Instituto Federal Sul Riograndense
Curso Superior em Engenharia Elétrica
Aprendizado de Máquina

Redes Neurais - Multi-Layer Perceptron

Oscar Schmitt Kremer.

Professor Ms. Lucian Schiavon

Pelotas
2019

Conteúdo

1 Objetivo

- Utilizando os dados do exercício 3 (Gaussianas), implementar uma rede neural utilizando o algoritmo *backprogapation*
- Utilizar a validação cruzada na implementação da rede (70-30%).
- Utilizar o valor do erro após cada época como critério de parada. Plotar o valor de erro em relação a cada época após o treinamento.
- Implementar tanto o modo de treinamento online quando o modo batch. Após, comparar qual dos modos converge mais rapidamente.

2 Metodologia

Por meio da linguagem python 3.6.4 implementou-se uma classe para definição da rede neural, onde a classe recebe, de modo arbitrário, o número de camadas e número de neurônios. Este tipo de implementação facilita a generalização, tornando possível que o código venha a ser utilizado em outras aplicações. Para geração dos gráficos foi feito uso da biblioteca matplotlib e para manipulação das matrizes envolvidas nos cálculos do algoritmo de aprendizado *backpropagation* usou-se numpy. A biblioteca scipy foi utilizada para leitura dos dados fornecidos para treinamento e teste, após o carregamento dos dados os mesmos foram separados em 70% para treinamento e 30% teste.

A rede usada tem como parâmetros: 2 neurônios para camada de entrada, 10 para camada intermediária e 1 para camada de saída, learning rate de 0.1 e fator de parada de 0.0001.

3 Resultados

3.1 Sem Batch

O erro quadrático médio do sistema pode ser analisado no gráfico mostrado na figura 1, onde o treinamento atingiu uma sensibilidade de $0,0001$ em 121 épocas, o desempenho para o conjunto de teste resultou em um erro médio de $0,033$.

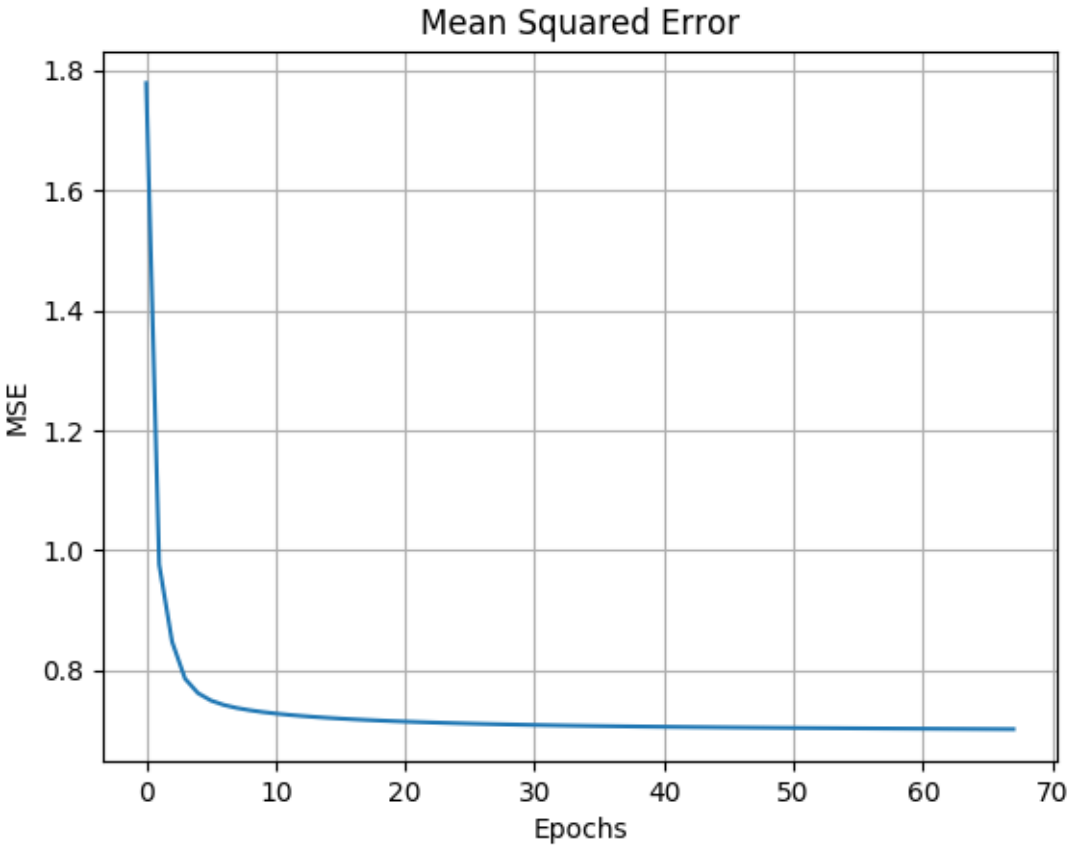

Figura 1: Error Quadrático Médio ao Longo das Épocas

3.2 Com Batch

O erro quadrático médio do para o caso com treinamento com batch aparace no gráfico mostrado na figura 2, onde o treinamento atingiu uma sensibilidade de $0,0001$ em 242 épocas, o desempenho para o conjunto de teste resulta em um erro médio de $1,333$.

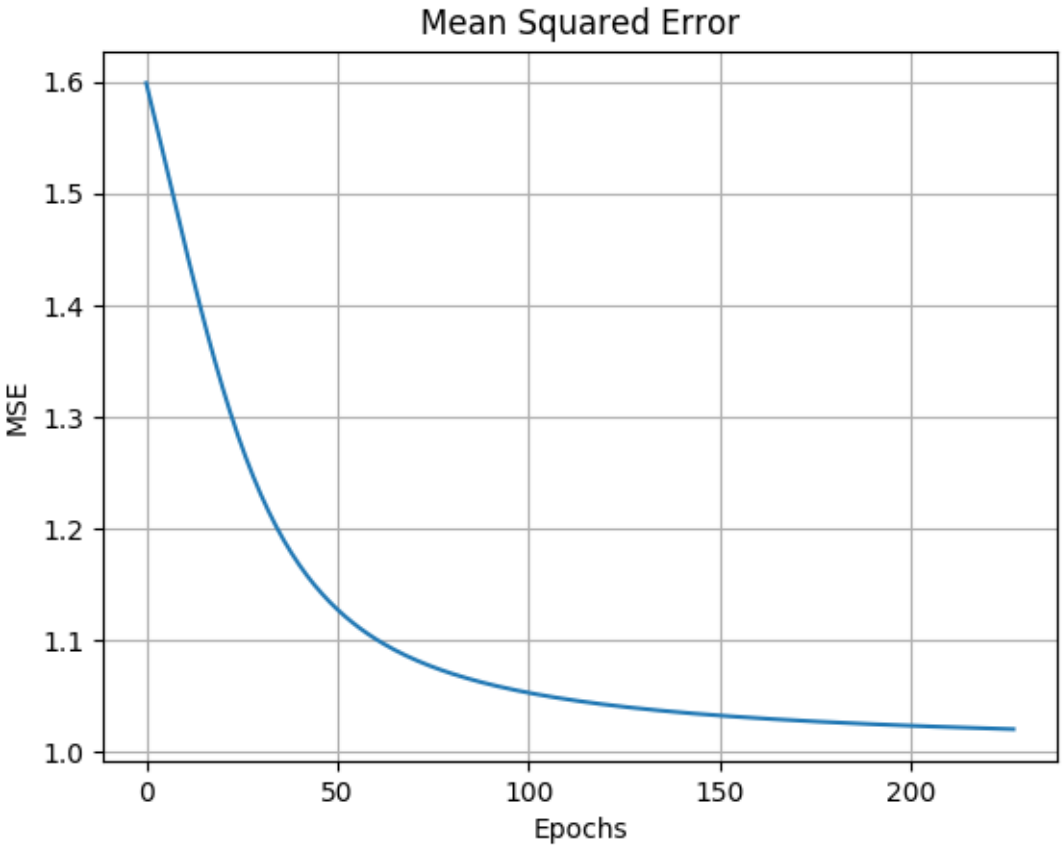

Figura 2: Error Quadrático Médio ao Longo das Épocas

4 Código

```
import scipy.io
import random
import numpy as np
import pandas as pd
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
from matplotlib import pyplot as plt
class MLP:
    def __init__(self, number_neurons, learning_rate,
        momentum, sensibility=0.000001):
        self.number_neurons = number_neurons
        self.sensibility = sensibility
        self.alpha = momentum
        self.eta = learning_rate
        self.initialize_weights()

    def initialize_weights(self):
        self.weights, self.bias = [],[]
        self.weights.append(np.random.rand(self.number_neurons[1],
        self.number_neurons[0], 3))
        self.weights.append(np.random.rand(self.number_neurons[2],
        self.number_neurons[1], 3))
        self.bias.append(np.random.rand(self.number_neurons[1], 3))
        self.bias.append(np.random.rand(self.number_neurons[2], 3))
        self.weights = np.array(self.weights)
        self.bias = np.array(self.bias)

    def fit(self, X, Y):
        self.epochs = 1
        pred = np.zeros(Y.shape[0])
        for j in range(Y.shape[0]):
            pred[j] = self.predict(X[j])
        self.error = []
        old_error = mean_squared_error(Y, pred)
        self.error.append(old_error)
        new_error = old_error
        while(True):
```

```
            old_error = new_error
            for j in range(Y.shape[0]):
                l_1 = self.weights[0][:,:,1].dot(np.transpose(X[j])) -
                self.bias[0][:,1]
                Y_1 = self.activation(l_1)
                l_2 = self.weights[1][:,:,1].dot(np.transpose(Y_1))  -
                self.bias[1][:,1]
                Y_2 = self.activation(l_2)
                delta = (Y[j] - Y_2)*self.dev_activation(l_2)
                self.bias[1][:,2] = (self.alpha+1)*self.bias[1][:,1] -
                self.alpha*self.bias[1][:,0] - self.eta*delta
                self.weights[1][:,:,2] = (self.alpha+1)*self.weights[1][:,:,1] -
                self.alpha*self.weights[1][:,:,0] + self.eta*delta*Y_1
                delta = np.transpose(self.weights[1][:,:,1]).
                dot((delta))*self.dev_activation(l_1)
                self.bias[0][:,2] = (self.alpha+1)*self.bias[0][:,1] -
                self.alpha*self.bias[0][:,0] - self.eta*delta
                self.weights[0][:,:,2] = (self.alpha+1)*self.weights[0][:,:,1] -
                self.alpha*self.weights[0][:,:,0] + self.eta*(delta.reshape(delta.s
                for i in range(2):
                    self.bias[i][:,0] = self.bias[i][:,1]
                    self.bias[i][:,1] = self.bias[i][:,2]
                    self.weights[i][:,:,0] = self.weights[i][:,:,1]
                    self.weights[i][:,:,1] = self.weights[i][:,:,2]

            for j in range(Y.shape[0]):
                pred[j] = self.predict(X[j])
            new_error = mean_squared_error(Y, pred)
            self.error.append(new_error)
            self.epochs+=1
            print(new_error)

            if abs(new_error - old_error) < self.sensibility:
                print(new_error)
                break

    def predict(self, data):
        Y_1 = self.activation(self.weights[0][:,:,1].dot(np.transpose(data)) -
        self.bias[0][:,1])
```

```
        return self.activation(self.weights[1][:,:,1].dot(np.transpose(Y_1)) -
        self.bias[1][:,1])

    def activation(self, x):
        return 1/(1+np.exp(-x))

    def dev_activation(self, x):
        return np.exp(-x)/((1+np.exp(-x))**2)

if __name__=='__main__':
    mat = scipy.io.loadmat('data/raw/data.mat')
    data = mat['data']
    X = data[:,:2]
    y = data[:,2]
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X, y)
    X_train = X_scaled[:int(0.7*len(y))]
    X_test = X_scaled[int(0.7*len(y)):]
    y_train = y[:int(0.7*len(y))]
    y_test = y[int(0.7*len(y)):]
    model = MLP([2, 10, 1], 0.1, 0,sensibility=0.0001)
    model.fit(X_train, y_train)
    pred = []
    print('------ TESTE--------')
    for i in range(y_test.shape[0]):
        prediction = model.predict(X_test[i])
        if prediction > 0.5:
            pred.append(1)
        else:
            pred.append(-1)
    pred = np.array(pred).reshape((y_test.shape[0]))
    print('EPOCAS - {}'.format(model.epochs))
    error_test = (y_test - pred)
    print('ERROR MEDIO TESTE - {}'.format(np.mean(error_test)))
    print('VARIANCIA TESTE - {}'.format(np.std(error_test)**2))
    plt.plot(model.error)
    plt.title('Mean Squared Error')
    plt.grid(True)
    plt.xlabel('Epochs')
```

```
plt.ylabel('MSE')
plt.savefig('plot.png')
```