Universidade Federal do Rio Grande do Sul
INF01017 - Aprendizado de Máquina
Redes Neurais

Emanoel Vianna e Renan Andrades
emanoel.fabiano@inf.ufrgs.br
rsandrades@inf.ufrgs.br

***Resumo*—O segundo trabalho da disciplina de Aprendizado de Máquina consiste na implementação de uma rede neural treinada via backpropagation. Composta por um número ajustável de neurônios e camadas, e que permita via linha de comando, informar a implementação, a estrutura de uma rede de teste e um conjunto de treinamento, e que retorne o gradiente calculado para cada peso.**

I. Introdução

As Redes Neurais Artificiais podem ser definidas como estruturas compostas por elementos de processamento simples adaptáveis densamente interconectados (chamados neurônios artificiais ou nós) que são capazes de realizar cálculos massivamente paralelos para processamento de dados e representação de conhecimento. A ideia de redes neurais artificiais não é replicar a operação dos sistemas biológicos, mas fazer uso do que se sabe sobre a funcionalidade das redes biológicas para resolver problemas complexos [2].

Seu entendimento passa pela noção do que é um perceptron, que é descrito como a soma ponderada de características de entradas associadas à pesos de conexão, normalmente números reais entre -1 e 1. A saída então passa por um limite ajustável, se estiver acima é classificado como 1, caso contrário 0.

Este trabalho é dividido em etapas de implementação, sendo a primeira delas a inicialização dos arquivos com as estruturas necessárias para a construção da rede, são três arquivos, o **network.txt** que contém a arquitetura da rede, onde a primeira linha é o fator de regularização e as linhas seguintes os números de neurônios em cada camadas da rede. O arquivo **initial_weights.txt** contém os pesos iniciais a serem utilizados pela rede, cada linha armazena os pesos dos neurônios de uma dada camada. E por fim o arquivo **dataset.txt** é o banco de dados, onde cada linha representa uma instância e cada coluna representa um atributo.

A segunda etapa de implementação consiste em processar os pesos iniciais associados entrada da rede e gerar uma saída predita, a partir dela então podemos calcular o erro/custo J da rede e rodar o backpropagate e calcular os gradientes e sua verificação numérica.

II. Implementação

A implementação deste trabalho foi feita em Python, especificamente utilizando o aplicativo web de código aberto que permite criar e compartilhar documentos que contêm código ativo, o Jupyter Notebook. Como um dos objetivos também era desenvolver o trabalho em conjunto, nossa dupla optou por utilizar o GitHub como plataforma de controle de versão.

Uma das especificações do trabalho era para não utilizar bibliotecas prontas para o desenvolvimento do trabalho, no entanto as bibliotecas auxiliares de manipulação de banco de dados e para cálculos estatísticos, **pandas** e **numpy** respectivamente poderiam ser utilizadas. Além delas também utilizamos as bibliotecas padrão do pythom, **random** para gerar números pseudoaleatórios, **math** para funções matemáticas.

No entanto neste trabalho nossa dupla enfrentou muitas dificuldades de implementação, e logo no começo do seu desenvolvimento, o que nos impossibilitou realizar os testes mais simples para os bancos de dados necessários para a avaliação. Focamos então em tentar desenvolver os dados de benchmark e debugar nosso código a partir deles, mesmo assim não conseguimos superar as dificuldades dos cálculos do gradiente, que como já mencionamos, parte essencial do trabalho. Conseguimos desenvolver um foward propagate, uma rodada do algoritmo de regressão logística sobre o neurônio sgmoide. Mesmo com algumas dificuldade na parte mais simples, avançamos e geramos uma saída predita conforme o arquivo de benchmark, podemos então calcular os valores de custo de cada exemplo e foram aí que começaram nossas reais dificuldades para avançar, testamos diversas formas de calcular o valor dos deltas e sempre gerando resultados muito diferentes daquilo esperado, o calculo do gradiente foi o grande problema, mesmo buscando outras referências além dos slides da aula algo que fazíamos durante o cálculo estava errado. Infelizmente não avançamos e ainda estouramos o prazo final de entrega pois tentamos mais um dia sem sucesso.

Como o treinamento de uma rede neural ocorre em duas fases, a de propagação e a de adaptação na qual ocorrem os ajustes dos pesos da rede, e conseguimos resolver a etapa de propagação, apesar da fase fundamental do trabalho de adaptação ter sido um desafio que não vencemos, resolvemos submeter mesmo assim o trabalho para alguma avaliação que possa ser construtiva para nós.

III. Funcionalidades

- Interface via linha de comando

Desenvolvemos também, conforme solicitado, uma interface para a compilação do código via linha de comando, em que a seguinte linha de código deve ser executada para testar a funcionalidade:

```
$ python3 backpropagation.py backprop2/network.txt
 backprop2/initial_weights.txt backprop2/dataset.txt
```

Onde separamos em duas pastas os dois exemplos de benchmark para avaliar os testes que fizemos, *backprop1* e *backprop2* respectivamente os exemplos 1 e 2.

- Estrutura dos dados

Para a inicialização dos dados contidos nos arquivos *txt* desenvolvemos as seguintes funções:

```
def read_initial_weights(path_file):
    '''função responsável em ler o arquivo de pesos inicias'''
    return open(path_file,'r+').readlines()

def read_dataset(path_file):
    '''função responsável em ler o arquivo de dataset'''
    dataset = list()
    file = open(path_file, 'r+')
    for lines in file:
        values_line = list()
        dataset.append([lines.rstrip()])
    return dataset

def read_def_network(path_file):
    '''função responsável em ler o arquivo de definição da rede neural'''
    file = open(path_file, "r+")

    network = list()
    for line in file:
        network.append(line.rstrip())
    return network
```

Figura 1. O código helper.py é responsável pela leitura dos arquivos .txt

- Fase de propagação

O sinal de entrada é propagado através de toda a rede, camada por camada, gerando os valores de saída preditos, que posteriormente serão comparados com os valores esperados para cálculo da função de custo.

```
def forward_propagate(network, row):

    inputs = row
    print('a', inputs)
    index = 0
    for layer in network:
        new_inputs = list()
        Zs =  list()
        As =  list()
        if index != len(network)-1:
            new_inputs.append(1.00000)
        for neuron in layer:
            activation = dot(neuron['weights'], inputs)
            Zs.append(activation)
            As.append(sigmoid(activation))
            neuron['output'] = sigmoid(activation)
            new_inputs.append(neuron['output'])
        print('z', Zs)
        print('a', As)
        inputs = new_inputs
        index = index + 1
    return inputs
```

Figura 2. Função responsável em propagar uma instância, e computar a ativação de cada neurônio

IV. Análise da implementação

Apesar das grandes dificuldades ainda algumas coisas puderam ser implementadas, acreditamos que estávamos próximos de encontrar o erro que nos custou o trabalho, por exemplo o cálculo dos deltas, conseguimos gerar o primeiro pois é o erro de saída da rede, simples de ser implementado, no entanto os de neurônios na camada oculta envolvem a soma ponderada dos deltas de neurônios na camada seguinte, multiplicados pela ativação do neurônio, o que não entendemos como fazer uma função para passar os paramentos que tínhamos disponíveis.

Parte fundamental para o calculo dos gradientes era o entendimento da fase de ajustes da rede neural, o calculo dos deltas e dos gradientes por consequência, nos impossibilitou de avançar no trabalho infelizmente.

```
Propagando entrada:  ['0.32000', '0.68000']
a ['1.00000', '0.32000', '0.68000']
z [0.74, 1.1192, 0.35640000000000005, 0.8744000000000001]
a [0.676995856238523, 0.7538402945133331, 0.5881686970847526, 0.7056604209420186]
z [1.9476913803325957, 2.1213580762283635, 1.4815357539775005]
a [0.8751946920726229, 0.8929618051152588, 0.8148044352153714]
z [1.6083096909157375, 1.6680482443653644]
a [0.8331765769648858, 0.8413154279501526]
[INFO] f(x):  [0.8331765769648858, 0.8413154279501526]
[INFO] Valor J:  0.7907366961135718  para a linha  ['0.32000, 0.68000; 0.75000, 0.98000']
[INFO] Saida predita:  [0.8331765769648858, 0.8413154279501526]
[INFO] Saida esperada:  ['0.75000', '0.98000']
[INFO] Derivada:  0.16547217353296462
delta4:  [0.08317657696488578, -0.1386845720498474]
```

Figura 3. Calculo dos neurônios de ativação e valores de custo que conseguimos implementar

V. Conclusão

Escolhemos utilizar a linguagem Python por se tratar de uma linguagem de programação de alto nível e que é amplamente utilizada em aprendizado de máquina, no entanto isso acabou sendo também um dificultador dado que é uma linguagem nova para ambos os membros do grupo. Uma dificuldade que acabou sendo positiva pelo aprendizado na linguagem, mas que acabou nos dificultando o desenvolvimento completo do trabalho.

O desenvolvimento da rede neural, utilizando os dados de benchmark não foi satisfatório, e conseguimos chegar apenas aos resultados esperados para o valor predito *fx*, valor de custo para os dois exemplos e o delta para a camada de saída.

Referências

[1] Data Science Academy. Deep learning book capítulo 12 – aprendizado com a descida do gradiente. https://bityli.com/ckS0c, 2019. [online; Acesso em 6 de novembro de 2020].

[2] Imad A Basheer and Maha Hajmeer. Artificial neural networks: fundamentals, computing, design, and application. *Journal of microbiological methods*, 43(1):3–31, 2000.

[3] Jason Brownlee. How to code a neural network with backpropagation in python (from scratch). https://bityli.com/fE5HX, 2019. [online; Acesso em 26 de outubro de 2020].

[4] Jason Brownlee. How to implement logistic regression from scratch in python. https://bityli.com/ViNF0, 2019. [online; Acesso em 26 de outubro de 2020].

[5] J. Grus. *Data Science from Scratch: First Principles with Python*. O'Reilly Media, 2019.

[6] Jake VanderPlas. *Python data science handbook: essential tools for working with data*. "O'Reilly Media, Inc.", 2016.