Redes Neurais Artificiais

Relatório de Implementação do Algoritmo Backpropagation

UFPA

Professor Adriana Castro

Turma TE05253

Discentes: Micael Fernandes – 201906840029

Implementação do Algoritmo Backpropagation em uma RNA

1. Introdução

Uma rede neural artificial pode ser entendida como uma implementação computacional que visa obter algum dado útil de uma base de dados. Tal rede pode ser implementada de diversas formas visando alcançar algum objetivo. Uma das primeiras redes neurais construídas foi Multi-Layer Perceptron (MLP) ou rede perceptron de múltiplas camadas.

A MLP consiste no cascateamento em diversas camadas de um neurônio básico de uma rede neural, o perceptron. Esse neurônio artificial pode ser dividido em pesos, entradas e saída. Cada parte sua tem uma relação matematica importante que torna possivel gerar algum resultado. Os pesos são os pesos sinápticos que influencia diretamente na saida do neurônio, a entrada é a fonte de dados que entra no neurônio e, influencia pelos pesos, gera uma saída que resulta de um cálculo sistemático. A Figura 1 exemplifica um perceptron.

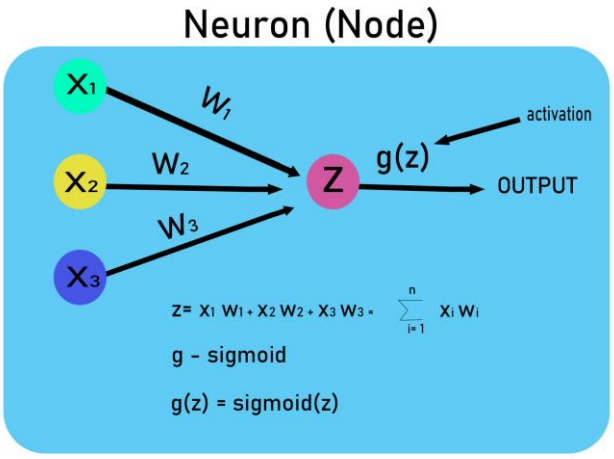

Figura 1. Exemplo de um Perceptron

Note que as relações matemáticas no perceptron envolve uma função especial chamada de função de ativação, a qual existem várias possibilidades de escolha. Ela impacta na maneira que o neurônio irá aprender, bem como a própria rede neural. É importante escolher a função de ativação mais adequada ao problema que se quer explorar/resolver.

Assim, visto o principal componente da rede neural em questão, é importe conhecer a estrutura de um MLP. A Figura 2 mostra um exemplo de MLP.

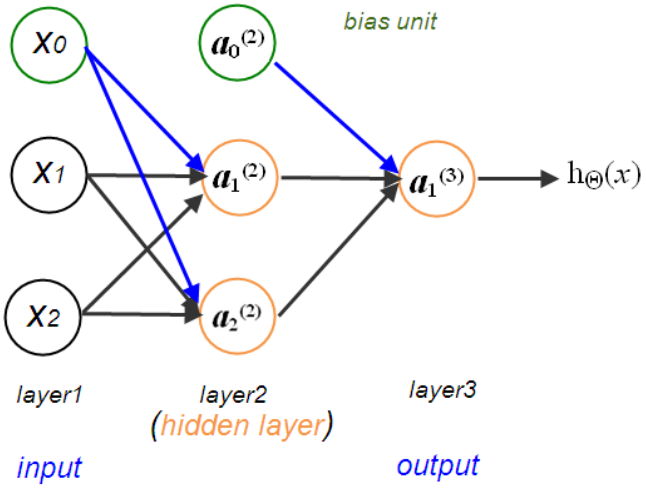

Figura 2. Exemplo de uma MLP

Uma camada dessa rede é um conjunto de perceptrons em sequência, cada um com seus respectivos pesos, entradas e saídas. A primeira camada da MLP é a camada de entrada por onde todos os dados de entrada (features) entram, em seguida a saída dos neurônios da camada de entrada se torna a entrada da primeira camada escondida, e essa repassa sua saída a entrada da próxima camada escondida, e isso se repete até chegar na ultima camada onde a rede neural produz seu resultado final.

2. O algoritmo de aprendizagem

O principal algoritmo de aprendizagem utilizado foi o backpropagation. Onde, com a saída obtida da rede neural, mede-se o erro produzido pela rede e atualiza os pesos sinápticos com base nesse erro. Essa mensuração do erro é retro propagado (de trás para frente) pela rede, de modo a ser possível atualizar todos os pesos da rede. O algoritmo de retropropagação do erro segue os seguintes passos:

1. Inicialização. Inicialize os pesos da rede aleatoriamente ou segundo algum método.
2. Processamento direto. Apresente um padrão à rede. Compute as ativações de todos os neurônios da rede e então calcule o erro.
3. Passo reverso. Calcule os novos pesos para cada neurônio da rede, no sentido retroativo (isto é, da saída para a entrada), camada a camada.
4. Teste de parada. Teste o critério de parada adotado. Se satisfeito, termine o algoritmo, senão volte ao passo 2.

Com essas informações, podemos discorrer a implementação da atividade.

3. Implementação

A implementação é feita em etapas, que vão desde a criação de dados para o treinamento, até a predição final após a construções do algoritmo.

A implementação foi realizada na linguagem orientada objeto Python, visto a grande facilidade de lidar com dados, matrizes e listas, além de possuir bibliotecas uteis para gráficos. O objetivo da rede neural é conseguir classificar qual letra A, B ou C está

sendo exibida numa matriz de entrada para a rede. Utilizando a lógica de programação em python, algumas bibliotecas para visualizar e tratar os dados, foi possível construir um programa que mostra transparentemente o funcionamento de uma MLP.

Inicialmente, é criado 3 matrizes que representam as letras A, B e C. Após isso, é mais adequado passar para a rede neural não uma matriz, mas sim um único array com todos os dados de entrada. Isso é feito transformando a matriz em um array do Numpy (biblioteca matemática e estatística do python). Após esse tratamento, inicia-se o código da MLP em si. A arquitetura resumida da rede implementada pode ser vista na Figura 3 abaixo.

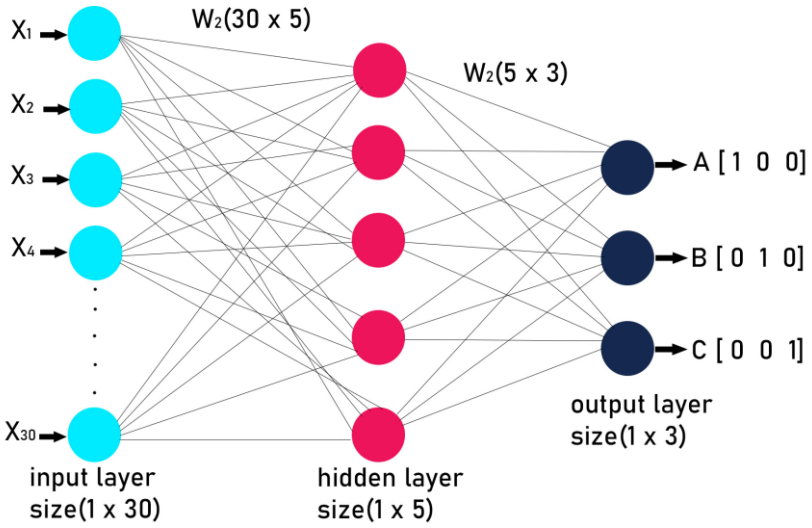

Figura 3. Resumo da rede implementada

Vale lembrar que nessa rede há também a presença do bias que faz parte do algoritmo backpropagation, onde cada camada, exceto a de saída, possui 1 bias. Fora o tratamento de dados, o código principal da rede segue abaixo:

```
import numpy as np
import matplotlib.pyplot as plt

# Função de ativação
def sigmoid(x):
  return(1/(1 + np.exp(-x)))

# Criação da MLP
# 1 Input layer(1, 30)
# 1 hidden layer (1, 5)
# 1 output layer(3, 3)

def f_forward(x, w1, w2,b1,b2):
  # hidden layer
  z1 = x.dot(w1) + b1# entrada vindo da layer 1
  a1 = sigmoid(z1)# saida da layer 2

  # Output layer
  z2 = a1.dot(w2) + b2# entrada vindo da camada anterior
  a2 = sigmoid(z2)# saida da ultima camada
  return(a2)
```

```
# Inicializa os pesos randomicamente (entre 0 e 1)
def generate_wt(x, y):
  l =[]
  for i in range(x * y):
    l.append(np.random.randn())
  return(np.array(l).reshape(x, y))

# Calculo do main square error (MSE)
def loss(out, Y):
  s =(np.square(out-Y))
  s = np.sum(s)/len(y)
  return(s)

# Backpropagation of error
def back_prop(x, y, w1, w2, b1, b2, alpha):
  # hidden layer
  z1 = x.dot(w1)# entrada vindo da layer 1
  a1 = sigmoid(z1+b1)# saida da layer 2

  # Output layer
  z2 = a1.dot(w2)# entrada vindo da camada anterior
  a2 = sigmoid(z2+b2)# saida da ultima camada

  # Erro na camada de saida
  d2 =(a2-y)
  d1 = np.multiply((w2.dot((d2.transpose()))).transpose(),
                (np.multiply(a1, 1-a1)))

  # Gradiente dos pesos
  w1_adj = x.transpose().dot(d1)
  db1 = d1[-1][-1]
  w2_adj = a1.transpose().dot(d2)
  db2 = d2[-1][-1]

  # Atualiza os pesos e bias
  w1 = w1-(alpha*(w1_adj))
  w2 = w2-(alpha*(w2_adj))
  b1 = b1-(alpha*db1)
  b2 = b2-(alpha*db2)

  return(w1, w2, b1, b2)

def train(x, Y, w1, w2, b1, b2, alpha = 0.01, epoch = 10):
  acc =[]
  losss =[]
  for j in range(epoch):
    l =[]
    for i in range(len(x)):
      out = f_forward(x[i], w1, w2, b1, b2)
      l.append((loss(out, Y[i])))
      w1, w2, b1, b2 = back_prop(x[i], y[i], w1, w2, b1, b2, alpha)
    print("Iteração:", j + 1, "======== acc:", (1-
(sum(l)/len(x)))*100)
    acc.append((1-(sum(l)/len(x)))*100)
    losss.append(sum(l)/len(x))
  return(acc, losss, w1, w2, b1, b2)
```

```
def predict(x, w1, w2, b1, b2):
  Out = f_forward(x, w1, w2, b1, b2)
  maxm = 0
  k = 0
  for i in range(len(Out[0])):
    if(maxm<Out[0][i]):
      maxm = Out[0][i]
      k = i
  if(k == 0):
    print("Imagem da letra A.")
  elif(k == 1):
    print("Imagem da letra B.")
  else:
    print("Imagem da letra C.")
  #mostra o resultado
  plt.imshow(x.reshape(5, 6))
  plt.show()
```

No código, é possível ver a utilização constante de lista, tuplas, recursividade e funções matemáticas para se concretizar o algoritmo backprogation. Nesse caso, obedecendo a lógica do python, é construído as funções e equações matemáticas que moldam o backpropagation. A função de ativação escolhida para esse algoritmo é a sigmoide, uma vez que ela apresenta um excelente desempenho para tarefas de classificação. No trecho “def sigmoid(x):” é possível ver como foi definida matematicamente no código.

Utilizando a biblioteca numpy, foi possivel construir as equações matemáticas que fazem parte do algoritimo, principalmente os sistemas lineares que fornecem a saida da rede neural. No trecho código anteriormente mostrado, podemos ver a construção da função “f_foward” que retorna a saída rede neural, que nesse caso é um vetor de tamanho 3 que indica qual letra está sendo representada na matriz. Ao analisar essa função vemos a função “.dot” do numpy que realiza a multiplicação de dois elementos, desse modo, o produto da entrada com os pesos é feita, tanto na primeira quanto na segunda camada, e o resultado é posto na função de ativação sigmoid para assim se obter o resultado. A função “generate_wt” é responsável pela criação dos pesos de maneira randômica, ou seja, atribui valores randômicos (entra 0 e 1) aos pesos de cada camada da rede neural, já os bias é criado separadamente na execução do código. A função “loss” realiza o cálculo do MSE (main square error) da rede neural a partir da saida mensurada e da saída verdadeira que foram passadas como parâmetro.

Ao final do código é criado a função “train” que essencialmente usa todas as outras funções citadas para realizar o treinamento complemento da rede neural a partir de sua entrada, uma taxa de aprendizado fornecida e do número de iterações desejada. Essa função também captura o erro quadráticos médio e a acurácia da rede neural que posteriormente será usado para elabora gráficos sobre a rede. E por fim temos a função “predict” responsável por realizar uma previsão a partir dos dados de entrada fornecidos, ou seja, ela testa a rede neural em si.

A execução da rede consistiu em criar os pesos e os bias e a partir disso utilizar a função “train” passando os parâmetros corretamente. Com isso, temos a rede neural treinada e os resultados obtidos ao final da execução.

4. Resultados

Após o devido treino da rede com 0.2 como taxa de aprendizado, bem como criação dos pesos e instanciações chegou-se nos seguintes resultados:

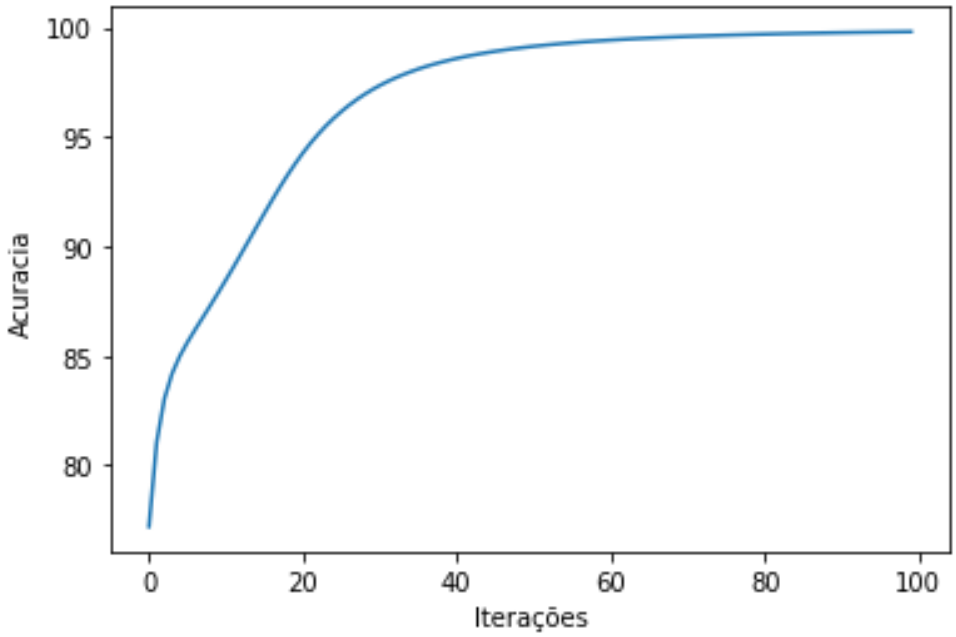

Figura 4. Gráfico Acurácia x Iterações

Na Figura 4 podemos ver a taxa de acerto da rede ao longo das iterações. É visível que a rede neural conseguiu aprender efetivamente a matriz de entrada e assim conseguiu prever melhor qual letra a matriz representa. Há muitos fatores que levaram a esse resultado, como o fato de ser um banco de dados simples e pequeno onde é mais rápido a classificação por meio da MLP, além disso, vale lembrar que esse experimento é didático e extremamente útil para compreender como uma rede neural básica funciona.

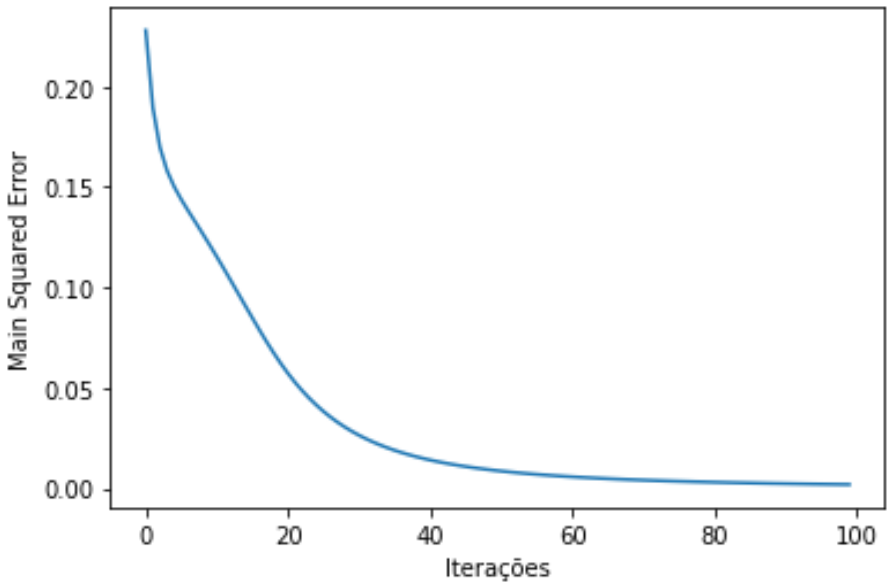

Figura 5. Erro quadrático médio vs Tempo

A Figura 5 mostra o erro de predição da rede pelo tempo, onde podemos ver que é coerente com o gráfico anterior, ou seja, com o avançar das iterações a rede se torna mais assertiva e completa seu treinamento. Vale lembrar que esses gráficos foram obtidos utilizando a biblioteca matplotlib do python que gera facilmente um gráfico informativo.

Ao testar a rede neural com a função “predict” e uma entrada com uma matriz que corresponde a letra B, temos a seguinte saída:

```
8 predict(x[1], w1, w2, b1, b2)
9
```

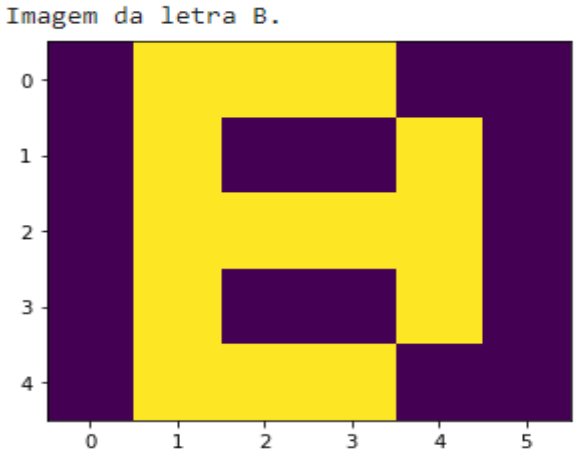

Figura 6. Saída do teste da RNA

Podemos notar que rede neural de MLP conseguiu classificar corretamente a matriz de entrada, mostrando que a rede neural aprendeu corretamente e cumpriu seu objetivo.

5. Conclusão

Com essa implementação, foi possível ver a evolução da rede neural com o passar das iterações, bem como entender a conexão dos neurônios e como um resultado é obtido. No código em python, seguiu-se a ideia de acompanhar o treinamento e visualizar os resultados. Apesar de código criado não ser escalável, flexível ou generalizador, o objetivo principal da implementação foi alcançado com eficácia.

Código fonte: RNA-MLP.ipynb - Colaboratory

Referências

How to Code a Neural Network with Backpropagation In Python (from scratch) (machinelearningmastery.com)

What Is Backpropagation? | Training A Neural Network | Edureka