Exercício 6p2

Arthur Felipe Reis Souza

April 21, 2024

1 Regularização

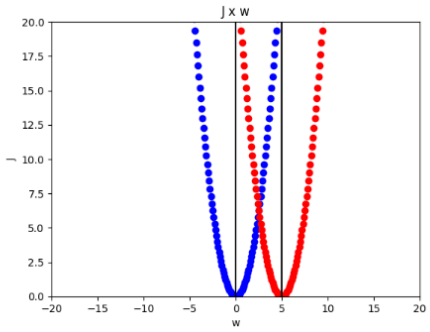
As técnicas de regularização são técnicas utilizadas em algoritmos de Machine Learning para evitar o sobreajuste(overfitting) do modelo. Há várias técnicas de regularização que são úteis para deixar o modelo não inviesado, sendo elas : Lasso Regularization (L1 Regularization), Ridge Regularization (L2 Regularization), DropOut, Elastic Net Regularization, Batch Normalization, Early stopping.

Cada uma dessas técnicas tem o intuito de reduzir o overfitting do modelo, por hora estudaremos as técnicas Lasso Regularization e Ridge Regularization, que consiste na adição de um termo penalizador na função de custo J. A adição desse termo penalizador na função de custo J, tende a reduzir o overfitting, pois agora a função de custo será multivariada, ou seja, terá dois termos que serão minimizadas em conjunto.

Para entender melhor a necessidade desse termo penalizador, é necessário entender a relação entre bias e variance. Bias é a média dos erros do modelo nos dados de treino, enquanto Variance é a média dos erros do modelo para os dados de teste. A adição desse termo penalizador trará um tradeoff conhecido como tradeoff Bias-Variance. Esse tradeoff indica que a adição de um termo penalizador irá reduzir o aumentar o Bias e reduzir o Variance. Ou seja, ele não se ajustará totalmente aos dados de treino e irá errar mais sobre os mesmos. Em contrapartida ele terá um melhor desempenho sobre os dados de teste, sendo uma melhor generalização e se aproximando mais de uma função geradora.

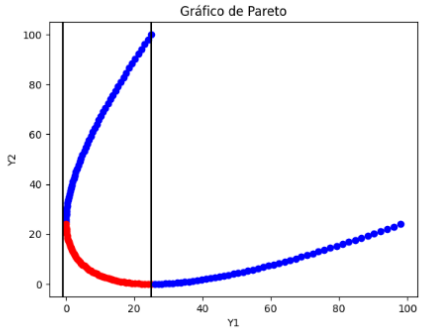
A imagem abaixo retrata o gráfico da função de custo em função dos pesos da rede :

```
[19]: x1_scat = np.arange(start = - 5, stop = 10, step = 0.1)
      Y1_scat = func_generator1(x1_scat)
      Y2_scat = func_generator2(x1_scat)

[20]: np.random.seed(123)
      plt.scatter(x1_scat, Y1_scat, color = 'blue')
      plt.scatter(x1_scat, Y2_scat, color = 'red')
      plt.xlabel('w')
      plt.ylabel('J')
      plt.axvline(x = 0, color = 'black')
      plt.axvline(x = 5, color = 'black')
      plt.xlim(-20, 20)
      plt.ylim(0, 20)
      plt.title('J x w')

[20]: Text(0.5, 1.0, 'J x w')
```



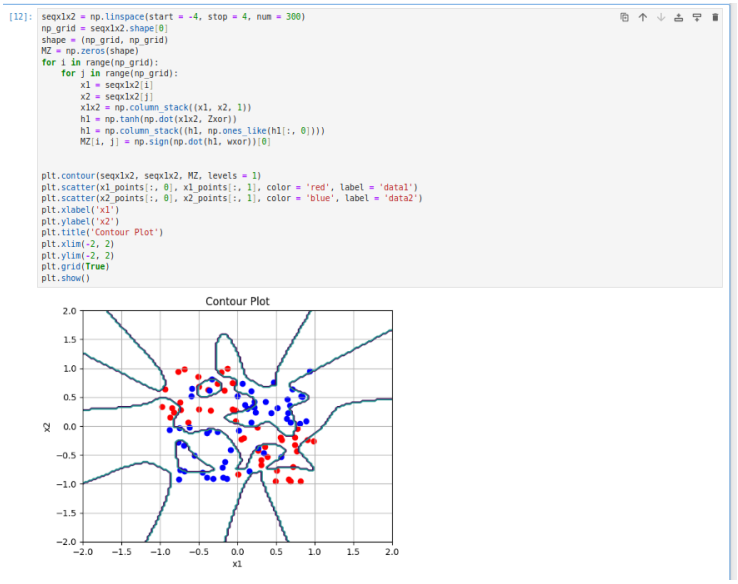
O gráfico de pareto é um gráfico que mostrará as regiões da minimização conjunta desses termos que serão minimizados. Para cada ponto do gráfico de Pareto, teremos 1 relação entre o termo de custo relacionado aos parâmetros w da rede e também da norma do vetor, que limitará o ajuste de parâmetros do modelo.

```
[21]: region1 = (x1_scat <= 0)
      region2 = (0 <= x1_scat) & (x1_scat <= 5)
      region3 = (5 <= x1_scat)

[24]: np.random.seed(123)
      plt.scatter(Y1_scat[region1], Y2_scat[region1], color = 'blue')
      plt.scatter(Y1_scat[region2], Y2_scat[region2], color = 'red')
      plt.scatter(Y1_scat[region3], Y2_scat[region3], color = 'blue')
      plt.title('Gráfico de Pareto')
      plt.axvline(x = 25, color = 'black')
      plt.axvline(x = -1, color = 'black')
      #plt.legend()
      plt.xlabel("Y1")
      plt.ylabel("Y2")
      plt.show()
```

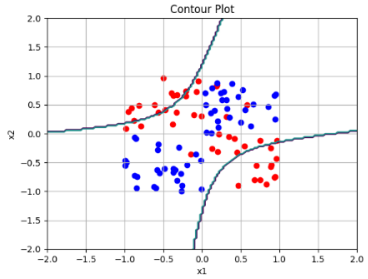


Analisando o gráfico de Pareto, é possível observar que a região destacada em vermelho será uma região onde as respostas serão desejadas, pois terá um erro baixo e uma norma baixa, evitando o overfitting. Portanto, os pontos nessa região são de interesse e também serão estudados pois há um tradeoff entre a função de erro da rede em função dos parametros, e também da norma dos parâmetros da rede.

O exercício é bastante simples e consistirá de apenas alterar, no exercício xor da última semana, valor de λ e observar a suavização da resposta. As imagens abaixo mostram, para o mesmo conjunto p = 100 de neurônios da camada intermediária, diferentes respostas do modelo.



```
[58]: seqx1x2 = np.linspace(start = -4, stop = 4, num = 300)
np_grid = seqx1x2.shape[0]
shape = (np_grid, np_grid)
MZ = np.zeros(shape)
for i in range(np_grid):
    for j in range(np_grid):
        x1 = seqx1x2[i]
        x2 = seqx1x2[j]
        x1x2 = np.column_stack((x1, x2, 1))
        h1 = np.tanh(np.dot(x1x2, Zxor))
        h1 = np.column_stack((h1, np.ones_like(h1[:, 0])))
        MZ[i, j] = np.sign(np.dot(h1, wxor))[0]


plt.contour(seqx1x2, seqx1x2, MZ, levels = 1)
plt.scatter(x1_points[:, 0], x1_points[:, 1], color = 'red', label = 'data1')
plt.scatter(x2_points[:, 0], x2_points[:, 1], color = 'blue', label = 'data2')
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Contour Plot')
plt.xlim(-2, 2)
plt.ylim(-2, 2)
plt.grid(True)
plt.show()
```



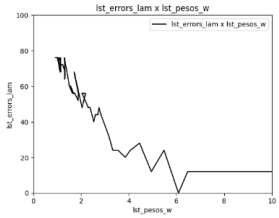
É possível observar, claramente que, enquanto o modelo sem regularização aprende os ruídos, o modelo com regularização não os aprende e tende a ser uma melhor generalização da função geradora do gráfico.

O segundo exercício consiste em plotar o gráfico da parcela do erro referente aos parâmetros w, e a norma dos vetores w. O gráfico está registrado abaixo :

```
[85]: pxor = 100
lst_errors_lam = list()
lst_pesos_w = list()
lam = 0
stop = 5
lst_lambda = list()
while lam <= stop:
    train_ELMspirals = train_ELM(X_trainxor, y_trainxor, pxor, control = True, lam = lam)
    wxor = np.array(train_ELMspirals[0])
    Hxor = np.array(train_ELMspirals[1])
    Zxor = np.array(train_ELMspirals[2])
    y_hatrainxor = test_ELM(X_trainxor, Zxor, wxor, True)
    y_hatestxor = test_ELM(X_testxor, Zxor, wxor, True)

    # Pegando os erros de treino e os erros de teste.
    sum_result = 0
    sum_errors = np.sum((y_trainxor - y_hatrainxor)**2)
    sum_result = sum_result + sum_errors
    lst_errors_lam.append(sum_result)

    lst_lambda.append(lam)
    lst_pesos_w.append(np.linalg.norm(wxor)) # Norma do vetor w.
    lam = lam + 0.1
```

```
[102]: # Transformando as listas em arrays numpy.
lst_errors_lam = np.array(lst_errors_lam)
lst_lambda = np.array(lst_lambda)
lst_pesos_w = np.array(lst_pesos_w)
plt.plot(lst_pesos_w, lst_errors_lam, color = 'black', label = 'lst_errors_lam x lst_pesos_w')
plt.title('lst_errors_lam x lst_pesos_w')
plt.xlabel('lst_pesos_w')
plt.xlim(0, 10)
plt.ylim(0,100)
plt.ylabel('lst_errors_lam')
plt.legend()
plt.plot()
```

```
[102]: []
```



É possível concluir, analisando o gráfico acima que o erro relacionado aos parâmetros w da função de custo tende a estabilizar com uma norma maior do que 6. Ou seja, plotando o gráfico da função de custo, o termo referente a diferença da saída real e saída do modelo se estabilizará em um ponto onde a diferença entre as saídas é 20, para um λ variando de 0 a 5, com 1 passo de 0.01.