Redes_Neurais_Lista1

April 13, 2018

1 Questão 2

1.0.1 Aprendizado supervisionado:

O aprendizado supervisionado pode ser comparado com um cenário em que há a presença de um professor, em que este detém conhecimento do ambiente desconhecido sobre o qual a máquina está tentando aprender. Esse conhecimento prévio é geralmente um conjunto de dados na forma entrada-saída.

Assim, o professor fornece à Rede Neural qual a saída desejada para uma determinada entrada e os parâmetros da rede são ajustados levando em consideração a informação fornecida pelo professor e um sinal de erro (Saída desejada - Resposta do sistema). O objetivo é que, com os ajustes dos parâmetros, seja possível tratar corretamente novas entradas (não-rotuladas).

Aprendizado por regressão e classificação são dois exemplos de aprendizado supervisionado. Em regressão, a saída é numérica. Por exemplo, em uma rede de telefonia celular, pode-se desejar prever qual a vazão de um usuário de acordo com sua localização em relação à estação rádio-base, seu histórico de vazão, etc. Em problemas de classificação, a saída da rede é um código indicando uma classe. Por exemplo, um cenário em que um banco deseja classificar os cliente em grupos de acordo com sua probabilidade de pagar um empréstimo (baixo risco, alto risco, médio risco) de acordo com alguns dados como: idade, salário, etc.

1.0.2 Aprendizado não-supervisionado:

Em aprendizado não-supervisionado não há a figura do professor. A rede neural dispõe apenas dos dados de entrada e o objetivo é encontrar padrões. Por exemplo, pode-se operar através do clustering, ou seja, formar grupos de dados por similaridade. Ou seja, o sistema tem o objetivo de representar os dados para tomada de decisões, previsões de entradas futuras, etc.

Um bom exemplo de aprendizado não-supervisionado está em algoritmos de compressão de imagens, em que pixels que representam tons de uma mesma cor podem ser agrupados (clustering).

1.0.3 Semi-supervisionado:

O aprendizado semi-supervisionado permite à rede aprender através de exemplos rotulados e não rotulados. Esse tipo de aprendizado surgiu pois exemplos rotulados são mais raros em comparação a exemplos não rotulados. Assim, os exemplos rotulados são utilizados para o aprendizado inicial, de forma que seja possível continuar o processo com os dados não rotulados.

É possível utilizar o aprendizado semi-supervisionado em problemas de classificação ou clustering. Por exemplo, em problemas de clustering, os exemplos rotulados são usados no processo

de formação dos clusters. Possuir informação prévia sobre os dados que irão formar os cluster geralmente rende melhores resultados.

Um exemplo é o algoritmo Co-training, introduzido em 1998, que precisa apenas de um pequeno conjunto de exemplos rotulados. Seu conceito principal é a utilização de dois classificadores que rotulam exemplos um para o outro, aumentando a precisão do processo de classificação. O artigo original que propôs o Co-training utilizou um experimento consistindo na classificação de páginas web como "página inicial de um curso acadêmico" ou não. O algoritmo classificou corretamente 788 páginas com apenas 12 exemplos rotulados inicialmente (95% de sucesso).

1.0.4 Aprendizado por reforço:

No aprendizado por reforço, assim como no aprendizado não-supervisionado, não há a figura do professor, ou seja, a Rede Neural não possui informação do ambiente. O aprendizado acontece através de uma contínua interação com o ambiente, com o objetivo de minimizar algum indicador de performance.

Em problemas de aprendizagem por reforço, a resposta do sistema é uma sequência de ações. De forma que seu objetivo é minimizar uma função de custo que na verdade representa o acúmulo do custo de diversas ações, tomadas em sequência. Ou seja, o algoritmo deve encontrar as ações que são melhores para o sistema em geral, pois nesse caso uma só ação não é mais importante, mas o conjunto certo de ações que leva ao resultado esperado.

2 Questão 4

2.0.1 Cálculo da matriz de covariância

```
In [15]: import numpy as np
         from numpy.linalg import inv

         #definindo os vetores das variáveis aleatórias
         x1 = np.matrix('5;2;4')
         x2 = np.matrix('4;6;7')
         x3 = np.matrix('3;2;4')
         x4 = np.matrix('3;6;5')

         #definindo os vetores aleatórios como uma matriz
         x = np.matrix('5 2 4; 4 6 7; 3 2 4; 3 6 5')
         print('Esse é o vetor X, formado pelos vetores aleatórios: \n\n',x,'\n')

         #calculando o vetor média
         x_media = np.matrix([ [ np.sum(x[:,0])/3 ], [ (np.sum(x[:,1]))/3] , [(np.sum(x[:,2]))/3
         print('Esse é o vetor média: \n\n',x_media,'\n')

         #definindo a matriz covariância
         x = x.transpose()
         s = np.cov(x)
         print('Matriz de Covariância S:\n\n',s)
```

```
Esse é o vetor X, formado pelos vetores aleatórios:
```

```
 [[5 2 4]
 [4 6 7]
 [3 2 4]
 [3 6 5]]

Esse é o vetor média:

 [[ 5.        ]
 [ 5.33333333]
 [ 6.66666667]]

Matriz de Covariância S:

 [[ 0.91666667 -0.66666667  0.        ]
 [-0.66666667  5.33333333  2.66666667]
 [ 0.          2.66666667  2.        ]]
```

2.0.2 Cálculo da forma fatorada da Matriz de Covariância

```
In [16]: from numpy import linalg as LA

         #cálculo dos autovalores e autovetores
         d,v = LA.eig(s)
         print('Autovalores da matriz S:\n\n ',d,'\n\nAutovetores da Matriz S: \n\n',v)

         #formando a matriz D
         d2 = np.matrix([ [d[0],0,0],[0,d[1],0],[0,0,d[2]]  ])
         print('\n Matriz diagonal dos autovalores (D):\n\n',d2)

         #print(d,'\n\n',v)

         v_1 = inv(v)

         print('\n Matriz S calculada pela forma fatorada: V*D*V^-1 \n\n',v*d2*v_1)
         print('\n É possível perceber que coincide com a matriz S calculada anteriormente.')
```

```
Autovalores da matriz S:

  [ 6.86860959  1.05395536  0.32743504]

Autovetores da Matriz S:

 [[-0.09776694 -0.85145428 -0.51523513]
 [ 0.87285487  0.17534257 -0.45538924]
 [ 0.47808577 -0.4942475   0.72605331]]
```

```
Matriz diagonal dos autovalores (D):

[[ 6.86860959  0.          0.        ]
 [ 0.          1.05395536  0.        ]
 [ 0.          0.          0.32743504]]

Matriz S calculada pela forma fatorada: V*D*V^-1

[[  9.16666667e-01  -6.66666667e-01   5.13653021e-16]
 [ -6.66666667e-01   5.33333333e+00   2.66666667e+00]
 [  4.65733420e-16   2.66666667e+00   2.00000000e+00]]

É possível perceber que coincide com a matriz S calculada anteriormente.
```

2.0.3 Cálculo de S^3 através da forma fatorada

```
In [17]: #calculo de s^3
         v*d2*d2*d2*v_1

Out[17]: matrix([[   3.95543981,  -27.81944444,  -14.66666667],
                 [ -27.81944444,  246.92592593,  135.11111111],
                 [ -14.66666667,  135.11111111,   74.37037037]])

In [18]: #calculo de s^3 de forma direta, para efeito de comparação
         s2 = np.matmul(s,s)
         np.matmul(s,s2)

Out[18]: array([[   3.95543981,  -27.81944444,  -14.66666667],
                [ -27.81944444,  246.92592593,  135.11111111],
                [ -14.66666667,  135.11111111,   74.37037037]])
```

2.0.4 Cálculo da Norma Euclidiana

Calculando a norma Euclidiana da matriz:

```
In [19]: np.linalg.norm(s, ord=2)

Out[19]: 6.8686095911265861
```

Calculando o traço da matriz S para comparação:

```
In [20]: s[0,0]+s[1,1]+s[2,2]

Out[20]: 8.25
```

2.0.5 Norma euclidiana dos vetores aleatórios

```
In [21]: print(np.linalg.norm(x1, ord=2))
         print(np.linalg.norm(x2, ord=2))
         print(np.linalg.norm(x3, ord=2))
         print(np.linalg.norm(x4, ord=2))
```

```
6.7082039325
10.0498756211
5.38516480713
8.36660026534
```

2.0.6 Norma de Mahalanobis dos vetores aleatórios

```
In [22]: print(np.sqrt((x1).transpose()*inv(s)*(x1)))
         print(np.sqrt((x2).transpose()*inv(s)*(x2)))
         print(np.sqrt((x3).transpose()*inv(s)*(x3)))
         print(np.sqrt((x4).transpose()*inv(s)*(x4)))
```

```
[[ 5.94506098]]
[[ 6.48796964]]
[[ 4.34093884]]
[[ 4.90853848]]
```

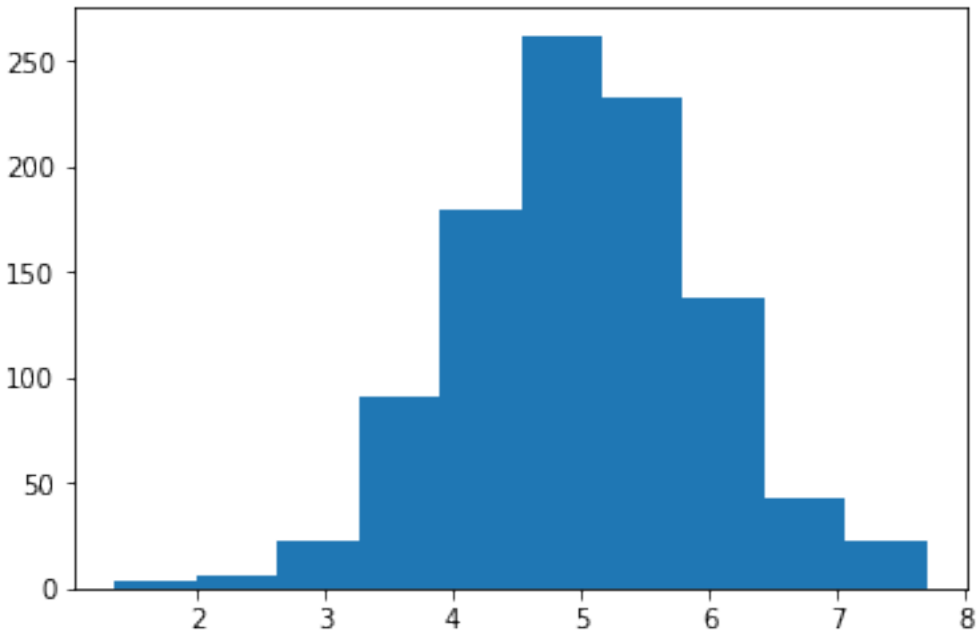
Se a matriz de covariância for a identidade, coincide com a Norma euclidiana:

```
In [23]: print(np.sqrt((x1).transpose()*(x1)))
         print(np.sqrt((x2).transpose()*(x2)))
         print(np.sqrt((x3).transpose()*(x3)))
         print(np.sqrt((x4).transpose()*(x4)))
```

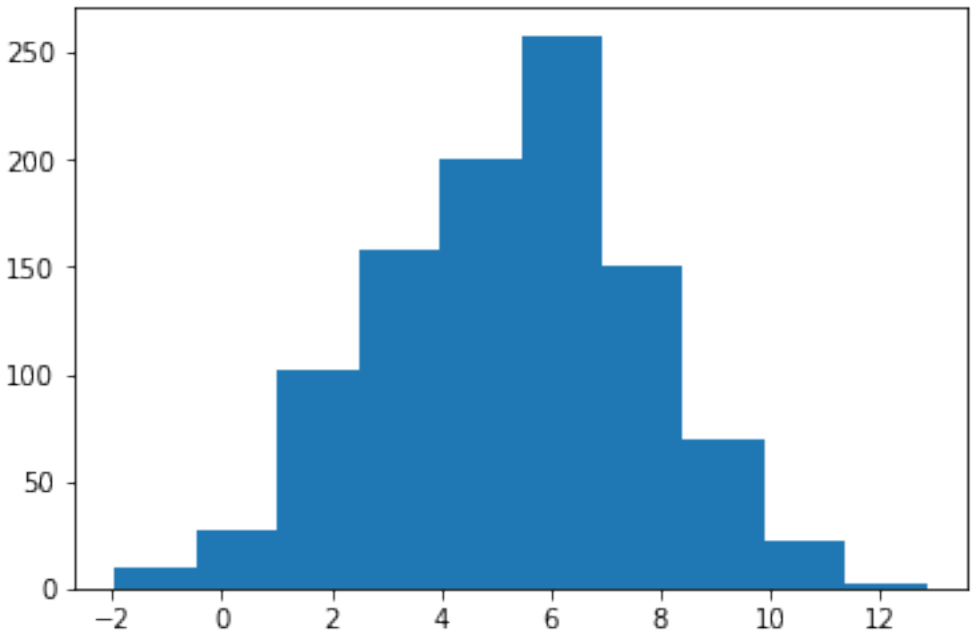
```
[[ 6.70820393]]
[[ 10.04987562]]
[[ 5.38516481]]
[[ 8.36660027]]
```

2.0.7 Distribuição de probabilidade dos vetores aleatórios

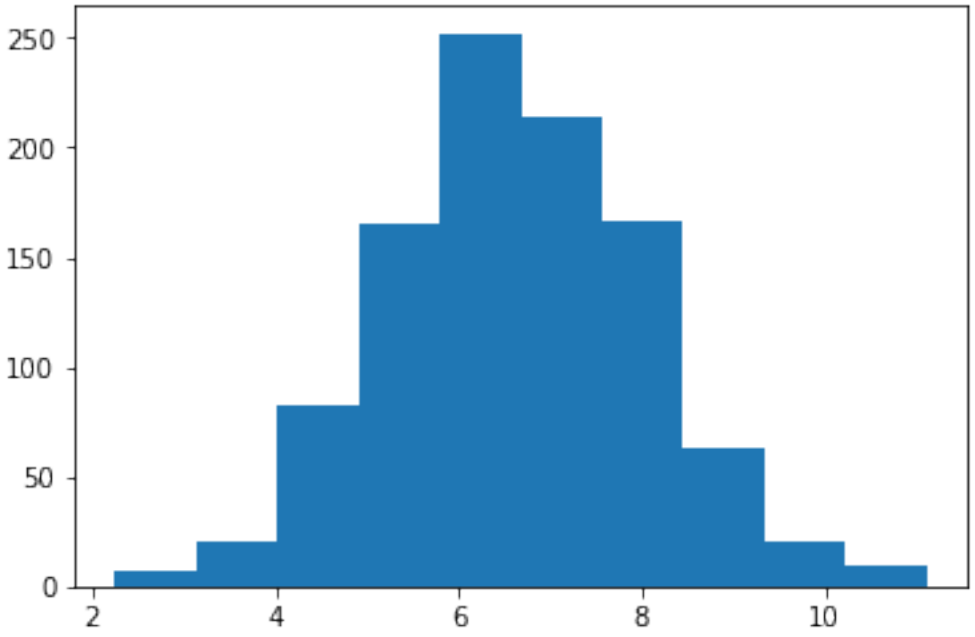
```
In [24]: import matplotlib.pyplot as plt
         g1 = np.random.normal(x_media[0,0], np.sqrt(s[0,0]), 1000)
         plt.hist(g1)
         plt.show()
```



In [25]:
```
import matplotlib.pyplot as plt
g2 = np.random.normal(x_media[1,0], np.sqrt(s[1,1]), 1000)
plt.hist(g2)
plt.show()
```



```
In [26]: import matplotlib.pyplot as plt
         g3 = np.random.normal(x_media[2,0], np.sqrt(s[2,2]), 1000)
         plt.hist(g3)
         plt.show()
```



3 Questão 6

3.0.1 Método do gradiente conjugado

Para esse método consideramos a minimização da função quadrática:

$$f(x) = (1/2)x^T Ax - b^T x + c$$

Como a forma é quadrática, há apenas um vector que minimiza f, que é o ponto crítico, solução de:

$$\nabla f(x) = 0$$

E, nesse caso:

$$f(x) = \nabla(1/2)(A^T + A)x - b = Ax - b$$

Assim, se encontrarmos o ponto de mínimo, ele será solução do sistema linear $Ax = b$
Consideramos métodos iterativos de optimização do tipo:

$$x^{(n+1)} = x^{(n)} + a_n r^{(n)}$$

de forma a que haja uma descida, ou seja, $f(x^{(n+1)}) < f(x^{(n)})$. O vector $r^{(n)}$ define a direcção de descida.

Dois vetores **x** e **y** são ditos com direção conjugadas se:

$$< u, v >_A = u.Av = u^T Av$$

$$< u, v >_A = u.Av = 0$$

Seja N a dimensão da matriz A e sejam d(0), ..., d(N-1) direcções conjugadas
Se considerarmos dˆ{(n)} como direcções de descida, temos a iteração

$$x^{(n+1)} = x^{(n)} + a_n d^{(n)}$$

Queremos agora encontrar o valor a_n que minimiza f, de entre os possíveis valores $x^{(n)} + ad^{(n)}$
De forma, semelhante, podemos obter

$$\frac{d}{da} f(x^{(n)} + ad^{(n)}) = r^{(n)} \cdot d^{(n)} - aAd^{(n)} \cdot d^{(n)}$$

e assim

$$a_n = r^{(n)} \cdot \frac{d^{(n)}}{d^{(n)}.Ad^{(n)}}$$

Sendo que:

$$r^{(n)} = b - Ax^{(n)}$$

Obtemos de forma genérica, um método de direcções conjugadas:

$$x^{(n+1)} = x^{(n)} + \frac{r^{(n)} \cdot d^{(n)}}{d^{(n)} \cdot ad^{(n)}} d^{(n)}$$

mas ainda não definimos as direcções $d^{(n)}$, apenas assumimos que existiam a priori, e eram conjugadas.

Assim, o caso particular do método das direcções conjugadas é o método do gradiente conjugado, em que elas são obtidas através do gradiente.

Recordamos que no caso linear o gradiente é dado pelo resíduo $r^{(n)} = b - Ax^{(n)}$. Através de um processo de ortogonalização de Gram-Schmidt, através dos sucessivos resíduos (gradientes) podemos construir as direcções $d^{(n)}$ que serão conjugadas.

Assim, podemos resumir o Método dos Gradientes Conjugados:

1. Dado $ xˆ{(0)}$ definimos $ dˆ{(0)} = rˆ{(0)} = b-Axˆ{(0)}$
2. Definimos $x^{(n+1)} = x^{(n)} + a_n d^{(n)}$, com $a_n = r^{(n)} \cdot \frac{d^{(n)}}{d^{(n)} \cdot Ad^{(n)}}$
3. Definimos $r^{(n+1)} = r^{(n)} - a_n Ad^{(n)}$, e $d^{(n+1)} = r^{(n+1)} + b_n d^{(n)}$, com $b_n = r^{(n+1)} \cdot \frac{r^{(n+1)}}{r^{(n)} \cdot r^{(n)}}$
4. Regressamos ao 2º passo, até alcançar critério de parada.

4 Questão 8

Função a ser minimizada:

$$f(x) = 8x_1^2 - x_1x_2 + 10x^2 + x_1 + x_2 - 5$$

Está sujeito à restrição abaixo:

$$g_1(x_1, x_2) = x_1 + x_2 = 0$$

Logo:

$$L(x_1, x_2, \lambda) = L(x_1, x_2) - \lambda R(x_1, x_2)$$

$$L(x_1, x_2, \lambda) = 8x_1^2 - x_1x_2 + 10x^2 + x_1 + x_2 - 5 - \lambda x_1 - \lambda x_2$$

Em seguida:

$$\frac{\partial L}{\partial x_1} = 16x_1 - x_2 + 1 - \lambda$$

$$\frac{\partial L}{\partial x_2} = -x_1 + 20x_2 + 1 - \lambda$$

Assim, podemos resolver o sistma com três equações:

$$\frac{\partial L}{\partial x_1} = 16x_1 - x_2 + 1 - \lambda$$

$$\frac{\partial L}{\partial x_2} = -x_1 + 20x_2 + 1 - \lambda$$

$$x_1 + x_2 = 0$$

Resolvendo o sistema:

$$16x_1 - x_2 + 1 = -x_1 + 20x_2 + 1$$

$$17x_1 - 21x_2 = 0$$

$$-17x_1 - 21x_2 = 0$$

Logo, de acordo com as expressões acima:

$$x_1 = x_2 = 0$$

$$\lambda = 1$$