UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA - ITEC
FACULDADE DE ENGENHARIA ELÉTRICA E BIOMÉDICA
CURSO DE ENGENHARIA ELÉTRICA

RODRIGO GOMES DUTRA 201607140080

JULIANO OLIVEIRA DE MIRANDA 201507140034

RELATÓRIO DE BACKPROPAGATION

Belém
2019

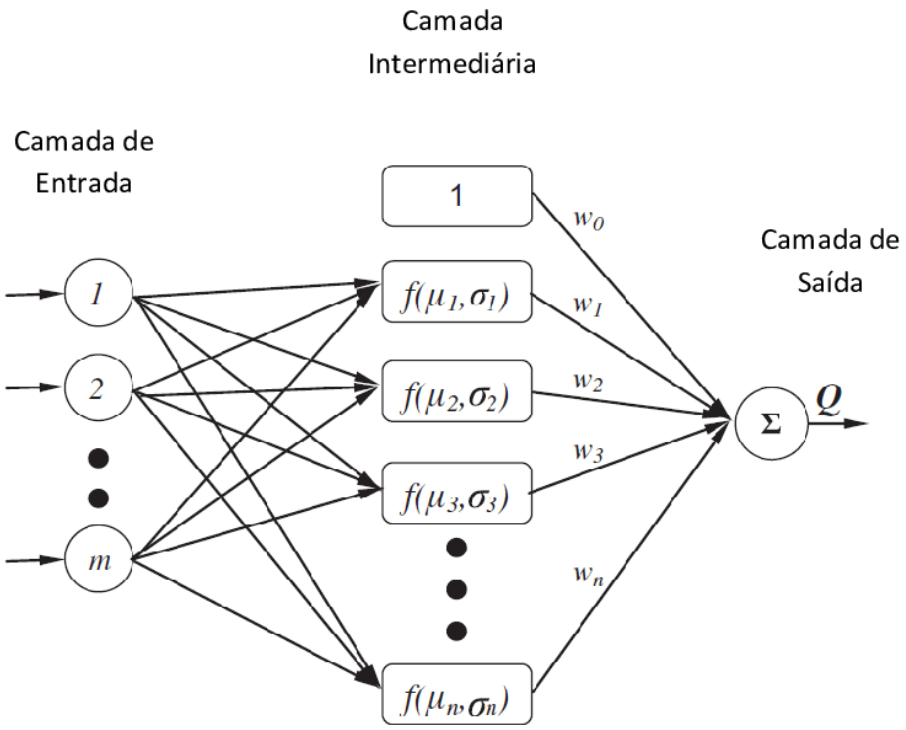
SUMÁRIO

1 INTRODUÇÃO

Este relatório tem como objetivo implementar o algorítimo de Backpropagation utilizando alguma linguagem de programação, para isso foi utilizada a linguagem Python3 utilizando-se do paradigma de orientação a objetos para a implementação do algorítimo de backpropagation em uma rede neural.

A rede neural será estruturada por uma camada de entrada, uma camada escondida e uma camada de saída, dessa forma totalizando 3 camadas. Cada camada terá um número variável de neurônios, sendo assim flexível para a aplicação em diferentes modelos. Além disso, cada camada da rede neural é interligada por pesos e uma função de ativação, neste relatório foi utilizado a função de ativação *sigmoid*.

Figura 1 – Estrutura da rede neural



Para testar o funcionamento da rede neural implementada em Python3 foi selecionada a equação da tensão de um capacitor em um circuito de RC série. Dessa forma, a rede neural tentará alcançar o resultado da equação da tensão do capacitor tendo as mesmas entradas da equação, assim sendo um problema de regressão.

2 DESENVOLVIMENTO

Como descrito anteriormente a rede neural será treinada para predizer a saída da equação da tensão de um capacitor em um circuito RC série. Para o desenvolvimento da equação foi definido que o circuito está descarregado para $t < 0$, assim essa equação para $t > 0$ pode ser definida como :

$$V_c = Vin * (1 - e^{\phi}) \tag{2.1}$$

Na qual, Vin é a tensão de entrada do sistema e ϕ pode ser definido como :

$$\phi = \frac{-t}{R * C} \tag{2.2}$$

Onde R é a resistência C é a capacitância e t é o tempo.

A rede neural em questão terá como dados de entrada um vetor tempo T , seguindo um caso no qual a resistência R e a capacitância C são constantes para todo t pertencente ao vetor T. Assim apenas t será uma variável na equação (2.2) e consequentemente na equação (2.1).

O treinamento da rede foi realizado da seguinte forma, primeiramente foi calculado os valores de saída vc utilizando as equações (2.1) e (2.2) para todos valores de t definidos na entrada da função. Os valores de entrada t e de saída vc foram salvos em vetores T e Vc respectivamente. Depois de definidos os vetores de saída e entrada foi realizado a quebra dos vetores em partes de 60% para treino e 40% para teste.

Assim como descrito na introdução, será utilizado o algorítimo de backpropagation para realizar o treinamento da rede. Este algorítimo possui 3 fazes, a propagação para frente(feed forward), o cálculo do erro da saída da primeira fase e a fase de retro propagação do erro (back-propagation).

Na fase de $feedfoward$ a rede neural irá produzir o sinal de saída, de forma que nesse método são utilizadas as entradas, os pesos, bias e a função de ativação de cada neurônio das camadas instanciadas. Dessa forma, será propagado os sinais de entrada mediados pelos pesos, multiplicados pela função de ativação e ajustados pelo Bias de cada camada.

O sinal de saída y_k de cada neurônio pode ser escrito da forma:

$$y_k = f(\sum_{i=0}^{n}(X_i * W_k i) + \theta_j) \tag{2.3}$$

Na equação (2.3), W a representa a matriz de pesos da camada j na qual está situado o neurônio, X o vetor de entrada do neurônio e θ o bias. Os subtítulos i e j significam respectivamente o índex do vetor de entrada do neurônio e a camada na qual o neurônio está situado, k é o índex do neurônio da camada j.

Na equação acima a função $f(x)$ é a função de ativação, nesse relatório será utilizado a função sigmoid. Para cada camada abaixo da camada de entrada, o vetor X_i contem todas as os valores saídas y_k dos neurônios da camada anterior.

Na segunda fase do algorítimo é necessário o calculo do erro de acordo com uma função de perda ou *loss*, como função de perda será utilizado o erro médio quadrático(MSE), que pode ser definida como:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{2.4}$$

Onde Yi é a saída predita pela rede na etapa de feed foward e $\hat{Y}_i$ é a saída da equação a qual a rede neural está sendo treinada para predizer.

A ultima fase do algorítimo de backpropagation é a retro propagação do erro, é nessa fase que será atualizado os dados dos pesos de acordo com o erro entre o valor predito e o valor desejado, para isto o é calculado a derivada parcial do erro em relação aos pesos, assim obtendo o gradiente da função erro.

O gradiente da função de erro indica a direção na qual os pesos devem ser alterados para que seja atingido o mínimo da função de erro. Dessa forma, para calcular o gradiente em questão utilizando a regra da cadeia, temos :

$$\frac{\partial MSE(Y_j,\hat{Y}_j)}{\partial W_j} = \frac{\partial MSE(Y_j,\hat{Y}_j)}{\partial \hat{Y}_j} * \frac{\partial \hat{Y}_j}{\partial Z_j} * \frac{\partial \hat{Z}_j}{\partial W_j} \tag{2.5}$$

Com $\hat{Z}_j$ sendo : $X_j * W_j + \theta_j$, Resolvendo cada componente de derivada parcial da equação (2.5) temos :

$$\frac{\partial MSE(Y_j,\hat{Y}_j)}{\partial W_j} = 2 * (Y_j - \hat{Y}_j) * \partial f(\hat{Y}_j) * X_j \tag{2.6}$$

Onde $f()$ é a função de ativação utilizada na rede neural.

Dessa forma, sendo j o subíndice da camada é possível calcular o gradiente descendente para para atualizar os pesos entre as camadas utilizando a equação (2.6). Entretanto, para utilizar a equação (2.6) é necessário definir a função de ativação, que no caso foi utilizado a função *sigmoid* definida como :

$$f(x) = \frac{1}{1+e^{-x}} \tag{2.7}$$

Visto que para utilização da função (2.6) é necessário a derivada da função de ativação, podemos definir a derivada da função sigmoid como:

$$f'(x) = f(x) - (1 - f(x)) \tag{2.8}$$

Com isto temos todas as equações necessárias para construir uma rede neural simples utilizando uma linguagem de programação como o Python3.

3 IMPLEMENTAÇÃO

Após equacionadas todas as equações que são necessárias para uma rede neural simples, treinada utilizando o algorítimo de backpropagation, é possível construir o algorítimo de uma rede neural, utilizando-se do paradigma de orientação de objetos para tal.

Dessa forma foi construído uma classe chamada *NeuralNetwork* :

```
class NeuralNetwork:
    def __init__(self, x, y, n=4):
        """
        argumentos:
        x: a entrada de treino
        y: a saida desejada no treino
        n: N mero de neur nios na camada escondida
        """
        self.entrada = x
        self.pesos1 = np.random.rand(self.entrada.shape[1],n)
        self.pesos2 = np.random.rand(4,1)
        self.y = y
        self.saida = np. zeros(y.shape)

    def feedforward(self):
        self.layer1 = sigmoid(np.dot(self.entrada, self.pesos1))
        self.layer2 = sigmoid(np.dot(self.layer1, self.pesos2))

        return self.layer2

    def backprop(self):
        d_pesos2 = np.dot(self.layer1.T, 2*(self.y -self.saida)* \
                   sigmoid_derivative(self.saida))
        d_pesos1 = np.dot(self.entrada.T, np.dot(2*(self.y-self.saida)*
    \
                   sigmoid_derivative(self.saida), self.pesos2.T)* \
                   sigmoid_derivative(self.layer1))

        self.pesos1 += d_pesos1
        self.pesos2 += d_pesos2

    def train(self):
        self.saida = self.feedforward()
        self.backprop()

    def predict(self,x):
        self.entrada = x
        self.saida = self.feedforward()
```

```
38          return self.saida
39
40      def mostrar_pesos(self):
41          print("Pesos1")
42          print(self.pesos1.flatten())
43          print("Pesos2")
44          print(self.pesos2.flatten())
```

Listing 3.1 – Rede Neural

Dessa forma a classe (3.1) permite a criação de redes neurais simples de 3 camadas com um número variável de neurônios na camada escondida.

Com o objeto da rede neural e a equações (2.1) e (2.2) foi possível construir um regressor baseado em uma rede neural simples.

Primeiramente foi alocado um vetor de entrada de 0 até 3 com intervalos de 0.01 entre os termos para ser o vetor T de tempo como variável da equação da tensão. Os valores de R e C escolhidos foram respectivamente : 5 e 0.1.

Definidas as entradas foi possível produzir o vetor de tensão Vc para cada índice do vetor de entrada T. Assim, foi realizado a normalização dos dados de T, e após isso os vetores T e V foram transformados em matrizes, ou vetores coluna, nas quais cada linha representava um item de t ou vc.Os vetores de T e V foram então particionados em partes de 60% e 40%, a parte de 60% foi destinada ao treino onde foi introduzida na classe $NeuralNet$, utilizando 4 neurônios na camada escondida.

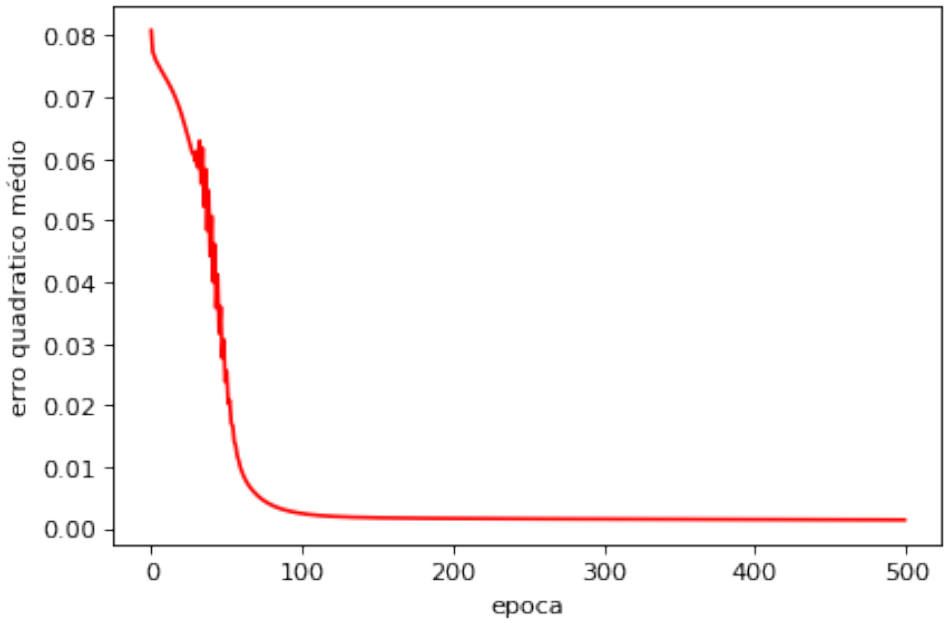
Posteriormente foi realizado o treinamento da rede com o método $train$ do algorítimo (3.1), esse treinamento foi realizado em $loop$ de 500 repetições, calculando-se o erro médio quadrático a cada repetição ou época.

Após o treinamento foi utilizada a parte de 40% da matriz T foi introduzida como argumento do método *predict* dessa forma produzindo o vetor de saída Y o qual foi comparado com a saída real da equação representada pela parte de 40% do vetor Vc.

4 RESULTADOS

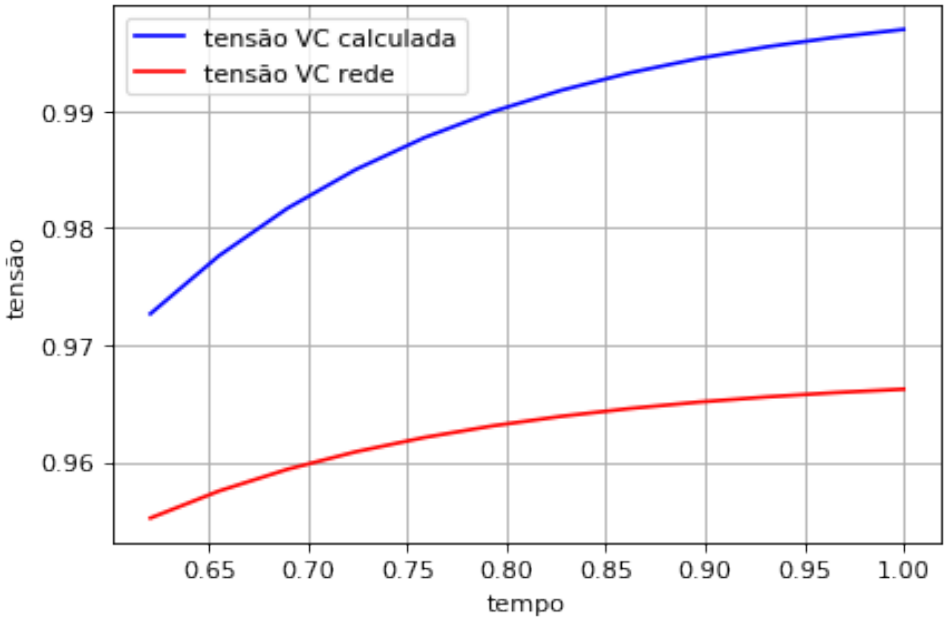
Realizando o procedimento descrito anteriormente foi possivel produzir o gráfico da função erro vs o número de épocas:

Figura 2 – Gráfico do erro



Com a rede treinada após 500 épocas foi predito o valor de saída para os dados de teste:

Figura 3 – Teste da rede neural



A evolução dos pesos pode ser observados nas tabelas abauxo:

Tabela 1 – Pesos antes do treinamento

Neuronio	Pesos 1	Pesos 2
1	0.17750217	0.28842405
2	0.12988619	0.80881573
3	0.78250621	0.3348163
4	0.48084812	0.86665015

Tabela 2 – Pesos após o treinamento

Neuronio	Pesos 1	Pesos 2
1	-6.35187434	-7.08600328
2	0.8177128	-0.79469751
3	2.63663661	1.00681257
4	4.66041337	3.00629355

5 CONCLUSÃO

Esse relatório apresentou a elaboração de uma rede neural simples, a qual utiliza o algorítimo de backpropagation para o seu treinamento, o mesmo demonstrou o funcionamento de rede neural de forma intima, desde os pontos mais internos como pesos e funções de ativação, até os métodos de feed foward e retro propagação do Erro.

Dessa forma o relatório exercita conhecimentos essenciais da disciplina de *Redes Neurais*. Assim, é possível concluir que a elaboração de uma rede neural do 0 é parte importante para o entendimento do funcionamento de uma rede neural complexa, a qual geralmente é aplicada por meio de bibliotecas.

Conclui-se também que até uma rede neural simples, de apenas 4 neurônios na camada escondida, foi capaz de predizer com um pequeno erro a saida da equação da tensão do capacitor.