lib_ppgee_mlp

May 27, 2021

1 Apêndice

1.1 Código do backpropagation desenvolvido pela equipe

[4]:
```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('grayscale')
```

1.2 Bilioteca

[5]:
```
class Neural_Network(object):
  def __init__(self, layers, epochs=200, eta=0.1):
    """
    Para criacao do objeto, deve-se passar um vetor com
    o numero de neuronios em cada camada. Esta classe
    so permite a criacao de rede MLP de uma unica camada
    """
    # parametros
    self.epochs = epochs
    self.eta = eta
    self.rms = []

    self.inputSize = layers[0]
    self.hiddenSize = layers[1]
    self.outputSize = layers[2]

    # pesos
    # np.random.randn - distribuicao "normal" (Gaussiana) de média 0 e␣
↪variância 1
    self.W1 = np.random.randn(self.inputSize, self.hiddenSize) # (30x30) matriz␣
↪de peso da camada de entrada para a oculta
    self.W2 = np.random.randn(self.hiddenSize, self.outputSize) # (30x1) matriz␣
↪de peso da camada oculta para a saída

  def forward(self, X):
    """
    Propagacao direta atraves da rede.
```

```
    Realiza o calculo da saida da rede a partir das entradas
    para isso, e utilizado a funcao de ativacao sigmoid
    """

    self.z = np.dot(X, self.W1) # produto escalar de X (entrada) e primeiro␣
↪conjunto de pesos 30x30
    self.z2 = self.sigmoid(self.z) # função de ativacao
    self.z3 = np.dot(self.z2, self.W2) # produto escalar da camada oculta (z2)␣
↪e segundo conjunto de pesos 30x1
    o = self.sigmoid(self.z3) # Funcao de Ativacao Final
    return o

  def sigmoid(self, s):
    """
    Função de Ativacao (Sigmoide)
    """
    return 1/(1+np.exp(-s))

  def Dsigmoid(self, s):
    """
    Derivada da Sigmoide
    """
    return s * (1 - s)

  def backward(self, X, y, o):
    """
    Aplicacao da propagacao reversa
    aqui e feito o ajuste dos pesos da rede
    """
    self.o_error = y - o # Erro na saída
    self.o_delta = self.o_error*self.Dsigmoid(o) # aplicando a derivada de␣
↪sigmóide ao erro

    self.z2_error = self.o_delta.dot(self.W2.T) # erro z2: quanto nossos pesos␣
↪de camada oculta contribuíram para o erro de saída
    self.z2_delta = self.z2_error*self.Dsigmoid(self.z2) # aplicando derivada␣
↪de sigmóide ao erro z2

    self.W1 += self.eta*X.T.dot(self.z2_delta) # Atualizando o primeiro␣
↪conjunto de pesos (entrada --> oculta)
    self.W2 += self.eta*self.z2.T.dot(self.o_delta) # Atualizando o segundo␣
↪conjunto de pesos (oculta --> saída)

  def train(self, X, y):
    """
    Realiza o treinamento da rede durante self.epochs
```

```
    e armazena o rms na variavel self.rms
    """

    for i in range(self.epochs):
        o = self.forward(X)
        self.backward(X, y, o)
        self.rms.append(np.mean(np.square(y - self.forward(X))/2))

def get_rms(self):
    """
    Coleta a evolucao do erro de treinamento da rede
    na variavel rms
    """
    return self.rms

def saveWeights(self):
    """
    Salva os pesos da rede nos arquivos
    w1.txt e w2.txt
    """
    np.savetxt("w1.txt", self.W1, fmt="%s")
    np.savetxt("w2.txt", self.W2, fmt="%s")
```