de data e hora, foi necessário transformar a data em índice do DataFrame, para isso, utilizou-se o código abaixo. Além disso, também trocou-se o separador decimal de vírgula para ponto.

Listing 3: Organizando os dados.

```
df['TEMPO'] = df[['DIA','MES','ANO','HORA','MINUTO']].apply(lambda x : datetime(year=x[2],mo
df.drop(['MINUTO','HORA','ANO','DIA','MES'],axis=1,inplace=True)
```

	DIA	MES	ANO	HORA	MINUTO	JPS_12B1
0	1	1	2008	0	0	7,90000009536743
1	1	1	2008	0	15	7,69999980926514
2	1	1	2008	0	30	7,69999980926514
3	1	1	2008	0	45	7,40000009536743
4	1	1	2008	1	0	7,40000009536743
5	1	1	2008	1	15	7,40000009536743
6	1	1	2008	1	30	7,40000009536743
7	1	1	2008	1	45	7,19999980926514
8	1	1	2008	2	0	7,19999980926514
9	1	1	2008	2	15	7,30000019073486

Figura 4: Dados Carregados.

```
df.rename(columns={'JPS_12B1': 'POTENCIA'},inplace=True)
df['POTENCIA']=df['POTENCIA'].str.replace(',', '.').astype(float)
df.set_index('TEMPO',inplace=True)
```

As 10 primeiras linhas dos dados tratados são mostradas na Figura 5.

	POTENCIA
TEMPO	
2008-01-01 00:00:00	7.900000e+14
2008-01-01 00:15:00	7.700000e+14
2008-01-01 00:30:00	7.700000e+14
2008-01-01 00:45:00	7.400000e+14
2008-01-01 01:00:00	7.400000e+14
2008-01-01 01:15:00	7.400000e+14
2008-01-01 01:30:00	7.400000e+14
2008-01-01 01:45:00	7.200000e+14
2008-01-01 02:00:00	7.200000e+14
2008-01-01 02:15:00	7.300000e+14

Figura 5: Dados Tratados.

Uma vez que os dados estão no formato adequado, podemos encontrar os valores máximos de potência semanais e diários a partir da função "Resample", como é mostrado abaixo.

Listing 4: Obtendo as máximas potências semanais e diárias.

```
maxima_semanal=df.resample('W').agg(['max'])
maxima_diaria =df.resample('D').agg(['max'])
```

Os dados de potência máxima diária e potência máxima semanal são ilustrados nas Figuras 6 e 7, respectivamente.

Por fim, montou-se o conjunto de dados para efetuar a previsão de demanda, para isso, adicionou-se N colunas com as N potências anteriores, essas que serão as entradas, e K colunas com as K

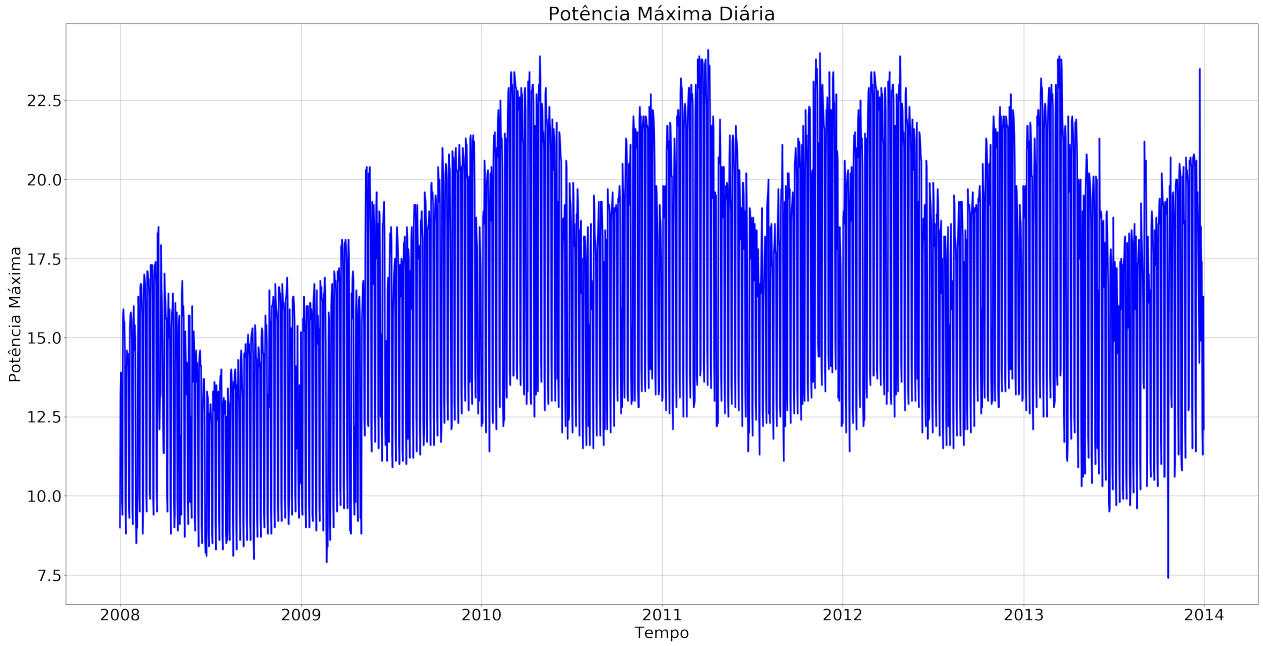

Figura 6: Dados de potência máxima diária.

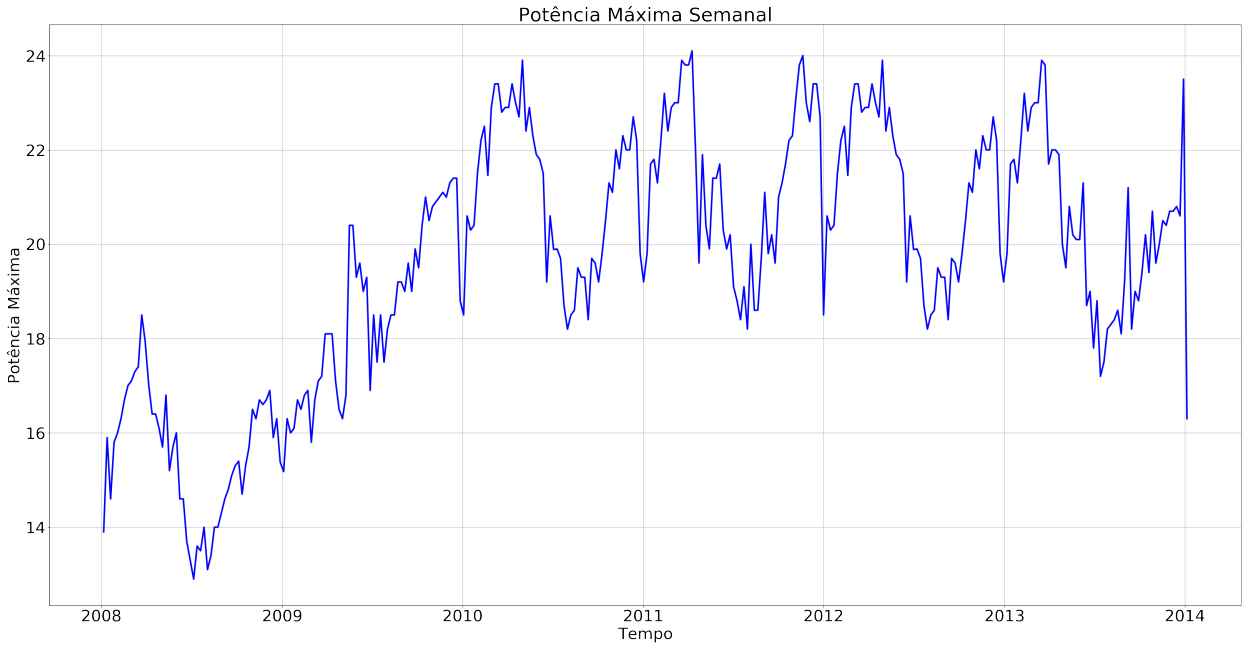

Figura 7: Dados de potência máxima semanal.

potências futuras, essas que serão as saídas. Abaixo é mostrado um exemplo em que temos 4 dados anteriores e iremos prever 2 dados posteriores.

Listing 5: Criando banco de dados para treinamento da rede neural.

```
n_anterior = 4
n_posterior = 2

x = maxima_semanal

for i in list(range(1,(n_anterior+1),1)):
    x_temp = maxima_semanal.shift(i).rename(columns={f'POTENCIA':f"POTENCIA(K-{i})"})
```

```
    x = pd.concat([x_temp,x],axis=1).dropna(axis=0)

for i in list(range(-1,-(n_posterior+1),-1)):
    x_temp = maxima_semanal.shift(i).rename(columns={f'POTENCIA':f"POTENCIA(K+{-i})"})
    x = pd.concat([x,x_temp],axis=1).dropna(axis=0)

x.to_csv('dados_demanda.csv')
```

As 10 primeiras linhas dos dados de treinamento para a rede neural são ilustrados na Figura 8, em que os valores de K até K-4 representam os dados de entrada e os valores de K+1 e K+2 representam os dados de saída.

	TEMPO	POTENCIA(K-4)	POTENCIA(K-3)	POTENCIA(K-2)	POTENCIA(K-1)	POTENCIA	POTENCIA(K+1)	POTENCIA(K+2)
0	03/02/2008	13.900000	15.900000	14.600000	15.800000	16.000000	16.299999	16.700001
1	10/02/2008	15.900000	14.600000	15.800000	16.000000	16.299999	16.700001	17.000000
2	17/02/2008	14.600000	15.800000	16.000000	16.299999	16.700001	17.000000	17.100000
3	24/02/2008	15.800000	16.000000	16.299999	16.700001	17.000000	17.100000	17.299999
4	02/03/2008	16.000000	16.299999	16.700001	17.000000	17.100000	17.299999	17.400000
5	09/03/2008	16.299999	16.700001	17.000000	17.100000	17.299999	17.400000	18.500000
6	16/03/2008	16.700001	17.000000	17.100000	17.299999	17.400000	18.500000	17.928948
7	23/03/2008	17.000000	17.100000	17.299999	17.400000	18.500000	17.928948	17.026842
8	30/03/2008	17.100000	17.299999	17.400000	18.500000	17.928948	17.026842	16.400000
9	06/04/2008	17.299999	17.400000	18.500000	17.928948	17.026842	16.400000	16.400000

Figura 8: Dados para treinamento da rede neural.

4 Rede Neural

4.1 Construção

Inicialmente importou-se as bibliotecas para efetuar a construção da rede neural, para isso, utilizou-se o módulo Python TensorFlow.

Listing 6: Importando bibliotecas para a criação e treinamento da rede neural.

```
import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Depois, importou-se os dados de treinamento e normalizou-se os mesmos. O processo de normalização é bastante recomendado, uma vez que a maioria das funções de ativação do módulo TensorFlow

possuem uma saída máxima igual à 1, dessa forma, iremos impedir que as funções de ativação dos neurônios saturem.

Listing 7: Importação e normalização.

```
df = pd.read_csv('dados_demanda.csv',sep=';')
df.drop(['TEMPO'],axis=1,inplace=True)
data = np.array(df)
data = tf.keras.utils.normalize(data,axis=1)
```

Posteriormente, separou-se o conjunto de dados em dados de treinamento e dados de teste, respeitando uma proporção de 70% e 30% para treino e teste respectivamente.

Listing 8: Separação dos dados.

```
x_train = data[0:216,0:5]
y_train = data[0:216,5:7]

x_test = data[216:308,0:5]
y_test = data[216:308,5:7]
```

Tendo os dados organizados, efetuou-se a criação da rede neural. A topologia utilizado apresenta 5 entradas, 2 saídas e 2 camadas ocultas. Utilizou-se como função de ativação de todas as camadas a Sigmoide. Além disso, de modo a encontrar a melhor topologia para a rede, efetuou-se uma varredura, variando o números de neurônios de ambas as camadas de 5 à 16.

Listing 9: Criação da rede neural.

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Input(5))
model.add(tf.keras.layers.Dense(16,activation=tf.nn.sigmoid))
model.add(tf.keras.layers.Dense(16,activation=tf.nn.sigmoid))
model.add(tf.keras.layers.Dense(2,activation=tf.nn.sigmoid))
```

Como função de custo utilizou-se o erro médio quadrático e efetuou-se o treino para um número de iterações igual a 5000.

Listing 10: Parâmetros de treino.

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

model.fit(x_train,y_train,epochs=5000)
```

A relação entre o número de neurônios nas camadas e o erro médio quadrático é mostrada no mapa de calor da Figura 9.

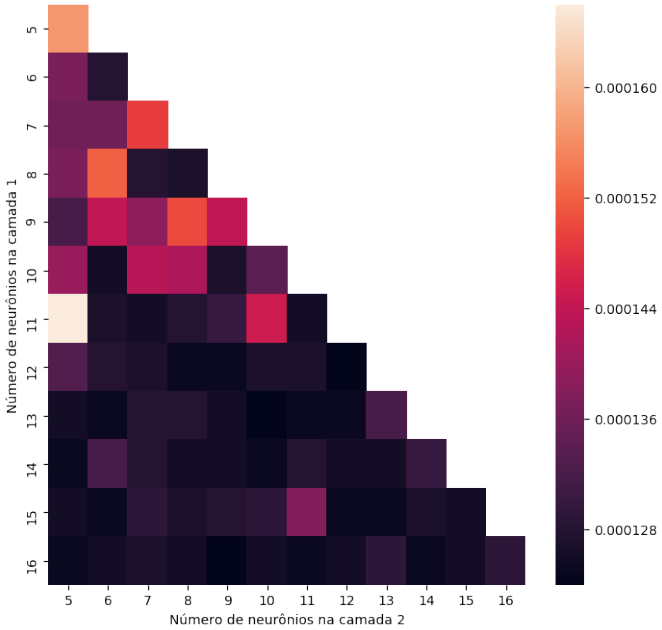

Figura 9: Relação entre o número de neurônios e os erros médios quadráticos.

4.2 Validação

O menor erro quadrático médio foi igual à $1,24 \cdot 10^{-4}$ e foi obtido para uma configuração de 12 neurônios em cada camada. Os valores de potência estimados pela rede neural para uma potência K+1 e para uma potência K+2 são mostrados nas Figuras 10 e 11, respectivamente.

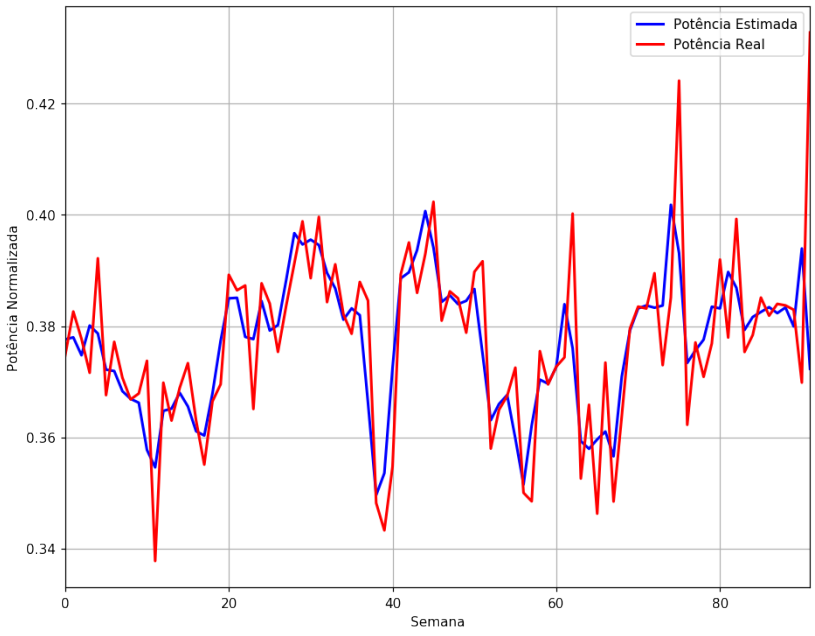

Figura 10: Valores K+1 estimados pela rede neural.

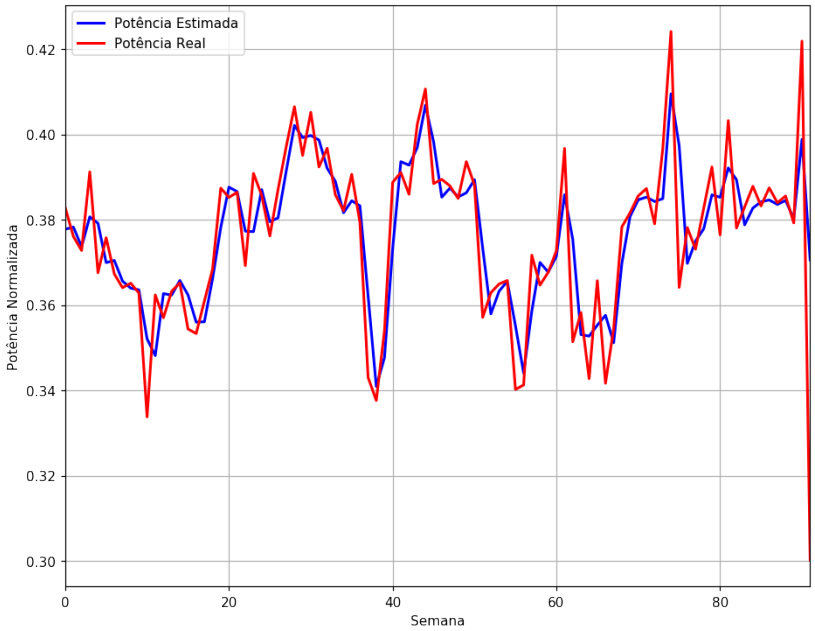

Figura 11: Valores K+2 estimados pela rede neural.

E possível perceber que para ambas as previsões a curva estimada pela rede neural seguiu bem a curva real de teste, apresentando erros relativamente pequenos. Isso fica claro ao vermos o histograma dos erros, em que a distribuição dos erros possui uma forma gaussiana com um pequeno desvio padrão.

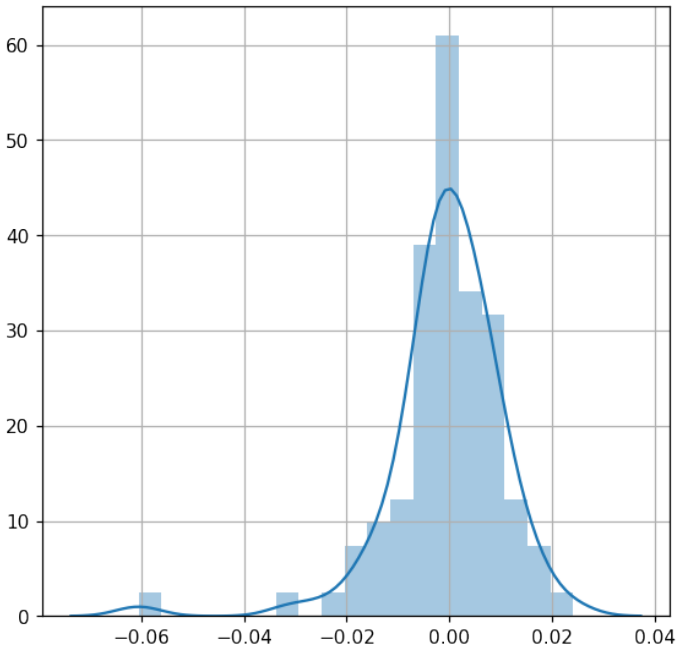

Figura 12: Histograma do erro.

5 Conclusões

Nesse trabalho, utilizando o conhecimento adquirido durante o semestre e a linguagem de programação Python, foi possível criar uma rede neural para efetuar a previsão de demanda de uma subestação localizada no estado da Paraíba, a rede neural criada é capaz de prever a demanda de duas semanas futuras a partir da análise dos dados de 4 semanas anteriores, durante o processo de validação obteve-se um erro quadrático médio igual à $1,24 \cdot 10^{-4}$ para a potência normalizada.