INSTITUTO FEDERAL DO SUL DE MINAS GERAIS
Campus Poços de Caldas

Método de Newton para o Aprendizado do Neurônio Artificial

Joaquim Augusto Pomarico[1], Marcos Vinícius Moreira[2], Otávio Messias Palma[3], Pablo Borges Martins[4], Yuri Fernandes de Oliveira[5],

joaquim.cioffi@alunos.ifsuldeminas.edu.br[1],
marcos.moreira@alunos.ifsuldeminas.edu.br[2],
otavio.palma@alunos.ifsuldeminas.edu.br[3],
pablo.borges@alunos.ifsuldeminas.edu.br[4],
yuri.oliveira@alunos.ifsuldeminas.edu.br[5]

Poços de Caldas, 09 de Dezembro de 2019

Resumo: *Para uma rede neural complexa, com parâmetros muito grandes, a tarefa de encontrar o mínimo dentre vários vales contido em tal superfície é muito difícil e custosa. Conforme visto na disciplina, um dos métodos para encontrar tal mínimo é a utilização do Método Gradiente Descendente. Todavia, para redes neurais grandes e com mais de uma camada oculta, o método gradiente se torna menos eficiente. Um dos problemas para uso do método gradiente em grandes redes é o fato do mesmo utilizar método de otimização para função de primeiro grau, assumindo sempre que a superfície da rede é um plano, sem contar com curvaturas. Uma das soluções para os pontos negativos do método gradiente é aplicação do método de Newton. O método de Newton é um método iterativo de aproximação, geralmente utilizado em problemas de otimização de segundo grau e para aproximação de funções.*

1 Introdução

- Utilização do Método de Newton para aprendizado do Neurônio Artificial ao invés do Método do Gradiente visto em sala.
- O algorítimo utilizado foi o RNA desenvolvido em sala junto do professor da disciplina, sendo que foram feitas alterações na evolução dos neurônios com a implementação do Método de Newton.
- Foi utilizada a linguagem de programação Java e as IDEs VSCode e NetBeans.
- Os resultados obtidos não foram compatíveis com os esperados. O método acaba por não convergir diversas vezes, mas quando converge o realiza de forma mais eficiente que o método do Gradiente.

2 Fundamentação teórica

- Em análise numérica, o método de Newton, desenvolvido por Isaac Newton e Joseph Raphson, tem como objetivo estimar raízes de uma função, para isto escolhe-se uma aproximação inicial para esta e calcula-se a equação da reta tangente (através de uma derivada) da função neste ponto e a interseção dela com o eixo das abcissas, a fim de se encontrar uma melhor aproximação para a raiz. Repetindo-se o processo, cria-se um método iterativo para encontrarmos a raiz da função.

- Fórmula do Método de Newton para funções de primeiro grau:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, n \in \mathbb{N} \tag{1}$$

Onde o valor inicial de x é estimado no começo da iteração.

- Fórmula do Método de Newton para funções de segundo grau:

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}, n \in \mathbb{N} \tag{2}$$

Onde o valor inicial de x é estimado no começo da iteração.

- No aprendizado do neurônio, queremos minimizar a fórmula do erro, ou seja, chegar na raiz da função. Utilizando o método de Newton precisamos calcular a derivada primeira e segunda da equação e utilizá-las para atualizar os pesos e o bias.

- Fórmula do erro:

$$erro = (y - f(v(w)))^2 \tag{3}$$

- Derivada do erro:

$$erro' = -2 * x_0 * (y - f(v(w))) \tag{4}$$

- Derivada segunda do erro:

$$erro'' = 2 * x_0^2 \tag{5}$$

- Atualização do pesos do neurônio:

$$w_{n+1} = w_n * \eta * \frac{x_0 * erro}{x_0^2} \tag{6}$$

3 Algoritmo

- O algoritmo é fundamentalmente o mesmo implementado em aulas com algumas pequenas alterações.

- A função de ativação definida para uso foi a Limiar segundo conselho do professor. A rede neural artificial escolhida foi o Perceptron, novamente sob conselho do professor, dessa forma, não é possível lidar com mais de 2 camadas.

- A Rede Neural Artificial Perceptron é inicializada com o número de entradas adequado ao caso e tem a função de ativação configurada junto do número de épocas e dos vetores de aprendizado (entrada e saída esperada).

- Durante o aprendizado, atualiza-se os pesos da entradas a partir do valor inicial estimado, que no caso do algoritmo é gerado de forma aleatória entre 0,0 e 1,0. A cada iteração nova, calcula-se o erro até que este seja zerado, o que significa que o valor desejado foi encontrado.
- Utilizando o método de Newton, diferente do Método do Gradiente, temos a possibilidade de estar dividindo por 0, dessa forma trata-se esse caso somente replicando o valor da entrada para a próxima iteração.
- No final da iteração mostra-se o erro total da época, caso seja 0 a execução foi bem sucedida, caso contrário não.
- No caso de uma execução bem sucedida o valor de saída indicado na linha abaixo será de acordo com os dados de entrada.
- A ideia de usar o algoritmo pronto veio da própria sugestão do professor que nos orientou a somente fazer as mudanças no código ao invés de refatorá-lo inteiro. Pode-se aproveitar do código pronto e focar nas alterações necessárias com mais afinco visto que não precisamos nos preocupar com a funcionalidade do código que ja fora definida como confiável previamente.

4 Recursos de implementação

- Foi utilizado apenas a linguagem de programação Java para realização do trabalho.
- As IDEs NetBeans e VSCode foram utilizadas para escrever o algoritmo, assim como o GitHub foi utilizado como ferramenta para versionamento de código. Além disso, foi utilizado o GitKraken como interface para versionamento.
- Nenhuma biblioteca se mostrou necessária na implementação do Algoritmo.
- Nenhuma base de dados se mostrou necessária na implementação do Algoritmo, apenas entradas baseadas em vetores de inteiros com suas respectivas saídas em outro vetor de inteiros.

5 Código fonte

- A execução do código se dá apenas pela execução da Classe Main.
- Na Classe Main, a variável dados[][] controla os dados de entrada, e saida[] os dados de saída esperada. A variável entrada[] define os valores que serão testados pelos neurônios e irão gerar sua resposta de acordo com o aprendizado feito anteriormente.
- Classe Main

```
1
2 public class Main {
3
4     public static void main(String a[]) {
5         // Define um Perceptron com 2 entradas
6         Perceptron p = new Perceptron(2);
```

```
7
8         // Define a função de Ativação como Limiar
9         FuncaoAtivacao fLimiar = new FuncaoLimiar();
10        p.setFuncaoAtivacao(fLimiar);
11
12        // Define o número de épocas
13        p.setNumeroEpocas(1000);
14
15        // Dados de entrada
16        double dados[][] = { { 0, 0 }, { 0, 1 }, { 1, 0 }, { 1, 1 }
      };
17
18        // Dados de saída
19        double saida[] = { 0, 0, 0, 1 };
20
21        // Realiza o aprendizado do nerônio artificial com os dados
      de entrada e saída
22        p.aprendizado(dados, saida);
23
24        // Entrada de teste, a resposta do neuônio nesse caso deve
      ser 0
25        double entrada[] = { 0, 1 };
26        double resposta = p.getSaida(entrada);
27
28        // Mostra a saída obtida pelo Neurônio Artificial
29        System.out.println("y = " + resposta);
30    }
31 }
```

- Classe Perceptron - Método aprendizado, responsável por atualizar os valores dos pesos.

```
1
2 public void aprendizado(double dados[][], double saidaDesejada[]) {
3         double erroTotal = Double.MAX_VALUE;
4         int epoca = 0;
5
6         // Faz enquanto o erro for diferente de 0 ou quando acabar o
      número de épocas
7         while (erroTotal > 0 && numeroEpocas > epoca) {
8             erroTotal = 0;
9
10            for (int j = 0; j < dados.length; ++j) {
11
12                // f(v(w))
13                double saidaDoPerceptron = getSaida(dados[j]);
14
15                // e = (y - f(v(w)))
16                double erro = saidaDesejada[j] - saidaDoPerceptron;
17
18                for (int i = 0; i < pesos.length; ++i) {
19
```

```
20                      // Caso erro ou entrada seja 0 não realiza a
    conta para não dividir por 0 e mantém o mesmo valor de peso da
    época anterior
21                      if (dados[j][i] != 0 && erro != 0) {
22                          // x(n+1) =  x(n) + ( f(x(n)) / f'(x(n)) )
23                          pesos[i] += (Math.pow(erro, 2)) /
    (dados[j][i] * erro);
24                      }
25                  }
26
27                  // Caso erro seja 0 não realiza a conta para não
    dividir por 0 e mantém o mesmo valor de bias da época anterior
28                  if (erro != 0) {
29                      // b(n+1) = b(n) + ( f(b(n)) / f'(b(n)) )
30                      bias += (Math.pow(erro, 2)) / (erro);
31                  }
32                  erroTotal += Math.sqrt(Math.pow(erro, 2));
33              }
34              epoca++;
35              // Exibe o erro total da época (pode ser no máximo igual
    ao número de entradas)
36              exibirEpoca(erroTotal, epoca);
37          }
38      }
39 }
```

6 Conclusão

- Para aplicação do método de Newton, foram encontradas diversas limitações. O método exige diversos critérios para que os valores venham a convergir e assim obter-se o resultado. No entanto, não foi possível realizar essas verificações sem prejudicar o tempo de execução do código.
- Como os valores iniciais são gerados de forma aleatória, em algumas execuções obtivemos os resultados esperados, mas na maioria os pesos divergem e tendem ao infinito, impedindo que o neurônio responda o grupo pertencente a entrada em questão.
- O grupo apresentou grande dificuldade em entender e associar o método de newton com o aprendizado do neurônio artificial pois além de haver material escasso para consulta, o código implementado ainda não apresentou o resultado esperado.